Pós-Graduação em Ciência da Computação

Uma Abordagem de Desenvolvimento Orientado a Modelos para a Integração entre Projetos de Mídia e Software no Domínio de Aplicações de TV Digital

Por

Raoni Kulesza

Tese de Doutorado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, FEVEREIRO/2013

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RAONI KULESZA

Uma Abordagem de Desenvolvimento Orientado a Modelos para a Integração entre Projetos de Mídia e Software no Domínio de Aplicações de TV Digital

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: Prof. Silvio Romero de Lemos Meira

RECIFE, FEVEREIRO/2013

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Kulesza, Raoni
Uma abordagem de desenvolvimento orientado a modelos para a integração entre projetos de mídia e software no domínio de aplicações de TV digital / Raoni Kulesza. - Recife: O Autor, 2013.
xxii, 171 p. : il., fig., tab.

Orientador: Silvio Romero de Lemos Meira.
Tese (doutorado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2013.

Inclui bibliografia e apêndice.

1. Engenharia de software. 2. TV digital. I. Meira, Silvio Romero de Lemos (orientador). II. Título.

005.1 CDD (23. ed.) MEI2013 – 088

Tese de Doutorado apresentada por **Raoni Kulesza** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Uma Abordagem de Desenvolvimento Orientado a Modelos para a Integração entre Projetos de Mídia e Software no Domínio de Aplicações de TV Digital"** orientada pelo **Prof. Silvio Romero de Lemos Meira** e aprovada pela Banca Examinadora formada pelos professores:

_____
Prof. Carlos André Guimarães Ferraz
Centro de Informática / UFPE

_____
Prof. Sergio Castelo Branco Soares
Centro de Informática / UFPE

_____
Prof. Vinicius Cardoso Garcia
Centro de Informática / UFPE

_____
Prof. Celso Alberto Saibel Santos
Departamento de Informática / UFES

_____
Prof. Fernando Antonio Mota Trinta
Departamento de Computação / UFC

Visto e permitida a impressão.
Recife, 25 de fevereiro de 2013.

_____
Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Para minha família...

Tek, Tereza, Maité, Uirá, Yuri,

Karla, Ernani, Roberta, Yasmin e Cauê.

AGRADECIMENTOS

Muitas pessoas ajudaram para a realização desta pesquisa, assim, gostaria de dizer, antes de tudo, um Muito Obrigado coletivo, para não esquecer de agradecer ninguém. No entanto, tenho alguns agradecimentos especiais.

Ao professor Silvio Meira pela aceitação de orientação e demonstração de confiança nas propostas da minha pesquisa.

Aos professores Celso e Manoel Neto pela sugestão de tema e parceria nos estudos e discussões que me ajudaram imensamente a alcançar os resultados e contribuições deste trabalho.

Ao professor Guido Lemos e a equipe do LAVID/UFPB que desde 2003 me ajudam dia-a-dia na melhoria significativa dos rumos das minhas pesquisas na área de sistemas multimídia. No contexto deste trabalho, gostaria de agradecer especialmente a Thales Ferreira pela imensa ajuda no amadurecimento das ideias e implementação das ferramentas. Também agradeço à Álan Guedes e Eduardo Alexandre pelos comentários e discussões que me ajudaram a avançar no tema.

Aos professores Carlos Ferraz e Sergio Soares pelos valiosos comentários da qualificação e que, sem eles, o caminho para alcançar os resultados seria muito mais longo. Agradeço também a ajuda de Paola Accioly no entendimento e uso do ferramental estatístico

À Karlinha pela força, amor, carinho, paciência e ajuda até mesmo durante um carnaval.

Ao meu professor-irmão Uirá pelos ensinamentos e ajuda nas áreas de desenvolvimento generativo e linhas de produto de software.

À minha irmã Maité e meu cunhado Nandinho que me incentivaram, me acolheram em Recife e sempre me fizeram acreditar que eu era capaz de terminar este trabalho.

E finalmente, ao meu pai Tek e a minha mãe Tereza, que sempre passaram para mim e meus irmãos o melhor dos seus corações.

Raoni Kulesza

Fevereiro de 2013

“*Quem me acompanha que me acompanhe: a caminhada é longa, é sofrida mas é vivida.”*

"Clarice Lispector”

RESUMO

O processo de convergência digital pode ser encarado sobre duas diferentes perspectivas. Na primeira, a convergência é vista como um casamento de tecnologias ou indústrias, por exemplo, através da integração das redes de Internet, telefonia móvel e televisão digital. Na segunda, a convergência pode ser vista como uma reunião de diferentes tipos de mídia por meio de uma tecnologia única. Este cenário tem favorecido o oferecimento de novos serviços e aplicações multimídia. O desenvolvimento de aplicações multimídia ainda é um desafio. Além da lógica de negócio já existente em outros tipos de software, essas aplicações oferecem uma interface gráfica sofisticada e integrada com diferentes objetos de mídia (imagens, gráficos 3D, áudio e vídeo). Neste contexto, é importante considerar três visões de projeto da aplicação: mídia, software e interface gráfica com o usuário. Atualmente, é possível identificar uma lacuna na definição de métodos sistemáticos e ferramentas de desenvolvimento que considerem esses três aspectos. Este trabalho procura tratar este tema num domínio específico das aplicações multimídia: TV Digital. O objetivo principal é propor uma abordagem de desenvolvimento orientado a modelos que procura uma melhor integração entre os projetos de mídia e software, uma vez que as principais abordagens de desenvolvimento dessa área tratam cada um desses projetos isoladamente. Nesta pesquisa defende-se a tese de que o desenvolvimento orientado a modelos pode aumentar a produtividade do desenvolvimento de aplicações de TV Digital, principalmente, as que possuem como requisito forte integração entre os objetos de mídia e a lógica da aplicação. Em particular, é realizada uma estruturação dos requisitos de uma família de aplicações; configuração das partes comuns e variáveis de cada categoria das aplicações através de um Modelo de *Features*; emprego de linguagens específicas de domínio para modelagem de visões que integram o projeto de mídia e projeto de software; e utilização de técnicas de metaprogramação para geração automática do código das aplicações. Para demonstrar esta tese, é descrita uma abordagem de desenvolvimento orientado a modelos no domínio específico de TV Digital e exemplos de uso. São também apresentados os resultados de uma avaliação envolvendo estudos empíricos, que buscou determinar a viabilidade da abordagem e os benefícios que podem ser alcançados com o emprego da mesma.

Palavras-chave:
Desenvolvimento dirigido por modelos, desenvolvimento generativo, linguagem de domínio específico, ferramentas de autoria, desenvolvimento de aplicações multimídia, televisão digital.

ABSTRACT

The process of digital convergence can be seen from two different viewpoints. In the first, convergence is seen as a merge of technologies or industries, for example, through the integration of Internet, mobile telephony and digital TV networks. In the second, the convergence can be viewed as a collection of different types of media by a unique technology. This scenario has favored offering new multimedia services and applications. The development of multimedia applications is still a challenge. These applications typically provide a sophisticated graphical interface and integrated with different media objects (images, 3D graphics, audio and video). In this context, it is important to consider three visions of application design: media, software and graphical user interface. Currently, it is possible to identify a gap in defining systematic methods and development environment that consider these three aspects. Our study, looks to address this issue in a specific domain of multimedia applications: Digital TV. The main objective is to propose a model-driven development approach that requires a better integration between media and software design, since the main development approaches in this area treat each of these designs individually. This research supports the thesis that model-driven techniques can increase Digital TV development productivity, especially through with a strong integration between media objects and application logic requirements. In particular, it addresses: (i) the specification of requirements for a family of applications; (ii) the configuration of common and variables parts of each category of applications through a feature model; (iii) the usage of domain-specific languages for modeling views that integrates media and software designs; and (iv) the usage of meta-programming techniques for automatic generation of application code. To demonstrate this thesis, we describe a model-driven development approach for the Digital TV domain, and usage examples. We also present the results of an evaluation involving empirical studies to determine the feasibility of the approach and the benefits that can be achieved.

Keywords:
Model-driven development, generative development, domain specific language, authoring tool, multimedia applications development, interactive digital television.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

ABREVIATURAS

ABNT	Associação Brasileira de Normas Técnicas
ACAP	*Advanced Application Platform*
ADL	*Architecture Description Language*
API	*Application Programming Interfaces*
BML	*Broadcast Markup Language*
CIM	*Computational Independent Model*
DSL	*Domain-Specific Language*
DSM	*Domain-Specific Modeling*
DVB	*Digital Video Broadcasting*
EMF	*Eclipse Modeling Framework*
FMP	*Feature Modeling Plugin*
FODA	*Feature Oriented Domain Analysis*
GSD	*Generative Software Development*
HTML	*Hypertext Markup Language*
IM	*Implementation Model*
IPTV	*Internet Protocol Television*
ITU	*International Telecommunication Union*
LPS	Linha de Produtos de Software
MBD	*Model-based Development*
MDA	*Model-driven Architecture*
MDD	*Model-driven Development*
MML	*Multimedia Modeling Language*
MOF	*Meta-Object Facility*
NCL	*Nested Context Language*
NCM	*Nested Context Model*
OMG	*Object Management Group*
OMT	*Object-modeling Technique*
PIM	*Platform Independent Model*
PSM	*Platform Independent Model*
RIA	*Rich Internet Application*
SBTVD	Sistema Brasileiro de TV Digital
TAL	*Template Authoring Language*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*

1 INTRODUÇÃO

O processo de convergência digital pode ser encarado sobre dois diferentes pontos de vista. No primeiro, a convergência é vista como um casamento de tecnologias ou indústrias, por exemplo, através da integração das redes de Internet, telefonia móvel e televisão digital. No segundo, a convergência pode ser vista como uma união de diferentes tipos de mídia por meio de uma tecnologia única (ALFONSI, 2005). Tal cenário tem favorecido o oferecimento de novos serviços e aplicações multimídia.

Estes serviços e aplicações, representados por *software* orientado para o usuário final (por exemplo: aplicações Web "ricas", para dispositivos embarcados, para dispositivos móveis, na área entretenimento e acesso à informações, de ensino à distância, jogos etc.), têm fortalecido a ideia de que uma apresentação gráfica sofisticada contribui para o sucesso da solução. Tais interfaces gráficas geralmente também são extremamente interativas, provendo uma rica experiência que na maioria das vezes faz o uso de elementos multimídia. As principais justificativas para este emprego, segundo Sundar (2004) e Hoogeveen (1997), são: (1) aumentar a eficiência e produtividade da interface com o usuário, (2) obter maior eficácia na transferência da informação e conhecimento; e (3) melhorar o grau de entretenimento no uso do aplicativo de software.

De acordo com Pleuss (2009), uma aplicação multimídia é definida como um software que possui interface com o usuário com pelo menos um objeto de mídia (áudio, vídeo, imagens, animações, gráficos 2D ou 3D) sendo que estes objetos, estão intimamente vinculados à lógica da aplicação. Isso implica que a relação entre os objetos de mídia pode incluir: (i) criação, exclusão, alteração de objetos de mídia a partir da lógica de aplicação em tempo de execução; e (ii) geração de eventos de objetos de mídia propagados para a lógica da aplicação. Já Engels e Sauer (2002), definem uma aplicação multimídia como sistemas de software interativos que unem e apresentam um conjunto de objetos de mídia independentes, de variados tipos, que têm relações espaços-temporais e dependências de sincronização, que podem ser dispostos em hierarquias de composições estruturadas, e podem ser relacionados com um modelo de ação baseado em eventos para possibilitar a interação e navegação. Dessa forma, a concepção dessas aplicações também deve envolver um processo de produção de mídias e tratar a questão da interação humano-computador.

1.1 Desafios atuais

O desenvolvimento de aplicações multimídia ainda é um tema em aberto. Além dos problemas relacionados à lógica de negócio, que devem ser tratados pela engenharia de software tradicional, essas aplicações tipicamente oferecem uma interface gráfica sofisticada e integrada com objetos de mídia (imagens, gráficos 3D, áudio e vídeo). Consequentemente, o processo de desenvolvimento deste

tipo de aplicação deve considerar o envolvimento de especialistas das áreas de *projeto de software*, *projeto de mídias* e *projeto de interação* (PLEUSS; HUSSMANN, 2011). O desafio é aliar ao processo de modelagem dos objetos multimídia e interativos, as técnicas tradicionalmente usadas na especificação de sistemas para solução de questões como: estruturação de requisitos, reutilização, comunicação em equipes multidisciplinares, definição de ferramentas etc. (MARQUES NETO, 2011)

Porém, se observamos os esforços anteriores para melhorar o suporte ao desenvolvimento de aplicações multimídia, é possível identificar que a maioria considera apenas um dos aspectos individuais do software. Por exemplo, existem plataformas[1] de distribuição e execução (*middleware*) que tratam questões relacionadas à portabilidade e ao acesso à infraestrutura (persistência de dados, redes de comunicação, segurança etc.) (CRUZ; MORENO; SOARES, 2008). Outros trabalhos procuram apoiar o desenvolvimento de aplicações multimídia oferecendo níveis mais altos de abstração e reutilização para o projeto de software. Engels e Sauer (2002) classificam esses trabalhos em: (a) interfaces de programação de aplicações (do inglês, *Application Programming Interfaces* – API) e arcabouços (*frameworks*); (b) linguagens e (c) ambientes integrados de programação. Por outro lado, é possível identificar propostas (NACK, 2005; BULTERMAN; HARDMAN, 2005; HANNINGTON; REED, 2007; HARDMAN, 2008; MUSBURGER; KINDEM, 2009; MARQUES NETO, 2011) que tratam questões do processo de produção, sincronismo temporal e disposição espacial de elementos multimídia com o objetivo de facilitar o processo criativo, que é fundamental para o projeto de mídias da aplicação. Estes trabalhos normalmente propõem: (i) modelos de processo, (ii) linguagens de programação declarativas e/ou (iii) ferramentas de autoria. Por fim, existem diversos estudos (PATERNO, 1999; CALVARY, 2002; CESAR, 2009) que se baseiam nos conceitos de desenvolvimento de interface gráfica baseado em modelos (SZEKELY, 1996) para tratar interação da aplicação com o usuário através de padrões de usabilidade e construção de interfaces gráficas.

A constatação é que enquanto a área de Engenharia de Software provê um suporte muito limitado para especificar, projetar e integrar objetos de mídia num software, as abordagens da área de Mídia e Interação focam apenas na criação e autoria dos objetos de mídia e sua interação com os usuários (AMOR; FUENTES; PINTO, 2004; LANG, 2006; PLEUSS et al., 2012).Dessa forma, é possível identificar três problemas no desenvolvimento das aplicações multimídia que são recorrentes (LANG; FITZGERALD, 2005; CAZENAVE; QUINT; ROISIN, 2011): (1) ausência de processos que considerem a natureza interdisciplinar e a importância de requisitos não funcionais que envolve os 3 (três) aspectos do desenvolvimento da aplicação; (2) falta de notações para permitir uma especificação integrada de um sistema em diferentes níveis de abstração e a partir de diferentes e visões do projeto da aplicação; e (3) carência de ambientes de desenvolvimento ou ferramentas de autoria que deem supor-

[1]Neste trabalho, por plataforma entende-se um conjunto coeso de subsistemas e tecnologias onde uma aplicação de software pode ser executada.

te ao desenvolvimento sistemático utilizando metodologias e técnicas definidas em (1) e/ou (2).

Considerando esses três problemas citados e restringindo a trabalhos no domínio específico de TV Digital (SCHWALB, 2004), pesquisas recentes (PLEUSS, 2009; MARQUESNETO, 2011; PROTA, 2012; VANOIRBEEK et al., 2012) apontam que a rota mais curta para resolver esse desafio é adaptar algumas das metodologias usadas na engenharia de software às restrições e peculiaridades do domínio multimídia. Mais especificamente, uma das abordagens de desenvolvimento de software que procura diminuir a complexidade da construção de aplicações de domínio específico é o Desenvolvimento Orientado a modelos (do inglês, *Model-Driven Development - MDD*) (KLEPPE; WARMER; BAST, 2003). A ideia principal do MDD está no fato de considerar a importância dos modelos em um processo de desenvolvimento software, não apenas como uma documentação para tarefas de desenvolvimento e manutenção, mas como parte integrante do software, assim como o código. O MDD permite a transformação de um modelo de concepção de alto nível em uma descrição do conjunto abstrato de elementos que o compõem e, posteriormente, a transformação desses elementos em código para uma plataforma específica.

Apesar das dificuldades técnicas e complexidade extra para se obter o suporte necessário para o uso do MDD, no final pode ser vantajoso empregar esforços extras em busca de seus benefícios, como (KLEPPE; WARME; BAST, 2003): (i) inclusão de uma fase (projeto) intermediária entre a especificação dos requisitos e a implementação; (ii) facilidade de comunicação devido a existência de representações do sistema em vários níveis utilizando uma linguagem específica de domínio (do inglês, *Domain Specific Language* – DSL) e (iii) possibilidade de automatização da geração de código através do uso de motores de transformação entre os modelos e o código final da aplicação. Além disso, existem vários relatos apresentados em Furtado (2012), Kärnä et al.(2009), Tolvanen e Kelly (2008) e Safa (2007) que demonstram que o MDD quando integrado com outras práticas da indústria, pode trazer ganhos de produtividade numa escala de 3 a 10 vezes.

Uma dessas abordagens complementares ao MDD é o Desenvolvimento Generativo de Software (do inglês, *Generative Software Development* - GSD) (CZARNECKI, 2004), que procura diminuir a distância semântica entre o espaço (domínio) do problema e o espaço (domínio) da solução, através de modelos de alto nível que protegem os desenvolvedores de software das complexidades da plataforma de implementação. A principal diferença entre o GSD e outras abordagens MDD é a grande utilização de tecnologias para automatização do desenvolvimento de software como, por exemplo, a meta-programação, a reflexão, a transformação de modelos etc. O objetivo é modelar e implementar "família de sistemas" de um modo que um determinado sistema pode ser automaticamente gerado a partir de especificações que refletem a solução descrita nos modelos, a partir em uma ou mais linguagens de domínio específico.

1.2 Tese

Este trabalho está contextualizado na premissa da literatura de que a combinação de conceitos e técnicas existentes no MDD em uma forma nova e inexplorada pode melhorar a produtividade na implementação de aplicações multimídia, quando comparado com outras abordagens que não utilizam o MDD.

Nesse contexto, o foco desta tese está na identificação de como aplicar o MDD através de uma abordagem de desenvolvimento de software no domínio de aplicações de TV Digital. A principal questão de pesquisa a ser respondida é: **O uso de uma abordagem de desenvolvimento orientado a modelos aumenta a produtividade, em relação a tempo de implementação, do desenvolvimento de aplicações de TV Digital quando comparada com abordagens que não utilizam o MDD?**

Esta pesquisa tem como objetivos esperados a exploração das seguintes questões de pesquisas referentes à união entre tecnologias e ambientes de desenvolvimento de TV Digital e MDD:

- Quais são as etapas necessárias para aplicar o MDD no domínio das aplicações de TV Digital, de forma a tratar sua natureza interdisciplinar de desenvolvimento, mais especificamente, o projeto de software e projeto mídia das aplicações?
- Como representar as características comuns e variáveis no desenvolvimento de aplicação de TV Digital?
- Quais notações utilizar para permitir uma especificação integrada de um sistema em diferentes níveis de abstração e a partir de diferentes visões do projeto em uma aplicação de TV Digital?
- Quais são os artefatos necessários para conseguir uma geração de código (semi) automática de aplicações multimídia no domínio de TV Digital?
- Como garantir que a aplicação gerada pela abordagem seja editada em ferramentas de autoria já existentes de forma a permitir uma edição temporal e gráfica mais avançada?
- Como avaliar o impacto do uso de uma abordagem generativa na produtividade do desenvolvimento de aplicações de TV Digital?

Como forma de avaliar este trabalho foi definida uma abordagem sistemática para a construção de três famílias de aplicações de TVD utilizando técnicas de desenvolvimento generativo de software. As principais contribuições deste trabalho são: (i) estruturação dos requisitos através da classificação de um conjunto de aplicações; (ii) configuração das partes comuns e variáveis de cada categoria das aplicações através de um Modelo de *Features;* (iii) emprego de linguagens de domínio específico para modelagem das aplicações identificadas em (i) e configuradas em (ii); e (iv) utilização de técnicas de metaprogramação para geração automática do código das aplicações. Foram também realizados estudos empíricos visando avaliar a abordagem proposta e determinar o impacto na produtividade do desenvolvimento de aplicações de TV Digital.

1.3 Definição do escopo

É importante deixar claro que o foco deste trabalho não foi pesquisar de forma individual e em profundidade as linhas de pesquisa de MDD e GSD. Ao invés disso, procurou-se combinar as técnicas já estabelecidas em uma metodologia nova, de maneira a gerar os benefícios das abordagens.

Adicionalmente, vale lembrar que foi priorizada a definição de uma abordagem útil do ponto de vista prático, ao contrário de uma cobertura muito extensa do ciclo de vida de desenvolvimento de um software. A principal motivação para esta estratégia foi permitir uma avaliação final através do uso da abordagem em cenários reais e que pudesse ser realizada sem muitos problemas. Acredita-se que assim foi viável a realização de estudos empíricos com um esforço compatível com um trabalho de doutorado e também com avaliações e contribuições mais precisas.

Com base nestes critérios, um primeiro ponto a ser ressaltado é em relação ao desenvolvimento orientado a modelos: o objetivo não está em definir um processo completo, mas as atividades e artefatos GSD relacionados às visões de configuração e mapeamento entre os espaços de problema e solução. Está fora do escopo, portanto, tratar de atividades como testes, manutenção e evolução dos artefatos gerados, bem como realizar a engenharia reversa, através da transformação de código em modelos. Optou-se por esta escolha por uma questão de foco de pesquisa.

Outro aspecto que dá margem a muitas possibilidades é o ambiente e tecnologias de suporte ao MDD. Acredita-se que ferramentas são parte essencial do MDD, e existem muitas opções para o desenvolvimento orientado a modelos. Neste trabalho, foram utilizadas ferramentas já existentes para MDD, em vez de construir alguma solução a partir do zero. A justificativa é que devido ao atual cenário da aplicação de MDD em outros domínios (que não inclui o de aplicações de TV Digital), já é possível contar com diversas opções concretas, avaliadas e que são efetivamente utilizadas na indústria (FURTADO, 2012).

Em resumo estão fora do escopo desta pesquisa:

- Tratar o projeto da interface gráfica do usuário que considera conceitos de usabilidade e experiência do usuário;
- A definição detalhada das atividades de um modelo completo de processo de desenvolvimento de aplicações multimídia;
- Atividades de manutenção dos artefatos, tais como aquelas relacionadas à edição posterior dos *templates* das aplicações, por exemplo;
- Atividades para planejamento e execução de testes; e
- Atividades para tratar mudança de requisitos durante as fases de projeto e implementação; e
- Desenvolvimento de novas tecnologias para o desenvolvimento generativo, tais como ferramentas para transformação ida-e-volta, ou ferramentas para validação de modelos e de transformações.

Mais detalhes sobre o escopo da pesquisa, em termos da abrangência da abordagem definida neste trabalho no que diz respeito ao ciclo de vida de desenvolvimento de software, encontram-se no Capítulo 5.

1.4 Estrutura da tese

Neste primeiro capítulo apresentou-se a motivação e objetivos da tese. O restante da tese está estruturado da seguinte maneira:

- No Capítulo 2 são discutidos os trabalhos relacionados;
- No Capítulo 3 discutem-se os fundamentos sobre a o desenvolvimento orientado a modelos, incluindo os principais conceitos, técnicas e abordagens existentes,
- No Capítulo 4 são explicados os conceitos de uma plataforma de TV Digital, exemplos de aplicações e tecnologias para desenvolvimento de aplicações para TV Digital utilizando o NCL;
- No Capítulo5 é apresentada uma visão geral e detalhada da abordagem proposta;
- No Capítulo 6 são descritos os três exemplos de uso utilizando a abordagem, bem como o projeto e resultados dos estudos empíricos da abordagem;
- No Capítulo 7 são debatidas as considerações finais, contribuições e trabalhos futuros; e
- O Apêndice A explica em detalhes o modelo NCM, base da linguagem NCL utilizada nos exemplos que avaliam a abordagem proposta no trabalho.

2 TRABALHOS RELACIONADOS

Este trabalho usou como referência vários trabalhos acadêmicos que tratam o desenvolvimento de aplicações declarativas para TVD. A restrição para aplicações declarativas se deve ao fato de que o foco desta tese é na integração de visões de projeto que utilizam como ponto inicial o modelo declarativo. Além disso, também foram considerados estudos que discutem os conceitos e viabilidade de uso do MDD em domínios específicos. Portanto, nesta seção é descrita uma análise desses estudos dividida em dois tópicos: (i) desenvolvimento de aplicações declarativas para TVD e (ii) metodologias de desenvolvimento MDD para domínios específicos.

2.1 Desenvolvimento de aplicações declarativas para TVD

As ferramentas de desenvolvimento de aplicações declarativas para TVD podem ser classificadas em duas categorias (SOARES NETO, 2010): ferramentas de Autoria com Foco na Aplicação e ferramentas de Autoria com Foco em *Templates* (Figura 2.1).

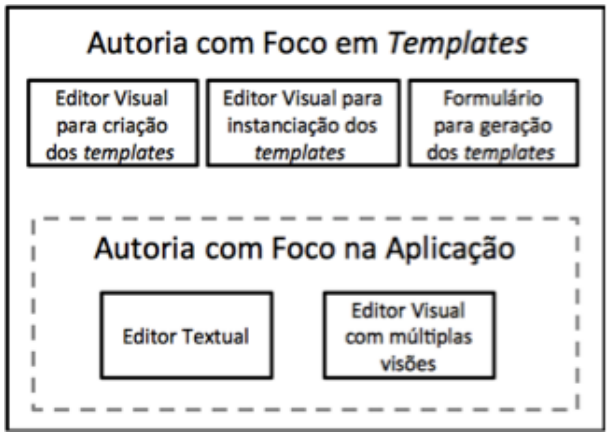


Figura 2.1: Ferramentas de autoria para TVD

O primeiro grupo corresponde a ambientes de desenvolvimento que procuram abstrair visualmente os detalhes de baixo nível da linguagem utilizada (código-fonte) por meio do emprego de um Editor Visual que pode possuir múltiplas visões das estruturas que representam a aplicação multimídia. Por exemplo, tarefas como a criação dos elementos de mídia, a sua localização espacial na tela podem ser feitas pelas ferramentas. Outras tarefas como a sincronização temporal e interatividade dos elementos com usuários também podem ser adicionados. Diante disso, um dos propósitos das ferramentas de autoria é usar métodos de programação visual que abstraiam do autor, toda, ou pelo menos parte, da complexidade de se utilizar uma linguagem de programação. Em Prota (2012) foi proposta uma ferramenta chamada de MoonDo-Eclipse e também realizada uma análise comparativa entre outras 5 (cinco) ferramentas existentes para o desenvolvimento aplicações declarativas para TVD: NCL Eclipse (AZEVEDO, 2011), Composer (LIMA et al., 2010; LIMA, MORENO e SOARES, 2011), GingaWay (BELTRÃO FILHO, 2008), LuaComp (SOUZA JUNIOR, 2009) e NCLite (ENCARNA-

ÇÃO, 2010). O estudo dividiu as propriedades das ferramentas considerando dois perfis: (1) Produtor (nesta tese representado pela equipe de mídia) e (2) Programador (nesta tese representado pela equipe de software). Para o primeiro, as seguintes funcionalidades foram definidas: (a) manipulação de dados da transmissão; (b) multiplicidade de visões; (c) manipulação gráfica e hipermídias; (d) potencialidade no uso de recursos do Ginga; e (e) configuração gráfica de layouts. Já para o segundo, foram considerados os seguintes requisitos: (a) definição de componentes gráficos; (b) manipulação gráfica de componentes; (c) suporte a scripts lua; (d) orientação arquitetural; (e) sugestão de código; e (e) evidência de erros e problemas. A proposta do MoonDo-Eclipse é permitir o desenvolvimento de aplicações utilizando um framework orientado a objetos e implementando usando a linguagem imperativa Lua. Apesar de melhorar as características para o perfil programador, o MoonDo-Eclipse gera o código da aplicação apenas na linguagem imperativa, impedindo a autoria criativa com uma ferramenta que possuem editores visuais das estruturas declarativas, por exemplo o Composer. Dessa forma, esta tese considera o uso das ferramentas Composer e NCL Eclipse de forma complementar, pois, como o próprio estudo realizado por Prota (2012) confirmou, a primeira (Composer) representa a melhor solução para o perfil do produtor e a segunda pode ser utilizada para edição textual do código-fonte da parte imperativa da aplicação. Mais detalhes sobre essas duas ferramentas são encontradas nas Seções 4.3.2 e 4.3.3.

O segundo grupo da Figura 2.1 corresponde a uma metodologia de desenvolvimento mais recente de domínio de TVD (SOARES NETO, 2010). De um modo geral, ela tem como objetivo a definição um processo de autoria destinado a cenários o nde se identifica a ocorrência de estruturas recorrentes de apresentação, isto é, que podem ser utilizadas em diferentes contextos. Tais estruturas são conhecidas como *templates* de composição (MUCHALUAT-SAADE, 2003). Essas abordagens ao invés de utilizarem ferramentas de autoria para aplicações finais, definem linguagens declarativas que permitem compor estruturas e, ferramentas para gerenciar os *templates* de composição. Há várias vantagens no uso do desenvolvimento baseado em *template* (SOARES NETO, 2010): (1) *templates* promovem a coerência de aplicações (*branding*), onde emissoras podem definir e seguir um mesmo estilo; (2) o fato das aplicações seguirem um mesmo padrão de interface recorrente, baseado no mesmo *template*, facilita o seu uso por parte dos usuários finais; (3) família de aplicações promovem naturalmente a indexação e agrupamento; (4) *templates* definem conceitos de domínio em aplicações com características similares, definindo um vocabulário próprio e um conjunto específico de restrições na forma que se compõem aplicações de uma família. As principais propostas que representam esse tipo de metodologia na área de TVD são: XTemplate (SANTOS; MUCHALUAT-SAADE, 2009) e TAL (do inglês, *Template Authoring Language*) (SOARES NETO, 2010).

XTemplate define uma linguagem para descrição de *templates* em NCL e onde o autor reusa especificações de apresentação e o relacionamento temporal das mídias definidas em um *template*. XTemplate possui um editor gráfico chamado de EDITEC (DAMASCENO; SANTOS; MUCHALU-

AT-SAADE, 2010) cujo objetivo é auxiliar o processo de criação de *templates* através de uma abordagem gráfica de fácil uso. O EDITEC é baseado em diversas visões integradas que juntas permitem definir os componentes genéricos do *template*, seus relacionamentos espaços-temporais bem como as repetições do *template* através das estruturas de repetição. À medida que a edição vai sendo realizada através dos elementos visuais, o código do *template* na linguagem XTemplate vai sendo gerado. As visões permitem separar os diferentes aspectos envolvidos na criação do documento *template*, proporcionando uma melhor visão global através das suas partes. A ferramenta possui três visões: a estrutural, que especifica a parte do vocabulário e o corpo do *template*; (ii) a visão de leiaute, para representar as regiões da tela onde as mídias serão exibidas; e (iii) a visão textual onde se apresenta o código do *template* correspondente a edição realizada. A visão estrutural é empregada para definir as principais entidades da linguagem bem como o aspecto comportamental do documento através dos relacionamentos espaço-temporal entre os componentes genéricos. Para isso, a visão estrutural é dividida em duas subvisões, uma que apresenta os elementos do vocabulário chamada de *Vocabulary View* e outra, que apresenta o corpo do *template* chamada de *Body View*. A sub-visão do vocabulário é voltada especificamente para diagramar os elementos estruturais como componentes genéricos, contextos bem como os tipos de conectores que farão parte do documento. Já na visão de corpo, os relacionamentos são definidos. Estas duas sub-visões são integradas de modo que os elementos definidos no vocabulário podem ser reutilizados na sub-visão do corpo para especificar os relacionamentos entre eles. O usuário pode tanto criar, como editar e apagar graficamente portas, componentes e elos. Ao arrastar um elemento para a área gráfica o usuário preenche as propriedades como o identificador, o seu tipo de mídia (imagem, texto, vídeo, etc) e o seu número de ocorrências no documento. Quando um elemento é selecionado, as suas propriedades também podem ser editadas. No EDITEC um modelo visual foi proposto para o suporte às repetições envolvidas nos elementos da linguagem, os quais são decorrentes da presença de grupos de componentes no *template*. O modelo é composto por diversas opções para representar graficamente estruturas de interação presentes no *template*, oferecendo suporte para especificar de forma gráfica os diversos tipos de relacionamentos (elos) entre os componentes. A principal vantagem do uso do XTemplate e EDITEC é favorecer o reuso com a possibilidade gerar um protótipo de uma aplicação de forma rápida. Por outro lado, duas desvantagens dessa abordagem são: (1) a necessidade do conhecimento de uma nova linguagem e; (2) dificuldade para integrar os *templates* com as mídias imperativas. Dessa forma, só é possível criar *templates* de aplicações que utilizem elementos da linguagem declarativa NCL, não contemplando as aplicações que possuem lógica de aplicação complexa e, consequentemente, não permite a integração com a equipe de software.

TAL é baseada na linguagem XML é vista como uma evolução de XTemplate por ser independente de linguagem declarativa e adicionar atividades no processo de desenvolvimento que não exigem um autor de documentos com conhecidos em linguagens específicas.Tal descreve um documento que não está completamente pronto para ser exibido, até o momento em que seja instanciado

com um conteúdo específico. A linguagem possui elementos para tratar o vocabulário por meio de componentes em aberto (genéricos), além de elementos de restrições e de relacionamentos entre os componentes. Para facilitar a geração da aplicação final, a abordagem TAL utiliza uma ferramenta chamada de *NCL Wizard* (SOARES NETO, 2010) que ajuda a especificar formulários usando uma linguagem chamada XWizard. Esses formulários são unidades independentes e simples para geração automática do código final da aplicação. Uma vez pronto um formulário, o usuário detém-se apenas no preenchimento de configurações requisitadas passo a passo para gerar uma aplicação. De forma geral, a abordagem utilizando TAL e NCLWizard envolve a participação de 3 (três) papéis: Editor de *Templates*, Editor de Formulários e Autor de Documentos (aplicações). O primeiro é responsável por criar os *templates* que podem gerar as aplicações utilizando TAL. O segundo define, utilizando XWizard, os formulários para configuração/parametrização de cada *template*. O terceiro por sua vez utiliza os formulários para gerar as aplicações. O uso de TAL traz todas as vantagens oferecidas por abordagens orientada a *templates*, assim como XTemplate. Porém, com a inclusão do NCL Wizard, a proposta procura facilitar a geração de aplicações finais por parte do editor de documentos sem a necessidade de conhecer a linguagem TAL. Para isso, adiciona mais um papel ao processo que é o responsável por criar os formulários.

Inspirado no MDA, o *StoryToCode* (MARQUES NETO; SANTOS, 2010) estrutura os requisitos em um modelo de concepção de alto nível. O processo de transformação desse modelo de alto nível em código, que é o principal objetivo do *StoryToCode*, parte do princípio que este modelo já está pronto. O processo de criação do modelo de alto nível define que: a equipe de engenheiros de software deve traduzir as informações obtidas nos documentos de cenários e *storyboards* (ambiente de produção de mídia) em um diagrama de elementos (ambiente software). Esta definição, que é tradicionalmente usada para concepção de softwares, não se mostra adequada para a concepção de aplicações multimídia, uma vez que ela não diminui a distância entre profissionais de produção de mídia e software na elaboração dos modelos de alto nível. A principal consequência disso é o aumento do retrabalho (pela equipe de software) nos ajustes necessários ao modelo antes de sua aprovação final (pela equipe de produção de conteúdo multimídia). Outro problema relacionado ao *StoryToCode* está na quantidade de trabalho necessária para finalizar o código gerado a partir do diagrama de elementos criado. Isso ocorre porque o diagrama de elementos que representa uma abstração dos requisitos da aplicação modelada não contempla as informações de instância. Esse diagrama descreve apenas a estrutura das informações que devem estar presentes em uma aplicação e não dos valores de instância dessa estrutura. Além disso, o *StoryToCode* possui representações apenas de estrutura da aplicação, não tratando outras visões que são essenciais para a elaboração das aplicações multimídia interativas, por exemplo interação e interface gráfica com o usuário. O *StoryToCode* também não modela características de aplicações mais recentes, que utilizam, por exemplo, serviços Web, uma vez que seu mode-

lo tem foco na estrutura da informação e não no tratamento de dependências ou comportamentos externos que possam afetar tal estrutura.

2.2 Metodologias de desenvolvimento MDD para outros domínios

Os trabalhos (PLEUSS; HUSSMANN, 2011; PLEUSS, 2009; PLEUSS; VITZTHUM; HUSSMANN, 2007) têm foco no desenvolvimento de aplicações multimídia interativas por meio da especificação de uma linguagem de domínio específico: MML (do inglês, *Multimedia Modeling Language*). Segundo a pesquisa, as linguagens de modelagem existentes não são suficientes para o desenvolvimento de aplicações multimídia, pois elas não cobrem nem a integração das mídias nem os aspectos gerais de interface com o usuário. Por outro lado, apesar das abordagens tradicionais do domínio multimídia fornecer amplo suporte para criação de mídia, elas negligenciam a lógica do aplicativo e os princípios da Engenharia de Software. O desenvolvimento de aplicações multimídia envolve 3 diferentes tipos de projeto: (i) projeto de software, desenvolvimento da lógica do aplicativo; (ii) projeto da interface gráfica com o usuário e (iii) projeto de mídias, pois a criação de objetos de mídia requer normalmente o conhecimento e ferramentas e conhecimentos (estudo da cor, forma etc.). A linguagem MML procurar endereçar natureza interdisciplinar existente no desenvolvimento das aplicações multimídia. Através de uma linguagem baseada na especificação UML 2.0, a MML procura integrar as diferentes equipes e os artefatos produzidos que permeiam os 3 diferentes tipos de projeto anteriormente descritos. Para tanto, MML define 5 tipos de modelos: (i) modelo estrutural, que representa entidades da aplicação (modelo de domínio e objetos de mídia); (ii) modelo de cenas, que auxilia o projeto da interface com o usuário, descrevendo como os objetos de mídia podem influenciar a escolha de diferentes cenas (telas da aplicação); (iii) modelo de interface abstrata, que define para cada cena, os elementos abstratos associados à interface do usuário; (iv) modelo de interface de mídia, responsável pela integração dos componentes de mídia com a interface com o usuário abstrata e derivado de modelo de interface abstrata e objetos de mídia do modelo estrutural e (v) modelo de interação, que especifica como os eventos de interface do usuário e os eventos de mídia podem disparar chamadas de operação nas entidades do domínio. O principal objetivo é permitir que o processo envolva diferentes especialistas pertencentes aos três tipos de projetos existentes: projeto de software; interface gráfica com o usuário e produção de mídias. O trabalho proposto nesta tese adota a MML como DSL. Porém, foi incluída uma etapa para estruturar os requisitos através de técnicas de análise de domínio (taxonomia e Modelo de *Features*) e, consequentemente, facilitar o projeto do domínio através da definição e configuração dos *templates* modelados com MML. Além disso, também foi avaliado o uso de MML num subdomínio específico de aplicações multimídia, através de: (i) extensão dos metamodelos de MML para tratar questões específicas de TVD; e (ii) emprego de técnicas de programação generativa para transformação de modelos para modelos e de modelos para código de aplicações.

Para o domínio de aplicações Web o trabalho (BERNARDI; DI LUCCA; DISTANTE, 2011) descreve uma abordagem de prototipagem rápida adotando MDD para permitir a geração automática de um protótipo totalmente funcional de uma Aplicação Web. É definido um processo de três passos: (1) definir um metamodelo para o projeto da aplicação; (2) desenvolver uma ferramenta de modelagem gráfica para o metamodelo definido e (3) desenvolver uma ferramenta para geração de código automática a partir dos modelos. Um ponto interessante é que o processo e as tecnologias utilizadas para implementar essa abordagem podem ser reutilizados para a abordagem de prototipagem rápida para modelos de design diferente e/ou plataforma diferente. Uma deficiência deste trabalho é não tratar o conceito de *templates* de software para a modelagem da aplicação, tornando inviável a estruturação de requisitos para domínios específicos.

Furtado (2012) apresenta uma proposta de desenvolvimento orientada a modelos para o domínio de jogos eletrônicos, utilizando diferentes técnicas de GSD para definir uma LPS que pode abordar vários subdomínios de jogos: modelo de *features*, definição de uma arquitetura de referência (a partir de *games engines*, bibliotecas e componentes), definição de DSLs para cada subdomínio e scripts de metaprogramação baseados em duas tecnologias específicas para jogos: *FlatRedBall* e XNA. O principal objetivo deste trabalho é definir uma metodologia de desenvolvimento específica de domínio, capaz de aumentar a reutilização de software e manutenção e promover uma ligação entre a análise de domínio e o projeto e implementação de domínio na área de jogos. Assim, novas estratégias de desenvolvimento podem ser aplicadas de acordo com o ambiente do jogo final (*engines*, *frameworks* e plataformas, por exemplo), resultando em uma estratégia de baixo acoplamento entre o domínio do jogo e as tecnologias de implementação. Apesar de não utilizar as mesmas tecnologias, a abordagem proposta nesta tese procura seguir alguns passos definidos em Furtado (2012). Porém, nesta tese não é definido nenhuma arquitetura de referência, fazendo com que os modelos dos *templates* sejam diretamente mapeados para o código das aplicações. Esta tese também levou em consideração o projeto experimental da avaliação empírica para um domínio específico (jogos), procurando fazer uma adaptação ao domínio de aplicações para TV Digital.

A pesquisa proposta por Fraternali, Comai e Bozzon (2010) ilustra uma abordagem MDD para o desenvolvimento de aplicações RIA (*Rich Internet Application*) que estende metodologias e notações pré-existentes para o desenvolvimento de aplicações Web tradicionais. As principais contribuições do trabalho: (1) identificação de que elementos do MDD podem ser aplicados para o desenvolvimento de aplicações RIA; (2) revisão das características específicas que uma aplicação RIA em relação a aplicações Web tradicionais para entender quais extensões são necessárias para as notações empregadas atualmente na Engenharia Web; (3) extensão de uma DSL pré-existente para atender o subdomínio das aplicações RIA; (4) avaliação do uso da DSL em um ambiente de desenvolvimento da indústria. Para tanto, o trabalho propõe um conjunto de metodologias MDD que devem ser integradas em ambiente de desenvolvimento para oferecer edição de modelos e funções de geração de código. Os

principais benefícios alcançados estão relacionados à produtividade de desenvolvimento através da melhoria da expressividade da arquitetura da aplicação e maior facilidade de desenvolvimento. Para tanto, foram utilizados modelos de uma DSL para geração de código de padrões arquiteturais em um ciclo de prototipagem rápida. Através de estudos de caso foi demonstrado que é possível especificar requisitos de aplicações RIA através de notações de alto nível e independente de plataforma e a partir daí transformar esses modelos em código da aplicação, considerando tanto o lado do cliente, como do servidor. Tal trabalho tem objetivos e questões de pesquisa bem próximas desta tese, mas, além de tratar domínio diferente de aplicações (Web), não realiza nenhum experimento para avaliar quantitativamente o uso da abordagem.

Cassou *et al.* (2011) propõem uma abordagem que abrange o ciclo de vida no desenvolvimento de um aplicativo no domínio da computação pervasiva. A metodologia proposta é fortemente apoiada por ferramentas. As principais contribuições do trabalho são: (i) definição de uma linguagem de domínio específico (*DiaSpec*) que fornece um framework conceitual que modela papéis aos atores e separação de interesses das aplicações; (ii) uma metodologia de desenvolvimento generativa apoiada por um conjunto de ferramentas (*DiaSuite*) e que abrange todo o ciclo de vida de desenvolvimento. Tal abordagem inclui um transformador de artefatos de arquitetura modelados em *DiaSpec* para código-fonte e um simulador (*DiaSim*), utilizado para testar as aplicações geradas num ambiente de computação pervasiva em determinadas plataformas de sistemas distribuídos. No contexto desta tese, seguimos a mesma estratégia de utilizar um ambiente de desenvolvimento (ferramentas) para apoiar a modelagem da arquitetura (através de artefatos que expões múltiplas visões do projeto da aplicação) e geração das aplicações da *DiaSuite*, ou seja, alguns passos sugeridos para o domínio de aplicações pervasivas, também foram utilizados para o domínio das aplicações multimídia, principalmente, a etapa de definir uma taxonomia de aplicações a partir da DSL *DiaSpec* (no caso da tese, utilizamos MML). Contudo, as tecnologias utilizadas são diferentes. Também não foram realizados estudos empíricos para análise quantitativa do emprego da metodologia proposta.

2.3 Considerações sobre o capítulo

A literatura na área de aplicações declarativas para TVD contém várias opções de metodologias e ferramentas para apoiar o seu desenvolvimento. Porém, os trabalhos existentes deixam em aberto duas lacunas: (1) definição de abordagens de desenvolvimento que integrem de modo sistemático as diferentes disciplinas (software, mídia e usabilidade) envolvidas no desenvolvimento desse tipo de aplicação, normalmente, os trabalhos focam em partes isoladas do problema; e (2) estabelecimento de ambientes de desenvolvimento que ofereçam suporte concomitantemente a autoria criativa e melhor estruturação da lógica de aplicação desse tipo de software. De modo geral, as ferramentas de autoria (tanto com foco em aplicação, como em *templates*) se focam no modelo declarativo ou no modelo imperativo, mas em nenhum momento na integração dos dois.

Já em outros domínios (jogos, aplicações Web ricas, sistemas de computação pervasiva etc.) é possível observar trabalhos que tratam melhor os dois desafios citados anteriormente utilizando abordagens orientadas a modelo. Porém, poucos trabalhos exploram uma avaliação utilizando métodos científicos que façam uso de experimentos controlados, deixando assim dúvidas sobre os benefícios do seu emprego.

Neste capitulo foram apresentados trabalhos acadêmicos nas áreas de ambiente de desenvolvimento para TVD e abordagens de desenvolvimento orientado a modelos, e que possuem alguma interseção com a proposta desta tese.

3 DESENVOLVIMENTO DE SOFTWARE ORIENTADO A MODELOS

Historicamente, o processo de desenvolvimento de software mostra-se caro, lento e incerto para as condições de negócio modernas. Tais dificuldades fizeram surgir o termo "a crise do software" no ano de 1968 numa conferência da OTAN, que justifica que a causa principal reside na complexidade inerente da natureza da tarefa de construir software. Outro motivo seria a necessidade constante de adaptar-se às mudanças dos requisitos dos usuários e das inovações tecnológicas. Conforme Brooks (1995) citou, a Engenharia de Software (termo também criado nessa conferência da OTAN) não conseguiu obter o mesmo sucesso da Engenharia de Hardware, que consegue seguir a Lei de Moore dobrando sua capacidade, representado pelo poder de processamento, a cada 24 meses.

Atualmente, a dimensão desse problema continua crescente, visto que aumentou a demanda por funcionalidades mais sofisticas e por sistemas de software mais confiáveis. Consoante Booch et al. (2004) apontou, "*software executa o mundo*". Portanto, é fundamental entender como encontrar as fontes dessa complexidade e o que se pode fazer para tratá-la de modo mais adequado. Neste capítulo é apresentado como as metodologias de desenvolvimento de software atuais tentam melhor lidar com "a crise do software".

3.1 Desenvolvimento baseado em modelos

A Engenharia de Software considera que desenvolver software deve ser encarado da mesma forma que engenheiros constroem outros sistemas complexos, tais como pontes, edifícios, navios e aviões. A ideia é considerar que software é um sistema complexo e seu processo de desenvolvimento deve ser encarado como um trabalho de engenharia, ou seja, deve existir um processo planejado, baseado em metodologias sistemáticas e apoiado por ferramentas. Do mesmo modo, cientistas e profissionais da computação procuraram durante as últimas 5 décadas criar abstrações que auxiliem o desenvolvimento de software através da utilização de modelos de alto nível que os protegem das complexidades de um ambiente computacional, como por exemplo, processador, memória, dispositivos de comunicação, plataformas de distribuição, etc (SCHMIDT, 2007). O valor dessas abstrações no desenvolvimento de software é substancial e pode ser observado na história das linguagens de programação e sistemas operacionais (MAHONEY, 2004). Na área de metodologias de desenvolvimento de software, um esforço que se destacou foi o livro de DeMarco (1979) que estabeleceu o conceito de desenvolvimento baseado em modelos (do inglês, MBD - *Model-based Development*). No MBD é destacado que o desenvolvimento de um sistema de software deve ser precedido pela construção de um modelo, assim como nas outras engenharias. O modelo de um sistema consiste na conceituação do domínio do problema e de sua solução. O modelo se foca sobre o mundo real, identificando, classificando e abstraindo os elementos que constituem o problema com o objetivo de organizá-los em uma estrutura formal.

O uso de modelos se baseia no conceito de abstração, que é uma das principais técnicas com que a mente humana enfrenta a complexidade. Por meio da ocultação do que é irrelevante num sistema complexo, é possível reduzi-lo a algo compreensível e manejável. Um exemplo é o uso de mapas em sistemas geográficos, que podem existir em diferentes escalas e representar apenas alguns aspectos do sistema. Um modelo também pode ser interpretado como um sistema. Igualmente, um modelo pode ser representado por outro modelo, por exemplo, outro nível de abstração. Analogamente, um mapa geográfico também pode ser visto como uma representação de um outro mapa mais detalhado do mesmo território. Quando se trata de software, é extremamente útil abstrair os detalhes tecnológicos de implementação e tratar dos conceitos do domínio de forma mais direta possível utilizando um ou mais níveis. Assim, um modelo de um sistema pode funcionar como um meio de comunicação entre usuários, analistas e programadores de forma a esconder ou diminuir os aspectos relacionados à sua tecnologia de implementação.

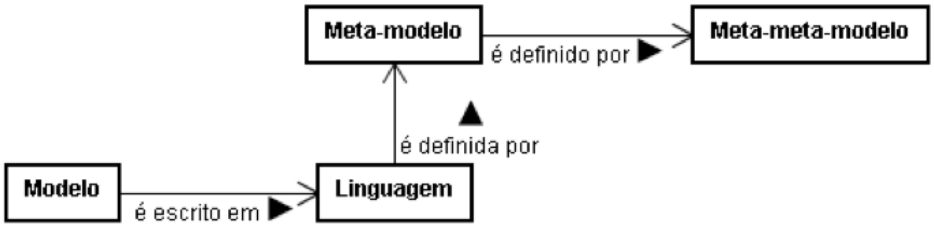


Figura 3.1: Relações entre modelo, linguagem de modelagem, metamodelo e meta-metamodelo

Todo mapa geográfico requer, pelo menos implicitamente, uma legenda, para interpretar os elementos que o compõem. Este conceito pode ser generalizado e aplicado aos modelos, uma vez que os mesmos devem ser elaborados a partir de uma linguagem de modelagem. Conforme a comunidade científica da área de modelagem observa, uma linguagem de modelagem é constituída por um conjunto de elementos que representa também um modelo e é denominado como um metamodelo. Ele define a estrutura, a semântica e as restrições (sintaxe) de um grupo de modelos (MELLOR, 2004).

Visto que um metamodelo também é um modelo, este necessita ser especificado por uma linguagem de modelagem. O modelo que possui os elementos utilizados para criação de um metamodelo é definido como um meta-metamodelo (KLEPPE; WARMER; BAST, 2003). A meta-meta modelagem configura uma hierarquia de metamodelos e tem como responsabilidade formar uma linguagem para a especificação de metamodelos (OMG, 2003). A Figura 3.1 apresenta a relação entre modelo, linguagem de modelagem, metamodelo e meta-metamodelo. Um modelo é descrito por intermédio da utilização de uma linguagem de modelagem, sendo que uma linguagem de modelagem é determinada por um metamodelo, que por sua vez é definido por meta-metamodelo.

Atualmente, quase todas as metodologias de desenvolvimento de software utilizam modelos, variando as classes de modelos empregadas em relação à representação e manipulação. Nesse sentido, destacam-se dois tipos principais: modelos matemáticos e modelos diagramáticos. O primeiro está re-

lacionado à utilização de modelos formais que representam o problema de forma precisa e rigorosa e utilizam notações de natureza matemática (por exemplo: Z, VDM, B e OCL). Apesar da vantagem de permitir a geração de sistemas robustos e consistentes, inclusive com a possibilidade de verificação e validação da especificação do software, tais modelos não foram adotados maciçamente na indústria, principalmente, pela complexidade da sua fundamentação matemática que impede o fácil entendimento e comunicação. O segundo tipo de modelagem é baseado em uma notação gráfica mais fácil de utilizar em relação aos modelos formais (por exemplo, UML). Embora representem diagramas mais fáceis de entender, esse tipo de modelagem também pode conter uma base formal para os modelos gráficos, garantindo as vantagens do primeiro método. Na seção a seguir é descrito o ciclo de vida do desenvolvimento baseado em modelos MBD, que pode utilizar diversos tipos de modelagem.

3.1.1 Ciclo de Vida baseado em modelos

Um ciclo de vida de desenvolvimento típico que se utiliza de MBD contém 6 fases (Requisitos, Análise, Projeto, Codificação, Testes e Manutenção) e é ilustrado na Figura 3.2.

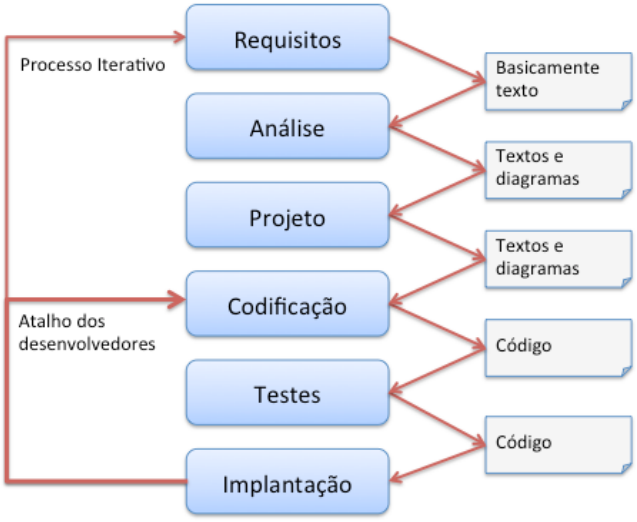


Figura 3.2: Ciclo de vida típico do desenvolvimento baseado em modelos (PONS; GLADINI; PEREZ, 2010)

Nas 3 (três) primeiras fases se constroem os modelos de requisitos (representado por linguagem natural), de análise e de projeto (esses dois últimos utilizam frequentemente diagramas visuais) e, posteriormente, realizadas as outras fases sem fazer mais o uso de nenhum diagrama, apenas código. Tais fases são executadas de um modo interativo-incremental, ou mesmo cascata. Na prática, apesar dos modelos ajudarem no início do desenvolvimento para a geração inicial do código-fonte, os desenvolvedores não sincronizam mais as atividades, gerando uma desatualização dos modelos perante o código-fonte desenvolvido. A seguir são discutidos esse e outros problemas do MBD.

O problema da produtividade, documentação e manutenção: como visto num processo de desenvolvimento baseado em modelos, a conexão entre os diagramas e o código vai se perdendo gradualmente quando se alcança a fase de codificação. Como ilustra a (ver: "Atalho dos desenvolvedores"), após a primeira interação, os desenvolvedores tendem a apenas realizar as modificações diretamente no código, normalmente com a justificativa que não há tempo para realizar as atualizações em diagramas e outros documentos de alto nível. Além disso, a atualização de diagramas e documentos é questionada, pois qualquer mudança se encontra no próprio código de todo modo. Em outras palavras, muitas vezes se considera a tarefa de documentação como uma atividade de sobrecarga adicional, valorizando como produtivo apenas a geração de código e negligenciado a documentação presente nos modelos. A principal consequência desse comportamento é dificultar a manutenção em médio prazo do sistema, principalmente, quando os desenvolvedores iniciais que detém o conhecimento do sistema não estão mais disponíveis. Uma solução proposta para esse problema é gerar a documentação diretamente no código-fonte do sistema. Porém, essa saída resolve unicamente a questão da documentação no nível mais baixo. Em contrapartida, a documentação de alto nível (representada por textos e diagramas) precisa ser mantida manualmente. Devido à complexidade dos sistemas atuais, é extremamente importante contar com documentação em níveis distintos de abstração. Desse modo, temos duas alternativas: ou se gasta tempo nas primeiras fases do processo de desenvolvimento para gerar os diagramas de alto nível e também na sua manutenção, ou se utiliza mais tempo na fase de manutenção para entender o software.

O problema da flexibilidade e das mudanças tecnológicas: a indústria de software possui uma característica especial que a difere de outras. A cada ano (ou mesmo em intervalo menor), surgem novas tecnologias que rapidamente se disseminam. As organizações precisam adotar essas novas tecnologias pelas seguintes razões:

- Os clientes exigem esta nova tecnologia (por exemplo, aplicações Web);
- São soluções para problemas reais (por exemplo, uso de XML para intercâmbio de dados); e
- As empresas fornecedoras de soluções deixam de dar suporte à velhas tecnologias e passam a atender apenas às novas (por exemplo, o suporte a OMT foi substituído por UML).

Devido a essas pressões e, também pela existência de vantagens, as organizações são forçadas a realizar a migração dos sistemas de software para uma nova tecnologia ou para uma versão mais nova da que foi utilizada anteriormente. Também existe o caso onde o software permanece utilizando uma tecnologia mais antiga, mas precisa se comunicar com sistemas novos, nos quais se utilizam novas tecnologias. Todos esses cenários de portabilidade precisam ser tratados, caso contrário, a estagnação pode ser prejudicial para a organização, pois ocasiona uma defasagem em relação aos concorrentes. Nesse caso, o problema surge da quantidade de esforço gasto em tarefas específicas da plataforma, esforço este que não é reutilizado em outras plataformas. Idealmente, as metodologias de desenvolvi-

mento de software deveriam tratar tal questão de forma mais conceitual e menos focado no esforço manual e repetitivo. Infelizmente, o MBD não trata essa questão com uma solução integral.

3.2 Desenvolvimento orientado a modelos

O desenvolvimento orientado a modelos (do inglês, MDD – *Model-driven Development*) ou desenvolvimento de software orientado a modelos (do inglês, MDSD – *Model-drivenSoftware Development)* apareceu com o objetivo de auxiliar a resolver os problemas citados na seção anterior (KLEPPE; WARMER; BAST, 2003). O adjetivo "orientado" (*driven*) procura se diferenciar da abordagem proposta anteriormente, que utilizava o adjetivo "baseado" (*based*), uma vez que o MDD enfatiza que os modelos devem ser tratados como artefatos centrais do desenvolvimento de software constituindo-se de:

- modelos que estão em conformidade com metamodelos
- transformações explícitas entre esses modelos

A proposta do MDD, de modo geral, é que o engenheiro de software não necessite manipular manualmente todo o código-fonte, se concentrando em modelos de mais alto nível e se protegendo de complexidades exigidas para implementação de diferentes plataformas tecnológicas. Os modelos são gerados de níveis mais abstratos para níveis mais concretos por meio de etapas sucessivas de transformação e refinamentos, até o código-fonte ser gerado numa última transformação. O diferencial é que devem ser utilizados mecanismos automáticos para transformação entre modelos e geração de código. Neste contexto, modelos não são apenas um guia ou uma referência que devem ser interpretados pelos desenvolvedores mas também, entidades que fazem parte do software e que a partir delas é gerado o código-fonte.

Para entendermos melhor a diferença entre MBD e MDD, é importante examinar o que Brown (2004) chama de "*espectro da modelagem*", onde ele ilustra uma escala de abordagens de desenvolvimento que faz desde o uso apenas de código-fonte, até as que fazem uso apenas de modelos Figura 3.3. No canto esquerdo da ilustração é possível notar que mesmo no cenário onde se faz o uso apenas de código-fonte, implicitamente os desenvolvedores geram modelos mentais e/ou informais. Todavia, a representação formal do sistema é realizada apenas pelo código-fonte. No segundo cenário, os desenvolvedores criam o código e, depois, utilizando alguma técnica de engenharia de reversa, geram os modelos apenas para visualizar o que existe no código-fonte. O terceiro cenário representa o uso de ferramentas de transformações de ida e volta (termo em inglês conhecido como *roundtrip engineering*) que sincronizam qualquer mudança nos modelos (definidos em primeiro lugar) para o código-fonte e vice-versa. Já no quarto cenário (programação de modelos) o modelo é o código e qualquer mudança no sistema só é acessível pelos modelos, uma vez que o código-fonte é gerado de forma transparente. O último cenário acontece quando utilizamos modelos formais, mas essas representações não são

transformadas ou utilizadas no código-fonte final da solução, mesmo que sirvam de base inicial (por exemplo, podemos gerar um modelo de entidade-relacionamento para começar a definição de um esquema de banco de dados, posteriormente, o projeto de banco de dados avança e não tem nenhuma relação com os modelos inicialmente gerados). Apenas o terceiro e quarto cenários são considerados metodologias MDD, enquanto que os outros são no máximo MBD.

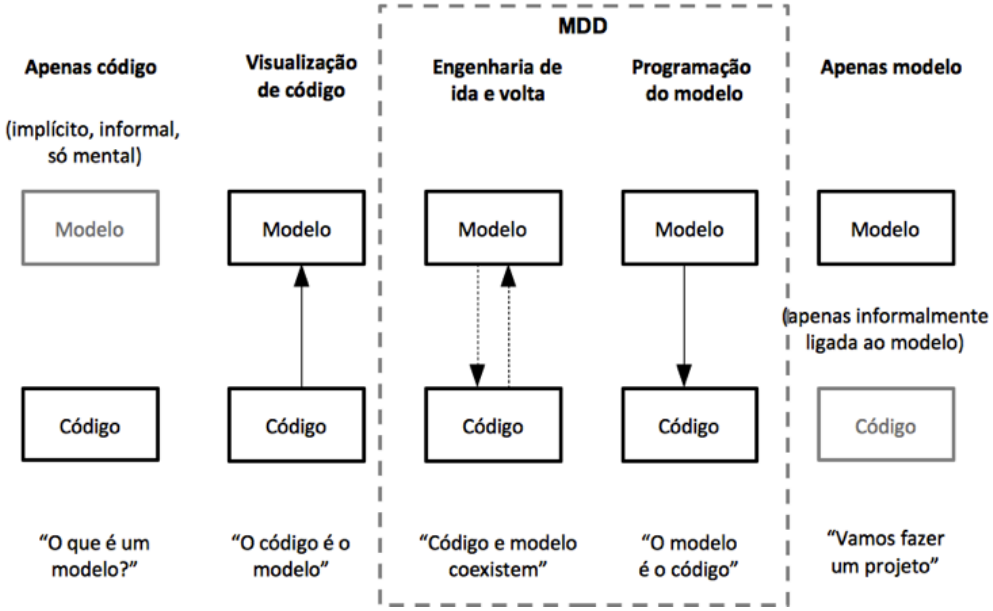


Figura 3.3: O espectro da modelagem. Adaptado de Brown (2004).

3.2.1 Principais elementos do MDD

Os elementos principais do MDD são identificados por Booch et al. (2004) como sendo:

- Uso de um nível maior de abstração em relação a outras metodologias. Tanto para a especificação do problema, como para a especificação da solução correspondente;
- Uso maior de mecanismos de automatização por meio de mecanismos computacionais para realizar as fases de análise, projeto e implementação;
- Uso de padrões propostos e/ou adotados pela indústria para facilitar a comunicação entre seus usufrutuários e também melhorar a interoperabilidade entre diferentes aplicações e produtos.

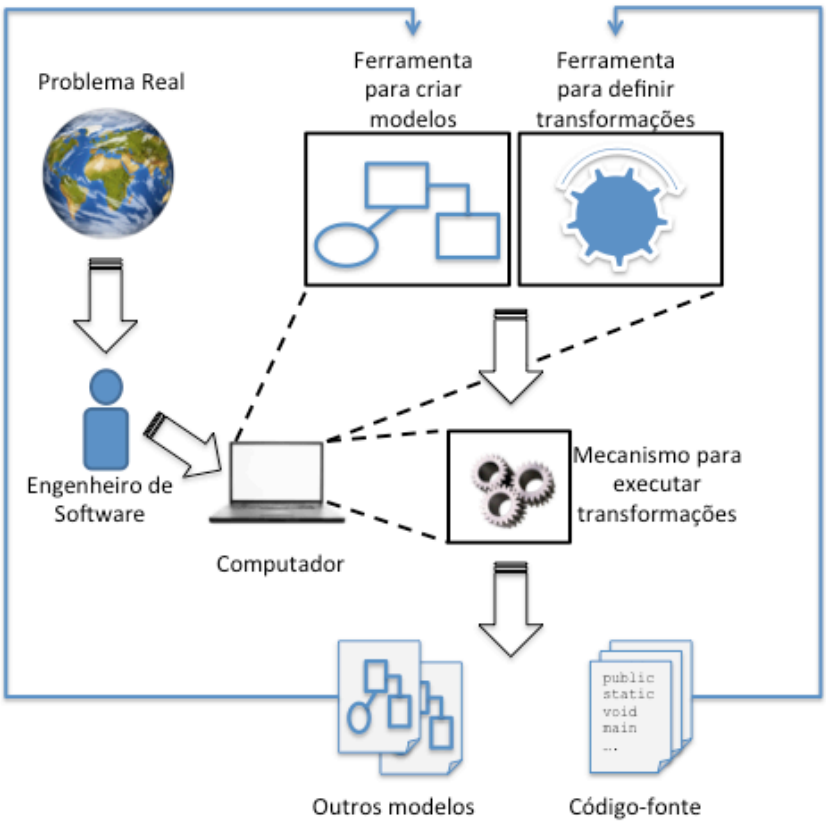


Figura 3.4: Principais elementos do MDD

Na Figura 3.4 é possível observar os principais elementos necessários para a utilização do MDD, onde inicialmente ocorre a definição dos modelos e das transformações pelo engenheiro de software a partir de uma investigação de um problema do mundo real. Um ponto importante a notar é que intervenção humana deve ser substituída o máximo possível por ferramentas e mecanismos que auxiliem a automatização de etapas do processo, por meio do apoio de um computador. Nessa ocasião, tanto para a criação dos modelos, quanto para a definição de transformações, é necessário o uso de ferramentas. A partir destas então é que podem ser gerados outros modelos e código-fonte dos sistemas de software.

3.2.2 Ciclo de Vida do MDD

O ciclo de vida do desenvolvimento de software utilizando MDD é exibido na Figura 3.5. Apesar de possuir as mesmas fases de uma metodologia tradicional de desenvolvimento de software, a principal diferença do MDD são os tipos de artefatos criados durante o processo de desenvolvimento. Estes artefatos devem representar modelos que devem ser interpretados por computadores para apoiar a geração de código ou transformação para outros modelos.

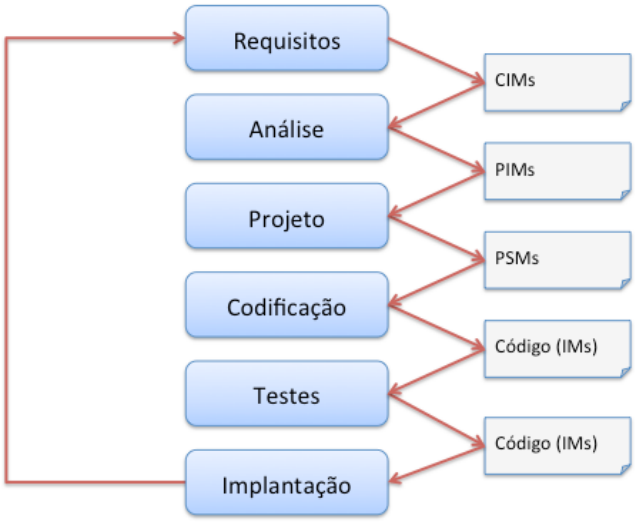


Figura 3.5: Ciclo de vida de desenvolvimento de software orientado a modelos (PONS; GLADINI; PEREZ, 2010)

A seguir uma lista dos principais tipos de modelos utilizados no MDD:

- Modelos independentes de computação (do inglês, CIM – *Computational Independent Model*)
- Modelos independentes de plataforma ou tecnologia de implementação (do inglês, PIM – *Platform Independent Model*)
- Modelos dependentes de uma plataforma ou tecnologia de implementação específica (do inglês, PSM – *Platform Independent Model*)
- Modelos que representam o código-fonte (do inglês, IM – *Implementation Model*)

Apesar das metodologias de desenvolvimento de software tradicionais utilizarem modelos, no MDD, as transformações entre modelos para modelos e modelos para código-fonte são sempre realizadas por meio de ferramentas. Outro ponto importante é o uso de modelos de níveis diferentes de abstrações, que facilita as transformações entre modelos para modelos e também a geração de código-fonte mais completo na etapa final.

3.2.3 Principais vantagens e desvantagens do MDD

Os principais benefícios do uso do MDD são alcançados tratando os problemas discutidos na Seção 3.1.1 e de acordo com Liddle (2011), Pons, Gladini, Perez (2010) e Kleppe, Warmer e Bast (2003) pode-se destacar:

- **Aumento da produtividade, documentação e manutenção** – apesar de necessitar de um esforço inicial para desenvolver ou adquirir as ferramentas de modelagem e de transformações, o MDD pode trazer um aumento na produtividade dos desenvolvedores em relação ao tempo de desenvolvimento, uma vez que é possível gastar mais tempo em modelos de mais alto nível, pois etapas repetitivas podem ser automatizadas com as transformações, ganhando tempo

e esforço em tarefas mais relevantes. Um único modelo também tem potencial para gerar uma quantidade e diversidade razoável de código. Como os modelos são elementos centrais no desenvolvimento, qualquer manutenção deve ser realizada diretamente neles, conservando-os consistente com o código-fonte. Desse modo, já que a documentação está atualizada, as tarefas de manutenção são facilitadas.

- **Melhor adaptação a mudanças de requisitos tecnológicos** – o MDD permite lidar com mudanças tecnológicas com mais flexibilidade, uma vez que utiliza modelos de alto nível independentes de detalhes de implementação ou de plataforma de execução. Nesse cenário, a mudança exigida fica concentrada nas transformações entre os modelos PIMs e PSMs, reutilizando os modelos independentes de tecnologia. Outro ponto positivo é que as decisões tecnológicas podem ser adiadas, uma vez que nas fases iniciais de modelagem, questões de implementação não precisam ser tratadas. Além disso, por conta da capacidade de transformar cada parte de um modelo para uma plataforma específica, o MDD pode tratar melhor software executa em um ambiente heterogêneo, tratando melhor requisitos de interoperabilidade.
- **Melhoria na comunicação e na qualidade do software** – a utilização de modelo ajuda na comunicação entre a equipe envolvida no desenvolvimento do software, pois o uso de abstrações de mais alto nível melhora o entendimento e auxilia ao tratar as expectativas em relação ao sistema que está sendo proposto. O uso de modelos em vários níveis de representação também possibilita que especialistas do domínio possuam um papel mais ativo no processo de desenvolvimento melhorando a identificação de conceitos de negócio. Por outro lado, utilizando outros modelos, especialistas em computação podem discutir melhor os aspectos técnicos. Adicionalmente, por oferecer especificações precisas do que será desenvolvido, os modelos podem contribuir para um gerente de projetos definir melhor as tarefas de desenvolvimento em relação a tempo, custo e escopo.

Já Thomas (2004) e Ambler (2003), descrevem como principais desvantagens do emprego MDD:

- **Aumento do investimento inicial, curva de aprendizado e complexidade:** as abordagens MDD normalmente dependem de maior investimento inicial e tempo de implantação do que abordagens tradicionais. Tais custos e esforços estão associados à construção de uma infraestrutura orientada a modelos que envolve desde o treinamento ou contratação de profissionais com conhecimento específico na área (construção de linguagens, ferramentas de modelagem, transformações e geradores de código) até a aquisição de software específicos. O uso de novos artefatos e ferramentas também pode adicionar uma complexidade adicional ao processo de desenvolvimento.

- **Geração de código-fonte menos flexível e com menor desempenho:** dependendo das regras de transformação criadas, uma abordagem MDD causa maior rigidez no software produzido, uma vez que boa parte do código é gerada automaticamente e fica fora do alcance do desenvolvedor. Outro aspecto da geração de código é que a mesma pode incluir código desnecessário, que pode diminuir o desempenho do software, quando comparado com código escrito manualmente.

3.3 Principais abordagens da indústria

A seguir são apresentadas as três principais abordagens da indústria que utilizam as práticas do MDD: OMG-MDA, Modelagem Específica de Domínio e Desenvolvimento de Família de Sistemas.

3.3.1 Abordagem OMG-MDA (Model-Driven Architecture)

OMG (do inglês, *Object Management Group*) é um consórcio da indústria que foi criado em 1989 com o objetivo de definir padrões para interoperabilidade de sistemas de objetos distribuídos. Sua primeira iniciativa está relacionada à definição do CORBA (do inglês, *Common Object Request Broker Architecture*), padrão para sistemas de middleware. Outro principal padrão homologado pelo grupo foi a UML (do inglês, *Unified Modeling Language*), que teve sua versão 1.1 lançada em 1997. Após a definição do UML, o OMG voltou seu esforço para a área de desenvolvimento de software orientado a modelos criando o padrão MDA (do inglês, *Model-Driven Architecture*) em 2001(OMG, 2003). Os três objetivos principais que o MDA procura no desenvolvimento do padrão são: (1) portabilidade; (2) interoperabilidade; e (3) reuso (OMG, 2003).

Para alcançar estes três objetivos listados, MDA descreve três principais camadas de abstração arquitetural chamadas de *pontos de visão* (já apresentados na Seção 3.3.1): (i) independente de computação (CIM); (ii) independente de plataforma (PIM) e (iii) específico de plataforma (PSM). O primeiro descreve o ambiente do sistema e seus requisitos numa terminologia que é familiar para profissionais do domínio do sistema, de forma a não ficar dependente de mudança arquitetural ou tecnológica. O segundo descreve a estrutura do sistema e suas funcionalidades, mas não descreve a plataforma tecnológica utilizada. O terceiro reflete uma solução tecnológica para os dois *pontos de visão* anteriores detalhando o que é importante para uma implementação do sistema em uma determinada plataforma. O objetivo principal é tratar melhor a separação dos requisitos, o projeto e as tecnologias de implementação numa arquitetura de software, de forma a permitir alterar qualquer um desses três elementos de modo disjunto.

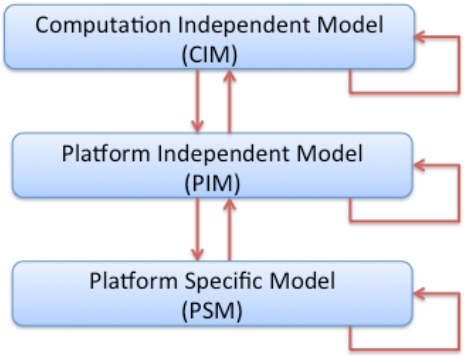


Figura 3.6: Camadas arquiteturais do MDA e transformações de modelos

Como a Figura 3.6 sugere, no MDA o mapeamento entre modelos (ou transformações) é um ponto chave. Cada seta na Figura 3.6 representa uma transformação de um modelo para outro e o MDA não define como essas transformações devem ocorrer. Outra questão é que as transformações podem ocorrer não apenas numa direção (setas de baixo para cima da Figura 3.6). Por exemplo, é possível existir transformações PIM-para-CIM e PSM-para-PIM (nesse caso, pode representar uma refatoração ou engenharia reversa). Mais um ponto importante é que podem existir mapeamentos entre pontos de visão para eles mesmos representando refinamentos dos modelos (por exemplo, CIM-para-CIM, PIM-para-PIM e PSM-para PSM). Também é possível existirem várias representações para cada *ponto de visão,* em outras palavras, vários modelos CIM, PIM e PSM.

As transformações entre CIM e PIM têm um desafio maior pois abrangem mais decisões e maiores possibilidades de interpretação dos requisitos. Já para as transformações entre PSM e código-fonte, normalmente tem-se uma automação bastante abrangente, uma vez que o PSM está fortemente relacionado com as tecnologias de implementação. Também é comum existir mais de uma representação PSM, uma vez que se pode gerar código para mais de uma plataforma.

Existe um guia no MDA que descreve um conjunto de tipos de transformação, técnicas e padrões (OMG, 2003). Uma forma de visualizar de forma geral essas recomendações é entender as camadas de modelagem e meta-modelagem do MDA composta por quatro níveis (Figura 3.7):

- M3: camada de meta-metamodelo: descreve conceitos que serão utilizados na camada de metamodelo. Como exemplo MDA, temos o MOF (do inglês, Meta-Object Facility), que descreve conceitos utilizados na UML. Uma analogia seria o conceito de Class na linguagem Java.
- M2: camada de metamodelo: apresenta conceitos que compõem uma linguagem de modelagem. O MDA cita como exemplos o uso de metamodelo UML (UML Metamodels) ou perfis UML (UML Profiles) ou um metamodelo específico de domínio (DSM Metamodel) criado e customizado para uma área de atuação particular ou segmento da indústria. Analogamente, na linguagem Java, existem os elementos (Attribute e Class) que podem compor uma classe.
- M1: camada de modelo de usuário: representa instâncias de modelos gerados a partir das definições dos metamodelos da camada M2. No caso da UML e um dado perfil, essa camada é re-

presentada pelos diagramas UML, e para o DSM, as instância de modelos específicos de domínio. De forma idêntica, na linguagem Java teria as definições das classes para um determinado sistema (na Figura 3.7 representadas pela classe *Carro*).

- M0: camada de instância de dados: constituídos de artefatos relacionados a objetos, registros e dados. Na linguagem Java, seria equivalente às instâncias de objetos.

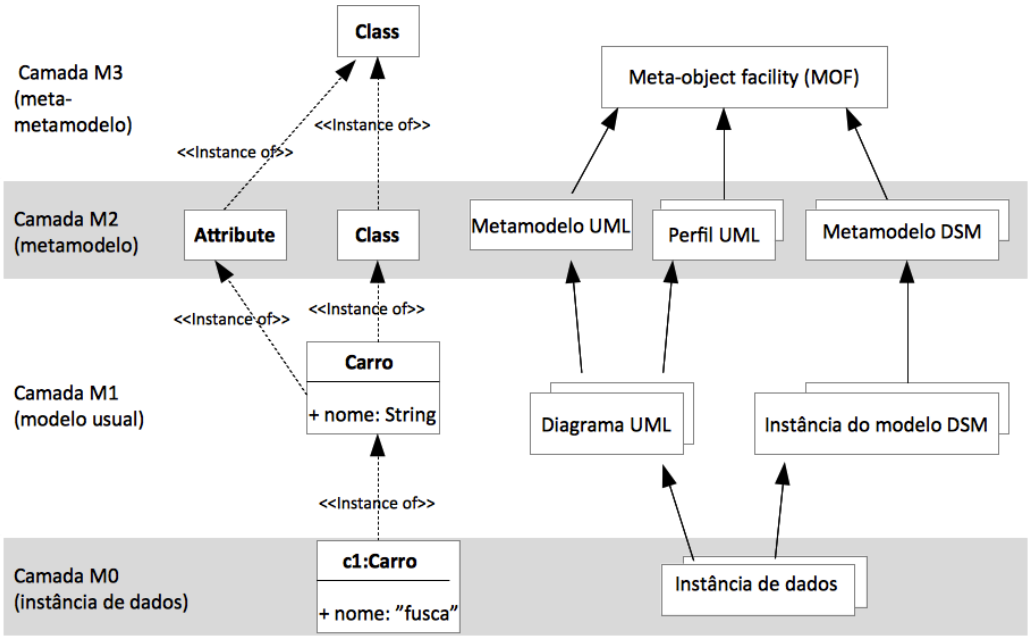


Figura 3.7: Camadas de modelagem e metamodelagem na MDA

3.3.2 Modelagem Específica de Domínio

Apesar do MDA também sugerir que um processo de desenvolvimento possa atender um domínio específico utilizando metamodelos DSMs (ver Figura 3.7) as abordagens MDA na prática utilizavam apenas UML ou Perfis UML, tratando da modelagem de domínios demasiadamente genéricos (SELIC, 2008). Numa direção oposta é que surgiram as abordagens que tratam uma modelagem específica de domínio (do inglês, DSM – *Domain-Specific Modeling*) e se baseiam em criar modelos que utilizam conceitos de um domínio particular utilizando linguagens específicas de domínio (do inglês, DSL – *Domain-Specific Language*). A ideia principal é que cada solução DSM se concentre em atender problemas com escopo menor de modo a obter um nível de automatização maior, uma vez que domínios menores possuem menor complexidade do que os de propósito geral e, por isso, são mais fáceis de definir. Consequentemente, as soluções DSM são aplicadas habitualmente para um produto particular ou ambiente ou plataforma específica. Uma questão crucial das abordagens DSM é que os metamodelos não precisam ser compatíveis com o meta-metamodelo MOF e nem com o metamodelo da UML, em outras palavras, é comum dizer que as abordagens DSM são metodologias MDD sem a obrigatoriedade de uso dos padrões da OMG.

Uma DSL é uma linguagem criada para uma classe específica de problema (FOWLER, 2013) e, normalmente, tem um conjunto pequeno de elementos representados por uma linguagem declarativa ou uma especificação passível de execução. As abstrações e notações de uma DSL devem possuir um poder de expressão com foco e restrito a um problema particular de modo a ser intuitivamente claras para especialistas desse domínio. Uma DSL é composta por três elementos: (i*) a sintaxe abstrata* que define os conceitos do domínio, e as relações e restrições que se aplicam a estes conceitos. Nas linguagens de modelagem, corresponde ao metamodelo que define a estrutura dos modelos que podem ser criados; (ii) a *sintaxe concreta* que provê um sistema para representar os conceitos do domínio de forma concreta.; e (iii) *a semântica* que define o significado dos elementos da sintaxe abstrata, e pode variar de acordo com o objetivo desejado. No contexto do MDD, a semântica é definida em forma de ações a serem executadas por um interpretador automático. Exemplos de DSLs populares são a SQL (do inglês, *Structure Query Language*) e a HTML (do inglês*, Hypertext Markup Language*). Já para domínios de negócios específicos, é possível encontrar DSLs nos mais diversos segmentos: agentes móveis, gerenciamento de equipamento de redes de computadores, bibliotecas de aceleração de vídeo, sistemas financeiros, projeto de hardware, etc.

Para aumentar a facilidade de uso e agilidade na especificação de um sistema (ou parte dele), várias DSLs permitem construir os modelos graficamente, tais DSLs são conhecidas como DSL visuais ou gráficas. Contudo, como Greenfield et al (2004) apontam, alterar a representação dos modelos sem aumentar seu nível de abstração não traz ganho de produtividade. Nesse sentido, experiências anteriores mostram que não adianta apenas usar UML e seus elementos para definir uma DSL, pois muitas vezes ao invés de simplificar, o novo conjunto de modelos aumenta a complexidade em alguns casos (TOLVANEN e KELLY, 2005).

Segundo Czarnecki (2004), algumas das vantagens que DSLs podem proporcionar são:

- Abstrações específicas do domínio: uma DSL fornece 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 é mais difícil 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 captu-

rado por uma DSL pode ser utilizado para disponibilizar um suporte mais inteligente por parte das ferramentas aos desenvolvedores.

3.3.3 Abordagens de desenvolvimento de família de sistemas

Como visto anteriormente, o DSM procura representar um domínio do problema utilizando abstrações de alto nível, especializadas em uma ou mais DSLs, para diminuir a distância até um domínio de solução. O objetivo é melhorar a qualidade e produtividade do desenvolvimento de software por meio de encapsulamento das complexidades das plataformas tecnológicas em modelos e, posteriormente, em transformações automáticas de artefatos de implementação que refletem a solução expressa nos modelos (SCHMIDT, 2007). Porém, apesar de encapsular o conhecimento necessário para gerar esses artefatos por intermédio da modelagem e transformações, o MDD não trata da questão da reutilização do conhecimento contido nos artefatos modelados para um determinado domínio. Nesse contexto, apesar de áreas distintas, é possível observar que as abordagens MDD e de reutilização de software procuram utilizar algum conhecimento prévio num processo de desenvolvimento de software com o objetivo de reduzir esforços repetitivos. Segundo Lucrédio (2009), esta característica comum se destaca em metodologias e técnicas sistemáticas que identificam pontos comuns e variáveis de um domínio, ou abordagens de desenvolvimento de família de sistema. Dessa forma a partir dessas observações, a seguir, são descritos alguns esforços da área de reutilização de software que podem auxiliar o desenvolvimento orientado a modelos.

Um dos primeiros esforços que teve grande visibilidade para alcançar o reuso de software foi o Desenvolvimento Baseado em Componentes (SZYPERSKI, 2002). A ideia é construir artefatos de implementação de software que: (a) são desenvolvidos independentemente e entregue como unidades; (b) têm interfaces explícitas e bem definidas para os serviços que oferece; (c) têm interfaces explícitas e bem definidas para os serviços que requer, podendo, inclusive detalhar suas dependências com o ambiente de execução; e (d) podem ser compostos com outros componentes, talvez após a customização de algumas propriedades, mas sem modificar os componentes em si (D'SOUZA e WILLS, 1999). Tal abordagem é considerada com um nível de reuso de baixa granularidade.

Posteriormente, surgiram as abordagens de Engenharia de Domínio, que ao invés de fazer o reuso de componentes individuais, estabelecem que é mais vantajoso reutilizar um projeto ou subsistema inteiro constituído de componentes e suas interconexões (GOMAA, 2005). Isso implica em poder empregar o reuso na especificação requisitos, projeto arquitetural, código-fonte e testes. O objetivo é aplicar uma metodologia com um nível de reuso de alta granularidade. Para tanto, a Engenharia de Domínio define um conjunto de atividades para coletar, organizar e armazenar a experiência do desenvolvimento de software de um determinado domínio (um projeto completo ou parte dele) na forma de artefatos reutilizáveis de um modo que possam ser aplicados em outros sistemas facilmente (CZAR-

NECKI e EISENECKER, 2000). Essas atividades são divididas em três fases básicas: análise, projeto e implementação de domínio, que por sua vez podem produzir os respectivos artefatos: (i) um Modelo de *Features*[3]; (ii) uma arquitetura do domínio; e (iii) componentes reusáveis e unidades de implementação.

A modelagem de *features* (KANG et al., 1990) procura identificar quaisquer conceitos ou características relevantes de um domínio do ponto de vista do produto de software que se pretende construir. O objetivo é simplificar a definição dos pontos comuns e variáveis de um domínio de modo a estabelecer um meio de comunicação intuitivo entre os interessados (do inglês, *stakeholders*) no uso e desenvolvimento do sistema (LEE e MUTHIG, 2006). A seguir são explicados os principais elementos para representar similaridades e variabilidades entre um produto de software em termos de requisitos funcionais e não-funcionais utilizando a técnica de modelagem de *features* (KANG et al., 1998):

- É possível identificar e classificar as *features* como sendo: (1) *de capacitação*- características visíveis para o usuário (serviços, operações e características não-funcionais); (2) *de tecnologia do domínio*- descrevem formas para se implementar serviços ou operações; (3) *de técnicas de implementação*- funções ou técnicas genéricas utilizadas para implementar serviços, operações e funções do domínio; (4) *de ambiente de operação*-representam o ambiente no qual as aplicações são executadas.
- Outra classificação diz respeito à presença da *feature* no produto final e podem ser: (1) o*brigatórias*- funcionalidades presente em todos os produtos; (2) *opcionais*- funcionalidades que podem estar ou não em um produto; (3) alternativas- funcionalidades mutuamente exclusivas, ou seja, apenas uma delas deve estar no produto.
- Existem também as relações estruturais e conceituais das *features* e que são de três tipos: composição, generalização e implementação. Há também regras complementares de relações de dependência ou exclusão mútua entre as *features*.

O modelo acima se baseia na presença ou ausência das *features*, e consegue representar uma gama enorme de variabilidades presente na maioria dos domínios. Entretanto, alguns autores propõem extensões para aumentar o poder de expressão dos modelos de *features*, por exemplo: uso de cardinalidades, atributos, extensão das categorias, modularização, etc. Mais detalhes podem ser encontrados em Czarnecki, Helsen e Eisenecker (2004).

Dando prosseguimento ao histórico da evolução de metodologias de desenvolvimento de software que buscam o reuso, no final da década de 90 surgiram propostas baseadas em Linhas de Produto de Software (LPS) (WEISS e LAI, 1999; CLEMENTS e NORTHROP, 2001), que consistem em

[3]Neste trabalho foi adotado o termo original em ingles "*feature*" para referenciar uma característica de um domínio, pois acredita-se que possui uma denotação mais direta e menos ambígua que sua tradução para o portugues (característica), que pode ser empregada em outros contextos.

um conjunto, ou família, de produtos pertencentes a um determinado domínio ou nicho de mercado, que tem como base uma arquitetura comum. O objetivo geral é ampliar a eficiência dos processos de desenvolvimento anteriores aproveitando ideias de experiências empíricas de outras indústrias, por exemplo, eletrônica de consumo e automobilística. Dentre os avanços é possível citar: (i) sistematização do desenvolvimento com reuso a partir de artefatos criados pela Engenharia de Domínio; (ii) definição de mecanismos de gerenciamento das variabilidades apoiadas por ferramentas de derivação de produtos (por exemplo, automatizar a sincronização de mudanças no Modelo de *Features* com os artefatos relacionados); e (iii) alinhar as atividades de desenvolvimento de software com requisitos de negócio.

Durante os últimos anos, a comunidade científica e industrial tem proposto várias metodologias para o desenvolvimento das LPS e, atualmente, são considerados como referência três tipos de processos: *Engenharia de Domínio*, *Engenharia de Aplicação* e *Gerenciamento*[4] (Figura 3.8).

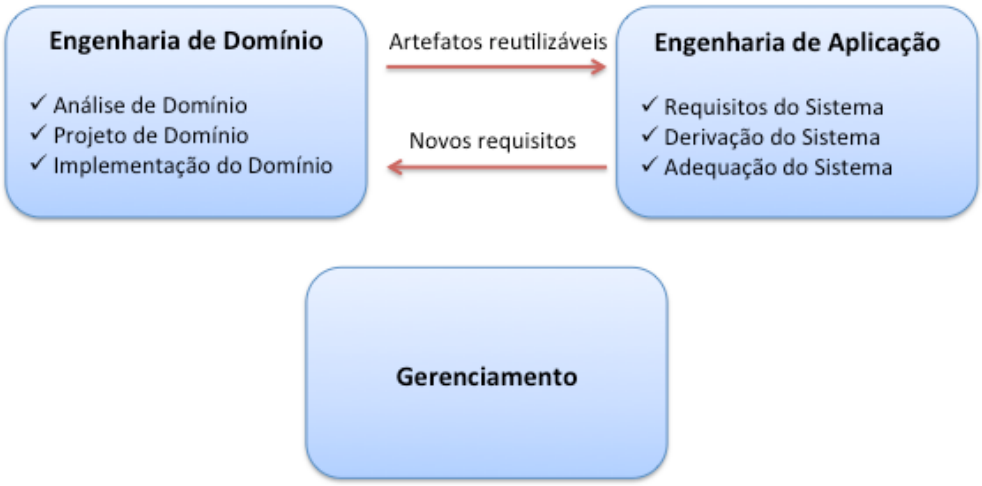


Figura 3.8: Principais processos no desenvolvimento de LPS (CZARNECKI, 2004)

A *Engenharia de Domínio* (também chamada de *desenvolvimento da linha de produto* ou *desenvolvimento de artefatos de núcleo*) já foi citada anteriormente e compreende a definição de artefatos (modelos de análise e projeto, componentes de software, DSLs, geradores de código-fonte, etc.) para habilitar o desenvolvimento com reuso. Assim como no desenvolvimento de um sistema individual, possui as três fases iniciais de desenvolvimento: Análise, Projeto e Implementação. Todavia, essas fases são executadas, considerando que a arquitetura de software será estabelecida, por artefatos que compõem a parte comum (*core*) e artefatos que compõem a parte variante (variabilidades) de um determinado domínio de sistemas (CZARNECKI, 2004). Portanto, os artefatos gerados devem refletir

[4]O terceiro processo (Gerenciamento) de LPS está fora do escopo deste trabalho, mas suas atividades incluem, entre outras: definir estrutura organizacional, alocar recursos, coordenação, treinamento, estratégia e plano de adoção. Mais detalhes consultar (WEISS e LAI, 1999; CLEMENTS e NORTHROP, 2001).

essas peculiaridades presentes em família de sistemas. Por exemplo: (1) na *Análise de Domínio*- Modelo de *Features* e modelo de requisitos anotados com variação; (2) no *Projeto do Domínio*- arquitetura baseada a componentes que abrange a família de sistema alvo, projeto de software detalhando variações possíveis nessa arquitetura e um plano de produção que indique como podem ser gerados a partir dos artefatos disponíveis (Projeto de Domínio); e (3) na *Implementação do domínio* DSLs (*wizards*, diagramas etc.) e geradores de código-fonte que auxiliam uma derivação manual ou automática dos artefatos reutilizáveis (componentes, biblioteca de classes, etc.).

Já a Engenharia de Aplicação (também conhecida como *desenvolvimento do produto*) tem como objetivo produzir o produto de software a partir de um conjunto de *Requisitos do Sistema* e combinação dos artefatos definidos na Engenharia de Domínio de modo a permitir um desenvolvimento com reuso. Uma das atividades desse processo é a *Derivação do Sistema*, onde, os desenvolvedores, com base nos requisitos desejados pelo(s) cliente(s), configuram um conjunto de *features* da LPS, e, então, o produto é automaticamente construído, a partir dos artefatos de implementação produzidos na Engenharia de Domínio.

Na Figura 3.8 também é possível observar que o processo de Engenharia de Domínio oferece ao processo de Engenharia de Aplicação os artefatos reutilizáveis. No sentido oposto, a Engenharia de Aplicação pode oferecer um conjunto de novos requisitos que inicialmente não foi previsto pela Engenharia de Domínio e, por conseguinte, não cobertos pelos artefatos reutilizáveis. É papel do engenheiro de aplicação a *Adequação do Sistema* de acordo com necessidades dos clientes, porém, novos requisitos devem ser notificados para o engenheiro de domínio sincronizar os artefatos reutilizáveis com os novos requisitos do produto (CZARNECKI, 2004).

Um mecanismo essencial para a construção de uma LPS é a implementação da derivação de produto (DEELSTRA, SINNEMA e BOSCH, 2005). Tal atividade pode ser dividida em duas principais ações. A primeira relacionada com a Engenharia de Domínio que consiste no gerenciamento dos modelos generativos e, por consequência, das variabilidades. A segunda associada à Engenharia da Aplicação, na medida em que permite construir os produtos da LPS a partir das configurações dos modelos, artefatos e seleção das *features* desejadas no produto (Engenharia de Aplicação). Uma das principais propostas que tratam especificamente a derivação de produtos, considerando uma família de sistemas, é o Desenvolvimento Generativo de Software (CZARNECKI, 2004). Neste trabalho são empregadas várias técnicas propostas por essa comunidade.

3.4 Ferramentas de Suporte ao MDD

O *Eclipse Modeling Framework* (EMF) é um framework da plataforma *Eclipse* que disponibiliza uma ferramentas para implementar práticas MDD. Ele disponibiliza um arcabouço para a especificação dos metamodelos do usuário e um conjunto de transformadores. No EMF, os metamodelos se-

guem como base o meta-metamodelo denominado *Ecore*, o qual constitui em uma alternativa mais simples e eficiente ao MOF (STEINBERG et al., 2008).

O framework é dividido em duas partes: o EMF e o EMF.Edit. O EMF em si é composto pelo meta-metamodelo *Ecore*, que é responsável pelo suporte na produção da especificação e representação dos metamodelos, e pelo gerador EMF, ferramenta de geração que cria uma implementação em Java do modelo especificado. O *EMF.Edit* é utilizado como um conjunto de classes para criar representações gráficas do modelo (editores gráficos), servindo de certa forma para manipulá-lo, através de operações de criação, edição e remoção dos objetos do modelo. O gerador EMF utiliza a API do *EMF.Edit* como base principal para a geração dos editores gráficos dos modelos do usuário.

A partir de métodos de geração de código, o EMF permite produzir uma implementação concreta em Java para manipular em memória as instâncias do modelo. Assim, tomando como entrada uma especificação formal de um modelo, um código em Java é produzido resultando em menor esforço no desenvolvimento da aplicação.

A Figura 3.9 apresenta as entidades envolvidas e a dinâmica no funcionamento da abordagem presente no EMF. A filosofia do EMF é partir de uma especificação (metamodelo do usuário) baseada no modelo *Ecore* e obter, por meio de geração de código, um editor visual que durante sua execução manipula em memória instancias deste metamodelo.

A primeira etapa no desenvolvimento consiste em definir o metamodelo do usuário (normalmente já descrito como um modelo conceitual) em termos dos conceitos presentes no modelo *Ecore*. Esta é a única tarefa que exige esforço do usuário, pois as tarefas restantes são feitas pelo próprio EMF. Com o metamodelo definido, a próxima etapa é gerar o editor a partir deste modelo. Para tornar o metamodelo independente e evitar que ele possa se misturar com conceitos exclusivos da geração de código, o EMF define o modelo denominado modelo gerador (*Generator Model* ou *genmodel* no EMF), sendo um modelo de geração exclusivo para gerar o código de manipulação do modelo (editor visual).

Assim como o *Ecore*, o modelo gerador é em si um modelo EMF. De fato, um modelo gerador provê acesso a todos os dados necessários para a geração, incluindo referências a parte do metamodelo *Ecore*, por meio de referências aos elementos presentes no metamodelo correspondente (STEINBERG et al., 2008). O editor é obtido a partir do modelo gerador.

A segunda etapa transforma o metamodelo em uma implementação em Java que é destinada para manipular o metamodelo. A geração pode ser enxergada em duas perspectivas diferentes. Na primeira é gerado um código responsável por manipular o modelo, permitindo criar e editar os elementos e atributos do modelo em memória (RAM).

Este código, normalmente chamado de código do modelo, é gerado na linguagem Java e mapeia cada conceito do metamodelo em termos de classes, atributos e métodos. Na segunda perspectiva de geração é possível gerar um editor visual do modelo que é destinado a editar o modelo visualmente.

Os visualizadores e editores gráficos, para certo modelo, apresentam e editam instancias reais do modelo usando visões gráficas do *JFace* (visão hierárquica em forma de árvore, tabela, lista, etc.) e a folha de propriedades do Eclipse. O editor visual pode ser gerado em um único *plugin* ou em dois *plugins* separados, o *Edit* e o *Editor*. O *Edit* corresponde a classes que manipulam o modelo independente da API do eclipse, já o *Editor* corresponde a classes de interface gráfica que são dependentes do Eclipse como o *JFace*.

No EMF a geração de código é realizada pela tecnologia JET (*Java Emitter Template*). O JET é um motor de transformação baseado em *templates* do tipo modelo para código. O modelo base é um modelo EMF e a saída são classes de implementação Java. Este passo do processo é chamado de tradução. O *template* utiliza uma sintaxe que é muito intimamente relacionada com a sintaxe do *Java Server Pages* (JSP).

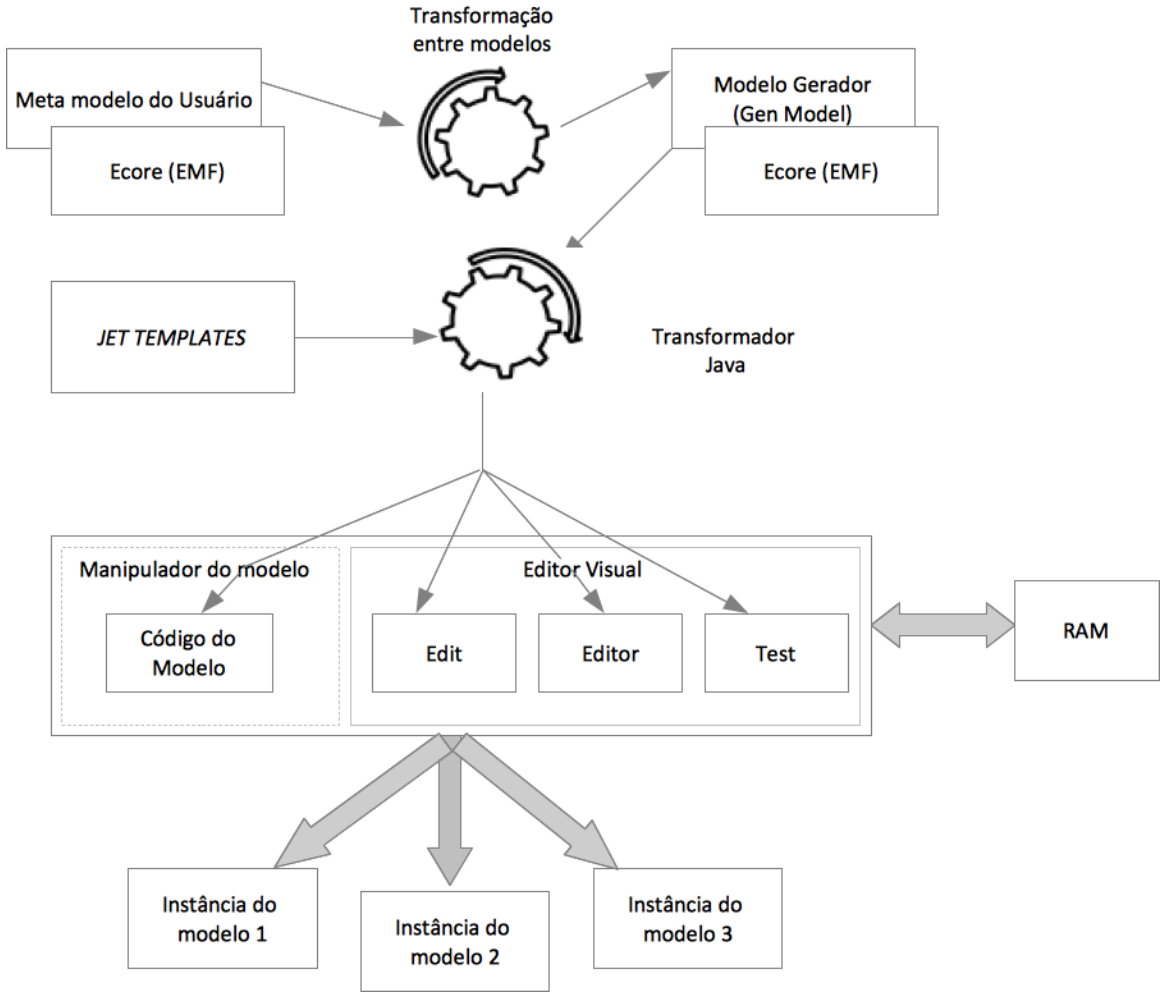


Figura 3.9: Funcionamento da plataforma EMF

3.5 Considerações sobre o capítulo

Neste capitulo foram apresentados os principais conceitos, histórico e técnicas do desenvolvimento orientado a modelos. As abordagens MDD procuram elevar o nível de abstração no qual a equi-

pe de desenvolvimento trabalha, ocultando detalhes da plataforma de execução e empregando mecanismos de automação do computador para auxiliar nas tarefas de mapeamento entre problema e solução e também nas tarefas repetitivas.

As vantagens no emprego dos mecanismos do MDD foram também apresentadas. Porém, deve-se observar a importância da existência de um processo de desenvolvimento bem definido, que ofereça um suporte à adoção de práticas comprovadas e sistemáticas. Caso contrário, os benefícios associados ao uso do MDD podem perder seu potencial e, ao mesmo tempo, aumentar o impacto das desvantagens. Na literatura, é possível encontrar diversas contribuições nas áreas de processos para MDD. Porém, o uso do MDD para auxiliar a integração de visões distintas de projeto no domínio de aplicações multimídia permanece pouco explorada.

Neste trabalho, procura-se combinar as abordagens propostas pela OMG-MDA, Modelagem Específica de Domínio e Desenvolvimento de Família de Sistemas num contexto específico de desenvolvimento de aplicações para TVD. Tal domínio será detalhado no próximo capítulo. Nesta pesquisa foram consideradas as tecnologias para modelagem e meta-programação do EMF. Tal uso se dá por questão de familiaridade e experiência anterior do autor com essa plataforma. A definição de pontos positivos e negativos em relação ao uso de outras abordagens e tecnologias faz parte do escopo desta pesquisa. De qualquer forma, mais informações sobre outras alternativas podem ser consultadas em Lucrédio (2009).

4 CONCEITOS DE TV DIGITAL

Este capítulo apresenta os conceitos básicos relacionados à televisão digital (TVD) por meio do funcionamento geral da sua plataforma, exemplos de aplicações e abordagens atuais para desenvolvimento de aplicativos neste domínio específico.

4.1 Plataforma de TV Digital

Num primeiro momento duas mudanças se destacam com a chegada da TV Digital (TVD): a melhoria na qualidade do áudio e vídeo e a possibilidade de recepção móvel devido ao emprego de técnicas mais robustas e eficientes de codificação e transmissão de sinais digitais. Além desses avanços, um sistema de TVD permite também a evolução na forma das pessoas acessarem o conteúdo por meio do envio de softwares e dados juntamente com os fluxos audiovisuais transmitidos pelas emissoras. Com esse recurso, conhecido como interatividade, o telespectador deixa de ser um elemento passivo no processo de comunicação, passando a ter condições de interferir no conteúdo enviado pela emissora de TV. Outro aspecto importante, é a possibilidade de integração com outras tecnologias, como é o caso da Internet. Com a convergência entre TV e Internet, o receptor, torna-se próximo às funcionalidades de um computador.

Considerando uma diversidade de soluções tecnológicas que podem ser adotadas para se definir um sistema de TVD, diversos órgãos de padronização concentraram esforços na especificação de padrões. No Brasil, o resultado desses esforços culminou no padrão brasileiro de TV Digital (SBTVD – Sistema Brasileiro de Televisão Digital).

Os diferentes padrões espalhados pelo mundo organizam um sistema de televisão digital baseado em uma arquitetura de camadas, agrupadas basicamente em dois grupos gerais. O primeiro grupo refere-se a tecnologias e padrões que implementam funcionalidades de baixo nível do sistema responsáveis pela modulação do sinal de difusão, multiplexação e transporte de fluxos elementares de áudio, vídeo e dados e, por fim, codificação e decodificação de áudio e vídeo. No segundo, encontram-se as especificações relacionadas à camada de *middleware* que é responsável pela execução das aplicações, independente de plataforma. Ele é embarcado em um hardware especial semelhante aos computadores pessoais, formando um receptor digital (mais conhecido como terminal de acesso ou *set-top box*) capaz de reproduzir áudio e vídeo de alta-qualidade e executar aplicações na TV.

O SBTVD foi baseado no padrão de TV digital japonês, com modificações na camada de compressão e na camada de *middleware*. No caso da compressão de vídeo, todos os padrões de TV Digital Terrestre empregam o MPEG-2. O Brasil, no entanto, emprega uma técnica de compressão de vídeo mais recente e mais eficiente, denominada de H.264. Com esta técnica de compressão de vídeo, é possível manter a qualidade de imagem, porém reduzindo sensivelmente a taxa de bits.

Como é esperado que o conjunto dos terminais receptores inclua equipamentos elaborados por fabricantes distintos, a camada do *middleware* pode permitir que as aplicações sejam produzidas independentemente de qual STB o espectador possua (FERNANDES, LEMOS e SILVEIRA, 2004). Torna-se então fundamental a padronização para oferecer aos desenvolvedores um ambiente de programação que seja comum a todas as plataformas de hardware existente.

O Ginga é a especificação de *middleware* do SBTVD, resultado da integração das propostas FlexTV (LEITE, 2005) e MAESTRO (SOARES, 2006), desenvolvidas por consórcios liderados pela UFPB e PUC-Rio no projeto SBTVD , respectivamente.

O FlexTV, proposta inicial de *middleware* para execução de aplicações imperativas do SBTVD, incluía um conjunto de APIs compatíveis com outros padrões além de funcionalidades inovadoras, como a possibilidade de comunicação com múltiplos dispositivos, permitindo que diferentes usuários pudessem interagir com uma mesma aplicação interativa a partir de dispositivos remotos. Já o MAESTRO foi a proposta inicial de *middleware* para a execução de aplicações declarativas do SBTVD. O foco era oferecer facilidade do sincronismo espaço-temporal entre objetos multimídia, utilizar a linguagem declarativa NCL e agregar as funcionalidades da linguagem de script da linguagem Lua.

O Ginga integrou estas duas soluções, chamadas de Ginga-J (SOUZA FILHO, 2007) e Ginga-NCL (SOARES, 2007), tomando por base as recomendações internacionais da ITU (ITU J200, 2001) Desta forma, o Ginga é subdividido em dois subsistemas interligados, também chamados de Máquina de Execução (Ginga-J) e Máquina de Apresentação (Ginga-NCL) conforme ilustrado na Figura 4.1.

Figura 4.1: Visão geral da arquitetura do *middleware* Ginga

Outro aspecto importante é que os dois subsistemas do Ginga não são necessariamente independentes, uma vez que a recomendação do ITU inclui uma "ponte", que deve disponibilizar mecanismos para intercomunicação entre os mesmos, de um modo que as aplicações imperativas utilizem serviços disponíveis nas aplicações declarativas, e vice-versa. Portanto, é possível a execução de aplicações híbridas em um nível acima da camada dos ambientes de execução e apresentação, permitindo

agregar as facilidades de apresentação e sincronização de elementos multimídias da linguagem NCL com o poder da linguagem orientada a objetos Java.

O Núcleo Comum Ginga (*Ginga Common Core*) é o subsistema do Ginga responsável por oferecer funcionalidades específicas de TV Digital comuns para os ambientes imperativo e declarativo, abstraindo as características específicas de sistema operacional e hardware para as outras camadas acima. Como suas principais funções, podemos citar: a exibição e controle de mídias; o controle de recursos do sistema (o canal de retorno, dispositivos de armazenamento); acesso a informações de serviço; sintonização de canais, entre outros. A máquina virtual Java é responsável pela execução das aplicações Java, bem como implementação da plataforma base da tecnologia Java no receptor.

4.2 Aplicações de TV Digital

A migração da TV analógica para a TV Digital cria oportunidades para a produção e desenvolvimento de novos serviços dos mais diversos tipos, por exemplo, os descritos na Tabela 4.1.

Tabela 4.1 Cenários de aplicações de TV Digital (SCHWALB, 2004)

Cenários	**Descrição**
Publicidade	Acesso personalizado aos conteúdos. Ex: redirecionar o usuário para um site na Web.
EPG	Busca e navegação através de *Guias Eletrônicos de Programas.*
VOD	Busca e visualização de conteúdos sob demanda.
Jogos	Jogos, normalmente descorrelacionados do conteúdo. Ex: dama, quebra-cabeça, etc.
Conteúdo adicional	Programação adicional. Seleção de uma câmera de vídeo entre várias câmeras possíveis ou seleção de regiões do vídeo para acessar informações adicionais sobre o programa.
Conteúdo educacional	Programação educativa. Ex: mudança de linguagem (áudio); carregar animações (ou simulações) sobre um tópico; responder a questões de múltipla escolha, etc.
Quizz	Aplicações de *quizz* (responder perguntas de múltipla escolha).
Noticiários	Noticiários. Ex: pequenas manchetes ou informações adicionais sobre a notícia
Esporte	Programas esportivos. Ex: melhores momentos, acessar estatísticas, mudar de câmera, etc.
Transporte	Reservas ou verificar horários de trens, ônibus ou aviões (comum na Europa)
Comercial	Serviços comerciais. Ex: *home banking*, *home shopping*, apostas, corridas de cavalo, etc.

Para melhor contextualizar o uso da abordagem apresentada neste trabalho propõem-se a organização das aplicações de TVD de acordo com 3 (três) categorias propostas por Bachmayer (2010): *Push-Delivery, Feedback channel e Distributed*. Esta taxonomia considera que qualquer aplicação multimídia é uma estória que tem o intuito de imergir o usuário no fluxo da sua narrativa, através da interação com um mundo virtual de comunicação entre os participantes deste mundo e acessando objetos de mídia num modo natural e guiado. O objetivo é contemplar os principais cenários de utilização

dos serviços de TVD de modo genérico, para permitir a evolução e inclusão de novas aplicações na metodologia de desenvolvimento. As descrições de cada categoria são apresentadas a seguir:

- ***Push delivery:*** agrupa os programas pertinentes à perspectiva tradicional da TV analógica. Apesar de acessar uma aplicação, o usuário possui limitações de interatividade diante do conteúdo produzido e transmitido pela emissora. São consideradas apenas as narrativas lineares e aquelas com possibilidade limitadas de opções. A modificação do desenvolvimento do fluxo da narração (conhecida como ramificação) é obtida com a adição de estórias paralelas dentro de um mesmo vídeo ou utilizando mais de um fluxo na transmissão. No primeiro, a mudança da narração é realizada pela emissora e, no segundo, o telespectador toma a decisão de mudança ao mudar de fluxo para acompanhar outro ponto de vista da estória atual. Nesses cenários, é importante observar que o usuário tem influência limitada para alterar o desenvolvimento da estória assim que ele recebe. Tal modelo permite a navegação e interatividade local através de dispositivos de interação como controle remoto, teclado ou joystick. A resposta da interação com a emissora é sempre indireta, utilizando telefone, SMS, fax, e-mail, entre outros. O conteúdo e os dados da aplicação se limitam ao que foi predefinido pela TV e que foi transmitido através de um canal unidirecional (difusão) controlado pela emissora (daí o nome *push*).
- ***Feedback channel***: a diferença deste modo em relação ao anterior está na presença de uma comunicação direta e bidirecional com a emissora. Tal conexão, também conhecida como canal de retorno, pode ser realizada através de várias tecnologias: linha discada, ASDL, *Wi-Fi*, *Wi-Max*, 3G, etc. Como consequência, existe a possibilidade de entrega de conteúdo individual, diferente do modo *Push Delivery*, onde o conteúdo interativo é coletivo e todos os usuários recebem as mesmas possibilidades de interação. Tal aspecto tem fundamental importância nas possibilidades de construção das narrativas das aplicações, pois agora é possível criar ramificações bem mais elaboradas. Por exemplo, é possível incluir na execução do programa interativo um desafio, condição de acesso ou obstáculo que deve(m) ser resolvido(s) por um ou vários usuários antes que o fluxo previsto na narrativa continue. A outra alternativa é a capacidade de guiar o usuário para caminhos específicos da narrativa limitando o número de opções a seguir de acordo com o resultado da interação com o usuário. Nesse contexto, três opções são possíveis: (i) "labirinto" com caminhos obrigatórios; (ii) "labirinto" com gargalos e (iii) "labirinto" dinâmico. Apesar de permitir formas mais elaboradas para interação com o conteúdo e as narrativas, este conjunto de aplicações pode reutilizar diversas estruturas existentes nos exemplos da categoria *Push Delivery*. Porém, todos estes novos caminhos podem ser enviados individualmente pela emissora através do canal de retorno (por exemplo, novas fases de um jogo), economizando recursos do receptor e não limitando a quantidade de opções permitidas.

- ***Distributed***: nesta categoria se encaixam aplicações mais recentes que exploram 2 (duas) peculiaridades encontradas também nas aplicações Web 2.0: (i) localização e processamento distribuído pela Internet ou em outros dispositivos dos elementos que podem compor a aplicação interativa (por exemplo, um serviço Web) e (ii) colaboração dos usuários na construção da narrativa. O objetivo da primeira característica é desenvolver aplicações que se beneficiam dos efeitos da rede, para se tornarem tanto melhores, quanto mais populares. Nesse caso, o benefício mais importante é agregar conteúdo, funcionalidades ou plataformas completas já disponíveis na rede. Um exemplo deste tipo de aplicação é a integração de uma rede social com a transmissão de algum evento ao vivo para compartilhar suas opiniões e sentimentos com os amigos enquanto assistem a TV (COPPENS; TRAPPENIERS e GODON, 2004). Para o caso de conteúdo produzido com a participação do consumidor, a colaboração pode ser explícita ou implícita. A primeira lida com sistemas que reagem a ações do consumidor através da manipulação do conteúdo transmitido ou criando elementos adicionais para o canal a partir dessas ações. A segunda, por sua vez, lida com sistemas que observam o consumidor e realiza ações em *background*. Um exemplo do primeiro caso é um sistema que auxilia uma emissora na escolha de um conjunto de saídas a partir de centenas de fluxos ao vivo, cobrindo um evento em tempo real onde o usuário pode atribuir notas ao conteúdo enviado e, dependendo da avaliação, a emissora pode modificar o conteúdo transmitido (MAC WILLIAMS e WAGES, 2008). Já para o segundo caso, em Lee et al. (2007) é descrito o *Emotional TV*, que busca a interação do usuário com a interpretação de emoções do telespectador através de uma bola sensitiva como dispositivo de entrada. De forma geral, esta categoria pode ser vista como um conjunto de aplicações que permitem consumir ou produzir conteúdo de/para funcionalidades ou plataformas já disponíveis na Web (por exemplo, um serviço Web). Segundo Ursu et al. (2008) esta categoria de aplicação é caracterizada como aplicações de narrativas evolucionárias, uma vez que o estado final da execução deste tipo de aplicação é não determinístico. Isso porque em nenhum momento do tempo é possível determiná-lo, já que o conteúdo da aplicação não é mais apenas gerado no emissor original (a emissora), mas por sistemas externos (por exemplo, serviços Web).

4.3 Desenvolvimento de Aplicações para TV Digital

A partir do momento em que as emissoras de TV passaram a transmitir dados e objetos junto ao conteúdo audiovisual dos programas, e os terminais de acesso conseguiram processar tal conteúdo como programas de computador, criou-se um novo mercado para desenvolvimento de software. Apesar de existirem softwares relacionados à infra-estrutura de transmissão e ao middleware, este trabalho aborda apenas o desenvolvimento de software relacionado aos aplicativos que são disponibilizados para o usuário final. Nesse contexto, além das aplicações, é comum também fazer uso de tecnologias

de suporte, como por exemplo, linguagens de programação e ambientes de desenvolvimento. Este trabalho tem foco no desenvolvimento de aplicações para TV Digital de forma geral, independente de plataforma de execução e paradigma de programação, porém, como avaliação da abordagem proposta nesta tese considerou apenas a geração de código-fonte para aplicações para a plataforma Ginga e utilizando paradigma declarativo da linguagem NCL, nas próximas seções são descritos os principais elementos da linguagem NCL e dois dos seus principais ambientes de programação: NCL Eclipse e NCL Composer.

4.3.1 NCL

As aplicações para TV digital são, normalmente, construídas usando abordagens linguagens declarativas, imperativas (algumas vezes também chamadas de procedurais) ou ainda, com o uso de uma abordagem híbrida, integrando os dois tipos de linguagens.

As linguagens declarativas tendem a ser mais intuitivas, num primeiro momento, portanto, mais simples para os programadores (também chamados autores, nesse caso), os quais definem: (1) restrições espaciais e temporais (sincronismo) entre as mídias que compõem a aplicação interativa e (2) tratamento de eventos de interação do usuário com aplicação. Uma linguagem declarativa enfatiza a descrição do problema por meio de elementos pré-definidos (*tags*), ao invés da sua decomposição em uma implementação algorítmica. Elas são linguagens de mais alto nível de abstração, usualmente ligadas a um domínio ou objetivo específico, onde o programador fornece apenas o conjunto das tarefas a serem realizadas, não estando preocupado com os detalhes de como elas serão executadas (SOARES, 2007). Como já dito, a linguagem declarativa utilizada no contexto do SBTVD é a NCL e do ponto de vista da Engenharia de Software, pode ser vista como uma linguagem específica de domínio.

As linguagens imperativas tendem a ser mais apropriadas para aplicações genéricas, orientadas a eventos, nas quais o programador possui um maior controle do código, sendo capaz de estabelecer todo o fluxo de controle e execução de seu programa. Neste caso, o software básico do receptor deve ser informado sobre cada passo a ser executado. Pode-se afirmar que, em linguagens imperativas, o programador possui um maior poder sobre o código, sendo capaz de estabelecer todo o fluxo de controle e execução de seu programa. A linguagem mais usual encontrada nos ambientes imperativos dos sistemas de TV Digital é a Java.

Uma outra possibilidade é permitir que o conteúdo declarativo possa referenciar o código-fonte imperativo e dessa forma a aplicação declarativa pode complementar suas funcionalidades e delegar a implementação de lógica de aplicação mais complexa para elementos procedurais (por exemplo, processamento de operações matemáticas, uso de estrutura de dados complexas, envio e recebimento de dados para fontes externas etc.). Estas aplicações, denominadas de híbridas, constituem uma

poderosa ferramenta de produção de conteúdo para TV digital, ao unir as vantagens dos paradigmas declarativo e imperativo.

Uma aplicação declarativa (também chamada de documento multimídia) pode ser definida como um conjunto de composições de fragmentos de informação, compostos por diferentes mídias, como texto, imagens estáticas, áudio, vídeo, animações etc. Para especificar uma aplicação declarativa, portanto, é necessário definir como esses fragmentos de informação se relacionam uns com os outros e isso deve incluir:

- **estrutura de conteúdos**: descrição de cada um dos componentes (fragmentos de informação) do documento.
- **estrutura lógica**: descrição da estrutura lógica dos documentos, isto é, descrição das relações lógicas entre os componentes de documentos.
- **estrutura de apresentação**: descrição da forma como os vários componentes devem ser apresentados, ou seja, onde (em que região, de qual dispositivo) e quando os componentes devem ser apresentados.

A divisão da descrição de um documento nessas três estruturas apresenta as seguintes vantagens:

- **separação da estrutura lógica e de conteúdos**: permite que conteúdos sejam reusados em diferentes partes da estrutura de um documento ou em outros documentos;
- **separação da apresentação e conteúdos**: permite que diferentes conteúdos sejam apresentados de diferentes formas. Exemplo: um texto pode ser representado graficamente por um editor de texto ou como áudio produzido por um sintetizador de voz;
- **separação da lógica** e apresentação: possível reusar a lógica para diferentes apresentações. Exemplo: adaptar a apresentação do documento a um dispositivo (celular, por exemplo) ou QoS diferentes.

Um modelo conceitual de um documento (aplicação declarativa) define mecanismos que permitem descrever documentos. Ele deve representar os conceitos estruturais dos dados, os eventos e relacionamentos entre os dados. De forma geral, os modelos conceituais são baseados no paradigma hipermídia.

O paradigma hipermídia, inspirado no paradigma hipertexto, permite que fragmentos de informação de várias mídias, representados por nós, se conectem entre si por elos associados a pontos de ancoragem (regiões marcadas) dentro dos nós. Assim, é possível navegar por um documento hipermídia (assim como em um hipertexto) de forma não-linear ou não sequencial, selecionando âncoras e seguindo os respectivos elos de um nó para outro. Isso permite definir relacionamentos entre os fragmentos de informação. Um documento com essa estrutura é denominado documento hipermídia. Por exemplo, ao selecionar por um nó, o usuário pode escutar um trecho de áudio ou assistir uma cena de um vídeo da mesma forma que vê os textos contidos em um hipertexto.

Um exemplo de modelo conceitual declarativo que utiliza o paradigma hipermídia é o XHTML, base para a linguagem HTML. O XHTML é muito simples e composto de quatro entidades básicas de acordo com a Figura 4.2. Resumidamente, um nó, também chamado de página, possui um identificador único (um *URI – Uniform Resource Identifier*), uma lista de âncoras e um conteúdo. O conteúdo é formado por um texto (com marcação de formatação) e outros objetos embutidos (W3C, 2002). Os objetos podem ser imagens, scripts, *applets* etc. Uma âncora é uma região (conjunto de informações) marcada de um nó. Âncoras podem ser definidas pela marcação de trechos de um texto de um nó ou pela marcação de todo o conteúdo de um objeto (Trechos de um nó ou todo um objeto). Em XHTML não é possível definir âncoras temporais relacionadas a entidades internas de um objeto, apenas âncoras espaciais.Elos definem relacionamentos de interação entre uma âncora de origem e uma âncora de destino. Elos são definidos embutidos em âncoras de origem.

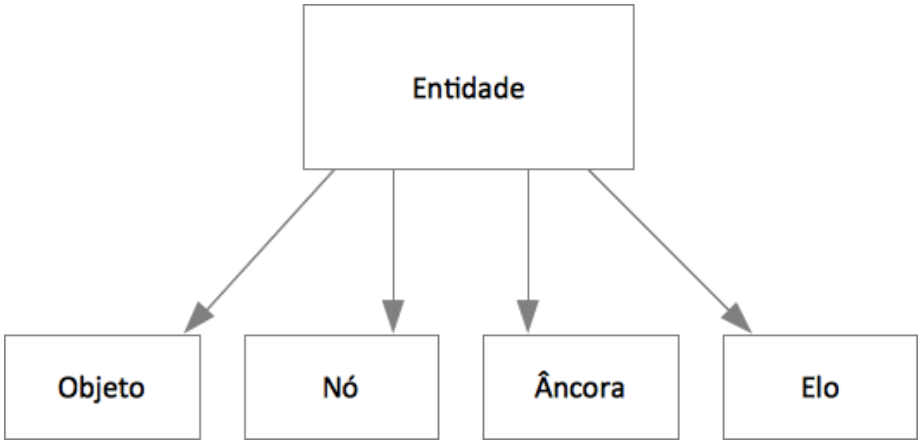


Figura 4.2: Modelo conceitual XHTML básico

A principal vantagem do modelo XHTML é sua simplicidade e essa característica é uma das responsáveis pela sua grande disseminação com linguagem para aplicações Web. A desvantagem é que não possui suporte a relacionamentos de sincronismo entres os objetos e a adaptação de conteúdo e qualquer outra forma de relacionamento tem que ser oferecida por objetos imperativos (*ECMAScrpit*, Java etc.). Assim, o modelo NCM e a linguagem NCL procuram evoluir as características do XHTML e HTML, uma vez que procuram tratar as desvantagens citadas e ao mesmo tempo, seguir o modelo declarativo e o paradigma hipermídia. A seguir são descritos os principais elementos da linguagem NCL.

4.3.1.1 Elementos da linguagem NCL

NCL é uma linguagem declarativa baseada no modelo conceitual NCM (mais detalhes, consultar o Apêndice A). NCL traz uma separação clara entre os conteúdos de mídia e a estrutura da aplicação. Um documento NCL apenas define como objetos de mídia são estruturados e relacionados no tempo e no espaço. Ela não restringe nem prescreve os tipos de conteúdo dos objetos de mídia de uma

aplicação. A NCL possui *tags* para definir todas as entidades do modelo conceitual NCM, como por exemplo:

- `<media>` para definir um nó de conteúdo;
- `<context>` para definir um nó de contexto;
- `<area>` para definir ancorar dentro do nó de conteúdo;
- `<switch>` para definir um nó switch;
- `<property>`, `<port>` e `<portSwitch>`, `<link>`, `<conector>` para definir propriedades, portas switch, elos e conectores, respectivamente. Essas entidades junto com <area> definem os relacionamentos de sincronismo espacial e temporal;
- `<descriptor Switch>` junto com `<switch>` permitem adaptar o conteúdo e a forma como o conteúdo é exibido.
- `<regionBase>` define a área de um dispositivo de saída onde a mídia pode ser exibida. Pode ser usada para definir múltiplos dispositivos de exibição;
- além disso, pode permitir edição ao vivo (em tempo de exibição), através de comandos de edição (`nclEditingCommands`), como `addNode`, `addAnchor`, para adicionar um nó e uma âncora, respectivamente. Na TV digital esses comandos podem ser transmitidos em descritores de eventos DSM-CC.

A NCL possui uma linguagem de script, a linguagem Lua, com um desempenho superior à *ECMAScript* (linguagem de script utilizada pelas principais linguagens declarativas para descrição de aplicações declarativas). Lua é uma linguagem de programação funcional e imperativa, pequena e leve. É dinamicamente tipada, interpretada a partir de *bytecodes* e tem gerenciamento automático de memória com coleta de lixo.

A NCL é utilizada como linguagem declarativa do middleware do Sistema Brasileiro de TV Digital (SBTVD) terrestre, o middleware Ginga. Em virtude disso, sua máquina de execução é denominada Ginga-NCL (e máquina Ginga-J). Recentemente, a linguagem NCL também foi padronizada como padrão ITU-T para serviços IPTV (ITU-T H.761, 2011).

As linguagens declarativas das principais plataformas de TV Digital existentes (com exceção do sistema brasileiro), como o DVB-HTML (padrão europeu), ACAP-X (padrão americano), BML (sistema japonês) são baseadas no modelo conceitual XHTML em conjunto com suporte a CSS, DOM E *ECMAScript*.

Como vimos, XHTML é simples e bastante disseminado por ser usado na Web, mas não possui suporte a relacionamentos de sincronismo e adaptação de conteúdo e por isso é necessário utilizar linguagens imperativas como *ECMAScript*. Todas as linguagens, DVB-HTML, ACAP-Xe BML apresentam pequenas extensões a XHTML, mas apenas BML define extensões para descrever alguns tipos de sincronismo.

Entretanto, mesmo as novas marcações precisam ser complementadas com códigos *ECMAScript*, para descrever a ação a ser executada no relacionamento de sincronização. Isso implica que o nível de abstração para os autores é baixo, obrigando os criadores de programas de TV interativa a trabalharem como programadores de Software (ao contrário de NCL que apresenta um alto nível de abstração).

Nas linguagens DVB-HTML, XDML e BML as características espaciais da apresentação de um componente podem ser definidas em um objeto separado, utilizando CSS, similar ao modelo de regiões e descritores de NCL.

4.3.2 NCL Eclipse

O NCL Eclipse (AZEVEDO, 2011), é uma ferramenta de edição textual para a linguagem NCL integrado ao ambiente do Eclipse através de um *plug-in*. O intuito dela é dar apoio à edição rápida do código fonte. Entre as principais funcionalidades implementadas pelo NCL Eclipse estão:

- coloração de elementos e atributos XML;
- exibição/ocultação elementos XML (permitindo que determinados elementos sejam escondidos ou exibidos pelo desenvolvedor);
- auto-formatação do código XML;
- validação do documento NCL;
- sugestão de código NCL de forma contextual (auto completar);
- navegação no documento como uma árvore;
- execução do documento NCL, entre outras; e
- integração com um repositório de aplicações NCL.

O objetivo principal do NCL Eclipse é apoiar o desenvolvimento textual, por isso não se preocupa em facilitar reaproveitamento de código ou mesmo tratar especificações de projeto mais abstratas (por exemplo, linguagens visuais para edição do código). Apesar de possuir uma funcionalidade de visualizar o layout da aplicação por meio das regiões, essa funcionalidade não se mostra muito útil quando se pretende obter uma previsão mais concreta do resultado da execução da aplicação, ou seja, para se acompanhar o desenvolvimento da aplicação é necessário e executar o código-fonte em todos os momentos. O NCL Eclipse permite que se obtenham aplicações de um repositório de aplicações da PUC-Rio, porém em nenhum momento se preocupa com a estruturação dos requisitos ou projeto da aplicação.

Em contrapartida, por ser desenvolvida em cima da plataforma de *plugins* do Eclipse, o NCL Eclipse tem como um ponto forte a facilidade de se integrar com outras ferramentas já existentes dessa plataforma (por exemplo, o *plugin* LuaEclipe, que permite a edição textual do código-fonte Lua). Pode-se afirmar que a ferramenta atende melhor profissionais da equipe de software (e não da equipe de

mídia). A Figura 4.3 mostra a interface do NCL Eclipse, onde é possível ver a coloração da sintaxe e sugestões automáticas de códigos (A, B e C). Também é exibido como a ferramenta pré-visualiza a disposição das regiões (D).

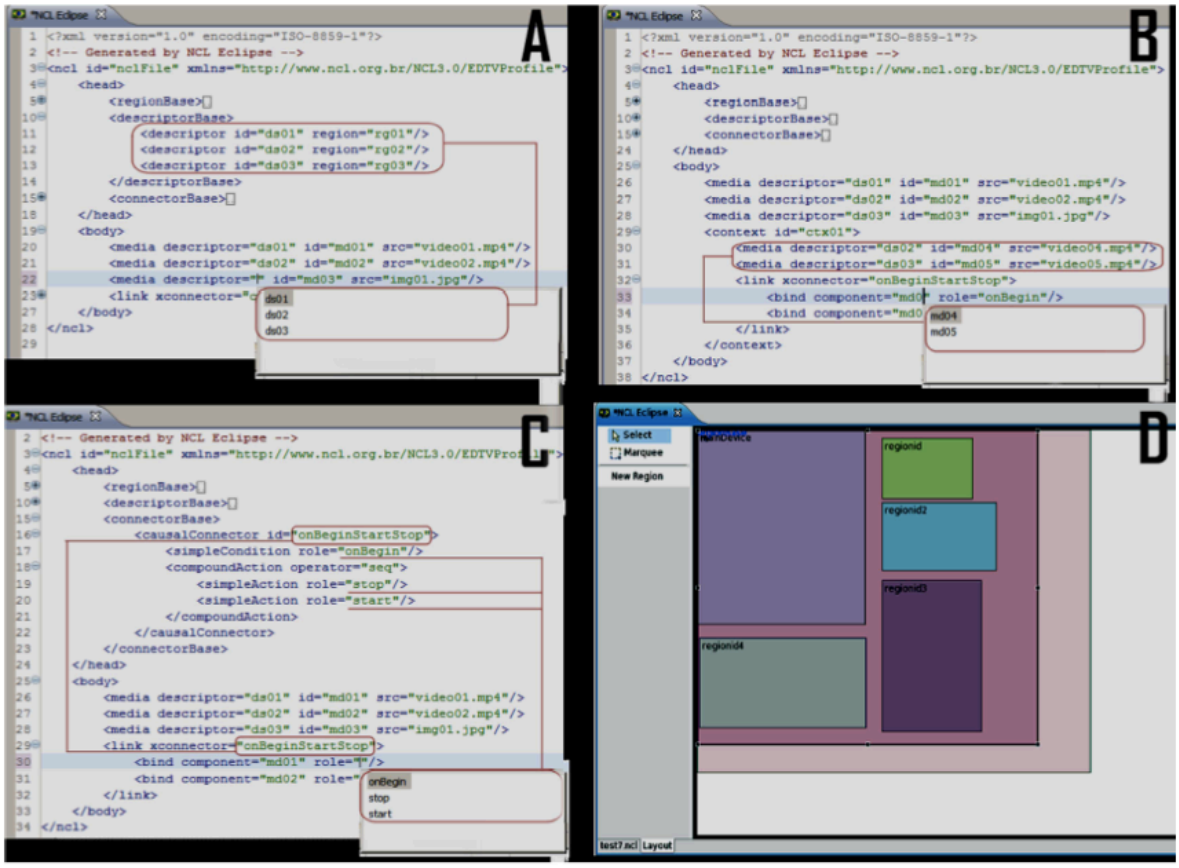


Figura 4.3: Visões disponíveis no NCL Eclipse (AZEVEDO, 2011)

4.3.3 NCL Composer

O NCL Composer (LIMA et al., 2010; LIMA, MORENO e SOARES, 2011) é uma ferramenta de autoria hipermídia para a codificação de programas audiovisuais interativos em NCL. Seu principal objetivo é possibilitar que usuários possam produzir código NCL por meio de abstrações visuais. A hipótese é que esta abordagem requer menos conhecimento especializado de NCL. A filosofia é construir o programa através de 4 (quatro) visões principais (Figura 4.4): estrutural (exibe, hierarquicamente, os objetos multimídia que fazem parte de uma apresentação e sua relação de sincronismo e interatividade), temporal (mostra a relação temporal entre os objetos), layout(mostra a relação espacial que existe entre os objetos) e textual (exibe o código fonte NCL de uma apresentação). No NCL Composer, cada visão mostra uma parte do programa NCL e, caso uma alteração ocorra em uma das visões, as outras visões são atualizadas automaticamente.

O foco da NCL Composer é a geração do código NCL de uma aplicação interativa por "não programadores". É importante ressaltar que essas visões não representam os diferentes níveis de abstração encontrados na MDD, apenas refletem o código-fonte NCL atual. O uso da ferramenta parte do princípio que todos os requisitos funcionais e não funcionais do programa já estão especificados desde o início, não tratando também da estruturação de requisitos ou de atividades de projeto de software.

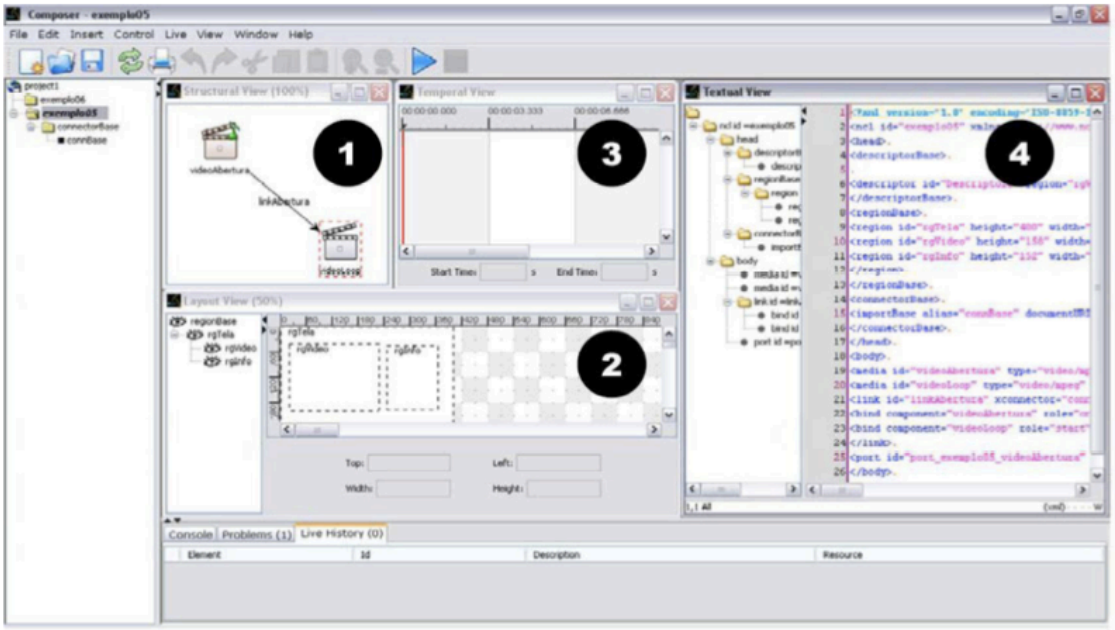


Figura 4.4: Visões do NCL Composer – (1) visão estrutural; (2) visão de layout; (3) visão temporal; (4) visão textual.

Apesar de o NCL Composer oferecer um mecanismo para integrar um código-fonte Lua por meio da visualização de um objeto (nó) de mídia imperativa, essa representação apresenta alguns problemas. A seguir são pontuadas situações nas quais o NCL Composer não deixa intuitivo, o suficiente, a utilização de mídias imperativas por autores de documentos:

- **Falta de exposição do código imperativo.** Uma mídia imperativa, quando criada na visão estrutural, não tem suas estruturas internas expostas (por exemplo, métodos, variáveis e propriedades de objetos). A exposição destas estruturas vem facilitar a identificação delas pelo autor de documentos. No NCL Composer atual o autor de documentos deve conhecer a mídia imperativa para poder encontrar o código (conteúdo) necessário para a aplicação. Desta forma, é necessário que o próprio autor atue na identificação das estruturas alvos dentro da mídia (métodos, variáveis, etc.), implicando em conhecimento da linguagem imperativa e perda de tempo para identificar tais estruturas. A Figura 4.5 apresenta um exemplo de mídia imperativa incluída na visão estrutural do NCL Composer. Após sua criação, a visão não disponibiliza de maneira simples quais as estruturas disponíveis nesta mídia. Para acessar algum método é preciso que o usuário crie os elementos de propriedades que representam os métodos que ele

queira acessar. Neste processo, o autor de documentos precisa interagir com o programador da mídia imperativa, ou quando na ausência deste, olhar internamente a implementação da mídia imperativa. Isto provoca uma dificuldade no uso deste tipo de conteúdo.

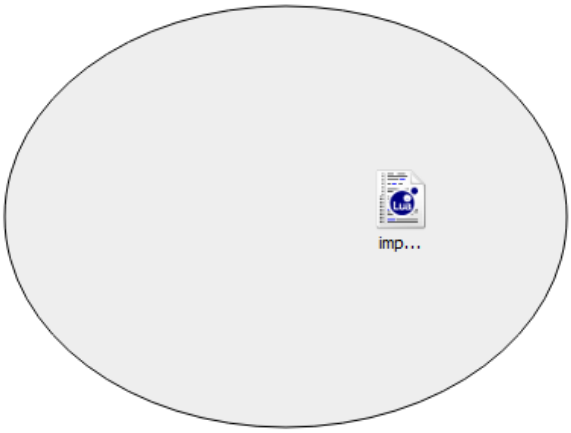


Figura 4.5: Representação de mídia imperativa no NCL Composer

- **Ter conhecimento de como usar as estruturas imperativas.** O uso das estruturas imperativas não é feita de maneira simples, necessitando, por exemplo, que o autor saiba qual a forma as ser empregada para poder usar um determinado método. Para isso, o autor deve ter conhecimento de qual ação deve ser utilizada neste caso. Isto implica a necessidade da interação do usuário quando no uso do método, isto é, o NCL Composer confia que o usuário especifique qual a ação a ser realizada para esta chamada de método. Aqui também implica que o usuário dependa (conheça) cada vez mais dos conceitos NCL. Este cenário é demonstrado na Figura 4.6 quando, no momento da ligação com um método da mídia imperativa (uso de método), a ferramenta pede ao usuário que especifique qual a ação a ser realizada, através da janela de diálogo para definição do link.

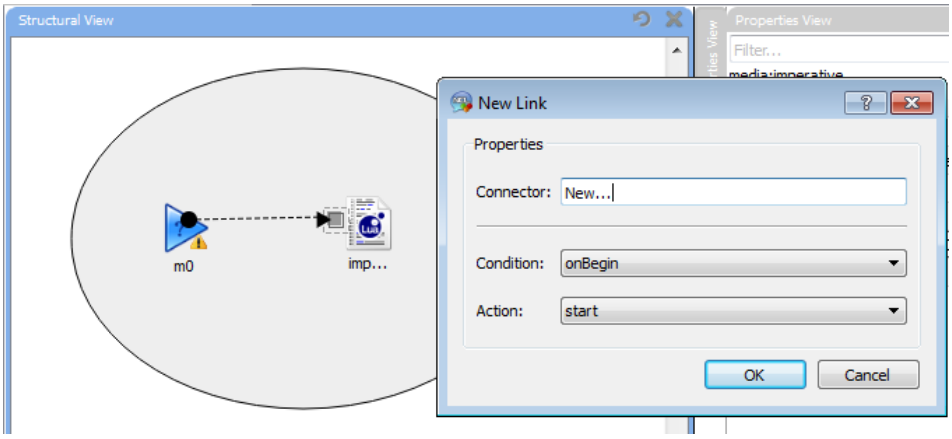


Figura 4.6: Utilização de recursos da mídia imperativa no NCL Composer

- **Ausência na especificação dos valores de cada parâmetro do método.** Não existe nenhum método visual que auxilie o autor a especificar quais os valores serão usados em cada parâme-

tro durante o uso do método. Um método pode possuir mais de um parâmetro e o valor para cada um é importante. Neste cenário, a Figura 4.7 apresenta a abordagem empregada para especificar os valores do parâmetro da chamada do método. Observa-se que não é possível especificar os valores caso o método possua mais de um parâmetro, nem mesmo quais são os parâmetros do método.

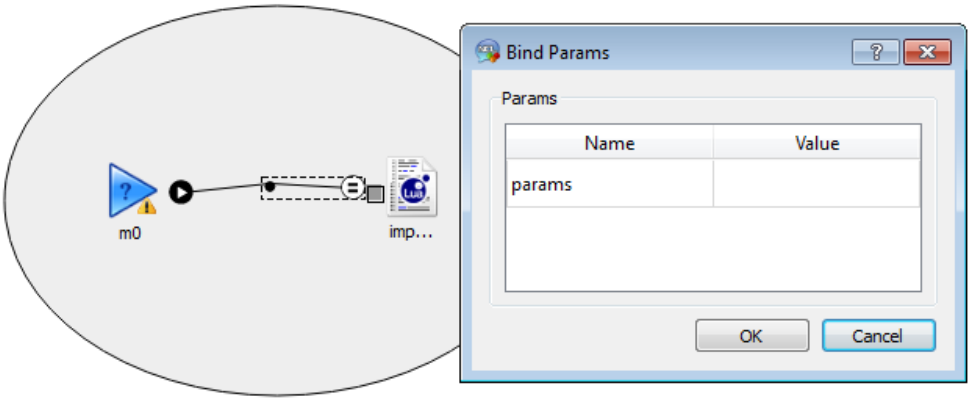


Figura 4.7: Passagem de valores de um nó de mídia NCL para um código-fonte Lua

- **Impossibilidade do uso de valores presentes em outras mídias para os parâmetros.** Ainda no problema anterior, não é possível especificar um valor para um determinado parâmetro que esteja presente em outras mídias. Por exemplo, às vezes é de interesse que o valor de um parâmetro advenha de um conteúdo presente em outra mídia ou variável global do documento.

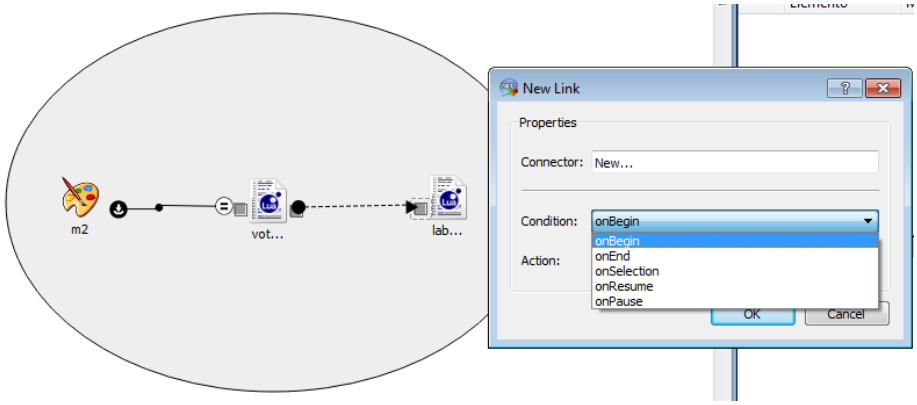


Figura 4.8: Tratamento de retorno de valores de um código-fonte Lua para um nó de mídia NCL

- **Ausência no tratamento do retorno de valores de mídias imperativas.** Não é possível receber um valor de uma mídia imperativa e reusá-lo em outro elemento de mídia. Este cenário é utilizado quando queremos armazenar o resultado de algum processamento e enviá-lo para outra mídia. A Figura 4.8 exemplifica esta situação ao não suportar o direcionamento do valor retornado pela mídia para ser reusado em outra mídia. Aqui também, possui semelhança com o segundo problema, o qual necessita que o autor saiba como repassar o valor para outra mídia, pedindo que o mesmo informe a condição na janela do link.

4.4 Considerações sobre o capítulo

Este capítulo se focou em apresentar o domínio de aplicações para TV Digital, mais especificamente, no contexto da plataforma Ginga, padrão de middleware para o ISDB-Tb. Foram apresentados alguns exemplos e categorias de aplicações. Além disso, foram explicados conceitos relacionados a NCL, que consiste numa linguagem de programação que herdou o paradigma hipermídia,utilizado no desenvolvimento de aplicações Web (XHTML), para aplicações de TVD. NCL também estende o modelo conceitual declarativo do XHTML para suportar um conjunto de comandos mais avançados. Entretanto, NCL faz uso de uma linguagem imperativa (Lua) para suportar a implementação de lógica de aplicação mais complexa.

Adicionalmente, foram detalhados dois ambientes de desenvolvimento que utilizam NCL e/ou Lua: NCL Eclipse e NCL Composer. Tais ambientes ilustram que o desenvolvimento de aplicações para TV Digital vem evoluindo do paradigma de "*implementação e teste*"[5] e autoria textual (NCL Eclipse), para o emprego de ferramentas visuais e orientadas a modelos (NCL Composer). Porém, como já apontado no Capítulo 2 (Seção 2.3) e pelas limitações apresentadas pelo NCL Composer (Seção 4.3.3), ainda se faz necessário melhorar dois aspectos: (1) definição de uma metodologia que integre diferentes visões de projeto; (2) desenvolvimento de ambientes de desenvolvimento que apóiem a integração dessas visões, principalmente, para o desenvolvedor que trabalha com uma linguagem declarativa.

Assim, o próximo capítulo propõe uma abordagem MDD que procura sistematizar as atividades e ao mesmo tempo estender o NCL Composer para melhor atender esses dois desafios e, consequentemente, melhorar o desenvolvimento de aplicações no domínio de TVD.

[5]Termo informal utilizado por Engels e Sauer (2002) para ilustrar o desenvolvimento de software baseado em ciclos curtos de escrita de código-fonte e execução num emulador, e que remete a práticas de engenharia de software da década de 70.

5 ABORDAGEM PROPOSTA

A abordagem aqui proposta foi definida após pesquisa da literatura e conhecimento prévio do autor deste trabalho na área de TV Digital. Foram estudados artigos científicos na área de desenvolvimento orientado a modelos e desenvolvimento de aplicações multimídia.

As etapas da abordagem foram definidas por meio de refinamentos sucessivos e procurando incorporar pontos fortes de outros trabalhos relacionados (analisados no Capítulo 2), por meio de práticas e tarefas já bem estabelecidas, como aquelas definidas pela comunidade de MDD, LPS, e/ou Desenvolvimento Generativo de Software (apresentados na Seção 3.3). Além de incorporar soluções pré-existentes diretamente aplicáveis (por exemplo, as ferramentas do EMF), também foram incorporadas melhorias em abordagens da literatura do domínio específico de aplicações de TV Digital, especialmente, no subdomínio de aplicações Ginga-NCL e na ferramenta NCL Composer (explicado na Seção 4.3.3).

Durante a definição das etapas foram desenvolvidas provas de conceito que, posteriormente, também foram utilizadas nas avaliações empíricas. Outro aspecto importante é que a divisão sugerida nas etapas da abordagem permitiu avaliar o trabalho de modo incremental, uma vez que a implementação dos mecanismos da última etapa (*Refinamento da Aplicação*) foram finalizados um tempo depois da penúltima (*Prototipação Rápida*). Encontros com outros pesquisadores especialistas e participação de outros alunos de pós-graduação com os quais o autor é colaborador também ajudaram a refinar e implementar a abordagem.

5.1 Visão geral

Para um entendimento inicial da abordagem proposta neste trabalho, foi definida uma divisão em 3 etapas (Figura 5.1): *Requisitos*, *Prototipação Rápida e Refinamento da Aplicação*. Mesmo abrangendo as principais fases de um processo de desenvolvimento tradicional da engenharia de software (Requisitos, Análise, Projeto, Implementação e Testes), o intuito não é definir um modelo de processo formal, mas ilustrar de forma geral os principais elementos da abordagem. Dessa forma, não são apresentados detalhes da definição e desenvolvimento dos artefatos e do ambiente de desenvolvimento, pois esses aspectos serão discutidos nas demais seções deste capítulo. Apesar da figura apresentar várias *Equipes de Mídia* e *Equipes de Software*, elas representam apenas uma equipe para cada disciplina. As setas tracejadas indicam responsabilidades de cada equipe sobre as atividades (retângulos com linhas contínuas) quando têm como origem uma equipe e representam artefatos quando têm como origem uma atividade. Outro ponto importante é que as atividades das etapas são apresentadas sequencialmente para auxiliar seu entendimento, no entanto, na prática essas atividades podem ser realizadas de modo iterativo e incremental para se adequar ao contexto de uma organização.

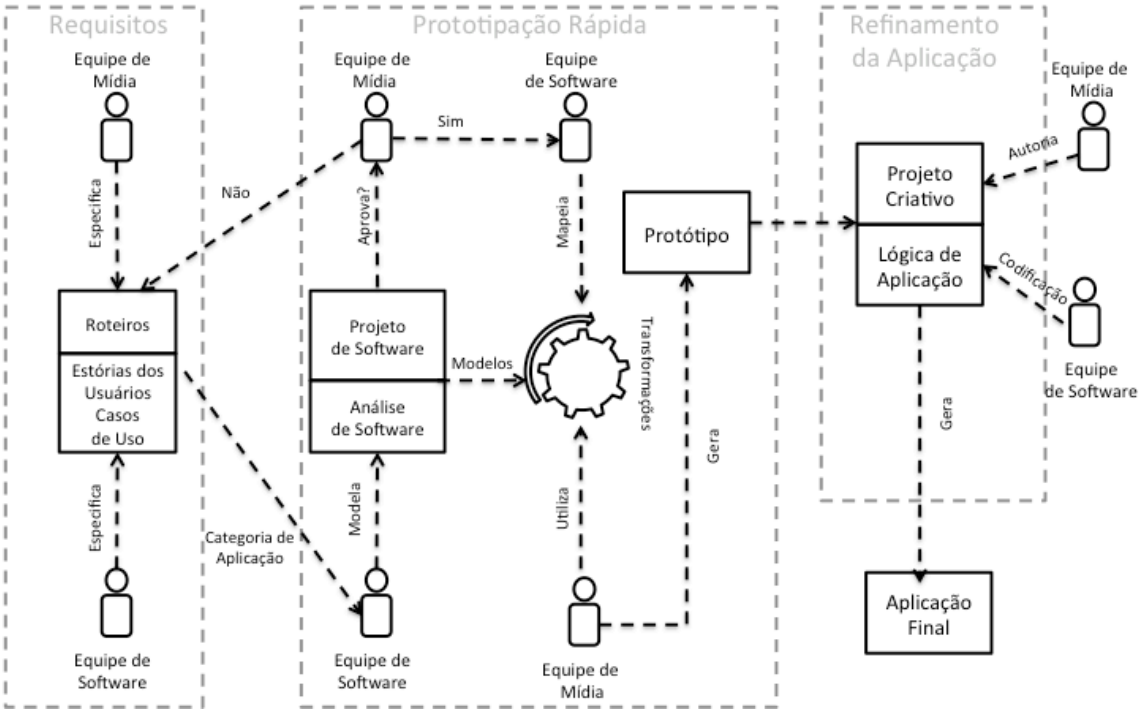


Figura 5.1: Visão geral da Abordagem

A primeira etapa (*Requisitos*) da abordagem tem como objetivo especificar os requisitos das aplicações, por intermédio de artefatos da área de produção de TV (estudo de viabilidade, desenvolvimento da ideia, projeto inicial do programa, teste de conceito, roteiros e etc.) e da área de desenvolvimento de software (requisitos funcionais, requisitos não-funcionais, requisitos do usuário, requisitos de sistema, casos de uso, estórias do usuário, esboço de projeto da interface gráfica com o usuário etc.). O primeiro conjunto de artefatos pode ser especificado por profissionais da *Equipe de Mídia* (diretor de produção, projetistas gráficos, produtores de conteúdo, analista de *marketing* etc.), e o segundo por profissionais de uma Equipe de Software (analista de negócio, analista de requisitos, analista de sistemas etc.). A partir daí, é possível definir um escopo que representa um conjunto de elementos de uma aplicação, em outras palavras, uma *Categoria de Aplicação*. Apesar de recomendar fazer uso da taxonomia apresentada na Seção 4.2 para auxiliar na elaboração dos possíveis cenários de uso da categoria definida, não faz parte do escopo deste trabalho detalhar essa fase, pois diversos trabalhos anteriores já realizaram estudos nesse contexto, inclusive abordando a integração de uma equipe multidisciplinar (GAWLINSKI, 2003; CHORIANOPOULOS, 2004; VEIGA, 2005; LULA, 2011). Tal etapa é ilustrada apenas para contextualizar as duas fases seguintes, que é o foco de investigação desta tese.

A segunda etapa (*Prototipação Rápida*) tem como objetivo receber os artefatos gerados na etapa de *Requisitos* e, a partir deles, realizar uma *Análise de Software* para elaborar um *Projeto de Software* com modelos e transformações que possibilitem a geração automática do código-fonte de uma estrutura prévia de uma *Categoria de Aplicação* de TV Digital, em outras palavras, um *Protótipo*. Tal termo é utilizado neste estudo baseado em trabalhos relacionados (BERNARDI, DI LUCCA e DIS-

TANTE, 2011; GAUFFRE, DUBOIS e BASTIDE, 2008; YU, 2008) e amplia o conceito inicial da engenharia de software[7]. Desse modo, o protótipo é desenvolvido a partir de modelos de linguagens específicas de domínio que são mapeados para transformações sucessivas até a geração de código-fonte que é utilizado na etapa seguinte da abordagem (*Refinamento da aplicação*). Como elucida a , a primeira atividade (Análise de Software) consiste na modelagem pela *Equipe de Software* de características comuns e variáveis da *Categoria da Aplicação*. Após tal atividade, é realizada, também pela *Equipe de Software*, a modelagem de várias visões (*Projeto de Software*) que são representadas por modelos diferentes. Tais modelos têm o objetivo de especificar uma estrutura geral da solução, a interface e os relacionamentos entre os diferentes aspectos que a compõe. Dessa maneira, é possível realizar uma integração inicial de vários tipos de elementos da *Categoria da Aplicação*, tal como, elementos da interface gráfica, objetos de mídia e lógica de aplicação. Visto que uma das peculiaridades do desenvolvimento de aplicações para TV Digital é exigir uma resposta rápida (GAWLINSKI, 2003), é possível, a partir do desenvolvimento iterativo e incremental dos artefatos dessas duas primeiras atividades (*Análise de Software e Projeto de Software*), um retorno rápido da *Equipe de Mídia* (produtores de conteúdo, projetista de interface gráfica etc.), que pode inclusive solicitar um refinamento de artefatos da etapa de *Requisitos* para melhor definição dos modelos do protótipo. Como atividade final, após a aprovação pela *Equipe de Mídia*, a *Equipe de Software* mapeia os *modelos* que representam as características e projeto arquitetural com o objetivo de permitir a configuração das transformações. Tais mecanismos são utilizados pela *Equipe de Mídia* para gerar o código-fonte de versões diferentes de uma estrutura principal de aplicação (*Protótipo*).

A terceira e última etapa consiste em realizar o *Refinamento da Aplicação* por meio de 2 (duas) atividades que são integradas num mesmo ambiente de desenvolvimento para gerar o código-fonte da *Aplicação Final*. A primeira atividade pode ser realizada pela *Equipe de Mídia* e utiliza funcionalidades de uma ferramenta de autoria para tratar do *Projeto Criativo* da aplicação: (1) configuração da disposição espacial (*layout*) e sincronismo temporal os objetos de mídia; (2) preenchimento de valores de instância que não foram definidos no protótipo; e (3) ajustes em comportamentos específicos da aplicação (interação do usuário). A segunda atividade consiste na codificação e/ou integração de módulos, bibliotecas e/ou componentes de software por parte da *Equipe de Software* utilizando editores textuais de linguagem de programação tradicionais. Esses artefatos implementam a *Lógica de Aplicação* complexa e específica da plataforma de execução da aplicação.

[7]Na literatura (KORDON, 2002), a prototipagem é definida como uma técnica utilizada na fase de elicitação de requisitos com o objetivo de verificar e validar os requisitos do usuário por meio de um artefato que representa uma visão simples da solução que se pretende desenvolver. Normalmente, tal artefato é descartado após sua aprovação.

A intenção de separar o desenvolvimento da aplicação nas duas últimas etapas (*Prototipação Rápida* e *Refinamento da Aplicação*) é unir as vantagens dos modelos de mais alto nível, com modelos específicos presentes em ferramentas de autoria. O primeiro grupo facilita a especificação da estrutura geral e as interconexões entre os elementos de uma aplicação. O segundo possibilita o projeto criativo da aplicação tratando de configurações de aspectos complexos de lógica de aplicação e específicos de plataforma. O objetivo é realizar a análise, projeto e implementação de uma aplicação de TV Digital utilizando refinamentos sucessivos em modelos e geração automática de código-fonte para facilitar a comunicação e integração entre a equipe de mídia e a equipe de software. Na seção seguinte são detalhados os modelos utilizados em cada etapa.

5.2 Visão MDA

Nesta seção é utilizada a nomenclatura e camadas arquiteturais do MDA (OMG, 2003) para facilitar o entendimento do uso de modelos em cada etapa da abordagem. Mais uma vez é importante ressaltar que a elaboração dos modelos é organizada em sequência, porém é possível retornar para um nível anterior durante qualquer momento da execução do processo para refinar um modelo ou transformação definida anteriormente.

Conforme já dito, a abordagem proposta parte de requisitos de software especificados por profissionais de equipes de mídia e software, tais como: *Roteiros*, *Estórias* dos *Usuários* ou *Casos de Uso*. Tais artefatos auxiliam na identificação dos principais conceitos e elementos de modo a definir o escopo de uma categoria de aplicação, que pode ver vista como uma *Família de Aplicação.* Dessa família é possível gerar diferentes variações de uma aplicação e, por isso, é elaborado um *Modelo de Features* para representar as características de tais aplicações no nível de análise de domínio (KANG et al., 1990). O conjunto dessas características identifica as similaridades e variabilidades, isto é, as funcionalidades ou propriedades que são comuns a todas as aplicações da categoria e aquelas que variam. O objetivo é estruturar os requisitos para facilitar as atividades seguintes relacionadas ao projeto de software da categoria da aplicação. No MDA os artefatos produzidos em ambas as atividades desta primeira etapa são conhecidos como *CIM* (do inglês, *Computational Independent Model*), uma vez que são independentes de representação computacional.

Na segunda etapa é utilizada a linguagem de modelagem visual MML para apoiar atividades de projeto (design) no desenvolvimento, de acordo com características comuns e variáveis presentes no modelo de features definido anteriormente. A adoção de linguagem específica de domínio (MML) tem 2 (dois) objetivos: (i) utilizar representações que contribuem para uma melhor estruturação da *Família da Aplicação*; e ao mesmo tempo, (i) promover a colaboração de profissionais da *Equipe de Mídia* e da *Equipe de Software*

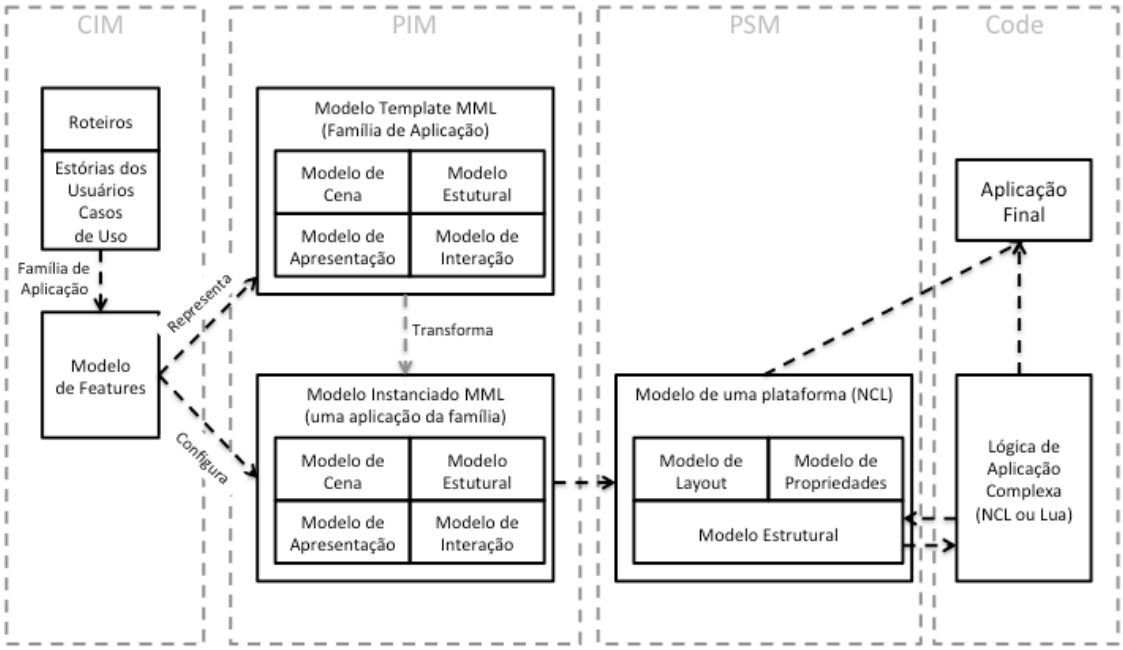


Figura 5.2: Visão da abordagem proposta de acordo do MDA

Considerando os dois objetivos citados, acredita-se que é possível gerar automaticamente um esqueleto de uma aplicação que facilitará a implementação final numa ferramenta de autoria para uma plataforma específica. A Figura 5.2 ilustra os quatro tipos de visões de projeto da MML: (1) modelo de cena; (2) modelo de apresentação; (3) modelo de estrutural; e (4) modelo de interação. O papel e detalhes desses modelos podem ser consultados na Seção 2.2. Os dois primeiros são de responsabilidade da equipe de mídia, e os dois últimos da equipe de software. O conjunto desses modelos pode ser visto como uma arquitetura de referência para uma determinada família de aplicações, sendo capaz de derivar diversos modelos diferentes decorrentes das configurações realizadas no *Modelo de Features*. Isso significa que nessa etapa é gerada uma transformação de modelo para modelo, antes de gerar os modelos para a próxima etapa. O conjunto de modelos que representa a família é chamado de *Modelo Template,* e o conjunto de modelos que representa uma versão de uma aplicação derivada é chamado de *Modelo Instanciado.* Mais detalhes da configuração e transformação entre o *Modelo de Features*, *Modelo Template* e *Modelo Instanciado* serão discutidos nas próximas seções deste capítulo. No MDA, todos os modelos gerados nesta etapa são conhecidos como PIM (do inglês, *Plataform Independent Model*), ou seja, são independentes de plataforma possuindo uma notação não ambígua e padronizada a fim de permitir a sua transformação em modelos e/ou código-fonte para plataformas específicas e diferentes.

A terceira etapa consiste em transformar um *Modelo Instanciado* de uma aplicação da Família de Aplicação modelada na fase anterior para um ou mais modelos de uma plataforma específica, no MDA conhecidos como PSM (do inglês, *Platform Specific Model*). Estes representam os modelos que estão associados a algum paradigma e linguagem de programação e são especializações dos modelos anteriores. Apesar de ser impossível mapear os modelos PIM para todas as tecnologias representadas

pelos PSM, a estruturação do *Modelo Template* num conjunto específico e limitado facilita essa transformação. Na ilustração (Figura 5.2), é representado o mapeamento do *Modelo Instanciado MML* para modelos compatíveis (*Layout, Propriedades e Estrutural*) com a ferramenta de autoria *Composer* NCL (ver Seção 4.3.3), que é o foco deste estudo. Entretanto, habilitando uma opção no momento da geração do código e uma vez implementado o mapeamento e transformações para outras plataformas, é possível aproveitar os modelos CIM e PIM para outras tecnologias de implementação.

A última etapa é responsável por transformar os modelos PSM, normalmente com ajustes detalhados e informações de instância (valores de propriedades), para o código-fonte da *Aplicação Final* específica para uma determinada plataforma alvo. Também é possível nessa etapa desenvolver ou reutilizar elementos de implementação que são visualizados e/ou configurados diretamente pelo código-fonte (exemplo, *Lógica de Aplicação Complexa*). Nesse caso, o ideal é que os modelos PSM facilitem a integração com as demais visões da aplicação. Este trabalho procura estender a visão do modelo estrutural do Composer NCL para facilitar a integração de elementos (componentes) de código-fonte escritos na linguagem Lua. Mais detalhes serão apresentados adiante.

5.3 Visão Generativa

Um detalhamento da abordagem é apresentado na Figura 5.3:

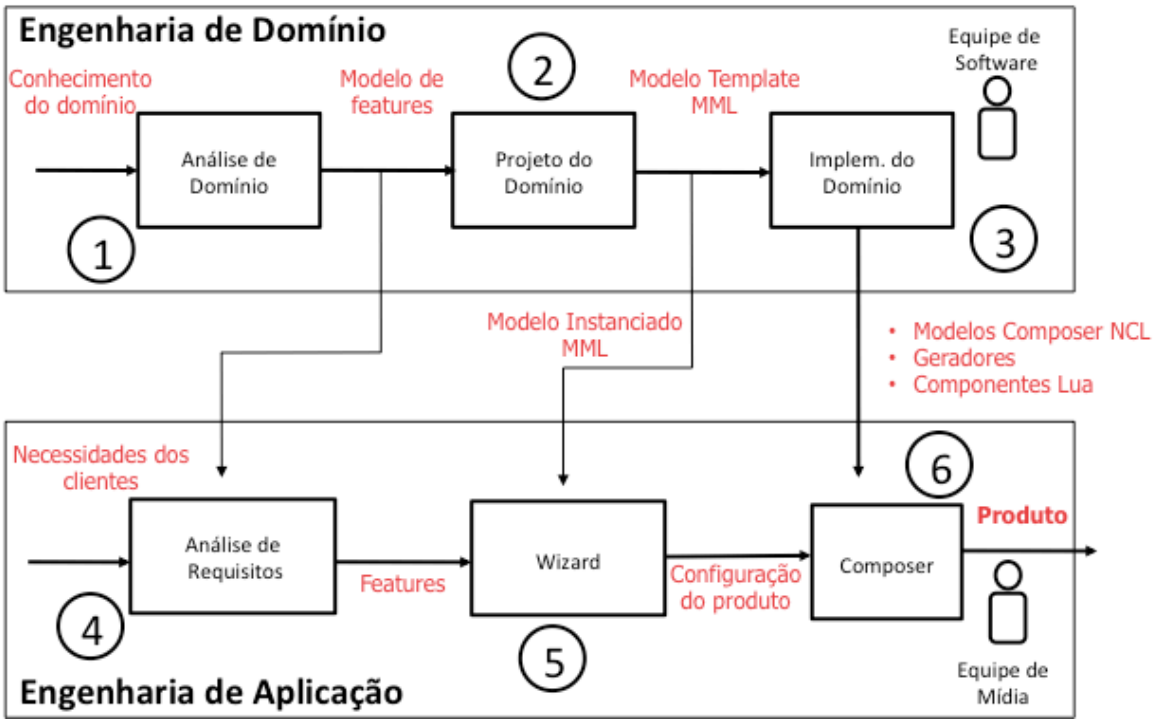


Figura 5.3: Visão generativa da abordagem

Seguindo a perspectiva apresentada por Czarnecki e Eisenecker (2000), também conhecida como Engenharia de Linhas de Produto e utilizada por metodologias de Desenvolvimento Generativo de Software. Na Figura 5.3, os itens descritos por caixas retangulares e texto em preto representam

atividades, e os itens em vermelho representam artefatos que são gerados durante o processo de desenvolvimento de software

Inicialmente são executadas as atividades da *engenharia de domínio* (área superior da Figura 5.3) para uma determinada família (ou categoria) de aplicação. Na *Atividade 1* (*análise de domínio*), um ator da *equipe de software* – especialista em domínio ou engenharia de requisitos – define o M*odelo de Features*. Ele representa o universo de possibilidades que um *Modelo Template* deve tratar. Com base no *Modelo de Features* produzido no passo anterior, um outro ator da *equipe de software* – engenheiro ou arquiteto de software – modela na *Atividade 2* (*projeto de domínio*), o *Modelo Template* utilizando a linguagem MML. O objetivo é contemplar todos os elementos pertencentes a todas as soluções válidas que são originadas a partir do *Modelo de Features* e que geram um *Modelo Instanciado* para cada conjunto de configurações definidas. Para expressar as variabilidades da aplicação, ainda na *Atividade 2* são anotados os elementos do *Modelo Template* utilizando dois tipos de anotações: (i) condição de presença (CP) e (ii) multiplicidade de elementos (M). Estas anotações são definidas em termos das características e podem ser avaliadas de acordo com uma determinada configuração que é escolhida no momento da derivação do produto (aplicação). Por exemplo, uma CP permite definir se um elemento do *Modelo Template* estará presente ou ausente no *Modelo Instanciado* e uma M pode indicar quantos elementos de um determinado tipo do *Modelo Template* estarão presentes no *Modelo Instanciado*. Na *Atividade 3* (*implementação do domínio*), um ator da *equipe de software* – engenheiro ou desenvolvedor de software – podem implementar: (1) estratégias de derivação automática ou manual (transformadores) que mapeiam os modelos MML (*Modelo Template* e *Modelos Instanciados*) em diagramas específicos de domínio (por exemplo, *Wizards*) e/ou uma plataforma (por exemplo, *Modelos Composer NCL*); e (2) componentes e/ou bibliotecas de software (por exemplo, módulos da linguagem Lua).

Na parte inferior da Figura 5.3 são ilustradas as atividades relacionadas à Engenharia da Aplicação. Na abordagem definida nesse trabalho, tais atividades podem ser executadas por profissionais da equipe de mídia (projetista de interface gráfica, produtor de conteúdo, programador de linguagens declarativas com pouca experiência em linguagens imperativas e etc.) e nesse caso esses atores são conhecidos como engenheiros de aplicação. A *Atividade 4* (análise de requisitos) corresponde em analisar uma especificação de requisitos e identificar que categoria ou família de aplicação atende tais necessidades, em outras palavras, se existe algum *Modelo de Features* definido pela engenharia de domínio que pode ser utilizado. No momento seguinte, na *Atividade 5* (*configuração do produto*), depois da escolha anterior (*features),* é realizada uma configuração manual (por exemplo, representado por opções de um *Wizard)*. Baseado nessas escolhas é gerado um *código base da aplicação (Esqueleto da Aplicação)*. Este processo de instanciação envolve primeiramente um mapeamento de modelo (*Modelo Template*) para modelo (*Modelo Instanciado*), depois uma transformação de modelo (*Modelo Instanciado*) para modelo (*Modelo Específico NCL*) e, em seguida, a geração do *Esqueleto da Aplicação*. Na

Atividade 6 (*implementação e testes*), o engenheiro de aplicação utiliza uma ferramenta de autoria (NCL Composer) para refinar o projeto criativo e implementar a lógica complexa da aplicação. Para gerar a aplicação final (*Produto*), é possível utilizar modelos específicos da plataforma e componentes de software desenvolvidos pela equipe de software da engenharia de domínio. Nas duas últimas seções deste capítulo são detalhadas o processo de modelagem, transformações e geração de código-fonte da *Atividade 5* e a extensão do *NCL Composer* para permitir a integração com a engenharia de domínio na *Atividade 6*.

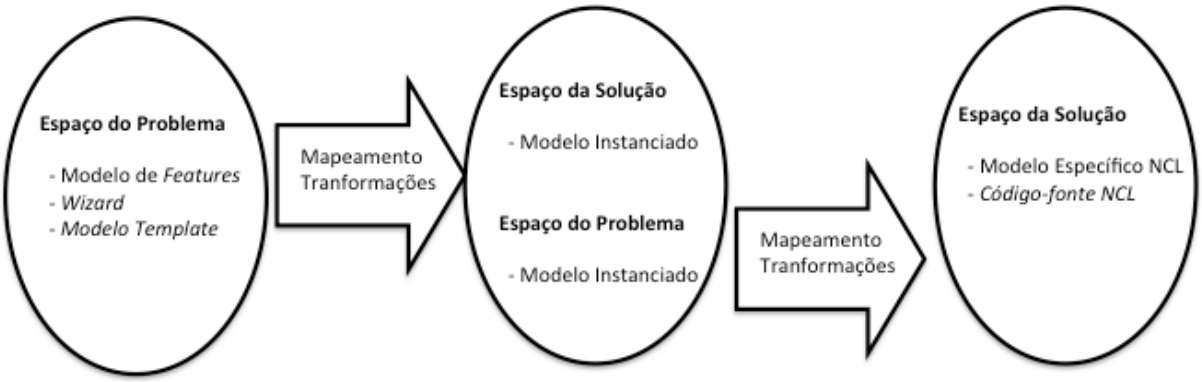


Figura 5.4: Encadeamento de mapeamentos da Abordagem (CZARNECKI, 2004)

De acordo com Czarnecki (2004), num primeiro momento o Modelo de *Features*, *Wizard e* Modelo *Template* representam o espaço do problema, e o Modelo Instanciado representa o espaço da solução. Num segundo momento os *Modelos Instanciados* representam o espaço do problema, e o Modelo Específico (NCL) e código-fonte da aplicação o espaço da solução (ver Figura 5.4). Tal situação é prevista e configura um caso de encadeamento de mapeamentos (do inglês, "*Chaining of mappings*"). Ainda segundo Czarnecki (2004), na *Visão de Configuração*, o Modelo de *Features*, a *Wizard* e o *Modelo Template* representam a configuração de conhecimento e na *Visão de Transformação* existem dois mapeamentos: (1) *Modelo Template* para *Modelo Instanciado* e (2) *Modelo Instanciado* para *Modelo Específico (NCL)* e código-fonte da aplicação.

5.4 Detalhamento da Prototipação Rápida

Nesta seção são detalhadas as tecnologias empregadas e os passos necessários para permitir a geração sistemática de protótipos ou esqueletos de aplicação para TV Digital utilizando a abordagem proposta.

5.4.1 Tecnologias empregadas

A Figura 5.5 apresenta os níveis de abstrações para definir o *Modelo Template* e, entre parênteses, suas respectivas tecnologias. No nível mais inferior é empregada uma abordagem visual com o *plugin* do Eclipse *UML2 Tools.* Esta ferramenta apresenta um conjunto de editores visuais para facili-

tar a geração dos modelos UML. Estes modelos são compatíveis com o *Metamodelo UML* definido no UML2, que é baseado no meta-metamodelo ECORE do EMF.

Para apoiar a definição do *Modelo Template* também são empregados o *Perfil MML* (estereótipos específicos do domínio de aplicações multimídia) e *Perfil de Variabilidades (*estereótipos para representar variabilidades em diagramas UML). Ambos são definidos também com base no *Metamodelo UML do plugin* do Eclipse *UML2*.

Já o *Modelo de Features* empregado na abordagem se apóia no FMP (*Feature Modeling Plugin*). O FMP é um *plugin* para o Eclipse que utiliza como base o framework EMF para tratar a criação e configuração de características em termos de modelos. Nele a criação das características é especificada usando a modelagem baseada na cardinalidade, a qual estende a modelagem de características original da metodologia FODA (*Feature Oriented Domain Analysis*) adicionando funcionalidades como: característica e cardinalidades de grupo, atributos de característica, entre outras (CZARNECKI e HELSEN, 2006). O uso deste *plugin* foi essencial para poupar tempo no desenvolvimento de uma solução que representasse as características presentes no *Modelo Template*, bem como a criação da configuração.

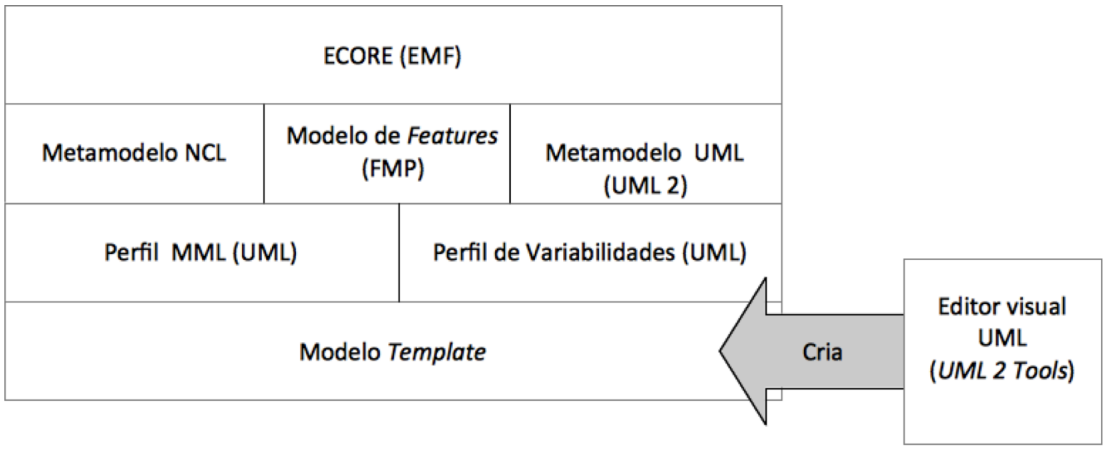


Figura 5.5: Tecnologias empregadas para criação dos modelos

5.4.2 Definição do Modelo de Features

A criação de um novo *Modelo de Features* no FMP é simples e utiliza o mecanismo de criação de arquivos do Eclipse. A edição das características é feita utilizando o editor gráfico baseado em árvore do próprio FMP. A Figura 5.6 apresenta a interface gráfica do FMP para criação do *Modelo de Features*. São ilustrados os principais tipos de características a serem utilizadas: (1) opcional (`Optional Feature`); (2) alternativa (`Alternative Feature`) sendo as suas características filhas `Fea-`

ture1 e *Feature2*, as duas possibilidades possíveis de escolha e; (3) uma característica *Feature3* com atributo de um determinado tipo (*STRING*) para representar valores de texto.

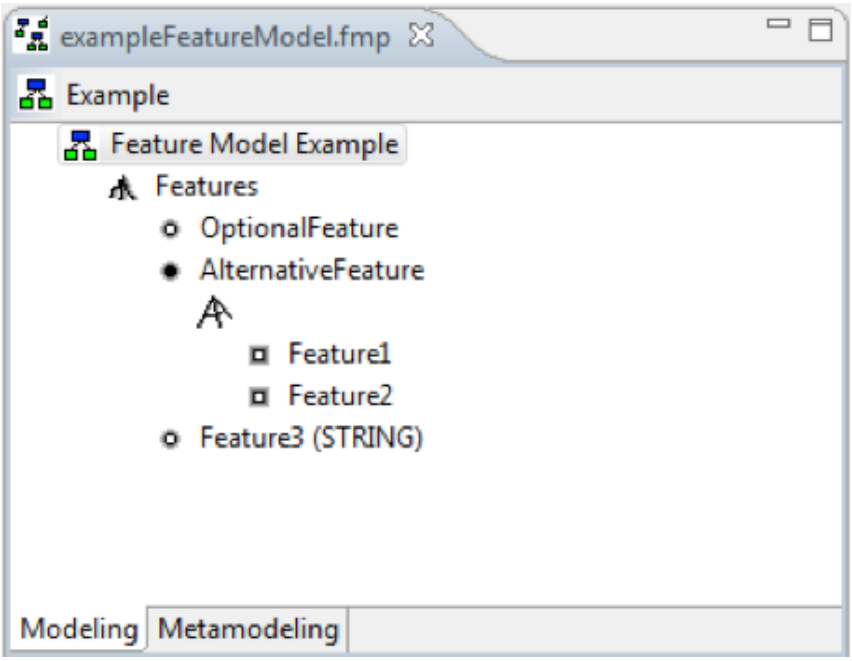
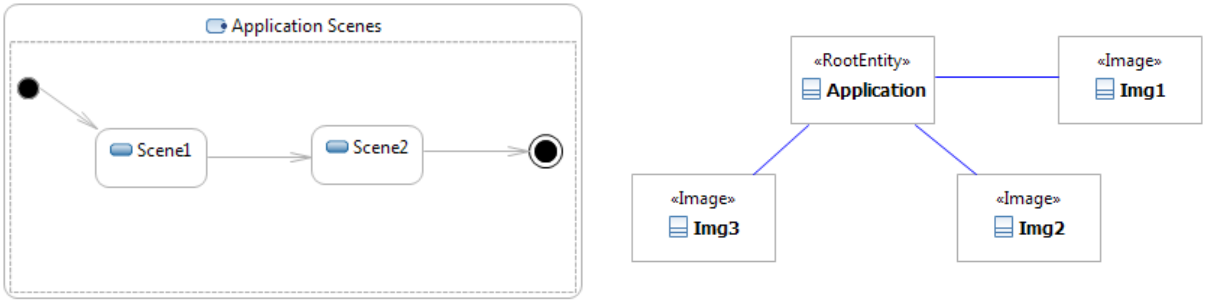


Figura 5.6: Exemplo de Modelo de *Features* no FMP

5.4.3 Definição do Modelo Template

A criação do *Modelo Template* começa pelo modelo estrutural e pelo modelo de cenas da MML. Em seguida, um modelo de apresentação e um modelo de interação são criados para cada cena. Após a criação dos elementos de cada modelo, o *Modelo Template* é definido aplicando o perfil de variabilidade aos elementos que sofrem variação no modelo. A seguir, são descritos alguns exemplos dos modelos MML.

A Figura 5.7 apresenta os diagramas da UML utilizados para representar visualmente um modelo de cena (Figura 5.7a) e um modelo estrutural (Figura 5.7b), respectivamente. Tais modelos procuram representar uma aplicação de TV Digital com duas "cenas". O modelo estrutural define uma classe para representar a aplicação em questão (`Application`) e três classes (`Img1`,`Img2` e`Img3`) para representar a existência de três objetos de mídia. As classes que representam objetos de mídia de imagem, são anotadas com o estereótipo `<<Image>>` e a classe que representa a aplicação como o estereótipo `<<RootEntity>>`,como definido na linguagem MML.



(a) (b)

Figura 5.7: Exemplos de modelo de cenas e modelo estrutural

Um aspecto importante para o modelo de cenas é que cada cena presente neste modelo deve conter uma referência para o seu respectivo modelo de apresentação e modelo de interação. Esta referencia é realizada por meio da inclusão de um elemento do tipo `packageimport` no elemento da cena. A Figura 5.8 apresenta uma visão em árvore dos elementos do modelo de cenas. No elemento do estado `Scene1` (Ponto 1) dois elementos `packageimport` referentes aos pacotes `scene1PresentationMode` e `scene1InteractionModel` são inseridos como filhos do elemento da cena.

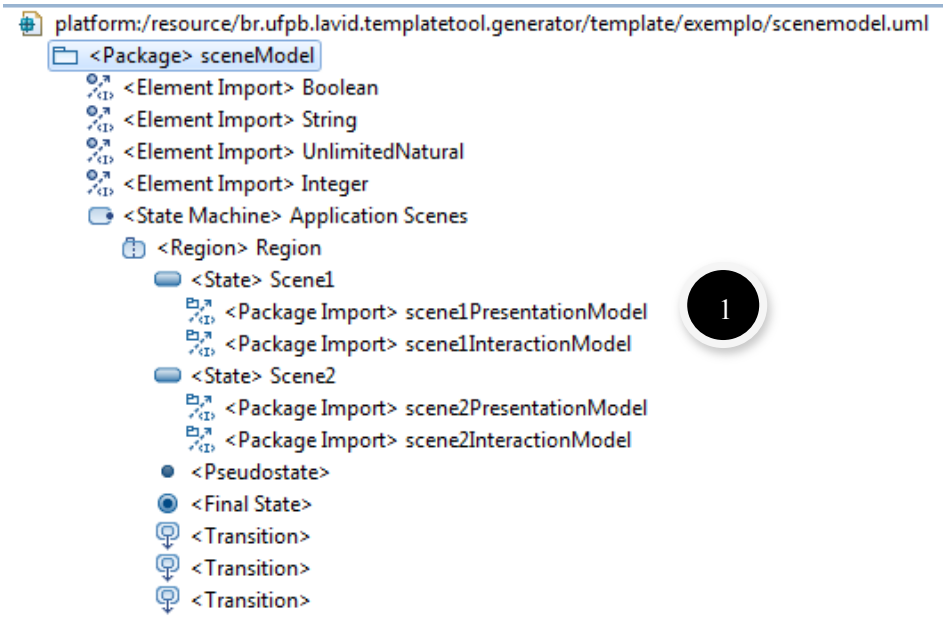


Figura 5.8: Mapeamento entre modelos de cenas e modelos de apresentação e interação

Como duas cenas estão presentes na aplicação, dois modelos de apresentação são criados, sendo um para cada cena. A Figura 5.9: apresenta o diagrama referente ao modelo de apresentação da segunda cena. Como nessa cena apenas duas imagens são exibidas, o modelo apresenta apenas duas classes (`Img2` e `Img3`) para representar as imagens a serem exibidas na interface, ambas com anotações do tipo `OutputComponent`, como indicado na linguagem MML. Cada imagem se relaciona com o objeto de domínio (do modelo estrutural) correspondente, que pode conter atributos da mesma.

A Figura 5.10 apresenta uma visão em forma de árvore do modelo de apresentação da segunda cena. O elemento do tipo `Package` (Ponto 1) é utilizado para representar o modelo de apresentação como um todo, já o pacote `PresentationUnit` (Ponto 2) é utilizado como um pacote para conter os elementos visuais presentes na cena, isto é, todos os `OutputComponent`.

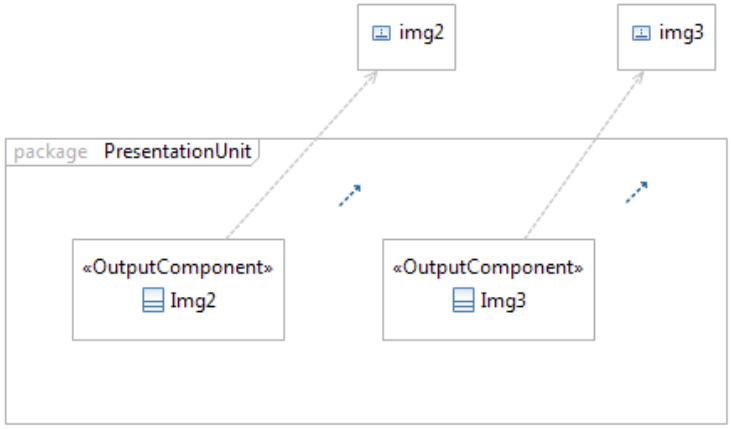


Figura 5.9: Modelo de apresentação da segunda cena da aplicação-exemplo

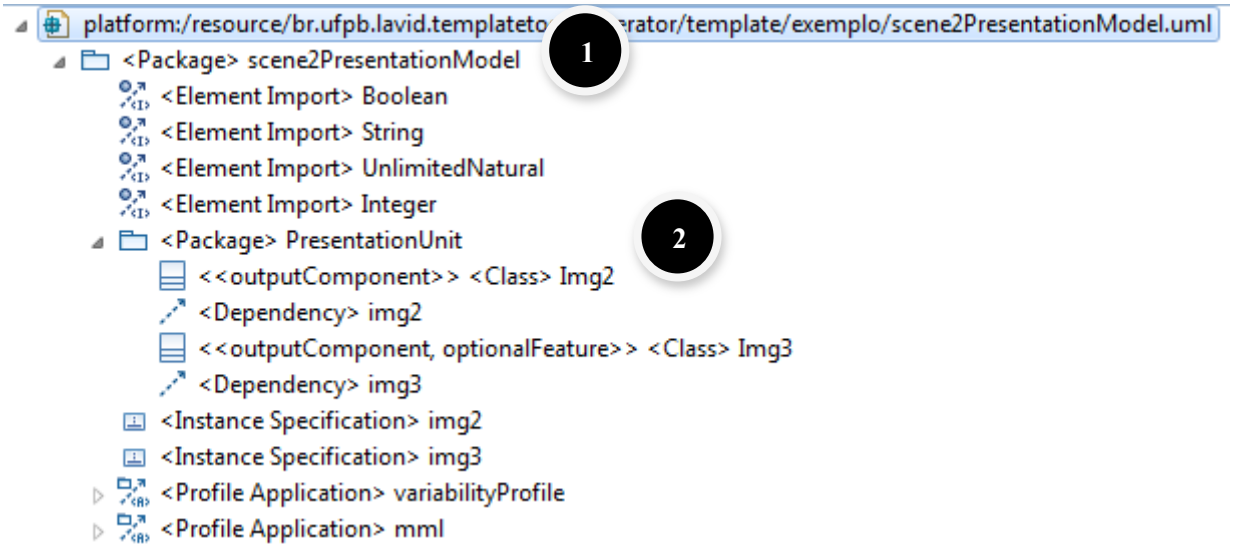


Figura 5.10: Visualização em forma de árvore do modelo de apresentação da segunda cena

O último passo do desenvolvimento do *ModeloTemplate* é a modelagem da interação. Como já dito, este modelo é definido a partir do diagrama de atividades da UML. A Figura 5.11 apresenta o modelo de interação da segunda cena da aplicação-exemplo. Ele descreve o comportamento com relação à apresentação dos objetos de mídia (inicio das imagens). O fluxo de execução inicia com o parâmetro de entrada `showScene2` (`ActivityParameterNode` da UML) e em seguida inicia as duas imagens da cena em paralelo. Este parâmetro de entrada possui dois fluxos com destino para o pino de entrada `start` (`InputPin` da UML) do elemento de ação (`CallBehaviorAction` da UML) `Img1` e

`Img2`, representando no diagrama abaixo pela seta até o pino de entrada `start`. Por último, temos a representação do fim da atividade no círculo da figura (`ActivityFinalNode` da UML).

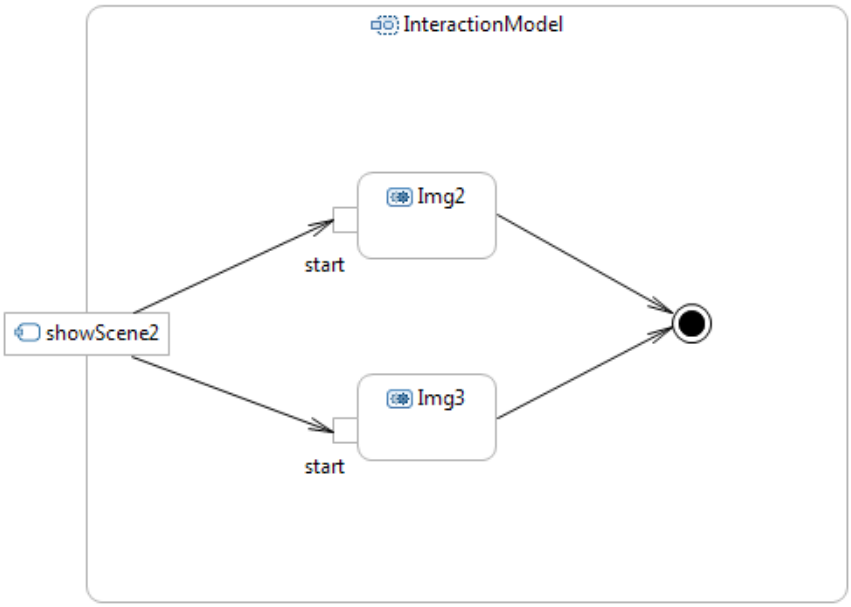


Figura 5.11: Modelo de interação da segunda cena

5.4.4 Mapeando o Modelo de Features para o Modelo Template

Uma vez definidos todos os modelos MML do *Modelo Template*, a próxima atividade é descrever os aspectos comuns e variáveis nos elementos pertinentes dos modelos. Para isso, é realizado um mapeamento entre o *Modelo de Features* e o *Modelo Template*.

O primeiro passo é criar outro diagrama UML utilizando o *pluginUML2 Tools* e utilizar estereótipos UML para criar um perfil de variabilidades associados a cada característica do Modelo de *Features,* lembrando que é possível acessar as características definidas no *Modelo de Features*do *plugin* FMP, uma vez que ele possui uma representação também em UML e acessível para o *plugin* UML2 Tools pela plataforma EMF.

O segundo passo é associar ao estereótipo criado (que representa uma característica) um elemento UML (classe, atributo, estado, pacote etc.). Na Figura 5.12 é ilustrado no perfil de variabilidade a definição de um estereótipo para a característica `OptionalFeature` do *Modelo de Features* da aplicação-exemplo. Nota-se que ele foi associado a um elemento `Class` do metamodelo UML.

O terceiro passo é aplicar o estereótipo criado no perfil de variabilidade em cada visão do *Modelo Template*. Por exemplo, na aplicação-exemplo, a terceira imagem (`Img3`) a ser exibida na segunda cena poderia associada à característica `OptionalFeature`, de modo a permitir que ela seja exibida ou não dependendo da configuração no *Modelo de Features* (ver Figura 5.13).

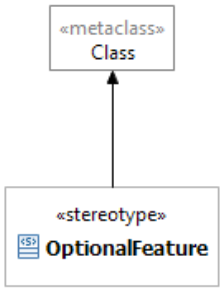


Figura 5.12: Estereótipo para a característica `OptionalFeature` associado a uma classe UML

(a) (b)

Figura 5.13: Anotações para mapeamento da característica `OptionalFeature`

O último passo é criar um formulário (*Wizard*) para a configuração das características. A Figura 5.14 ilustra a criação de formulário escolher a presença ou não da característica `OptionalFeature`, que na prática, poderia representar a exibição de uma figura (Img3) exibindo uma propaganda na exibição da segunda cena da aplicação-exemplo. Atualmente, o processo de criação da interface gráfica do formulário e mapeamento para uma arquivo de configuração do Modelo de Features é realizado manualmente.

Figura 5.14: Formulário (*Wizard*) para uma característica opcional

5.4.5 Transformação para o Modelo Instanciado

O Modelo Instanciado é originado a partir de uma transformação modelo para modelo e tem como entrada: (1) conjunto de modelos (visões MML) do *Modelo Template* anotados com as variabili-

dades e (2) configuração definida pelo Wizard. O processamento dos modelos utiliza um manipulador do *plugin* UML2.

Como o *Modelo Template* é formado por vários modelos distintos, a transformação começa pelo modelo estrutural e em seguida passa para o modelo de cenas. À medida que os elementos do modelo de cenas são processados, os modelos de apresentação são armazenados para posterior análise. Uma vez gerado o modelo de cenas do *Modelo Instanciado*, inicia-se a geração de cada modelo de apresentação.

Apesar das peculiaridades de cada modelo em particular, o algoritmo de identificação das variabilidades anotadas é igual para todos. Nesse cenário, cada modelo é processado recursivamente começando pelo elemento raiz (um pacote UML que agrupa os elementos do modelo) e percorrendo a árvore em um percurso em largura, isto é, verificando primeiramente os elementos de cada nível e os seus filhos da esquerda para a direita. A seguir alguns detalhes sobre o processamento das anotações em cada modelo.

Após a identificação das variabilidades, é realizado outro processamento para gerar um novo modelo (instanciado) para cada modelo MML (por exemplo, apagar um elemento ou multiplicá-lo quantas vezes for necessário). A seguir são descritas peculiaridades desse procedimento em cada modelo dependendo do tipo de variabilidade:

- **Variabilidade Condição de Presença (PC) -** a abordagem proposta permite anotar qualquer elemento UML, porém quando os elementos possuem uma hierarquia de pai e filho, a remoção de um elemento pai, implica na remoção dos seus elementos filhos, ou seja, só é necessário realizar a anotação estereótipo no elemento de nível mais alto. No modelo estrutural as anotações mais comum ocorrem em classes e associações, já no modelo de cenas, nos estados e transições.
- **Variabilidade de Multiplicidade (M) -** no modelo estrutural as classes anotadas com a multiplicidade são classes que poderão possuir uma ou mais instancias (objetos). Logo, estas anotações são para classes que participam em uma associação e possuem multiplicidade de um ou mais elementos. O processamento desta anotação refere-se apenas a modificar o valor da multiplicidade da associação para ser uma quantidade de instancias informada na configuração, ao invés dela ser indefinida. No modelo de cenas, uma cena com a variabilidade de multiplicidade implica na replicação de quantos elementos de cena forem necessários, como também da criação de modelos de apresentação para cada cena. Quando a cena é replicada, os elementos ligados a ela, como transições, também são replicados no Modelo Instanciado.

O algoritmo utilizado nessa atividade e na próxima (Seção5.4.6) para transformação entre modelos é baseado na "abordagem *template*" (CZARNECKI e ANTKIEWICZ, 2005; CZARNECKI e HELSEN, 2006). Nesta metodologia, a ordem e lógica do processamento de cada modelo MML podem ser alteradas explicitamente pelo desenvolvedor, não exigindo nenhuma política fixa. Além disso,

a transformação de um modelo para outro pode gerar vários tipos de elementos, também definidos pelo desenvolvedor, ou seja, um modelo do *Modelo Template* pode introduzir além de qualquer elemento UML, outros tipos de artefatos, por exemplo, trechos de código-fonte, arquivos de configuração etc. Tais propriedades permitem definir transformações flexíveis para atender especificidades de domínio. Por exemplo, no caso da plataforma Ginga-NCL, muitas vezes pode ser necessário também associar a um elemento declarativo do tipo `<media>` ou mais arquivos de código-fonte na linguagem Lua, arquivos de texto e arquivos que representam objetos de mídia (imagens estáticas, áudio, vídeo etc.).

5.4.6 Derivação da aplicação

A atividade final da etapa da Prototipação Rápida é a derivação do código-fonte parcial da estrutura da aplicação a partir do *Modelo Instanciando*. Esta derivação é realizada em dois passos: (1) mapeamento do *Modelo Instanciando* para o modelo NCL e (2) transformação do mapeamento de (1) para gerar o código-fonte NCL.

Assim como os elementos do *Modelo de Features* e da MML, os elementos da NCL são definidos como um metamodelo utilizando o framework EMF. Ele é gerado a partir da especificação *XML Schema* do NCL[8]. Dessa forma, um conjunto de classes geradas pelo EMF para manipular o Modelo Instanciado faz o mapeamento em memória dos elementos do metamodelo NCL. O objetivo desse mapeamento é associar os conceitos existentes no *Modelo Instanciado* para os elementos do metamodelo NCL. Como resultado, a partir de um conjunto de modelos MML é gerado um único modelo NCL. A seguir são descritos alguns mapeamentos de elementos do modelo MML para o modelo NCL de um trecho aplicação-exemplo (Figura 5.15).

```
1 <context id="ctxScene2">
2     <port id="showScene2_img2" component="img2" />
3     <media id="img2" src="./media/img2.png" descriptor="img2Desc"/>
4 </context>
```

Figura 5.15: Exemplo de código NCL derivado

Inicialmente, é analisado o modelo de cenas e para cena, o seu modelo de apresentação e modelo de interação. Uma cena é mapeada para um elemento de contexto da NCL (linha 1). No modelo de apresentação as classes são mapeadas para os objetos de mídia (linha 3). O tipo de mídia gerado (texto, imagem, vídeo etc.) e a localização do seu arquivo podem ser definidos no diagrama estrutural. Caso não sejam especificados na modelagem, o gerador utiliza arquivos de mídia padrão que podem

[8]Schemas XML NCL versão 3.0. Disponível em: http://www.ncl.org.br/pt-br/schemasxml

ser atualizadas na ferramenta de autoria. No modelo de interação cada fluxo de uma atividade é mapeado para uma porta diferente. O parâmetro `showScene2` da Figura 5.11 é mapeado para o elemento Img2, que no arquivo NCL é representado pelo elemento `<media id="img2">`.

A Figura 5.16 apresenta um resumo do mapeamento do modelo MML para o modelo NCL. Para cada Modelo Instanciado, representado por um dos quatro tipos de modelo: estrutural, cena, apresentação e interação, o mecanismo de transformação procura por elementos específicos de cada tipo de diagrama UML. Por exemplo, no diagrama de atividades, busca-se: parâmetros, fluxos e ações. Para o diagrama de estado, as transições e para o diagrama estrutural são consultadas as propriedades. Cada elemento presente em um dos diagramas UML é mapeado para um elemento NCL.

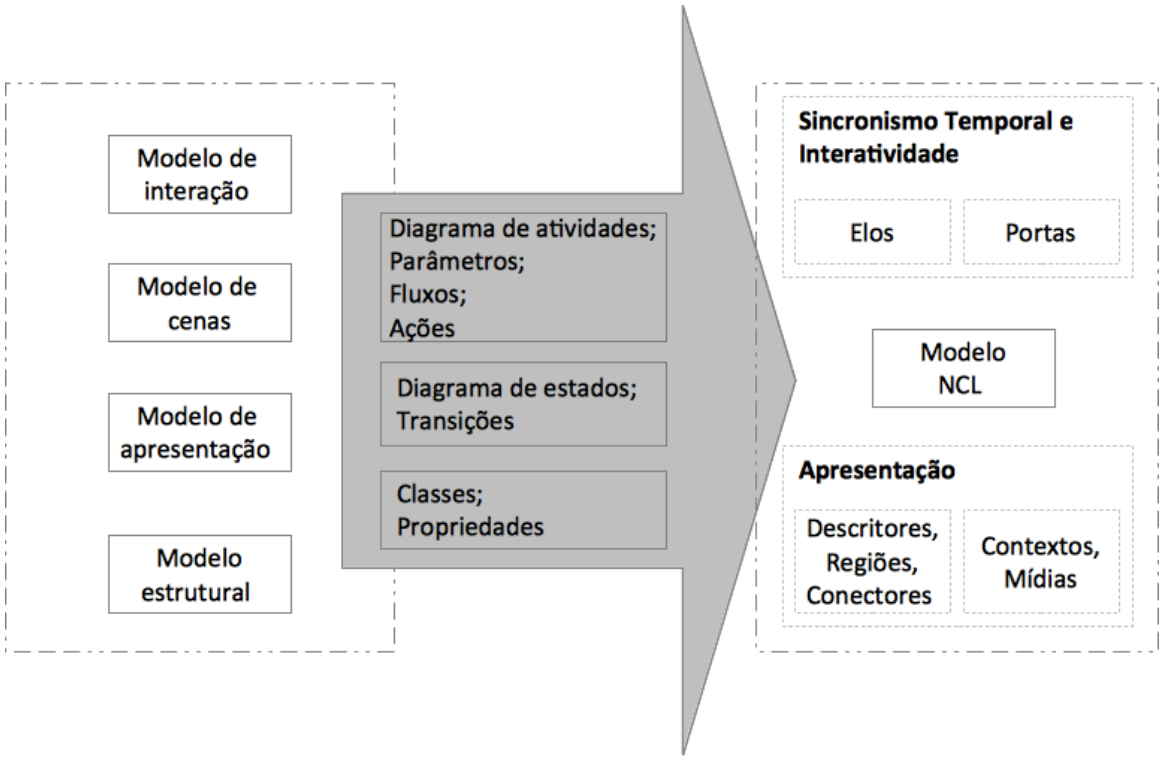


Figura 5.16: Resumo do mapeamento do modelo instanciado para o modelo NCL

5.5 Detalhamento do Refinamento da Aplicação

A criação e transformação do *Modelo Template* (detalhado na seção anterior) permite a geração de código-fonte base de uma aplicação de TV Digital compatível com o Ginga-NCL. O próximo passo da abordagem proposta é editar esse código utilizando alguma ferramenta de autoria para refinar a estruturar gerada e adicionar elementos mais complexos (por exemplo, projeto criativo e a lógica de aplicação específica) para obter o código-fonte final da aplicação. Apesar de ser possível utilizar um editor textual (por exemplo, o NCL Eclipse) para acelerar o preenchimento da sintaxe da linguagem declarativa, o ideal é utilizar um editor visual, pois este tipo de ambiente traz facilidades para desenvolvedores inexperientes, normalmente o perfil dos profissionais da equipe de mídia.

Uma das soluções mais recentes que atende os requisitos de autoria visual para aplicações Ginga-NCL é o NCL Composer (ver Seção 4.3.3). Por meio das suas abstrações visuais, essa ferramenta permite rapidez na edição de valores de atributos, edição espacial e temporal dos objetos de mídia sem necessitar que o desenvolvedor lide diretamente o código-fonte NCL. Tais funcionalidades são ideias para um projeto criativo da aplicação, também conhecido como projeto de mídia.

Entretanto, conforme explicado na Seção 4.3.3, o desenvolvimento de aplicações para TV Digital envolve também o projeto de software. Essa atividade, por exigir conhecimento de uma linguagem imperativa, é realizada por um profissional da equipe de software (desenvolvedor mais experimente). O papel dele é codificar a lógica de aplicação mais complexa, não suportada pelos elementos da linguagem declarativa, e realizar a integração com os elementos do projeto de mídia. Essa última tarefa pode ter apoio do NCL Composer, já que sua visão estrutural permite a representação de arquivos de scripts Lua como nós de mídia. Dessa forma, uma vez disponibilizados (desenvolvidos ou reutilizados) os scripts Lua, os profissionais envolvidos no projeto de mídia poderiam realizar a integração com o projeto de software. Porém, como já explanado na Seção 4.3.3, esse mecanismo apresenta vários pontos negativos que podem dificultar ou inviabilizar essa tarefa por parte da equipe de mídia.

5.5.1 Requisitos para reuso de objetos de mídia imperativa

Esta seção procurou detalhar algumas extensões recomendas no NCL Composer para facilitar a integração da visão estrutural com objetos de mídia imperativos. O objetivo principal é melhorar a integração do projeto de mídia e projeto de software na etapa de Refinamento da Aplicação da abordagem proposta neste trabalho.

Um código-fonte em Lua pode ser desenvolvido de modo independente do documento NCL. De qualquer forma, no momento da integração é necessário definir pelo menos um objeto de mídia no documento NCL que referencie um arquivo do código-fonte Lua. Consequentemente, é necessário representar no documento NCL a ligação com elementos do código imperativo Lua e vice-versa.

Considerando apenas a visão do desenvolvedor NCL, ele precisa identificar que trechos do código-fonte Lua (funções e variáveis) estão disponíveis e se atendem às suas necessidades. Apesar de contar com o auxílio da equipe de software, o desenvolvedor NCL deve ser também capaz de realizar essa tarefa, visto que o código-fonte Lua pode ter sido desenvolvido por terceiros. Portanto, é importante abstrair ao máximo o conteúdo da mídia imperativa de modo a permitir o uso simples e rápido para o desenvolvedor NCL. Assim, o principal requisito que essa proposta tenta atender no contexto de integração de documento NCL e mídia imperativa é: permitir uma ligação com o código presente na mídia imperativa de modo a identificar suas estruturas internas disponíveis (por exemplo, funções, variáveis ou propriedades de objetos) e facilitar a representação e ligação desses elementos na autoria criativa da aplicação NCL.

A partir dessa necessidade foram definidas 2 (duas) extensões para a visão estrutural do NCL Composer:

- **Extensão 1**: identificar automaticamente num arquivo de código-fonte Lua associado a mídia imperativa seus elementos disponíveis e a partir daí representar visualmente quais são as interfaces de entrada (recebe eventos) e interfaces de saída (retorna eventos) para o documento NCL;
- **Extensão 2**: definir elementos visuais para representar uma ação que é associada a uma das interfaces de entrada do objeto de mídia imperativa e um evento que é associado a respostas de uma das interface de saída do objeto de mídia imperativa.

Nas próximas duas seções essas extensões são detalhadas.

5.5.2 Identificação e representação de interfaces de entrada e saída numa mídia imperativa

A primeira extensão é responsável por auxiliar o desenvolvedor NCL a identificar e representar facilmente que elementos estão disponíveis num objeto de mídia imperativa. Para facilitar o processo de identificação é utilizado um mecanismo de **anotações** no código-fonte Lua. As anotações são inseridas explicitamente no código imperativo e relacionadas diretamente às estruturas do código: funções e variáveis. Dessa forma, a descoberta do conteúdo disponível em uma mídia imperativa é feita de forma direta.

Um método é anotado usando uma anotação `@Method` e uma variável ou propriedade de objetos com `@Variable`. Ambas as anotações devem ser inseridas como um comentário da linguagem Lua antes da sua declaração. A anotação do método e da variável deve acompanhar uma definição de outros atributos: (a) `name`: representando o nome do método ou da variável em questão; (b) `params`: determinando os parâmetros do método; (c) `outputValue`: o valor de retorno do método, caso exista; e (d) `description`: uma breve descrição do método ou da variável. A Figura 5.17 apresenta a sintaxe completa do conjunto dessas anotações.

```
@Method(name="methodName", params="param1, param2, ..., paramN",
outputValue="outputValue", description="brief description" )


@Variable( name="variableName",description="brief description"  )
```

Figura 5.17: Anotações para identificar elementos do código-fonte Lua

A Figura 5.18 ilustra um exemplo de uso das anotações associado a uma mídia imperativa que possui um método (`getValue`) que recebe algum valor da mídia (fluxo de entrada) e um método (`returnValue`) que retorna um valor. No exemplo, o método `getValue` consulta um valor associado a uma chave de uma coleção de dados (tabela) e repassa o valor obtido para o método `returnValue`. Esta, por sua vez, cria um evento (`counterEvt`) que encapsula o valor (`value`) que será enviado para o documento NCL. Ressalta-se que é necessário lançar (`event.post`) um evento de início (`start`) e fim (`stop`) de atribuição para atualizar o atributo (de nome `tagValue`) de nó mídia NCL que chamou o código-fonte Lua. É importante observar que o método `getValue` é anotado com `@Method`, que possui o nome do método "`getValue`", o parâmetro "`tag`" e o valor de saída "`tagValue`".

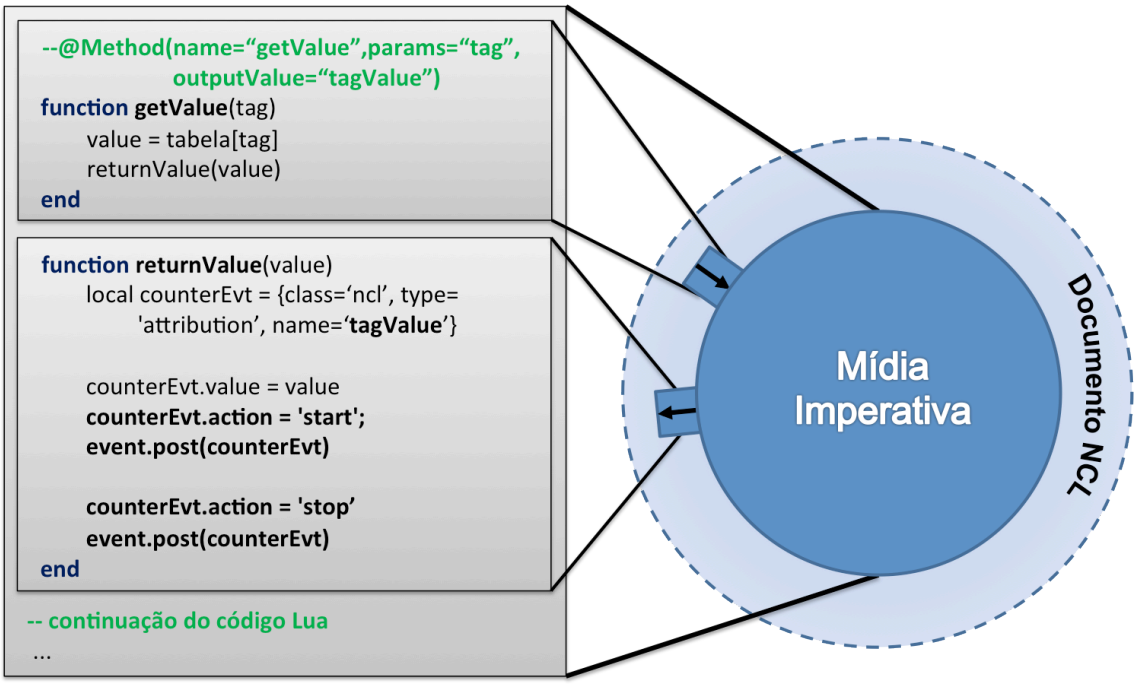


Figura 5.18: Exemplo de uso de anotações em uma mídia imperativa

Para que a visão estrutural do NCL Composer exponha os elementos anotados de uma mídia imperativa foram definidos dois novos elementos visuais: **pino de entrada** e **pino de saída**. A ideia é estender a semântica de uma ancora em um nó de mídia NCL para incluir, além do elemento associado, o sentido da direção (entrada ou saída) do processamento em um nó de mídia imperativa.

A Figura 5.19 ilustra como é o mapeamento entre os pinos e o código imperativo interno à mídia. Um par de pinos (pino de entrada e de saída) pode estar associado a uma determinada estrutura da mídia imperativa, seja um método, variável ou propriedade de objeto. Se associado com um método que não retorna um valor, o pino de saída correspondente não se faz necessário. Para um mesmo elemento, também podem existir mais de um pino de saída, quando são retornados mais de um valor. Os

pinos de entrada e saída em conjunto formam uma espécie de interface de comunicação com elementos do documento.

A partir das anotações inseridas no código-fonte Lua e representações por pinos de entrada e saída, a mídia imperativa passa a ser integrada na visão estrutural de um modo mais direto. O processo de criação da mídia imperativa é idêntico ao existente atualmente no NCL Composer, a única diferença é que no momento em que o arquivo do código-fonte Lua é associado a uma mídia imperativa, é realizado a descoberta das anotações que descrevem as funções, variáveis e propriedades de objetos acessíveis.

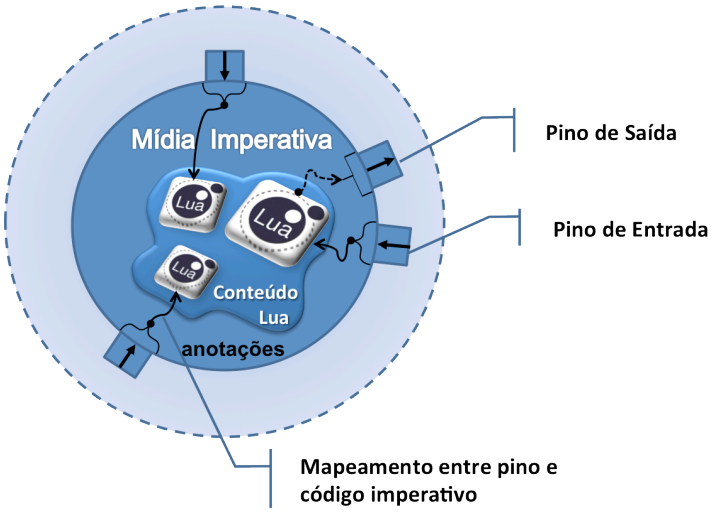


Figura 5.19: Pinos de entrada e pinos de saída numa mídia imperativa Lua

5.5.3 Associação da mídia imperativa com elementos do documento NCL

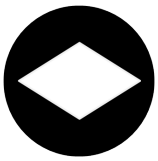
Na visão estrutural atual do NCL Composer, os objetos de mídia imperativos, assim como qualquer outro objeto de mídia NCL, são representados como uma entidade visual. Eles podem estar associados a elos que permitem que eles ativem (papéis de condição) ou recebam ações a partir de eventos padrão do NCL (apresentação, seleção e atribuição). Porém, como explicado na Seção 4.3.3, a semântica para utilizar esses três tipos de eventos não mapeia de modo intuitivo o conteúdo que pode existir num objeto de mídia imperativa. Assim, também é proposta outra extensão para melhorar a integração do projeto de mídia e projeto de software no NCL Composer. Ela está relacionada à forma como os pinos de entrada e saída são associados a outros elementos do documento NCL e define outro **tipo de ação (**`call`**)** e outro **tipo de evento (**`onGetValue`**)**.

A nova ação "`call`" permite criar um elo e associar a algum papel de ação dele, um pino de entrada da mídia interativa. Isso permite, por exemplo, que um evento de condição do NCL seja direta e visualmente mapeado para uma chamada de um método de um objeto de mídia imperativa. Na Figura 5.20 é exibido o ícone escolhido para representar a ação de "`call`". Na ação, a parte exterior do

circulo é clara e o símbolo interno escuro, seguindo assim o mesmo padrão de aparência dos ícones de eventos de ação utilizados na ferramenta NCL Composer.

Outro ponto importante é que uma chamada de método na mídia imperativa pode envolver a presença de parâmetros, dessa forma a visão estrutural foi alterada para facilitar a especificação dos valores para cada parâmetro. Dependendo da quantidade de parâmetros do método, a visão deve oferecer a possibilidade de especificar valores diferentes para cada parâmetro em separado. No caso de variáveis, o autor sempre está restrito a especificar apenas um valor que será atribuído a variável em questão.

(a)



(b)

Figura 5.20: Tipo de ação `call` (a) e tipo de evento `onValueGet` (b)

A especificação do valor para um determinado parâmetro pode ter 2 (duas) formas distintas: (1) uma *string* formada por letras e/ou números; e (2) o conteúdo presente na propriedade de uma outra mídia. O segundo caso abre possibilidades maiores para parametrização do comportamento da mídia imperativa, pois além de receber valores de objetos de mídia que representam conteúdo multimídia (imagem, texto, áudio, vídeo etc.), também é possível receber valores de outras mídias imperativas.

O novo evento "`onValueGet`" permite disponibilizar para a parte declarativa o valor presente num pino de saída de um mídia imperativa. A Figura 5.20b ilustra o ícone utilizado para este novo tipo de evento, onde a parte exterior do circulo é escura e a parte interna clara, seguindo novamente o padrão de aparência dos ícones de eventos utilizados na ferramenta Composer.

O principal objetivo da criação dos pinos de entrada e saída, da ação `call` e do evento `onValueGet` é disponibilizar automaticamente o acesso ao conteúdo existente na mídia imperativa Lua. Logo, quando um objeto de mídia (nó de mídia) NCL qualquer se liga a uma mídia imperativa por meio de um pino de entrada ou de saída, a exibição da ação `call` e do evento `onValueGet` é realizada de modo automático pela visão estrutural. Dessa forma, se facilita e acelera o uso de mídias imperativas no documento NCL. A Figura 5.21 apresenta uma mídia imperativa com uma ação de `call` (lado esquerdo) e um evento `onValueGet` (lado direito), os quais são exibidos automaticamente quando no momento da criação dos elos e *bind* com o pino de entrada e pino de saída, respectivamente.

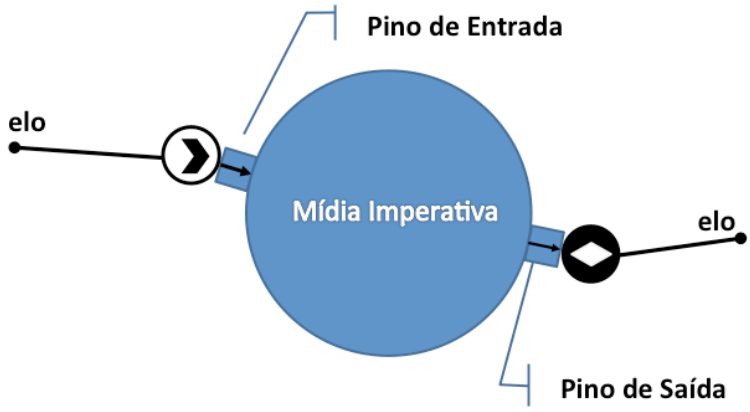


Figura 5.21: Integração entre elos, eventos, ações e pinos de entrada e saída

5.5.4 Exemplo de uso de mídia imperativa como um serviço Web

Para melhor entendimento da proposta de extensões ao NCL Composer apresentadas nas últimas duas seções deste capítulo será explicado o uso de mídia imperativa que implementa um serviço Web que tanto recebe chamadas (ação `call`), como o retorna de valores (evento `onValueGet`). Para ter uma natureza mais prática, foi escolhido o serviço *TinyWebDB*[9], que permite armazenar e recuperar dados na forma chave e valor num banco de dados remoto da Web.

O TinyWebDB possui um método (`storeavalue`) para armazenar um valor correspondente a uma chave e outro método (`getvalue`) para recuperar valores a partir de uma determinada chave. Para cumprir os requisitos de interação com este serviço, os pinos (representados por pequenos retângulos vermelhos com setas de entrada e saída na mídia imperativa) que fazem parte da mídia são: dois pinos de entrada (1) `storeValue`, armazena no servidor uma dupla chave; valor (2) `getValue`, recupera um valor armazenado a partir de uma chave. E um pino de saída (1) `tagValue`, que é utilizado para ter acesso ao valor retornado pelo pino de entrada `getValue`.

Na aplicação, a interação no uso do serviço Web é realizado do seguinte modo: um valor é especificado pelo usuário num campo de texto e esse valor é enviado para o servidor quando selecionado um outro botão. Para recuperar o valor armazenado no servidor no passo anterior, o usuário pressiona outro botão e o valor é exibido na tela.

[9]TinyWebDBService. Disponível em: http://goo.gl/416Br

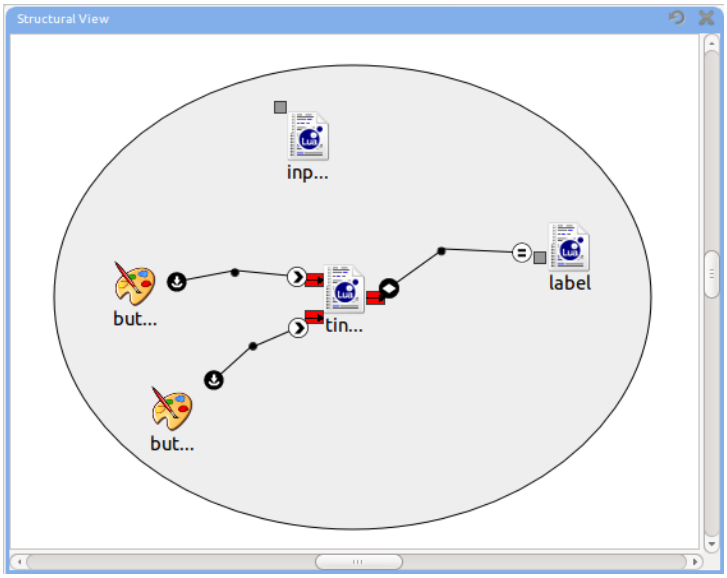


Figura 5.22: Visão estrutural do exemplo TinyWebDB

A Figura 5.22 apresenta a visão estrutural da aplicação. Duas mídias de imagens `buttonStore` e `buttonGet` são usadas para fazerem o papel do botão que envia o valor e do botão para recuperar o valor do servidor, respectivamente. Além dos botões, a mídia `inputText` é usada para o usuário digitar o texto a ser enviado e a mídia `label` para exibir o valor recuperado do servidor.

O elo entre o botão `buttonStore` e `tinyWebDB` modela o envio do valor presente na mídia `inputText` para o servidor. Este elo é ativado quando o usuário pressiona o botão, fazendo com que uma ação `call` seja realizada sobre o pino de entrada `storeValue` (Figura 5.23) da mídia `tinyWebDB`. Esta ação faz com que o método `storeValue` correspondente no serviço Web seja chamado e assim a aplicação apresenta uma comunicação com o parte "servidor" do serviço Web.

```lua
require 'lib/TinyWebDBClient/tinyWebDBClient'

--@Method(name="storeValue", params="tag,value")
function storeValue(tag, value)
    tinyWebClient = TinyWebDBClient("usfwebservice.appspot.com", "80")
    tinyWebClient:storeValue(tag, value)
end

--@Method(name="getValue", params="tag",
--        outputValue="tagValue")
function getValue(tag)
    tinyWebClient = TinyWebDBClient("usfwebservice.appspot.com", "80")
    tinyWebClient:getValue(tag, getResponse)
end

--Método para processar a resposta da requisicao enviada ao WS
function getResponse(value)
    local outputEvent = {
        class = 'ncl',
        type  = 'attribution',
        name  = 'tagValue',
    }
    --sinaliza app NCL de que o resultado foi recebido
    outputEvent.value = value
    outputEvent.action = 'start'; event.post(outputEvent)
    outputEvent.action = 'stop';  event.post(outputEvent)
end

function eventHandler(evt)
    ...
end

event.register(eventHandler)
```

Figura 5.23: Código-fonte da mídia imperativa que acessa serviço TinyWebDB

O método `storeValue` possui dois parâmetros para serem especificados (`tag` e `value`) e os mesmos são configurados na criação do elo entre as mídias `buttonStore` e `tinyWebDB`, que exibe a janela apresentada na Figura 5.24. Neste exemplo, os valores dos parâmetros são originados de fontes distintas. O primeiro parâmetro `tag` é configurado para possuir o valor fixo "`1`" e o segundo parâmetro `value` é configurado para possuir o valor relativo a uma mídia presente na visão estrutural. Este valor é especificado para a propriedade `text` da mídia `inputText`, a qual é responsável por conter o valor de entrada digitado pelo usuário.

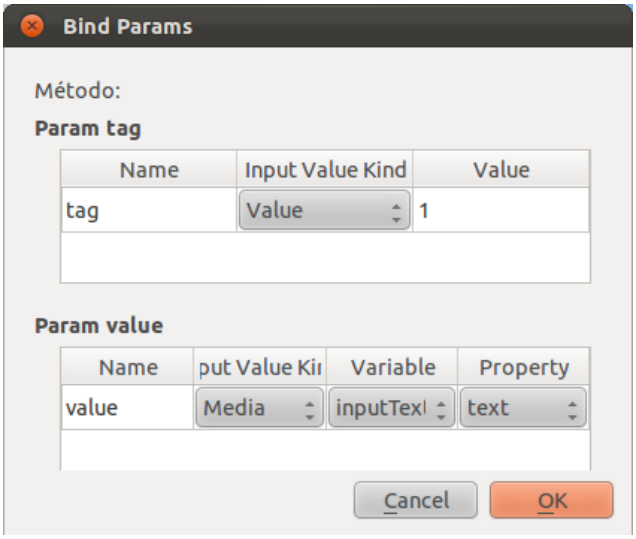


Figura 5.24: Tela de configuração elo entre o `buttonStore` e o `tinyWebDB`

A recuperação e a apresentação do valor do servidor é iniciada pelo elo que liga as mídias `buttonGet` e `tinyWebDB`. O elo é ativado quando o usuário pressiona o botão, o qual dispara em seguida a ação de `call` sobre o pino `getValue`. Este pino de entrada é chamado com apenas um parâmetro referente à `tag` que se deseja recuperar o valor.

Para apresentar o valor na mídia `label`, primeiramente o valor é recuperado através do pino de saída `tagValue`. Ele foi criado automaticamente para disponibilizar o valor retornado pelo servidor Web. Quando o valor fica disponível no pino (ou seja, a resposta com o valor do servidor é enviada) ele deve ser exibido na mídia `label`. Este comportamento é feito pelo elo que possui a condição `onValueGet` da mídia `tinyWebDB` e a ação de armazenar propriedade `text` da mídia `label`.

5.6 Considerações sobre o capítulo

Este capítulo descreveu a abordagem proposta neste trabalho. Para um melhor entendimento foram apresentadas 3 (três) visões gerais que se preocuparam em detalhar atividades MDD no contexto de desenvolvimento de aplicações para TVD com foco na integração entre os projetos de mídia e software. Independente da visão descrita, as atividades sempre foram agrupadas em duas partes: *Prototipação Rápida* e *Refinamento da Aplicação*. Estas etapas posteriormente foram detalhadas para explicar o uso das técnicas de modelagem específica de domínio e desenvolvimento de família de aplicação.

A etapa de *Prototipação Rápida* realiza atividades de estruturação de requisitos, análise e projeto de software, estruturando uma família de aplicação por meio de atividades de engenharia de domínio e engenharia de aplicação. Para tanto, ela foi definida com os seguintes passos: (1) utilização do *Modelo de Features* para representar as características comuns e variáveis de uma categoria (família) de aplicação; (2) modelagem do *Modelo Template*, baseado no *Modelo de Features* e utilizando as visões MML (linguagem específica de domínio), para representar a arquitetura de referencia da categoria da

aplicação; (3) transformação do *Modelo Template* para o *Modelo Instanciado*, este último representando uma aplicação dentro das inúmeras possibilidades em uma família de aplicação; e (4) derivação do código-fonte da aplicação a partir do *Modelo Instanciado* para uma plataforma específica. Um ponto importante desses passos é que apenas o último é dependente da plataforma-alvo, ou seja, todos os anteriores podem ser reaproveitados na modelagem para outras plataformas de TVD ou mesmo outros domínios, por exemplo, aplicações Web que utilizam HTML e *ECMAScript*. Cabe também destacar a utilização de formulários (*Wizards*) que permitem uma configuração de alto nível de abstração dos transformadores de modelos e geradores de código-fonte da plataforma EMF. Em outras palavras, uma vez realizada a Engenharia de Domínio, que é representada pela capacidade de gerar versões de uma categoria de aplicação a partir de *Wizards*, a Engenharia de Aplicação, representada pela geração do código-fonte do protótipo, consiste basicamente da seleção de opções nos formulários.

A etapa de *Refinamento da Aplicação,* que inicia a partir do código-fonte gerado *na Prototipação Rápida*, foi detalhada com um levantamento de necessidades e especificações das extensões no NCL Composer para permitir melhor integração entre as visões estrutural (projeto de mídia) e o código-fonte das mídias imperativas (projeto de software). De modo geral, sua implementação consiste na extensão de uma linguagem visual para permitir que a equipe de mídia integre gráfica e automaticamente artefatos de software desenvolvidos (e descritos por anotações) pela equipe de software. Também foi comentado como funcionam essas novas funcionalidades no NCL Composer e suas vantagens por meio da descrição de um exemplo de uso.

Uma vez que já foi descrita de forma geral e em detalhes a abordagem MDD proposta neste trabalho, o próximo capítulo explica as avaliações da abordagem utilizando provas de conceitos e experimentos controlados, bem como os resultados obtidos.

6 AVALIAÇÃO DA ABORDAGEM

Este capítulo apresenta duas metodologias utilizadas para avaliar a viabilidade, dificuldades e benefícios do uso da abordagem proposta. O objetivo é identificar evidências que reforcem a ideia de que uma abordagem de desenvolvimento orientada a modelos pode trazer ganho de produtividade quando considerado o projeto de mídia e projeto de software no domínio de aplicações para TVD.

A Seção 6.1 utiliza as técnicas de persuasão e implementação (SHAW, 2001) para ilustrar o emprego da abordagem com 3 (três) exemplos e também demonstrar o uso prático dos protótipos desenvolvidos para apoiar as etapas de Prototipação Rápida (Ferramenta baseada no Eclipse EMF) e Refinamento da Aplicação (versão estendida do NCL Composer).

A Seção 6.2 utiliza a técnica de avaliação baseada em estudos empíricos (SHAW, 2001) por meio do projeto de experimentos controlados para coletar e analisar dados do uso abordagem em comparação com outras metodologias.

6.1 Avaliação por persuasão e implementação

Esta seção busca ilustrar o uso a abordagem proposta através de 3 (três) provas de conceito, mais especificamente, tratando um caso específico de aplicação multimídia: aquelas enviadas aos receptores por uma emissora de TV Digital juntamente com o conteúdo audiovisual ao vivo, utilizando o canal de difusão que está sob seu controle e numa plataforma com ou sem canal de comunicação com a emissora ou com a Web.

Um fator importante que foi considerado como critério para escolha das aplicações foi reduzir do escopo da diversidade de possibilidades existentes para tratar problemas que consideram a necessidade de integração entre projeto de mídia com um projeto de software, ou seja, aplicações que possuem uma lógica de aplicação mais complexa. Assim, foi priorizada a escolha de aplicações que se encaixam na categoria *Distributed* (Seção *4.2*).

6.1.1 Exemplo de uso 1: TV Voting

A primeira aplicação escolhida é conhecida como *TV Voting*, que permite ao usuário participar de uma enquete pela TV com envio da resposta e recebimento de resultado “ao vivo” através de canal direto com um serviço Web que representa um “Servidor de enquetes”.

A Figura 6.1 exibe uma das aplicações propostas que é composta por 3 (três) cenas obrigatórias: *Cena 1*: com ícone de interatividade (inicia interação); *Cena 2*: exibe a pergunta e opções de respostas para a votação; e *Cena 3* com confirmação do envio da resposta e, se existir comunicação, resultado parcial ou final da enquete. Além disso, pode existir uma adicional *Cena 4,* que detalha a opção escolhida e confirmação do voto.

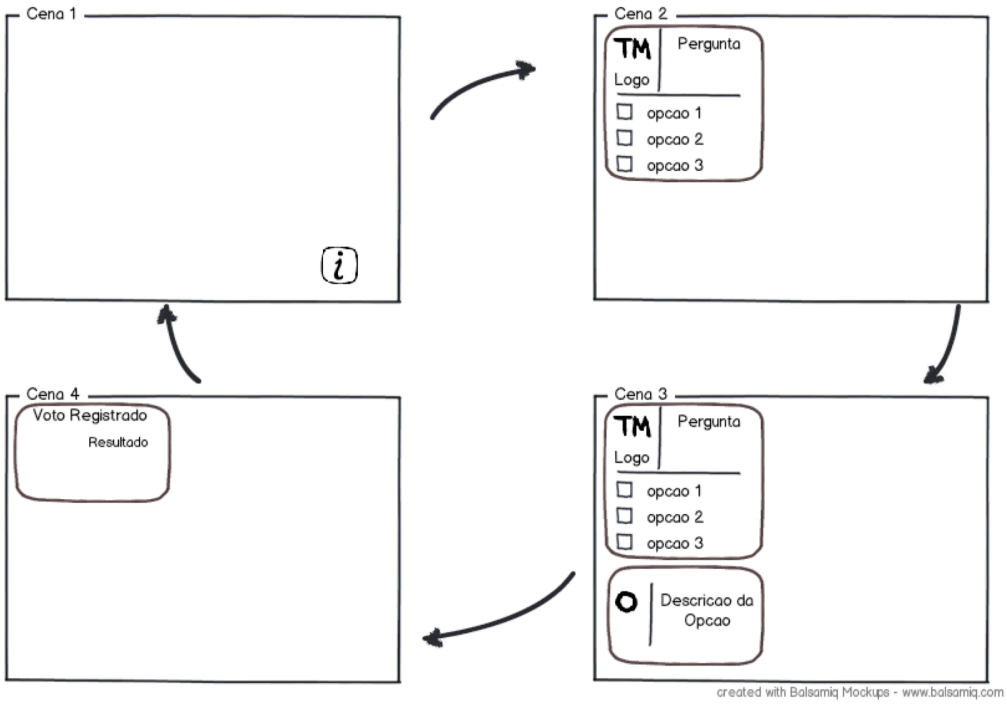


Figura 6.1:*Storyboard* da Aplicação da *TV Voting*

6.1.1.1 Passo 1 – Especificação do Modelo de Features

O Modelo de *Features* definido para descrever as possíveis soluções de Modelo *Template TV Voting*, é apresentado na Figura 6.2. O conjunto de características relacionadas aos aspectos estruturais (`Structural`) e aos valores das instâncias dos *templates* (*DataInstantiation*) são exibidos na Figura 6.2a e na Figura 6.2b, respectivamente. O modelo `Structural` define, além de outras opções variáveis, a característica alternativa `QuestionFormat`, que indica as possibilidades da enquete com múltipla escolha ou da enquete com resposta aberta. A opção de exibir ou não as informações para a resposta escolhida pelo usuário é tratada pela característica opcional `ChoiceInformation`. A característica `InteractionMode` representa o modo de interação do usuário com a enquete e possui as opções para indicar uma interação sem comunicação (`Indirect`) ou com comunicação (`ReturnChannel`) , este último possuindo um conjunto de característica alternativas (`WebService`, `HTTP` e `Socket`). Já o modelo `DataInstantiation` do *TV Voting* possui as opções para preencher os valores relativos a questão através de `Question`, as respostas através de `Answers`, informações do modo de

interação com `Interaction` e, por último, opções para especificar as imagens dos backgrounds das cenas, inclusive a cena opcional de confirmação.

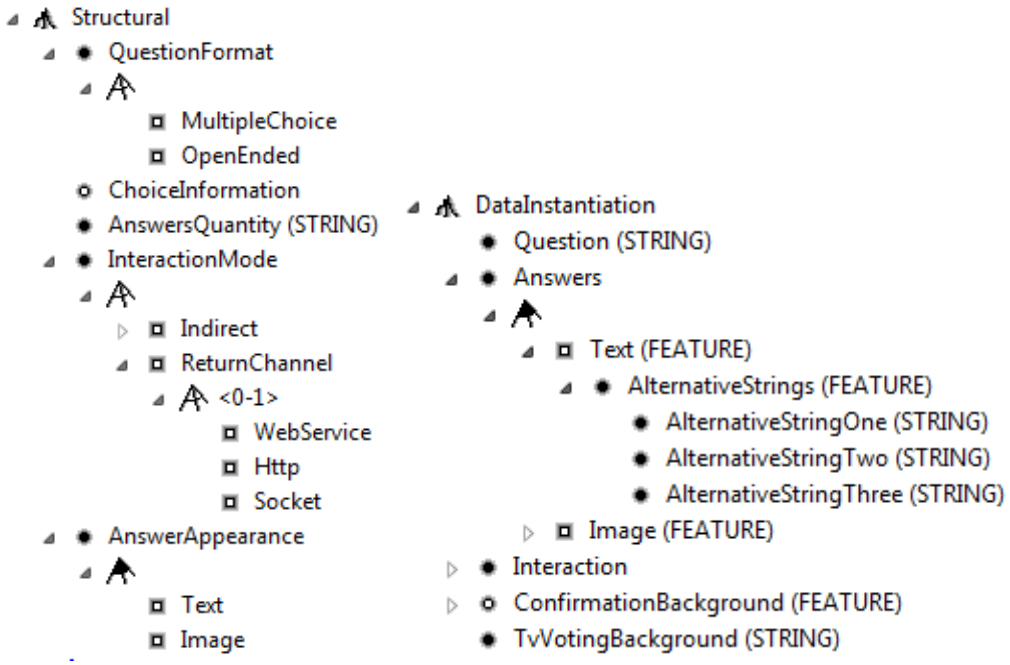


Figura 6.2: Modelo de Features da *TV Voting*

6.1.1.2 Passo 2 - Projeto do Modelo Template e variabilidades associadas

O *Modelo Template* de *TV Voting* é descrito através dos modelos relacionados às visões estruturais e comportamentais da linguagem MML. Dessa forma, o *template* é composto pelos seguintes modelos: (i) estrutural, (i) de cena, (iii) de apresentação e; (iv) de interação. Além disso, as variabilidades de cada modelo são especificadas como estereótipos UML que são anotados junto aos elementos do modelo. Cada estereótipo é mapeado para uma determinada característica. Como descrito na Seção 5.4.3, cada *Modelo Template* não se limita a especificar apenas uma solução da família de modelos, mas sim conter a união de todas as possíveis soluções de *Modelo Template*. A seguir ilustramos alguns diagramas UML para o template de *TV Voting*.

De acordo com o MML, a primeira modelagem a ser realizada é a estrutural, nela se evidencia os elementos do domínio, bem como o seu relacionamento com as respectivas mídias. Na Figura 6.3 é ilustrado o modelo estrutural de *TV Voting*. Ele é composto pelos elementos do modelo (classes, associações etc.) e das anotações dos mesmos, realizadas pelos estereótipos. Com a aplicação dos estereótipos mapeiam-se os elementos do modelo para as características. O modelo contém uma classe para a enquete (`Poll`), que possui uma pergunta e duas ou mais respostas (`Answer`), e por fim o resultado parcial da votação (`Result`). A aplicação não permite a geração de perguntas com múltipla escolha.

Ainda nesse modelo, fica explícito os elementos do domínio associados aos elementos de mídia `AnswerIcon`, `AnswerText`, `PollBackground`, respectivamente, um objeto de mídia imagem e um objeto de mídia texto para perguntas e um objeto de mídia imagem para compor o fundo da cena de votação. Outro aspecto importante é a representação do modo de interação sem canal de retorno (`ReplyData`) e com canal de retorno (`ReturnChannel`) que permite configurar o protocolo de acesso e associá-lo a aplicação de forma modular.

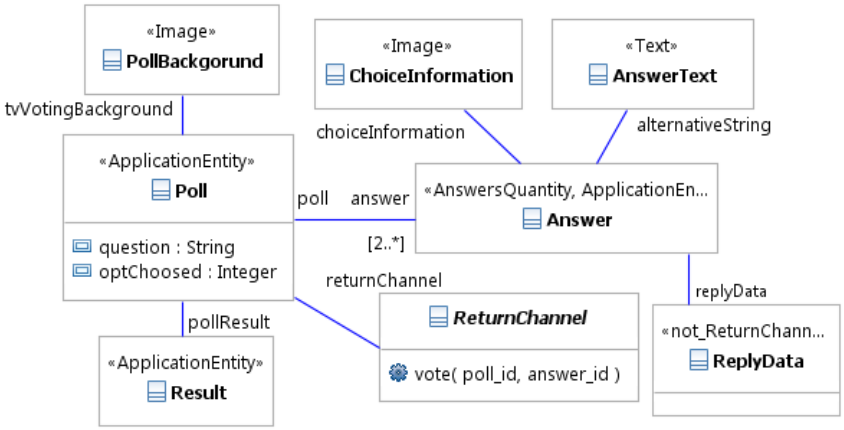


Figura 6.3: Modelo estrutural da *TV Voting*

Continuando com a modelagem, o modelo de cenas deve especificar o comportamento geral da aplicação através de uma máquina de estados com um conjunto de estados (cenas) e de transições entre eles. A Figura 6.4 apresenta o *template* com quatro estados para representar a cena com o ícone de interatividade (`Start`), a cena de votação (*TvVoting*), a cena opcional para exibir as informações das respostas (`InfoAnswer`) e a cena para exibir a confirmação e o opcionalmente o resultado da votação (*Result*).

Tal modelo foi baseado nas possibilidades de cenas e sequência entre elas de acordo com as seguintes variações: (i) permitir ou não a exibição das informações adicionais para a resposta escolhida e (ii) adição de cenas de informações adicionais de acordo com a quantidade de respostas existentes. Para a primeira variação, associada a presença da característica opcional `ChoiceInformation` (Figura 6.2) o modelo exibe a cena `InfoAnswer` da resposta escolhida e, após o voto ter sido confirmado, a cena com o resultado da votação é exibido (ambas os estados são aplicados o estereótipo `ChoiceInformation`). Caso contrário, na ausência de `ChoiceInformation`, é exibido apenas a cena de votação (`TvVoting`) e em seguida a cena do resultado (estado `Result` com aplicação do estereótipo `not_ChoiceInformation`). Para a segunda variação, associada à característica `AnswersQuantity`, a cena é anotada com a variabilidade de multiplicidade com o intuito de criar cópias da cena `InfoAnswer` específicas para cada resposta. Na modelagem, cada cena representa uma

tela da aplicação e esta tem a sua interface com o usuário especificada através do modelo de apresentação de cena, explicado a seguir.

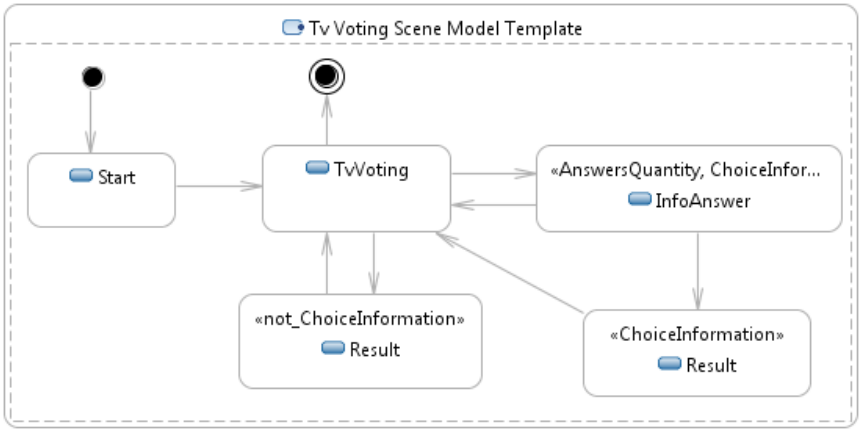


Figura 6.4: Modelo de cenas da *TV Voting*

O modelo de apresentação da cena de votação (Figura 6.5) descreve os elementos da interface gráfica através das classes `AnswerText`, `QuestionText` e `Background`, as quais representam respectivamente os textos das respostas, da questão e uma imagem para compor o fundo da cena. Elas são anotadas com o estereótipo `OutputComponent` de modo a representar um componente abstrato para exibição de dados.

Os valores dos componentes visuais a serem exibidos são representados tomando como base uma dependência partindo da classe para o objeto do modelo estrutural, o qual é responsável por armazenar os dados. Por exemplo, o texto da resposta é obtido utilizando a dependência "`src`" cujo valor é a propriedade `alternativeString` especificada no elemento `answer`. O estereótipo `AnswersQuantity` define a variabilidade com relação à quantidade de respostas especificada para a aplicação.

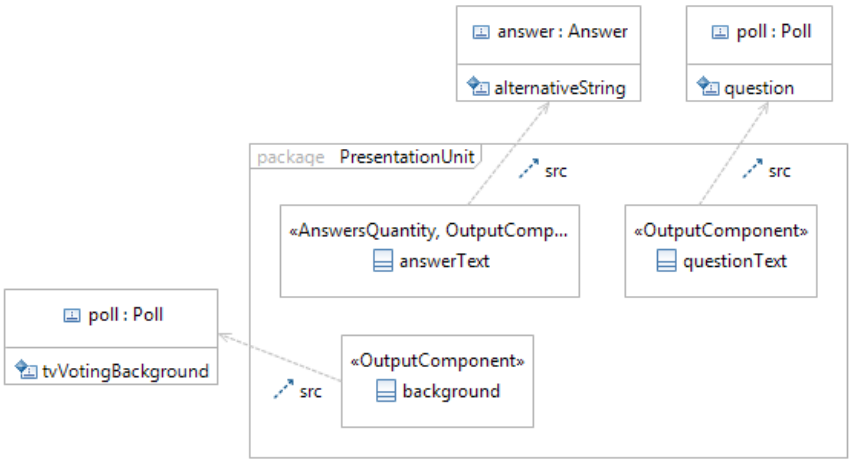


Figura 6.5: Modelo de apresentação da cena de votação da *TV Voting*

O modelo de interação é um diagrama de atividades da UML anotado com as variabilidades. A partir dele, deve-se derivar o relacionamento temporal e a interatividade entre os objetos de mídia como definido no modelo NCM (SOARES; RODRIGUES, 2005).

Pelo fato do modelo NCM não ser considerado no projeto inicial da linguagem MML, o modelo de interação precisou ser definido com uma semântica próxima aos conceitos presentes no modelo NCM. A iniciativa neste sentido foi fazer com que o diagrama de atividades trate os conceitos de elos baseado na relação da causalidade. Em uma relação causal, uma *condição* deve ser satisfeita para que ações possam ser disparadas.

Para isso, uma ação no diagrama de atividades representa um objeto de mídia, o seu pino de saída uma condição e seu pino de entrada uma determinada ação. O fluxo do diagrama de atividades modela a ocorrência do evento e as ações que são realizadas após o disparo do evento.

A Figura 6.6 apresenta o modelo de interação para a cena `InfoAnswer`. O ponto 1 trata a inicialização dos componentes da interface gráfica (apresentação dos objetos de mídia), representando os fluxos existentes entre o parâmetro da atividade `showTvVoting` e os pinos de entrada `start` pertencentes aos componentes presentes na interface da cena de informação da cena. Entre os elementos estão o *background* da cena, o botão de votação, entre outros elementos.

O fluxo do ponto 2 especifica a votação do usuário em uma determinada opção, a qual acontece quando o usuário aperta o botão verde do controle remoto durante a cena. Este fluxo começa a partir do pino de saída "`onSelectionGREEN`" do componente `buttonVoting` (componente de interatividade aplicado com estereótipo `ActionComponent`). Este pino de saída representa a condição que está atrelada ao usuário pressionar o botão verde do controle remoto. Em seguida é chamada a ação `sendPollOption` do componente `returnChannelInteractionLua`, a qual é responsável por enviar o voto para o serviço Web. Após a chegada do resultado pelo pino de saída `onResultReceived`, o resultado é armazenado na propriedade `result` do componente `globalVar` e em seguida é atribuído o valor `true` para propriedade `voting` deste mesmo componente, indicando assim que houve a votação na resposta. A última ação realizada é a ação de transição para a cena de resultado, a qual é feita pela chamada do componente de transição de cenas `Result` (componente com aplicação do estereótipo `ExitTo`).

O fluxo do ponto 3 especifica a ação do usuário em querer voltar para a cena anterior, ou seja, a cena de votação. Após o usuário pressionar o botão vermelho sobre o componente `buttonBack` (componente de ação aplicado com estereótipo `ActionComponent`), a propriedade `voting` do componente `globalVar` recebe o valor `false` para indicar que não houve votação e em seguida é chamada a ação de `stop` para indicar o término da cena.

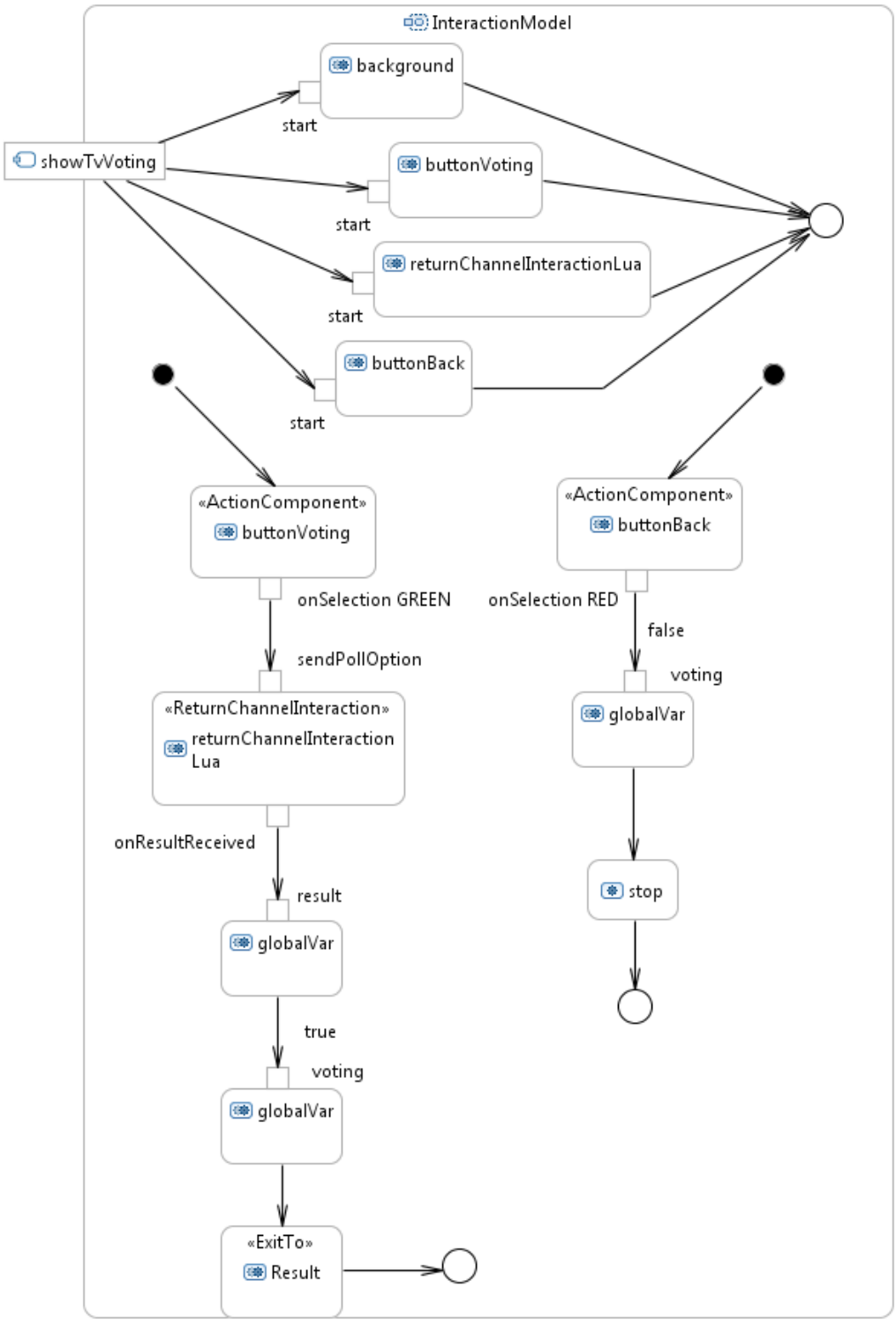


Figura 6.6: Modelo de interação da cena de votação da *TV Voting*

6.1.1.3 Passo 3 – Instanciação dos Modelos Abstratos de Templates

Um *Modelo Instanciado* representa o processamento das variabilidades do *Modelo Template* (descritas na seção anterior) de acordo com as opções do engenheiro da aplicação. Estas escolhas são satisfeitas através de uma configuração do *Modelo de Features* que é criada pelo *Wizard* e são descritas no passo seguinte. A seguir são ilustradas duas instâncias (*Modelos Concretos de Templates*): estrutural e de cenas. Assumiremos que a configuração escolhida foi uma enquete com perguntas e respostas do tipo textual, com (três opções respostas, com tela de informação das respostas e canal de retorno.

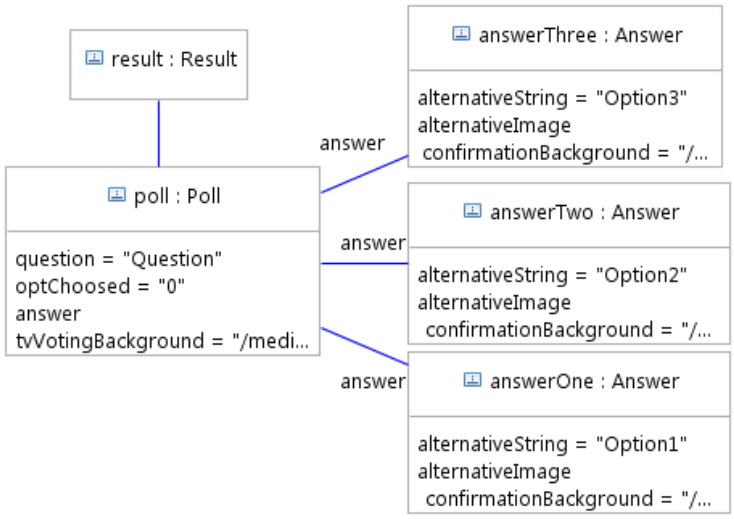


Figura 6.7: Modelo Instanciado estrutural da *TV Voting*

A instância do modelo estrutural da Figura 6.7 é obtida pela transformação do modelo estrutural num diagrama de objetos. Assim, cada classe, com exceção da classe *ReplyData*, é mapeada para um objeto. A classe *ReplyData* representa as informações da comunicação indireta e foi removida pelo motivo da escolha da interação por canal de retorno. Os objetos têm seus "*slots*" criados de acordo com as propriedades das classes e seus valores preenchidos através do *Wizard*.

O modelo de cenas (Figura 6.8) instanciado representa o fluxo passando pela cena *InfoAnswer*, ao invés de ir direto para a cena de resultado. Neste sentido, a cena *InfoAnswer* possui uma variabilidade de multiplicidade e assim três cenas são criadas, uma para cada resposta. Além de criar os estados, as transições são mantidas como descritas no template. A cena de resultado permaneceu por ser um elemento comum entre as instancia do template, logo quando o voto é computado a cena de resultado é apresentada.

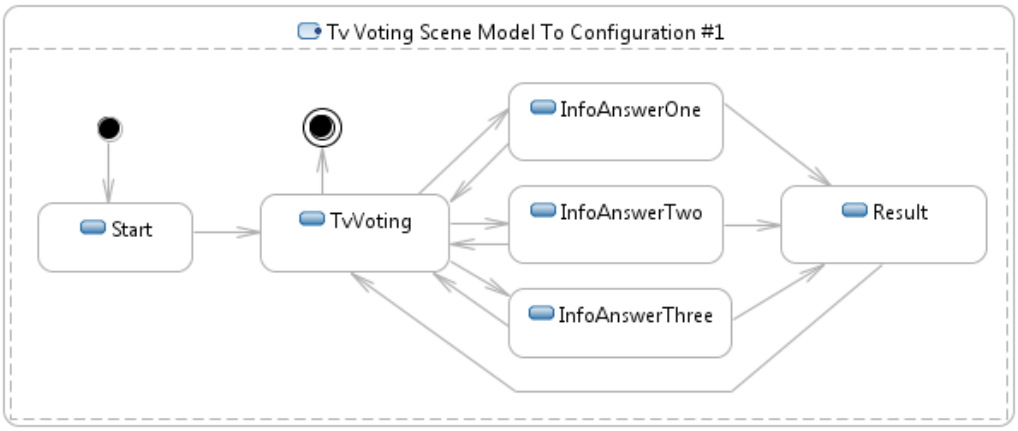


Figura 6.8: Modelo Instanciado de cenas da *TV Voting*

6.1.1.4 Passo 4 – Geração da aplicação

Na geração da aplicação é disponibilizado um conjunto de formulários de aplicações (*Wizard*) que guia o usuário através de uma série de passos, para que o preenchimento das informações pendentes dos *templates* (variabilidades) aconteça e a aplicação seja gerada. O objetivo é prover uma interface de alto nível para o desenvolvedor que deseje criar e personalizar aplicações definidas utilizando a abordagem. A seguir, uma descrição dos formulários do *Wizard* empregados para geração de versões da aplicação *TV Voting*.

O *Wizard* é composto por três passos, sendo cada passo um formulário diferente. No primeiro passo o engenheiro da aplicação escolhe a aplicação a ser gerada e nos passos seguintes as configurações da aplicação. No caso do *TV Voting* o próximo passo é o formulário da Figura 6.9 e em seguida o da Figura 6.10.

No formulário, na Figura 6.9, apresenta as variações das características estruturais. Por exemplo, é informada a quantidade de respostas e a escolha de ter ou não tela de informação. A tecnologia do canal de retorno e o formato da questão são representados com caixa de seleção que tratam apenas uma escolha entre as opções presentes no painel. No formulário da Figura 6.10 apresenta o preenchimento dos valores para as características valoradas, por exemplo, informar os valores textuais da pergunta e das respostas e as imagens correspondentes a cada tela.

A última ação é acionar a geração, que vai utilizar a configuração recém criada e o *Modelo Template* para gerar o *Modelo Instanciado* que representa uma instanciação da aplicação.

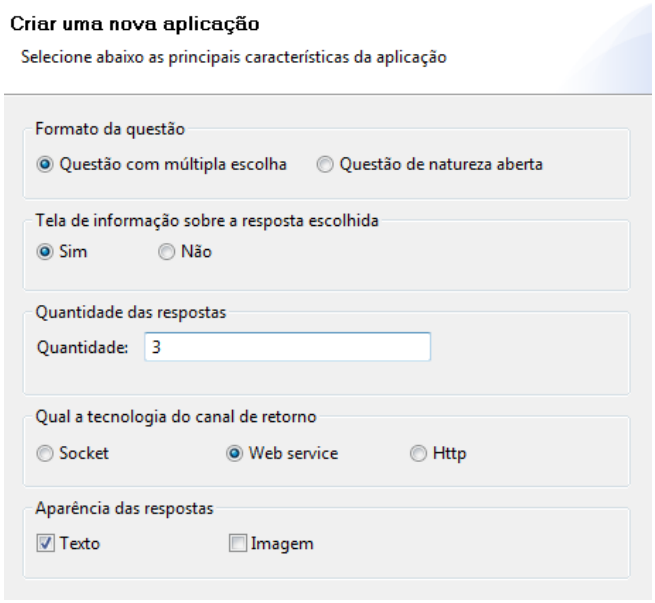


Figura 6.9: Formulário para configuração das características estruturais da *TV Voting*

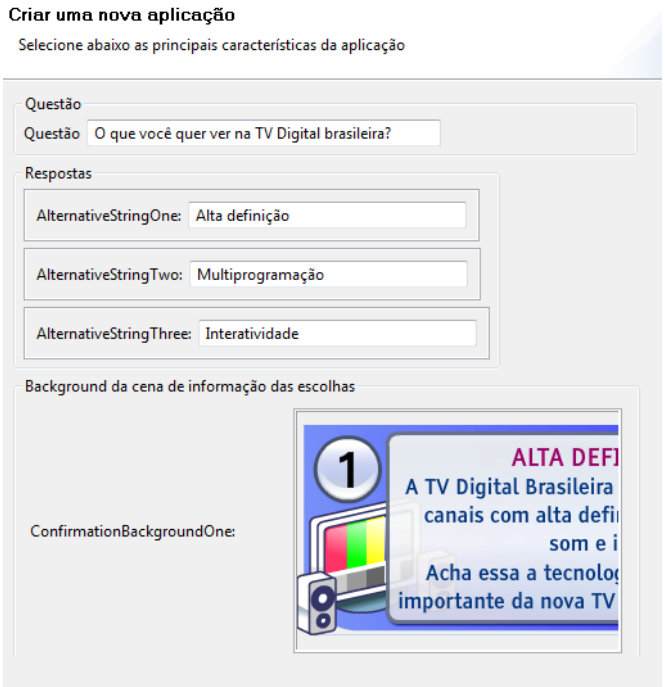


Figura 6.10: *Wizard* com configuração das características de valores da TV *Voting*

A geração de código utiliza o framework do EMF. Neste caso, a transformação recebe como entrada os modelos estruturais e de cenas. Para cada estado do modelo de cenas é realizado um mapeamento para o código NCL correspondente. Internamente a cada cena é necessário gerar a interface gráfica através do modelo de apresentação referente à cena e a especificação da interação através do modelo de interação.

6.1.1.5 *Passo 5 - Integração com Mídia Imperativa Lua*

A Figura 6.11: apresenta a visão estrutural referente à cena de votação. O envio da opção para o serviço é feita quando a tecla verde do controle remoto for pressionada para a mídia de imagem `aButton`. Após a tecla verde ser pressionada, o processo de votação é realizado pela mídia de serviço `pollVoting`. Esta mídia possui um processo interno responsável pelo envio da opção ao serviço.

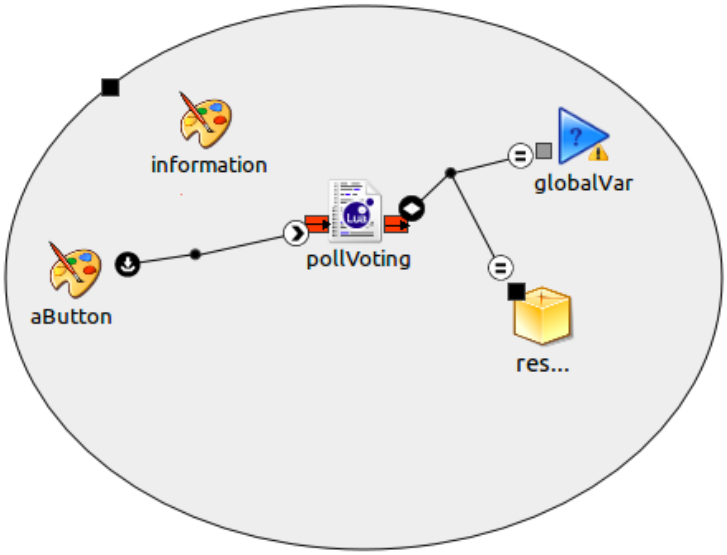


Figura 6.11: Visão estrutural do uso da mídia imperativa de votação

A votação na enquete ocorre pelo pino de entrada `sendPollOption` pertencente a mídia `pollVoting`, o qual representa o nome do método presente na mídia do serviço Web. A execução deste método ocorre através de um elo que liga a mídia de botão `aButton` e o pino de entrada `sendPollOption`. O elo criado especifica uma condição que é disparada quando o usuário pressionar a tecla verde e a partir daí uma ação é executada, representando uma chamada de método sobre o pino de entrada. Este método corresponde ao envio do voto pela mídia do serviço Web. Para criar o elo desta ligação o usuário precisou informar apenas a condição, já que a ação a ser feita (chamada de método sobre o pino de entrada) é determinada automaticamente.

A Figura 6.12 apresenta o código Lua da mídia de serviço Web bem como a anotação `@Method`. O método `sendPollOption` (linha 5) é anotado com o nome do método (`sendPollOption`), os parâmetros (`idPoll` e `option`) e o nome do evento de saída (`resultReceived`). A implementação do método chama o método do serviço responsável pelo envio do voto utilizando o módulo cliente denominado `PollServiceComunication` e o seu método `sendOption` (linha 7). O serviço Web testado encontra-se no endereço a3tv.lavid.ufpb.br.

Quando a resposta do método do serviço Web chega, o método `getResponse` (linha 11) é executado e neste momento o método `returnResultReceived`, o qual encapsula o envio do resultado recebido, é chamado para enviar o resultado como um evento. Com o envio do evento o resultado

é deixado disponível no pino de saída `resultReceived` da mídia `pollVoting`. Assim o resultado, proveniente do serviço, pode ser acessado por outras mídias através do evento `onValueReturned`.

```
require 'lib/PollServiceComunication/pollServiceComunication'

--@Method(name="sendPollOption", params="idPoll,option",
--        outputEvent="resultReceived")
function sendPollOption(idPoll, option)
    pollService = PollServiceComunication("a3tv.lavid.ufpb.br", "8080")
    pollService:sendOption(idPoll, option, getResponse)
end

--Método para enviar o resultado como um evento de saída
function getResponse(result)
    returnResultReceived(result)
end

function returnResultReceived(result)
    local outputEvent = {
        class = 'ncl',
        type  = 'attribution',
        name  = 'resultReceived',
    }
    --sinaliza app NCL de que o resultado foi recebido
    outputEvent.value = result
    outputEvent.action = 'start'; event.post(outputEvent)
    outputEvent.action = 'stop';  event.post(outputEvent)
end

function eventHandler(evt)
    ...
end

event.register(eventHandler)
```

Figura 6.12: Código Lua da mídia imperativa de votação

6.1.2 Exemplo de uso 2: Slide-show

A segunda aplicação definida é conhecida como *Slide-show*. Esta aplicação tem como objetivo exibir uma coleção de slides (objeto de mídia com imagem e/ou texto) que pode ter relação temporal com outros objetos de mídia ou responder a eventos do usuário, por exemplo, mover de slide para outro. A origem dos slides pode ser do tipo local (enviado pela emissora), remoto (recuperado por uso do protocolo HTTP) ou distribuído (oferecido pelo acesso de algum serviço Web, por exemplo, *Flickr* ou *Picasa*).

A Figura 6.13 exibe uma versão da aplicação proposta que é composta por 4 cenas obrigatórias: *Cena 1*: título do slide-show (inicia interação); *Cena 2, Cena3 e Cena 4*: são exibidas numa sequência controlada por um intervalo de tempo fixo (definido pela emissora) ou por meio da interação do usuário com setas direcionais (ou setas de navegação). Quando uma dessas três cenas é carregada, caso o usuário aperte o botão `Enter` do controle remoto, a aplicação exibe uma informação complementar sobre a imagem sendo exibida. A origem e quantidade de fotos podem variar de acordo com o que for escolhido do Modelo de *Features*. A configuração da quantidade de cenas possíveis numa aplicação deve ser definida ainda na especificação do *Modelo Template*.

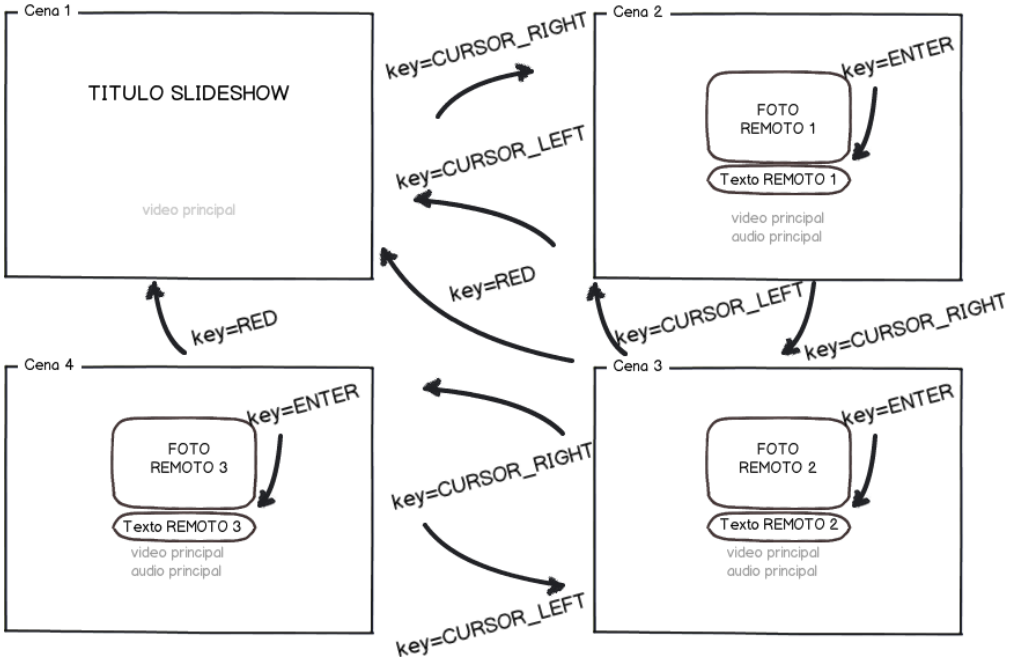


Figura 6.13: *Storyboard* da aplicação *Slide-show*

6.1.2.1 Passo 1 – Especificação do Modelo de Features

O *Modelo de Features* que descreve as possíveis soluções do Modelo *Template Slide-Show* é apresentado na Figura 6.14. Nela é possível observar o conjunto de características relacionadas aos aspectos estruturais (`Structural`) e aos valores das instâncias dos *templates* (`DataInstantiation`). Neste exemplo as características valoradas são adicionadas no modelo estrutural como filhos das características estruturais. O modelo `Structural` define, além de outras opções variáveis, a característica alternativa `Transition`, que indica as possibilidades do *Slide-show* em exibir as imagens com transição manual, isto é, pela interação do telespectador, ou com transição de acordo com um tempo especifico (transição automática). A exibição das imagens pode voltar novamente para a primeira imagem (circular) ou não, ou seja, parar na última imagem (característica `ListCircular`). As imagens do *Slide-show* podem se encontrar localmente (característica`LocalImages`), isto é, as imagens são enviadas juntamente com a aplicação, remotamente (característica`RemoteImages`) as imagens encontram-se em um servidor Web (característica `Repository`).

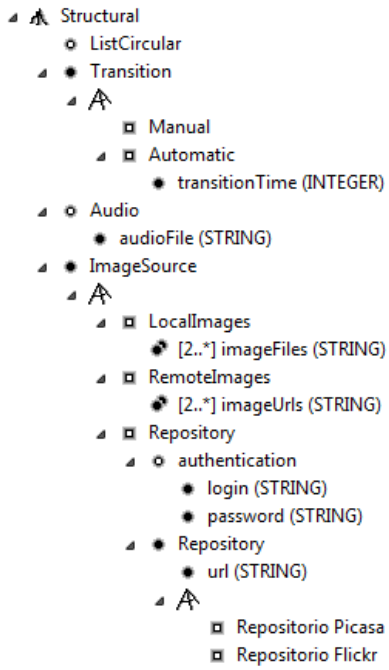


Figura 6.14: Modelo de Features da *Slide-Show*

6.1.2.2 Passo 2 - Projeto do Modelo Template e Variabilidades Associadas

Na aplicação *Slide-Show*, o modelo estrutural (Figura 6.15) contém uma classe (`SlideShow`), que representa propriedades gerais da aplicação, tais como: o tempo de transição (`transitionTime`), o caminho do áudio (`audio`), entre outros. Também possui uma associação com um repositório. O repositório é uma classe abstrata que é estendida por classes para cada tipo de repositório: local, remoto e dinâmico. Para cada uma destas classes aplica-se o seu respectivo estereótipo com modelando assim uma variabilidade alternativa para o tipo de repositório. O repositório possui um `DataSource`, que é responsável por armazenar os objetos relacionados as imagens da aplicação.

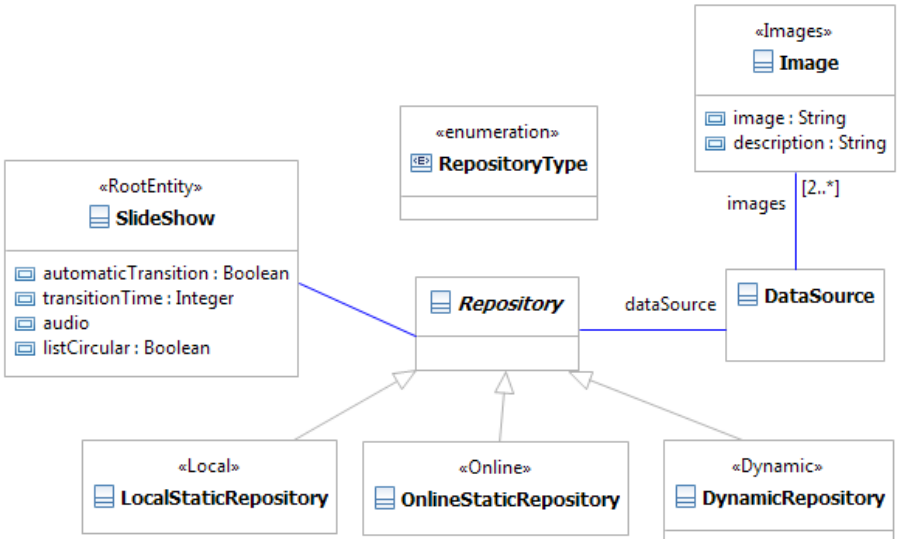


Figura 6.15: Modelo estrutural da *Slide-Show*

Continuando com a modelagem, o modelo de cenas deve especificar o comportamento geral da aplicação através de uma máquina de estados com um conjunto de estados (`cenas`) e de transições entre eles. A Figura 6.16 apresenta o *template* com três estados para representar a cena com o ícone de interatividade (`Start`), a cena de apresentação das imagens (`Slides`) e a cena opcional para exibir as informações relacionadas a erros de comunicação com o canal de retorno.

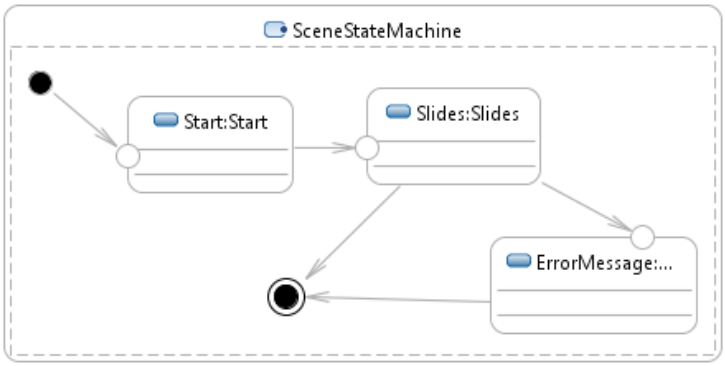


Figura 6.16: Modelo de cenas do *Slide-Show*.

O modelo de apresentação da cena de *slides* (Figura 6.17) descreve os elementos da interface gráfica através das classes `pic`, `label` e `playback`, as quais representam as imagens a serem exibidas, as descrições de cada imagem e áudio para tocar durante a apresentação. Elas são anotadas com o estereótipo `OuputComponent` de modo a representar um componente abstrato para exibição de dados.

Os valores dos componentes visuais a serem exibidos são representados tomando como base uma dependência partindo da classe para o objeto do modelo estrutural, o qual é responsável por armazenar os dados. Por exemplo, o caminho da imagem é obtido utilizando a dependência "`src`" cujo valor é a propriedade `image` especificada no objeto `dataSource`. O estereótipo `Images` (aplicado em `pic`) define a variabilidade de quantas imagens serão apresentadas durante a aplicação. O componente `pic` é duplicado para tratar as imagens do tipo local (estereótipo `Local`) e do tipo remoto (estereótipo `Online`), isto é, do servidor Web.

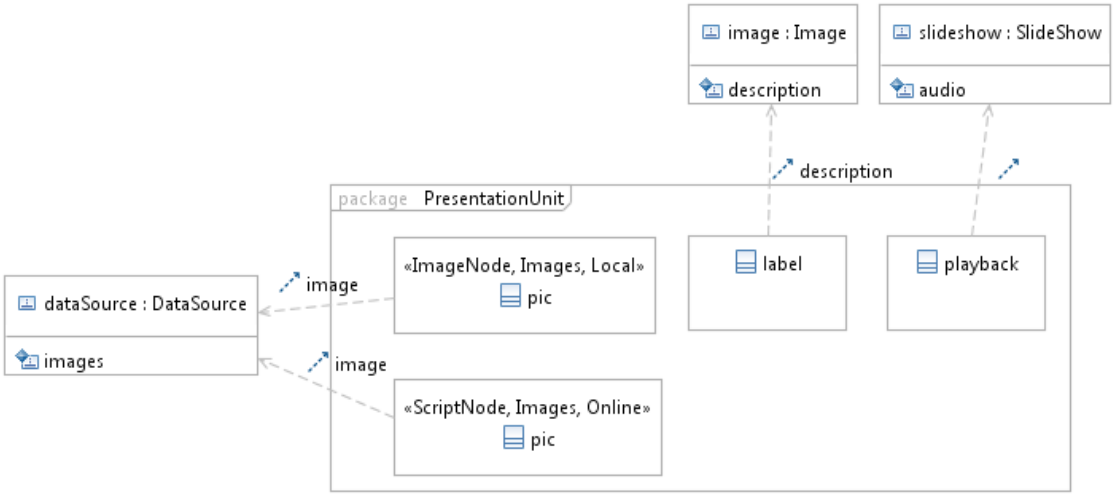


Figura 6.17: Modelo de apresentação da cena de *slides*

Para a modelagem da interação, seguimos o mesmo princípio já apresentado no exemplo anterior. A Figura 6.18 apresenta o modelo de interação para a cena `Slides`. Este diagrama especifica a apresentação encadeada das imagens presentes no slide-show. Primeiramente o componente `pic1` é apresentado por um tempo específico, quando este tempo termina, a imagem seguinte é iniciada. Este processo continua para quantas imagens existirem. O estereótipo de multiplicidade `Images` é aplicado para os componentes `pic1` e `picN` indicando uma replicação da estrutura de terminar um componente e começar outro para quantas imagens forem necessárias.

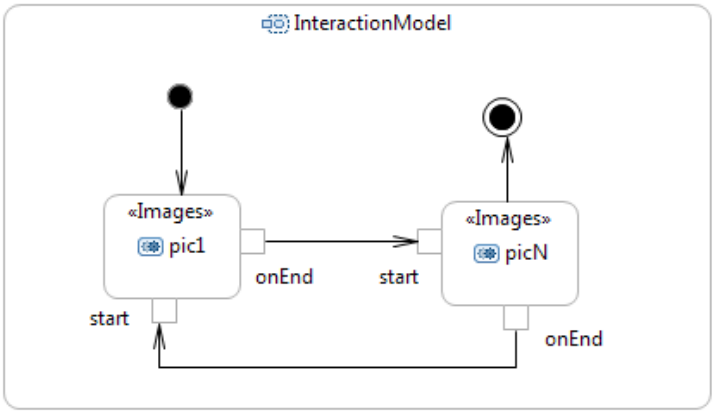


Figura 6.18: Modelo de interação da cena slides.

6.1.2.3 Passo 3 – Instanciação dos Modelos Abstratos de Templates

A instância do modelo estrutural da Figura 6.19 é obtida pela transformação do modelo estrutural no diagrama de objetos. Assim, cada classe é mapeada para um objeto. O *slide-show* foi instanciado com transição automática (`automaticTransition` com valor *true*), sem apresentação circular (`listCircular` com valor `false`) e tempo de transição de 8 segundos (*transitionTime* com valor *8*). O *slide-show* escolhido foi do tipo `Remote`, o qual instanciou um objeto do tipo `OnlineStaticRe-`

pository para ser o repositório do slideshow. Duas imagens (objetos images0 e images1) foram escolhidas e suas URLs são preenchidas pela configuração.

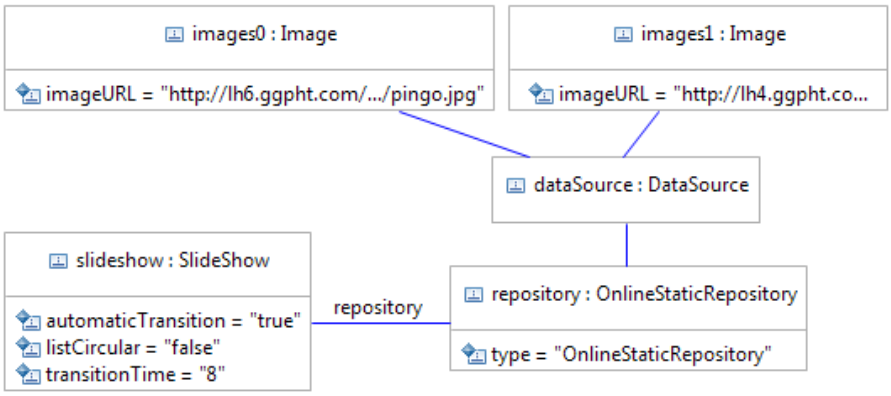


Figura 6.19: Instância do modelo estrutural da *Slide-Show*

A instância do modelo de interação (Figura 6.20) representa o fluxo passando pela primeira imagem e em seguida iniciando a próxima imagem pic2. Este diagrama foi gerado para duas imagens.

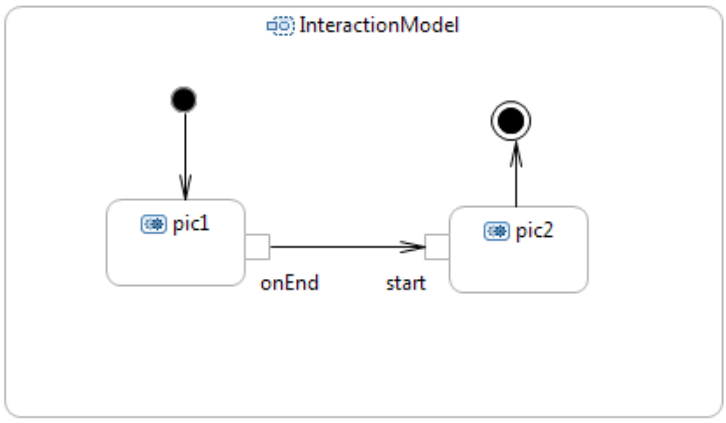


Figura 6.20: Instância do modelo de interação da cena *Slides*

6.1.2.4 Passo 4 – Geração da aplicação

Na geração da aplicação é disponibilizado um conjunto de formulários de aplicações (*Wizard*) que guia o usuário através de uma série de passos, para que o preenchimento das informações pendentes dos *templates* (variabilidades) aconteça e a aplicação seja gerada. O objetivo é prover uma interface de alto nível para o desenvolvedor que deseje criar e personalizar aplicações definidas utilizando a abordagem. A seguir, uma descrição do formulário do *Wizard* empregado para geração de versões da aplicação *Slide-Show*.

O *Wizard* é composto por um passo. Neste passo o engenheiro da aplicação escolhe as configurações da aplicação. No formulário da Figura 6.21 apresenta as variações das características. Por

exemplo, é informado que a transição é automática com o tempo de 8 segundos, o tipo remoto com as URL's das imagens preenchidas no campo de texto.

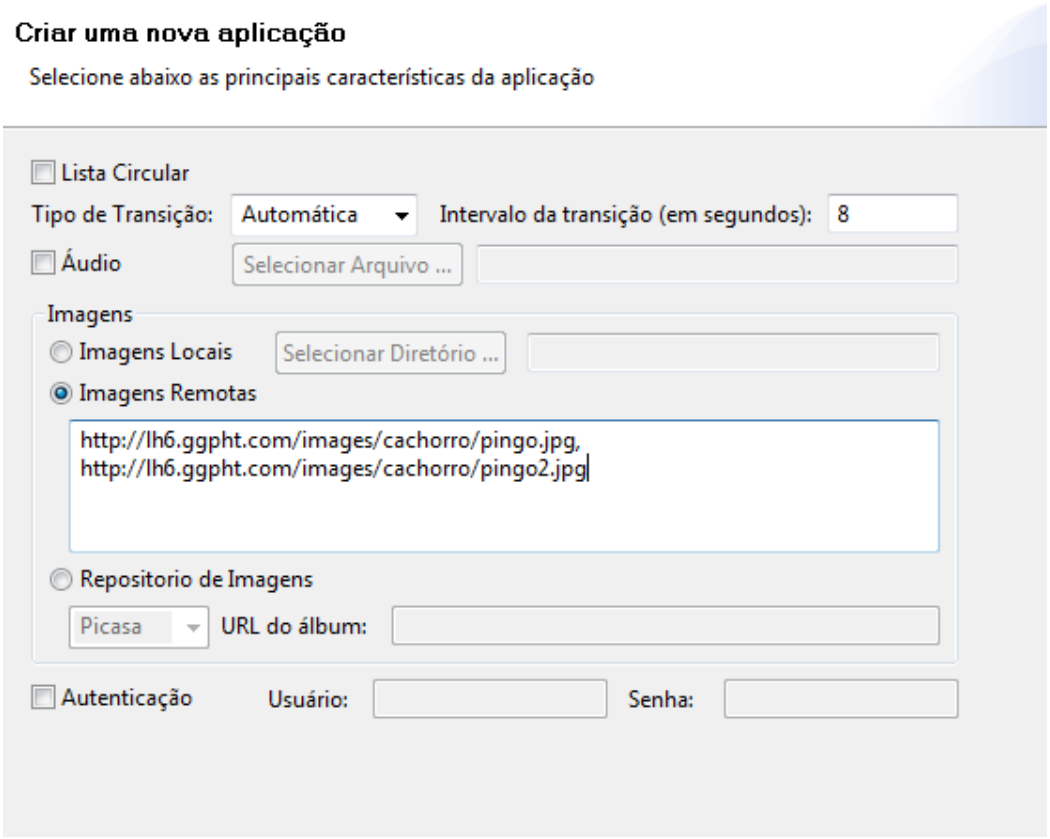


Figura 6.21: Formulário para configuração das características

A última ação é acionar a geração, que vai utilizar a configuração recém criada e o *Modelo Template* para gerar o *Modelo Instanciado* que representa uma instanciação da aplicação.

A geração de código utiliza o framework do EMF. Neste caso, a transformação recebe como entrada os modelos estruturais e de cenas. Para cada estado do modelo de cenas é realizado um mapeamento para o código NCL correspondente. Internamente a cada cena é necessário gerar a interface gráfica através do modelo de apresentação referente à cena e a especificação da interação através o modelo de interação.

6.1.2.5 Passo 5 - Integração com Mídia Imperativa

A Figura 6.22 apresenta a visão estrutural da aplicação Slide-Show. O inicio da apresentação das várias imagens em um intervalo de tempo é feita quando a mídia de áudio `audio` é iniciada. Após o inicio da mídia de áudio, ocorre a exibição da primeira imagem `picture1`. Cada imagem é tratada como uma mídia imperativa Lua responsável por exibir uma imagem a partir de uma URL, a qual pode se referir a um endereço na Web ou a um arquivo local. A exibição de cada imagem é feita através da ação de chamada de método sobre o pino de entrada `showImage` (representado pelas caixas vermelhas com setas em cada mídia imperativa na Figura 6.22) Esta ação realiza a chamada do método

`showImage` da mídia imperativa, o qual é responsável pelo acesso à imagem e o desenho da mesma na tela. Este método possui como parâmetro a URL da imagem a ser apresentada. Quando uma imagem é terminada a próxima imagem é exibida, seguindo o mesmo processo até a última imagem `picture3`.

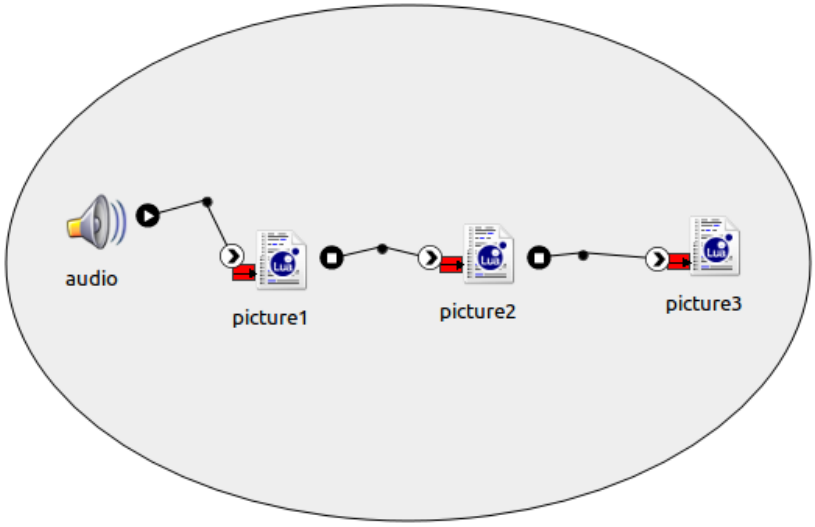


Figura 6.22: Visão estrutural do uso da mídia imperativa para apresentar uma imagem.

A Figura 6.23 apresenta o código Lua da mídia de serviço Web bem como a anotação `@Method`. O método `showImage` (linha 4) é anotado com o nome do método (`showImage`), um parâmetro (`URL`). A implementação do método chama o módulo `image` responsável por exibir a imagem na tela (linha 6).

```
1  local image = require 'image'
2
3  --@Method(name="showImage", params="URL")
4  function showImage(URL)
5      image.setURL(path)
6      image.showImageFromURL()
7  end
```

Figura 6.23: Código-fonte da mídia imperativa `image`

6.1.3 Exemplo de uso 3: Weather TV

A terceira aplicação modelada consiste numa aplicação com informações sobre o tempo de uma lista de cidades. O tempo é exibido com o valor numérico da temperatura e também por um ícone que indica o estado atual do clima. Assim como a aplicação de Slide-show, a origem dos valores das temperaturas pode ser do tipo local (enviado pela emissora), remoto (recuperado por uso do protocolo HTTP e servidor controlado pela emissora normalmente) ou distribuído (oferecido pelo acesso a algum serviço, por exemplo, *Yahoo! Weather*). A quantidade de cidades pode ser definida pela emissora, mas caso a aplicação recupere os valores de origens não-locais, essa lista pode ser dinâmica.

A Figura 6.24 exibe uma versão da aplicação proposta que é composta por 3 cenas obrigatórias: *Cena 1*: ícone para iniciar interação, se usuário selecionar a tecla azul (BLUE) aplicação vai para

a Cena 2; *Cena 2:*aplicação exibe um menu com a lista de cidades disponíveis. Caso o usuário selecione a tela vermelha (RED), a aplicação volta para a cena inicial. *Cena 3:* aplicação acessa a opção (cidade) escolhida na Cena 2 e consulta a origem associada, recuperando o valor solicitado da temperatura e exibindo o ícone de acordo com uma regra pré-definidas. Na Cena 3o usuário tem a opção de voltar para a Cena 2, pela tecla verde (GREEN) ou voltar para a Cena 1, pela tecla vermelho (RED).

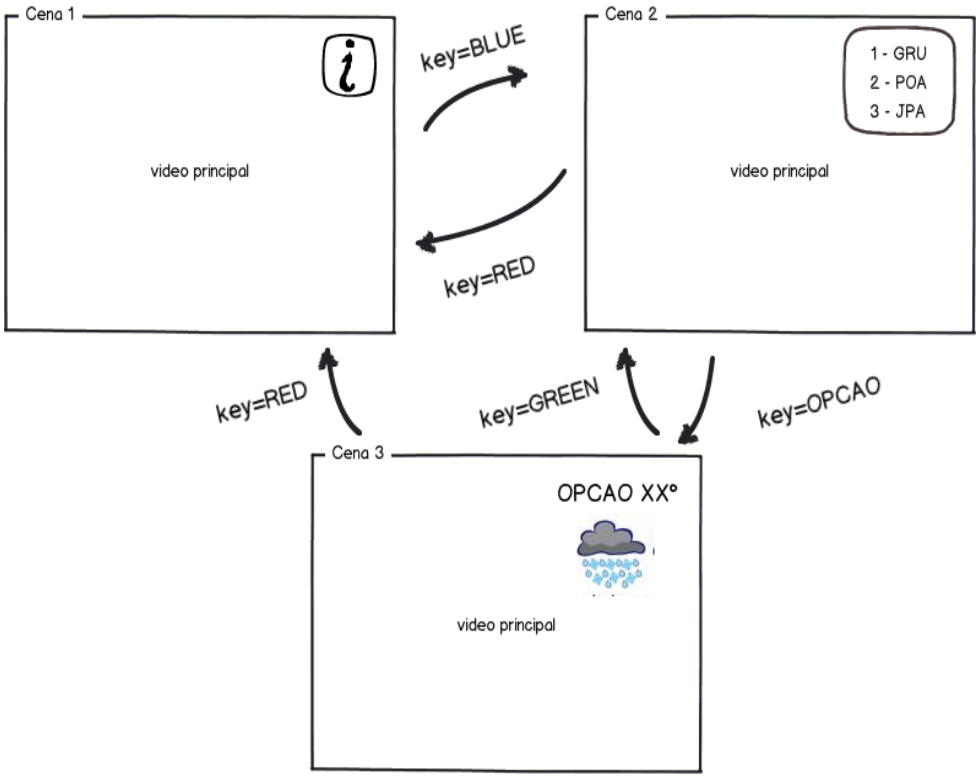


Figura 6.24: Storyboard para a aplicação *Weather TV*

6.1.3.1 Passo 1 – Especificação do Modelo de Features

O *Modelo de Features* do template para o *Weather*TV é apresentado na Figura 6.25. O conjunto de características relacionadas aos aspectos estruturais (`Structural`) e aos valores das instâncias dos *templates* (`DataInstantiation`) são exibidos no ponto 1 e no ponto 2, respectivamente. O modelo define, além de outras opções de variações, a característica alternativa `TemperatureUnit`, que oferece duas escolhas (entre Celsius e Fahrenheit) para a unidade de temperatura da aplicação. A opção de existir ou não localização geográfica é tratada pela característica opcional `GeoLocalization`. A característica `WeatherSource` representa três modos alternativos de distribuição da aplicação: na forma local (característica `Local`) a aplicação exibe estaticamente o clima apenas das capitais brasileiras, no modo remoto e serviço (características `Remote` e `Service`) a aplicação de clima apresenta em tempo real (a partir de um servidor) o tempo da cidade do telespectador ou de outra cidade próxima a ele. A característica valorada `NumberOfCities` representa o número para escolha de cidades próximas ao telespectador. Por padrão esse numero é de quatro cidades.

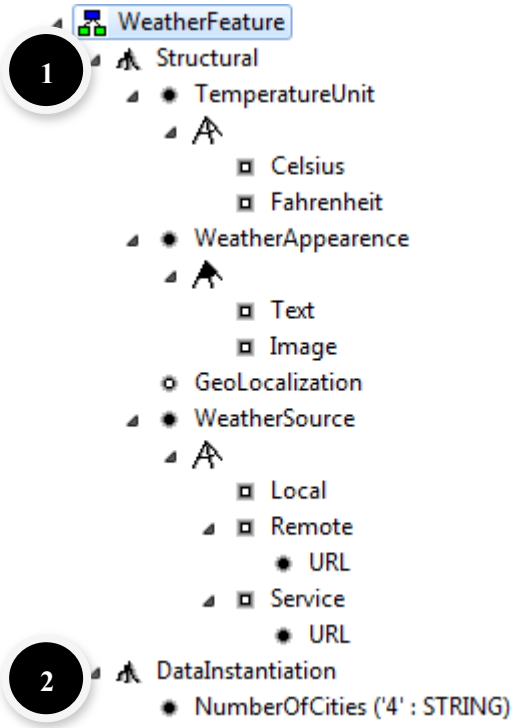


Figura 6.25: Modelo de Features da *Weather TV*.

6.1.3.2 Passo 2 - Projeto do Modelo Template e Variabilidades Associadas

Seguindo os mesmos passos dos exemplos anteriores, o modelo estrutural (ver Figura 6.26) da aplicação *Weather TV* contém uma classe para a aplicação (`Weather`), a qual possui associações com um ou mais fornecedores de condições do tempo (`WeatherCondition`).

Ainda nesse modelo, ficam explícitos os elementos do domínio associados aos elementos de mídia `TemperatureLabel`, `ConditionImage` e `CityLabel`, respectivamente, um objeto de mídia de texto para exibir a temperatura, um objeto de mídia de imagem para a condição do tempo (chuvoso, ensolarado, nublado, etc.) e um objeto de mídia textual para apresentar a cidade do clima atual. Cada classe desta possui um estereótipo de acordo com o seu tipo, são eles `Text` e `Image`. As classes deste modelo não possuem variações, logo não foi aplicada nenhuma variabilidade.

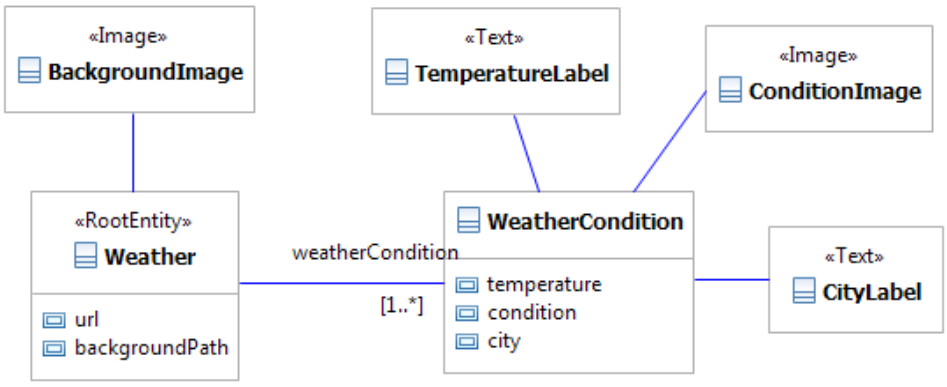


Figura 6.26: Modelo estrutural do *Weather TV*

Continuando com a modelagem, o modelo de cenas deve especificar o comportamento geral da aplicação através de uma máquina de estados com um conjunto de estados (cenas) e de transições entre eles. A Figura 6.27 apresenta o *template* com três estados para representar a cena com o ícone de interatividade (Start), a cena para escolha da cidade (`ChooseCity`) e por fim a cena para exibir o clima tempo da cidade escolhida (`CityWeather`). A cena `ChooseCity` apresenta um conjunto de cidades para o telespectador escolher uma cidade para exibir as suas informações de tempo na cena `CityWeather`. Caso seja utilizado o esquema de localização, a cidade é escolhida automaticamente e repassada para a cena seguinte.

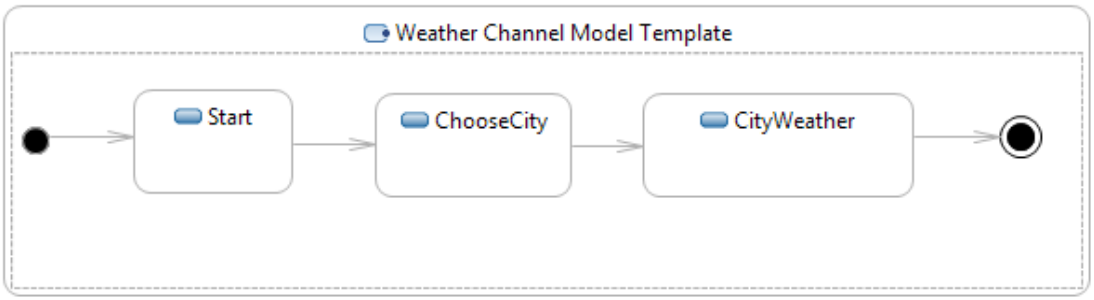


Figura 6.27: Modelo de cenas da *Weather TV*

O modelo de apresentação da cena (Figura 6.28) de condição do tempo para uma cidade descreve os elementos da interface gráfica através das classes `cityLabel`, `temperatureLabel` e b*ackground*, as quais representam respectivamente os textos da cidade, da temperatura e uma imagem para compor o fundo da cena. Além destes `OutputComponent`, o diagrama possui os OC dos vários tipos de condição do tempo como chuvoso (`rainyCondition`), ensolarado (`sunnyCondition`) e nublado (`cloudyCondition`). Como na MML, as classes são anotadas com o estereótipo `Ouput-Component` de modo a representar um componente abstrato para exibição de dados.

Os valores dos componentes visuais a serem exibidos são representados tomando como base uma dependência partindo da classe para o objeto do modelo estrutural, o qual é responsável por armazenar os dados. Por exemplo, o texto da cidade e da temperatura é obtido utilizando as dependências `temperature` e `city` com destino para o objeto `weatherCondition` relativo a classe `WeatherCondition` do modelo estrutural. Junto com as propriedades do objeto e o nome do relacionamento o conteúdo para os OC pode ser determinado. No diagrama evidencia a presença de dois OC's para a temperatura em Celsius ou em Fahrenheit, possuindo em ambos a aplicação dos estereótipos Celsius e Fahrenheit, respectivamente.

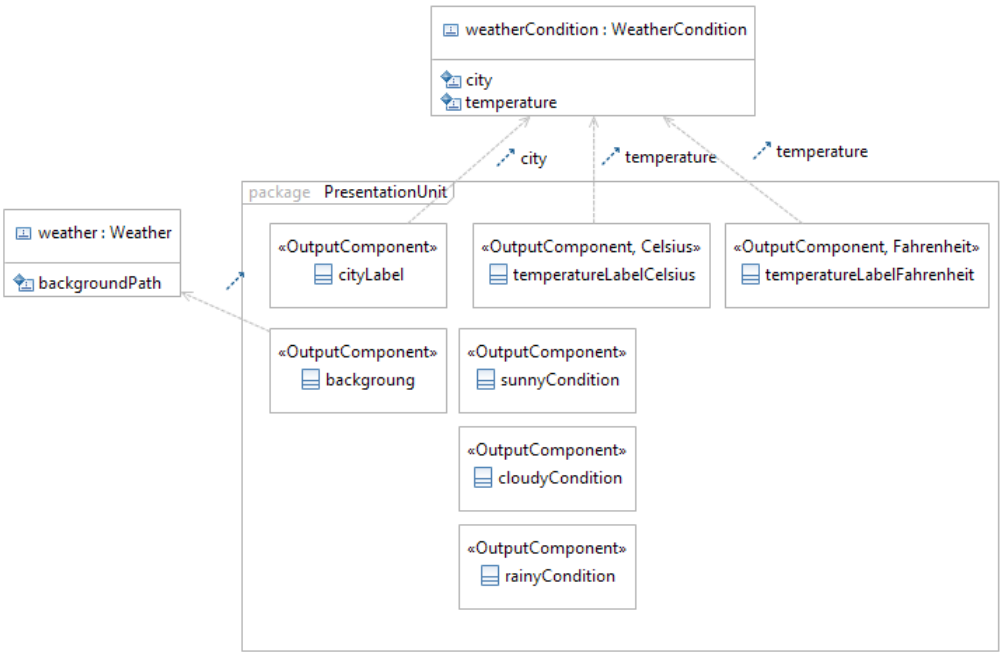


Figura 6.28: Modelo de apresentação da cena de condição do tempo *WeatherCondition*

Seguindo os mesmos princípios da modelagem das outras aplicações já descritas neste capítulo, a Figura 6.29 apresenta o modelo de interação para a cena que exibe o clima da cidade. O ponto 1 trata a inicialização do componente de *background* (apresentação dos objetos de mídia), representando os fluxos existentes entre o parâmetro da atividade `showCityWeathere` os pinos de entrada `start` do componente `background`. O fluxo entre o parâmetro da atividade e o pino de entrada `getCityWeather` do componente `WeatherServiceClient` permite realizar a chamada ao serviço Web para obter as informações do clima relativo a cidade. Quando a resposta é recebida do servidor cada valor é disponível nos pinos de saída `weatherCondition`, `temperature` e `city` do componente `WeatherServiceClient`. O condição (ponto 2) permite `iniciar` os componentes das imagens de cada condição do tempo condicionalmente associado a um determinado valor, isto é, dependendo do valor retornado pelo pino de saída `weatherCondition` uma imagem da condição do tempo respectiva é iniciada. Se o valor retornado for *sunny* o componente `sunnyCondition` é iniciado, bem como se for `cloudy` o componente `cloudyCondition`, e assim por diante. Os valores da temperatura e da cidade são exibidos respectivamente pelos pinos de entrada `text` pertencente aos componentes de texto `temperatureLabel` e `cityLabel`. Para o valor ser exibido, existe um fluxo entre os pinos de saída e o pio de entrada de texto destes componentes.

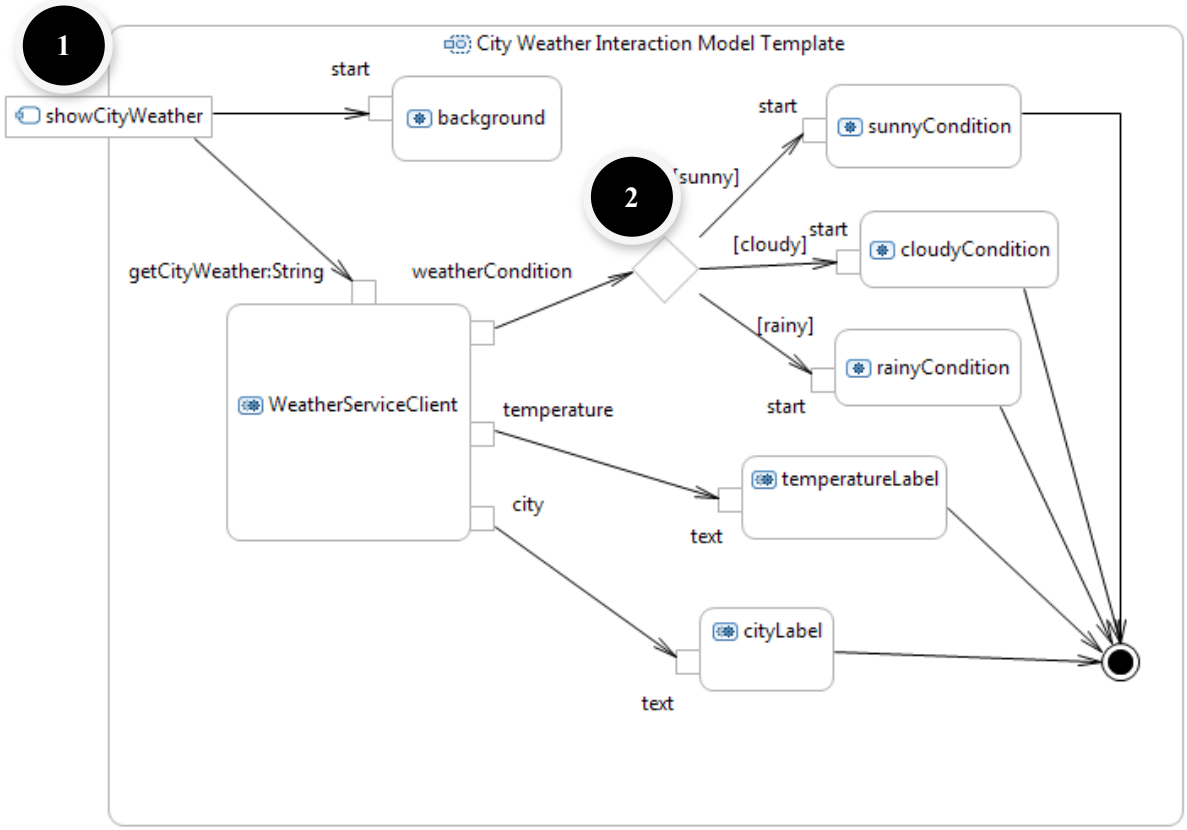


Figura 6.29: Modelo de interação da cena do clima da cidade

6.1.3.3 Passo 3 – Instanciação do Modelo Template

A instanciação do *Modelo Template* foi realizada para uma aplicação de clima tempo local, o qual possui todo o seu conteúdo determinado pela configuração. A Figura 6.30 mostra que foram escolhidas três cidades próximas. Os objetos têm seus "*slots*" criados de acordo com as propriedades das classes e seus valores preenchidos através do *Wizard*.

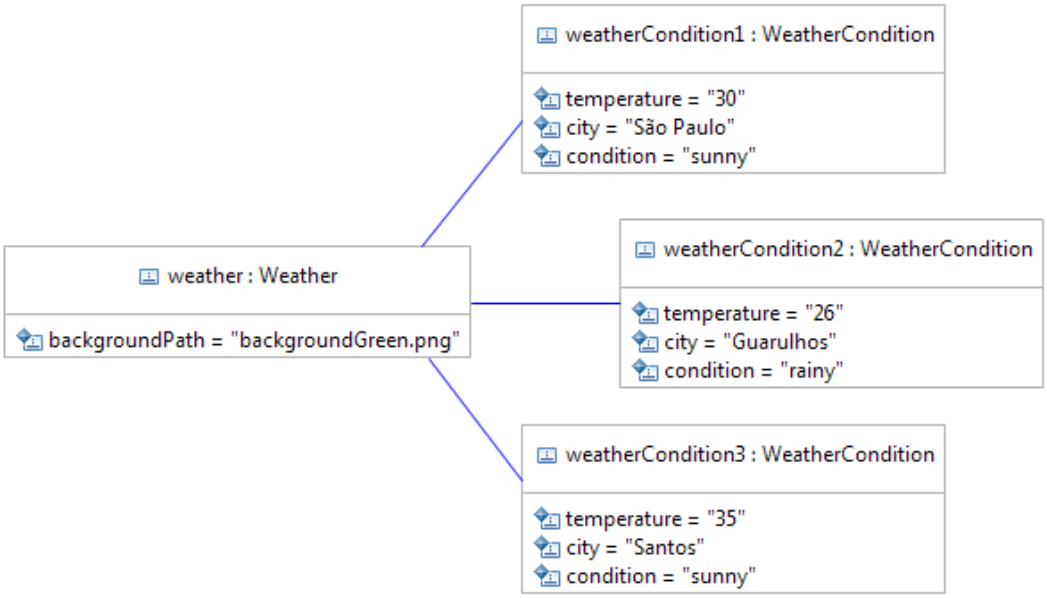


Figura 6.30: Instância do modelo estrutural da *Weather TV*

6.1.3.4 Passo 4 – Geração da aplicação

Na geração da aplicação é disponibilizado um conjunto de formulários de aplicações (*Wizard*) que guia o usuário através de uma série de passos, para que o preenchimento das informações pendentes dos *templates* (variabilidades) aconteça e a aplicação seja gerada. O objetivo é prover uma interface de alto nível para o desenvolvedor que deseje criar e personalizar aplicações definidas utilizando a abordagem. A seguir, uma descrição dos formulários do *Wizard* empregados para geração de versões da aplicação *Weather TV*.

Neste caso, o *Wizard* é composto por um passo. Neste passo o engenheiro da aplicação escolhe as características estruturais e os valores de instanciação. A Figura 6.31 apresenta o formulário relativo às variações das características do `weather`. Por exemplo, é possível escolher entre a temperatura ser em Celsius ou Fahrenheit, escolher a aparência da aplicação entre texto ou imagem e ambos e se aplicação será local, remota ou com o uso do serviço Web.

A última ação é acionar a geração, que vai utilizar a configuração recém criada e o *Modelo Template* para gerar o *Modelo Instanciado* que representa uma instanciação da aplicação.

A geração de código utiliza o framework do EMF. Neste caso, a transformação recebe como entrada os modelos estruturais e de cenas. Para cada estado do modelo de cenas é realizado um mapeamento para o código NCL correspondente. Internamente a cada cena é necessário gerar a interface gráfica através do modelo de apresentação referente à cena e a especificação da interação por intermédio do modelo de interação.

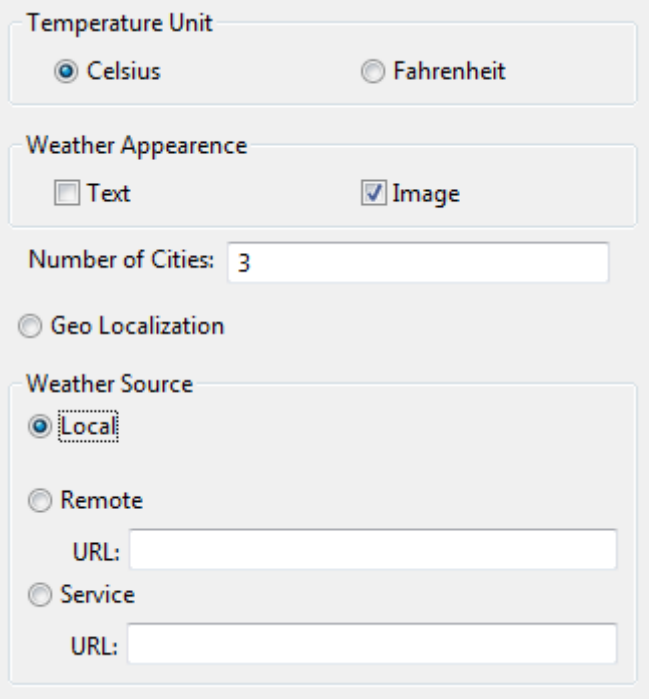


Figura 6.31: Formulário para configuração das características da aplicação

6.1.3.5 Passo 5 - Integração com Mídia Imperativa Lua

A Figura 6.32 apresenta a visão estrutural da aplicação de clima tempo. Quando a imagem de fundo `background` é iniciada, é disparada uma ação de chamada de método sobre o pino de entrada `getcityWeather` da mídia de serviço Web `weatherClient`. Em outros termos, esta ação realiza uma requisição ao serviço Web para consultar o clima e obter a resposta contendo as informações do clima da cidade.

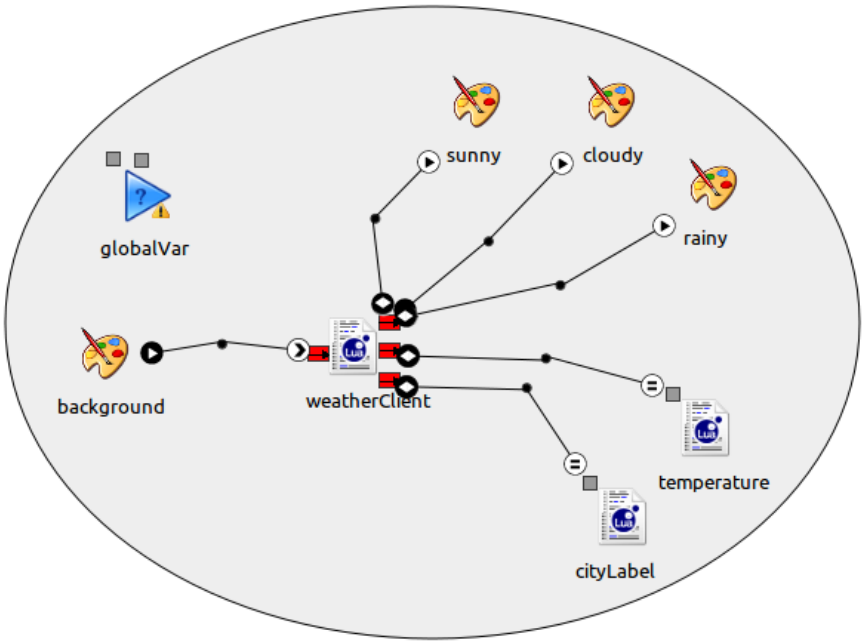


Figura 6.32: Visão estrutural da aplicação instanciada *Weather TV*

Além do pino de entrada (mapeado para o único método do serviço `getWeather`), na figura a mídia `weatherClient` possui outros três pinos de saída. Como estamos interessados em mais de uma informação presente na resposta (condição do tempo, temperatura e cidade), este pino de saída enquadra-se no segundo tipo de pino de saída. Dessa forma, o pino de saída da resposta é desmembrado em outros três pinos de saída, sendo um para tratar o recebimento da condição do tempo, da temperatura e outro da cidade.

A Figura 6.33 apresenta o código Lua da mídia de serviço Web bem como a anotação `@Method`. O método `getCityWeather` (linha 5) é anotado com o nome do método (`getCityWeather`), dois parâmetros (`woeid` e `unity`) e três eventos de saída (`weatherCondition`, `temperature` e `city`). A implementação do método chama o método do serviço responsável pela consulta do clima da cidade utilizando o módulo cliente denominado `weatherServiceClient` e o seu método `getCityWeather` (linha 7). O serviço Web utilizado é do *Yahoo! Weather*.

Quando a resposta do método do serviço Web chega, o método `getResponse` (linha 10) e o mesmo extrai as informações de interesse (condição do tempo, temperatura e `cidade`) da resposta e envia o evento de saída com o valor. No código foi exemplificado apenas o tratamento da condição do tempo através do método `returnWeatherCondition` (linha 19). Este método, que encapsula o valor da condição do tempo, é chamado para enviar a condição do tempo como um evento. Com o envio do evento o valor vindo do serviço é deixado disponível no pino de saída `weatherCondition` da mídia `weatherClient`. Assim o valor da condição do tempo, pode ser acessado por outras mídias através do evento `onValueReturned`.

```
1  require 'lib/WeatherServiceClient/weatherServiceClient'
2
3  --@Method(name="getCityWeather", params="woeid,unity",
4  --         outputEvent="weatherCondition,temperature,city")
5  function getCityWeather(woeid, unity)
6      weatherClient = WeatherServiceClient("http://weather.yahooapis.com/forecastrss",
7                                           "80")
8      weatherClient:getCityWeather(woeid, unity, getResponse)
9  end
10
11 function getResponse(response)
12     ...
13     local weatherCondition = getWeatherConditionFromResponse(response)
14     ...
15
16     returnWeatherCondition(weatherCondition)
17     ...
18 end
19
20 function returnWeatherCondition(condition)
21     local outputEvent = {
22         class = 'ncl',
23         type  = 'attribution',
24         name  = 'weatherCondition',
25     }
26     --sinaliza app NCL de que o resultado foi recebido
27     outputEvent.value = condition
28     outputEvent.action = 'start'; event.post(outputEvent)
29     outputEvent.action = 'stop';  event.post(outputEvent)
30 end
```

Figura 6.33: Código-fonte Lua associado mídia imperativa `weatherClient`

A ação de chamada de método possui um parâmetro para com o código da localidade e a unidade da temperatura que deseja retornar. Estes valores são determinados, respectivamente, a partir das propriedades `woeid` e `unity` da mídia `globalVar`. O usuário realiza esta configuração pela janela de configuração de parâmetros associando cada parâmetro a sua respectiva propriedade.

A temperatura é apresentada pela mídia imperativa `label` responsável pela exibição da informação textual. Como visto, esta mídia possui um pino de entrada `text` que possui como parâmetro o texto a ser desenhado na tela. Para exibir a temperatura retornada pelo servidor, o próprio valor presente no pino de saída `temperature` é usado como valor para o parâmetro da chamada de método sobre o pino de entrada `text` da mídia `label`, em outras palavras, a chamada do procedimento para exibir o texto da temperatura na mídia `label` é feita com o valor do pino de saída `temperature` da mídia do serviço Web. Este elo é gerado automaticamente sem a interação do usuário, pois apenas transfere o valor do pino de saída para ação de chamada de método no pino de entrada.

A imagem referente à condição do tempo é apresentada de forma condicional, ou seja, para cada condição do tempo (ensolarado, chuvoso, etc.) existe uma imagem correspondente. Este exemplo usa três mídias de imagens: mídia `sunny` para tempo ensolarado, mídia `rainy` para tempo chuvoso e a mídia `cloudy` para tempo nublado. Cada mídia é apresentada dependendo do valor presente no pino de saída `weatherCondition` da mídia do serviço Web. Por exemplo, caso o valor retornado seja "`sunny`" a mídia `sunny` é apresentada.

Para isso criou-se um elo para apresentar cada mídia. Sendo ele formado por uma condição composta, referente à condição `onValueReturned` e ao valor corresponde a mídia que será apresentada,e uma ação de inicio de mídia.

6.2 Avaliação com estudos empíricos

A Engenharia de Software é uma área multidisciplinar que aborda aspectos técnicos e humanos. Nesse contexto, desenvolver pesquisas na área de engenharia de software exige a escolha de métodos científicos compatíveis com os objetivos da pesquisa (WOHLIN et al, 2000). Um dos métodos científicos mais conhecidos é o método empírico. O método empírico utiliza estudos primários baseados em evidências para avaliar modelos propostos. Segundo Wohlin et al. (2000), estudos dessa natureza são convenientes para obter resultados objetivos e estaticamente significativos para entender, controlar, prever e melhorar processos de desenvolvimento de software. Tais características são ainda mais importantes quando se propõe uma nova metodologia. Um estudo empírico pode ser realizado através de:

- ***surveys***: realizados com o objetivo de avaliar e analisar uma população, da qual uma amostra é retirada (BABBIE, 1990). Esta avaliação ocorre, geralmente, através de entrevistas, questioná-

rios e formulários. Assim, um *survey* tem objetivo como entender, explicar, explorar e descrever uma população.

- **estudos de caso**: executados para investigar um fenômeno em um intervalo de tempo específico. Este fenômeno, pode ser a adoção de uma metodologia, ferramenta ou tecnologia da Engenharia de Software. Diante disso, os estudos de caso são adequados no contexto de uma avaliação industrial (YIN, 1994).
- **experimentos**: utilizados com o objetivo de observar uma relação causa-efeito. Um exemplo de causa seria uma tecnologia e o objetivo seria observar o efeito do uso desta tecnologia no desenvolvimento de um software. Deste modo, estes elementos da teoria (causa e efeito) são estruturados em elementos observáveis, ou seja, elementos do estudo experimental. Um experimento coleta métricas para comparar duas ou mais causas (também conhecidos como tratamentos) ao mesmo tempo em que controla variáveis que podem influenciar o efeito. Adicionalmente, os resultados de cada efeito são expressos em termos de nível de significância definidos por análise estatística de modo a evitar conclusões geradas por situações aleatórias.

Neste trabalho utilizamos experimentos para avaliar de forma quantitativa uma abordagem de desenvolvimento orientado a modelos no domínio de aplicações para TV Digital.

A Figura 6.34 traz uma visão geral dos principais elementos de um experimento. A ideia é observar a relação causa-efeito diante dos aspectos da teoria. Para tanto, os elementos da teoria (causa e efeito), são estruturados em elementos de observação (tratamento e resultados), ou seja, elementos de execução do estudo experimental. Esses elementos do estudo experimental são as variáveis independentes, fatores e níveis do experimento que podem ser controladas e modificadas durante o tratamento para interpretar a causa e, as variáveis dependentes, obtidas durante os resultados para interpretar o efeito.

A realização do experimento torna possível obter informações precisas de um conjunto de variáveis. Para conseguir estas informações, é imprescindível um alto nível de controle sobre o ambiente e a execução do experimento. Outro ponto importante do experimento, além do controle, é a sua capacidade de repetição, de forma que ele possa ser executado novamente, e atestar se produz os mesmos resultados.

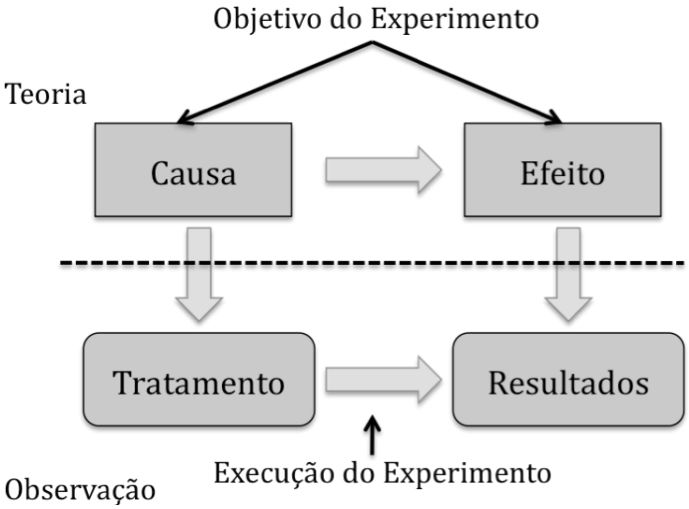


Figura 6.34: Visão Geral de um experimento baseado em Wohlin et al. (2000)

Com o objetivo de estruturar melhor os elementos dos experimentos, neste trabalho é utilizado o processo definido por Wohlin et al. (2000) para realização de estudos experimentais em 5 (cinco etapas): (i) definição: (ii) planejamento; (iii) operação do experimento; (iv) análise e interpretação dos resultados; e (v) apresentação dos resultados. O objetivo dessas etapas é estruturar os elementos importantes do experimento para permitir o foco no controle e nas variáveis significativas para a obtenção dos resultados. Os experimentos também terão sua validade avaliada através de 4 (quatro) tipos de ameaças: (conclusão, construção, interna e externa). Nas próximas seções são descritas as características comuns das fases de definição, planejamento, avaliação de validade e projeto experimental para os 3 (três) experimentos realizados neste estudo. Após isso, cada experimento é detalhado individualmente.

6.2.1 Definição dos experimentos.

O objetivo geral dos experimentos realizados nesse trabalho é avaliar o uso de uma abordagem de desenvolvimento orientado a modelos apoiada por algum ambiente de desenvolvimento para aplicações de TV Digital. Mais especificamente, o objetivo é avaliar o impacto do uso da abordagem por meio da medição do tempo de implementação de requisitos de uma aplicação.

De modo geral, seguindo o formato de Wohlin et al.(2000), a definição do estudo empírico deste trabalho está baseada nos seguintes conceitos-chave:

- **O objeto de estudo** são abordagens de desenvolvimento de aplicações para TV Digital que tratam a integração entre projeto de mídia e projeto de software utilizando a linguagem declarativa NCL e a linguagem imperativa Lua;
- **O propósito** é comparar as abordagens com foco no tempo de implementação de aplicações;

- **Do ponto de vista** do desenvolvedor de software que lida com a parte declarativa (documento NCL) da aplicação, ou seja, o responsável pelo projeto de mídia da aplicação, mas que necessita realizar integração com lógica de aplicação complexa, representada por um projeto de software que especifica artefatos de software na linguagem Lua.
- **No contexto** de Desenvolvimento Orientado a Modelos no domínio específico de aplicações de TV Digital.

Resumindo tais conceitos em uma única expressão, os experimentos consistem em: Analisar diferentes abordagens de desenvolvimento de software MDD ou não, com o propósito de compará-las com respeito a tempo de implementação de aplicações de TV Digital do padrão ISDB-Tb, do ponto de vista do desenvolvedor NCL que trata o projeto de mídia da aplicação.

Sob esta visão, é possível definir a seguinte hipótese geral que se deseja investigar no experimento:

"O uso de abordagens distintas traz medidas diferentes de tempo no desenvolvimento de aplicações para TV Digital?

A métrica relacionada ao tempo de implementação de uma aplicação tem como unidade de medida os segundos.

6.2.2 Planejamento dos Experimentos

6.2.2.1 Seleção de Contexto

Em relação à seleção de contexto (WOHLIN et al., 2000), os experimentos foram realizados em ambiente de laboratório para melhor controle, e dessa forma, tem caráter *off-line*. Apesar de entender que profissionais na área de TV Digital representam melhor o contexto de estudo, foram utilizados apenas estudantes como sujeitos dos experimentos por conta da facilidade de acesso. Todavia, uma vez que o estudo procura avaliar a integração do projeto de mídia e software realizado por uma linguagem declarativa NCL, a seleção dos sujeitos se preocupou em utilizar indivíduos com pouca experiência em linguagens de programação imperativa e realizou treinamentos para procurar uniformizar o conhecimento da linguagem NCL.

No que diz respeito aos problemas abordados, foram escolhidas duas aplicações que são reais e que possuem o requisito de integração com lógica de aplicação complexa, porém, de modos diferentes. Para diferenciar do termo aplicação mais geral, neste capítulo será utilizada o termo **aplicação-problema** para identificar as aplicações desenvolvidas nos experimentos. Os experimentos tratam de

um contexto específico do domínio de aplicações de TV Digital, pois, apesar de usar aplicações reais, os resultados obtidos só podem ser analisados em cenários nos quais se exige o uso de lógica de aplicação complexa em aplicações de TV Digital, característica é cada vez mais presente nesse tipo de software.

6.2.2.2 Variáveis

O tempo de implementação representa a métrica de produtividade de desenvolvimento da aplicação-problema e por isso é **variável dependente** ou variável resposta do experimento. É importante observar que as medidas de tempo de implementação em todos os experimentos devem considerar apenas as fases da Engenharia da Aplicação e não considera o esforço realizados nas fases de Engenharia de Domínio (ver Figura 5.3). Já para **variáveis independentes**, que definem como empregar as abordagens, foram definidas: (1) implementação de aplicações NCL que utilizam o código-fonte Lua apenas para implementar lógica de aplicação complexa; (2) especificação dos requisitos da aplicação-problema que será desenvolvida utilizando uma linguagem simples e precisa fazendo uso de *storyboards* e *checklist*; (3) implementação da aplicação-problema de modo iterativo e incremental com aumento da complexidade a cada iteração; e (4) utilização de ambientes de desenvolvimento de aplicações NCL e Lua que são compatíveis com o padrão ABNT definindo pelo SBTVD.

6.2.2.3 Fatores e níveis

Os experimentos apresentam um único fator, que trata da abordagem a ser utilizada. A definição dos níveis levou em consideração que os experimentos foram realizados em épocas diferentes, e por isso, as diferentes abordagens estavam disponíveis. Nos primeiros experimentos apenas a primeira parte da abordagem deste trabalho estava disponível e, por isso, foi considerado apenas o uso de uma abordagem com Prototipação Rápida e outra com uso do NCL Eclipse e NCL Composer. Na execução do último experimento já foi possível utilizar a segunda fase da abordagem (Refinamento da Aplicação) e o NCL Composer. Assim foram definidos 3 (três) níveis diferentes que foram utilizados nos experimentos de acordo com sua data de execução.A seguir seus identificadores são apresentados:

- M1: emprego das ferramentas NCL Eclipse e NCL Composer, representando o uso de uma abordagem dirigida por modelos que utiliza autoria textual e autoria visual das aplicações, mas sem nenhum suporte à estruturação de requisitos e à integração entre o projeto de mídia e projeto de software;
- M2: emprego da ferramenta NCL Eclipse e de um *plugin* que permite a geração rápida de protótipos de aplicações, representando o uso de uma abordagem dirigida por modelos que utiliza autoria textual e implementa a etapa de *Prototipação Rápida* proposta neste trabalho;
- M3: emprego da ferramenta NCL Eclipse, um *plugin* que permite a geração rápida de protótipos de aplicações e uma versão estendida do NCL Composer, representando o uso de uma

abordagem dirigida por modelos que utiliza autoria textual e implementa as etapas de *Prototipação Rápida* e *Refinamento da Aplicação* propostas neste trabalho.

6.2.2.4 Hipóteses

As hipóteses aqui levantadas foram obtidas a partir da hipótese geral estabelecida durante a definição do experimento, e devem contemplar as variáveis dependentes e independentes (WOHLIN et al., 2000). Consequentemente, a partir da hipótese geral que foi a motivação para a execução do experimento, foram derivadas as hipóteses nulas, e alternativas. Tais hipóteses, após a execução do experimento, serão submetidas a testes estatísticos, que fornecem informações definitivas para rejeitar (ou não) as hipóteses nulas em favor das hipóteses alternativas. Estes testes são realizados durante a análise dos resultados do experimento, e constituem a principal fonte de informação para as conclusões que serão obtidas no estudo experimental

Considerando Mx, x= 1, 2, e 3 as metodologias utilizadas nos experimentos e a métrica TI (Mx) (tempo de implementação da aplicação-problema), as seguintes hipóteses nulas e as respectivas hipóteses alternativas foram definidas:

- **Hipóteses Nulas** ($H_0$1 e $H_0$2): as abordagens de desenvolvimento são semelhantes com relação às propriedades de tempo de implementação (TI) no desenvolvimento de aplicações de TV Digital

$$H_0 1: TI\ (M1) \cong TI(M2)$$

$$H_0 2: TI(M1) \cong TI(M3)$$

- **Hipóteses Alternativas** ($H_1$1 e $H_1$2): as abordagens de desenvolvimento são diferentes com relação às propriedades de tempo de implementação (TI) no desenvolvimento de aplicações de TV Digital

$$H_1 1: TI\ (M1)\ \neq\ TI(M2)$$

$$H_1 2: TI\ (M1) \neq TI(M3)$$

6.2.2.5 Objetos do Experimento

Os objetos deste experimento são as especificações das aplicações-problema que servem de entrada para as metodologias utilizadas nos estudos. Nos experimentos foram utilizadas como aplicação-problema as 2 (duas) primeiras aplicações provas de conceito da abordagem proposta neste estudo: *TV Voting* (6.1.1) e *Slide-Show* (6.1.2). A especificação de cada aplicação-problema foi dividida em 4 (quatro) iterações considerando a evolução da implementação das funcionalidades de acordo com a complexidade. Essas especificações foram descritas utilizando *Storyboards* (roteiros), gráfico e textuais, além de uma *checklists* de funcionalidades que deveria ser atendida para cada interação. Adicionalmente, também foram disponibilizados para os sujeitos todos os objetos de mídia (arquivos de ima-

gens, áudio, vídeos) que compõe a aplicação multimídia, visto que o estudo não tem como objetivo analisar a etapa de produção multimídia da aplicação, e sim, as questões relacionadas a etapa do projeto de mídia do documento NCL e sua integração com o projeto de software implementado com a linguagem Lua, ou seja, aspectos mais relacionados a Engenharia de Software.

6.2.2.6 Instrumentação

Em relação à instrumentação do experimento, segundo Wohlin et al. (2000), podemos utilizar 3 (três) elementos: objetos, diretrizes e ferramentas de suporte. Nos experimentos realizados neste estudo, estes elementos são:

- Objeto: especificação da aplicação de TVD
- Diretrizes: um conjunto de *Storyboards* que descreve cada etapa do desenvolvimento da aplicação-problema dividida em iterações, uma lista de itens (*checklist*) foi utilizada para verificar o término do desenvolvimento das iterações, todos os objetos de mídia foram disponibilizados antes do início do desenvolvimento, um documento com instruções gerais para os sujeitos do experimento, e, por fim, também foram disponibilizados código-fonte Lua que servem de base para implementação da lógica de aplicação requerida (por exemplo, um módulo para acesso a dados remoto pelo protocolo HTTP).
- Ferramentas de suporte: Um laboratório com um ambiente de desenvolvimento e testes de aplicações Ginga-NCL foi utilizado para dar suporte à implementação do ambiente de experimentação. Todas as máquinas possuíam a mesma configuração. Tal ambiente, além de permitir a coleta das métricas de forma precisa, também possibilita testes de execução das aplicações em situações bem próximas de um ambiente real e de produção. O tempo de implementação de cada aplicação-problema foi calculado como a diferença entre início do experimento (igual para todos os participantes) e a finalização de todas as funcionalidades requeridas para a aplicação-problema. A determinação se aplicação estava completa ou não a cada interação era realizada por um dos executores do projeto. O tempo foi coletado individualmente para cada participante utilizando a ferramenta automática. Além de possuir um contador de segundos, essa ferramenta também permitia realizar pausas. Essas interrupções eram realizadas quando um participante desejava realizar alguma atividade não relacionada ao desenvolvimento da aplicação, por exemplo, atender um telefone, ir ao banheiro, parar para comer ou beber algo etc. Para evitar que os participantes ativassem a função de pausa da ferramenta, mesmo quando ainda estavam realizando atividades de desenvolvimento, foi instruído para os participantes que eles só deveriam interromper a contagem do tempo, após a permissão de algum executor do experimento. Do mesmo modo, também era necessário pedir permissão para retomar o contador. Dessa forma, foi possível controlar melhor as interrupções mal intencionadas. Adicio-

nalmente, os executores do experimento também monitoravam constantemente os participantes em relação ao uso correto da ferramenta de coleta de tempo.

6.2.2.7 Projeto Experimental

A análise do experimento tem como objetivo comparar os dados coletados durante o desenvolvimento do objeto do experimento (a aplicação-problema) através do objeto de controle (a abordagem) a fim de observar se a hipótese nula pode ser rejeitada.

O estudo aqui planejado analisa o uso de uma metodologia a partir do tempo de implementação de uma aplicação de TV Digital que possui como requisito uma integração entre o projeto de mídia, representado pelo documento NCL e o projeto de software, representado pelo código-fonte Lua.

Cada experimento foi caracterizado como um estudo envolvendo dois fatores. O primeiro diz respeito aos sujeitos, já que conhecimento e habilidades técnicas podem influenciar diretamente nossas medições e devem ser controlados. O segundo está relacionado à definição da aplicação-problema a ser desenvolvida pela abordagem, além de tratar aspectos relevantes da integração dos projetos de mídia e software, pois a mesma deve controlada, uma vez que sua complexidade ou simplicidade pode induzir a erros de avaliação.

Considerando esses fatores os experimentos deste estudo utilizam um plano experimental de quadrado latino. O uso dessa configuração em avaliação de metodologias de desenvolvimento em Engenharia de Software já bem avaliada por simulações (SÁNCHEZ, 2011) e sua aplicação já foi demonstrada em estudos na área de abordagens de Linhas de Produto de Software (ACCIOLY, 2012; ALMEIDA, 2010). A Figura 6.35 mostra uma configuração de quadrado latino seguindo o modelo de "linhas cruzadas com colunas embutidas dentro das réplicas" (SÁNCHEZ, 2011), dispondo os sujeitos em linhas e as aplicações-problema nas colunas. Para cada linha e coluna de uma réplica de um quadrado, a abordagem utilizada deve aparecer somente uma vez. Para cada abordagem avaliada (tratamento) um sujeito desenvolve uma aplicação-problema. A posição inicial das abordagens e das aplicações-problema deve ser aleatória. Os resultados foram analisados utilizando os testes de análise variância ANOVA (BOX; HUNTER; HUNTER, 2005). Ferramentas visuais também foram empregadas como suporte a análise dos resultados. A seguir são explicados os detalhes da execução e resultados de cada experimento.

	Aplicação 1	**Aplicação 2**		**Aplicação 1**	**Aplicação 2**	
Sujeito 1	M1	M2	**Sujeito 3**	M2	M1	
Sujeito 2	M2	M1	**Sujeito 4**	M1	M2	...
	Quadrado 1			**Quadrado 2**		

Figura 6.35 Projeto de experimento com quadrado latino

6.2.3 Primeiro Experimento

O primeiro experimento teve como objetivo avaliar exclusivamente a etapa de *Protipação Rápida* da abordagem proposta (ver seção 5.1). Essa restrição aconteceu por que na época a extensão do NCL Composer que implementa a etapa de *Refinamento da Aplicação* ainda estava em estágio de desenvolvimento, sendo inviável utilizá-la na prática. A configuração deste experimento é exibida na Figura 6.36. É possível observar que existem duas abordagens e cada uma é utilizada para desenvolver duas aplicações: P1 equivale a aplicação de *TV Voting* e P2, a aplicação de *Slide Show*. Considerando o projeto do quadrado latino (explicado na seção anterior), um sujeito que implementa uma aplicação P1 utilizando a abordagem *Com Prototipação,* vai obrigatoriamente, num segundo momento, implementar a aplicação P2 utilizando a outra abordagem (*Sem Prototipação*).

Na abordagem *Sem Prototipação,*o NCL Eclipse é utilizado para edição textual do código Lua e o NCL Composer do documento NCL, uma vez que o NCL Composer não tem suporte a nenhuma edição de código-fonte Lua. Por outro lado, a abordagem *Com Prototipação* faz uso de um *plugin* do Eclipse para derivar uma estrutura inicial da aplicação e partir daí realizar uma autoria textual do código no NCL Eclipse ou autoria visual com o NCL Composer. A diferença principal entre as duas abordagens é que a última realiza automaticamente a estruturação dos requisitos e uma análise do problema. Isso é realizado com o uso de um *Wizard* que permite gerar automaticamente o código-fonte de um projeto (estrutura) genérico de uma categoria de aplicação. É importante ressaltar que a derivação do protótipo não contempla a geração de código-fonte Lua, muito menos sua integração com o documento NCL. Para as duas aplicações, a ferramenta gera apenas um diretório para colocar os scripts disponibilizados para o experimento, e, dependendo da configuração definida no *Wizard*, ela pode gerar nós de mídias no documento NCL. Se esses nós forem do tipo mídia imperativa, então eles devem ser preenchidos e integrados com outros nós de mídia ou código-fonte Lua manualmente.

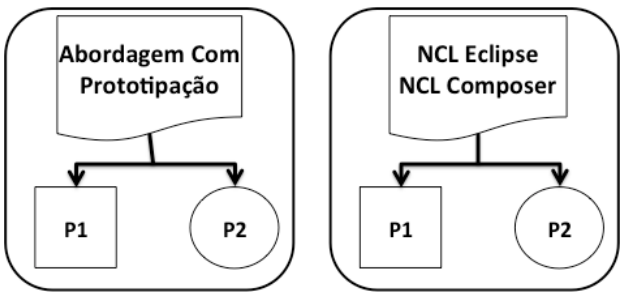


Figura 6.36: Configuração do primeiro experimento

6.2.3.1 Participantes

Este primeiro experimento teve a participação de 20 estudantes que participavam de um processo de seleção de estágio para um laboratório de pesquisa da Universidade Federal da Paraíba (UFPB). Os estudantes cursavam entre o segundo e sétimo período do curso de graduação em Ciência

da Computação da UFPB Campus I localizado na cidade de João Pessoa e tinham em média de 1 a 2 anos de experiência com programação. Nenhum deles teve contato com a linguagem NCL ou Lua anteriormente, e a maioria conhecia a linguagem HTML. Por conta da grande quantidade de sujeitos, além do autor deste trabalho, mais dois alunos de pós-graduação da UFPB ajudaram como executores do experimento.

Execução do Experimento

O experimento foi realizado no período de 02 de julho de 2012 a 6 de julho de 2012 num laboratório da UFPB. Os três primeiros dias foram dedicados a ministrar um treinamento de duração de 24 horas (8 horas diárias) sobre as linguagens NCL e Lua no contexto de desenvolvimento para aplicações para TV Digital. Nesses dias também foram apresentadas todas as ferramentas empregadas no experimento. Os últimos dois dias foram utilizados para a execução das atividades de desenvolvimento das aplicações no experimento.

A definição das 10 réplicas dos quadrados latinos foi realizada inicialmente, pela formação das duplas por meio de um sorteio. Cada computador foi numerado e à medida que o estudante entrava no laboratório, ele retirava uma ficha aleatoriamente para definir sua bancada e, consequentemente, sua réplica de quadrado latino. Depois foi sorteado qual seria a primeira aplicação-exemplo a ser desenvolvida, e por fim, o sorteio de que abordagem seria utilizada inicialmente pelo sujeito da primeira linha do quadrado latino (Sujeito 1 da Figura 6.35).

6.2.3.2 Análise dos dados

Os dados coletados durante a execução do experimento foram analisados utilizando o *R Statistical Software*[10]. Para facilitar a criação do arquivo de entrada para esse software, todos os dados foram antes unificados numa planilha. O formato do arquivo de entrada é o mesmo utilizado por Accioly (2012), visto que foi empregado o mesmo projeto de experimento.

A primeira análise realizada diz respeito à distribuição dos dados utilizando um gráfico do tipo "*box-plot*" (JAIN, 1991). Este apresenta medidas de quartis, média, mediana e pontos-limite. Na Figura 6.38 é possível observar que no geral a abordagem *Com-Prototipação* tem valores menores para o tempo de implementação. Os valores das medianas se encontram relativamente distantes. Na Tabela 6.1 também é possível verificar que o valor médio do tempo de implementação da abordagem Com-Prototipação representa um ganho de produtividade na ordem de 1,34. Acredita-se que desvio padrão encontra-se alto por conta da heterogeneidade de nível de formação dos participantes.

[10]R: Projeto para computação estatística. Disponível em: http://www.r-project.org/

Tabela 6.1 Medidas de tempo médio e desvio padrão do primeiro experimento

	Com-Prototipação	**Sem-Prototipação**
Tempo médio (em segundos)	10254	14036
Tempo médio (em horas e minutos)	2h:51m	3h:54m
Desvio padrão	2850	4075

Adicionalmente, também foram observados os resultados individuais de cada sujeito. A Figura 6.39 mostra que em todos os casos a abordagem *Com-Prototipação* obteve um tempo de implementação menor do que a abordagem *Sem-Prototipação*.

$$Y_{lijk} = \mu + \tau_l + \tau\alpha_{li} + \beta_j + \gamma_k + \varepsilon_{lijk}, onde:$$

Y_{lijk} Resultado para l-ésima réplica, i-ésimo sujeito, j-ésima aplicação-problema e k-ésima abordagem
μ Média dos valores respostas
τ_l Efeito da l-ésima réplica
$\tau\alpha_{li}$ Efeito entre a l-ésima réplica e o i-ésimo sujeito
β_j Efeito da j-ésima aplicação-problema
γ_k Efeito da k-ésima abordagem avaliada
ε_{lijk} Erro experimental

Figura 6.37: Modelo linear utilizado no experimento

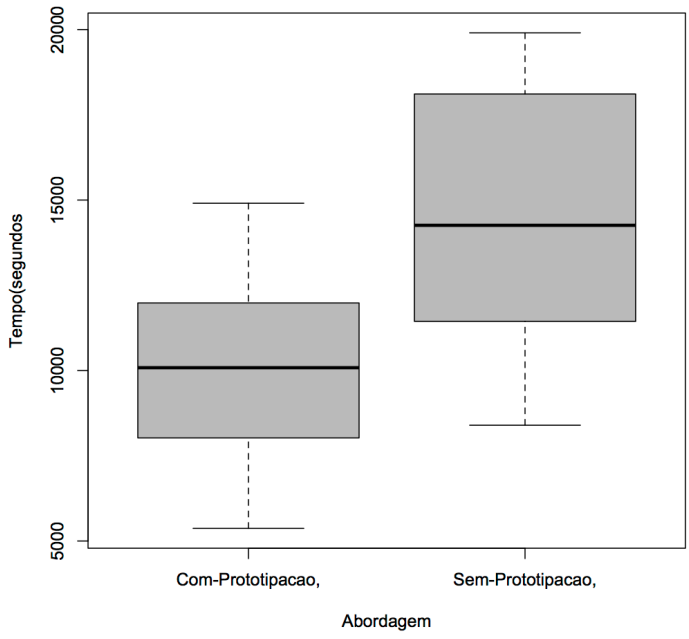


Figura 6.38: Gráfico *box-plot* do primeiro experimento

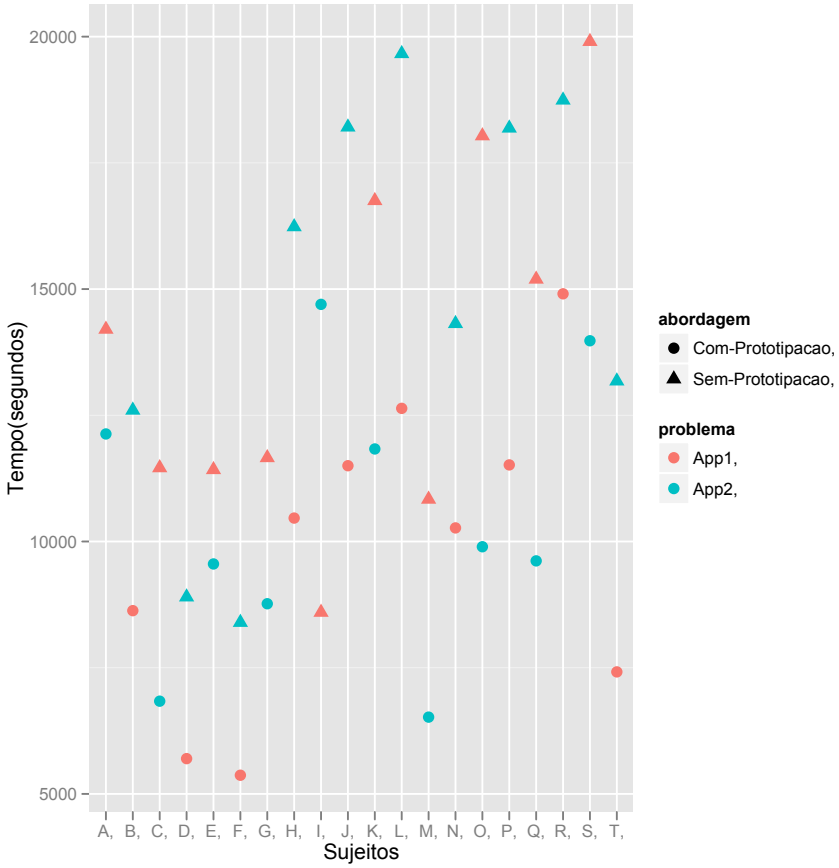


Figura 6.39: Resultados individuais para cada abordagem

A análise estatística dos experimentos foi realizada utilizado o modelo linear (KEMPTHORNE e HINKELMANN, 1994) apresentado na Figura 6.37 e que atende o modelo de replicação de quadrado latino de "linhas cruzadas com colunas embutidas dentro das réplicas". Porém, antes de realizar o teste estatístico, mais dois processos foram aplicados aos dados.

O primeiro, recomendado por Sakia (1992), consiste em aplicar uma técnica de transformação do tipo *Box-Cox*, que permite verificar se os resultados obtidos possuem anomalias de não-aditividade e não-normalidade. Esse diagnóstico consiste em avaliar se uma curva gerada pelo método tem seu valor máximo entre 0 e 1. Caso sim, não é necessário fazer nenhuma transformação nos dados. Conforme mostra a Figura 6.40, é possível verificar que não é necessário alterar os dados deste experimento.

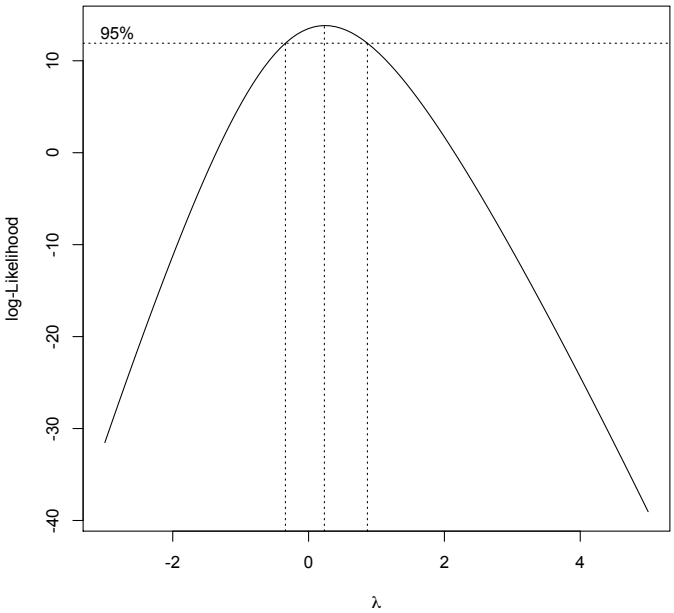


Figura 6.40: Método Box-Cox para o primeiro experimento

O segundo passo é executar o Teste de Aditividade de Tukey (BOX; HUNTER; HUNTER, 2005) que verifica se o modelo de efeitos é aditivo, ou seja, verifica se os fatores das colunas e linhas do quadrado latino não afetaram de modo significativo as respostas. No contexto deste experimento, por exemplo, se a variação do tempo de desenvolvimento das duas aplicações-problema escolhidas podem ser comparadas diretamente. Este teste possui as seguintes hipóteses:

H_0: o modelo é aditivo

H_1: H_0 é falso

O resultado do Teste de Aditividade de *Tukey* neste experimento retornou um valor de p de 0.163, não dando evidência significativa para rejeitar a hipótese H_0, ou seja, o modelo é aditivo.

Uma vez garantida a aditividade, a ANOVA foi aplicada sobre o modelo para comparar o efeito das duas abordagens (tratamento) no tempo de desenvolvimento das aplicações (variável resposta). A Tabela 6.2 mostra os resultados da execução do teste estatístico. Logo (pelo valor em destaque vermelho), é possível afirmar que, com um intervalo de confiança de 95% (valor de p < 0,05), podemos rejeitar a hipótese $H_0 1$ da Seção6.2.2.4, ou seja, a abordagem M1 teve efeito significativo sobre o tempo de desenvolvimento das aplicações em relação a outra abordagem M2.

Tabela 6.2 Resultado da ANOVA do primeiro experimento

	Df	Sum Sq	Mean Sq	F value	p-value
replica	9	217272763	24141418	5.633	0.000924
problema	1	6225999	6225999	1.453	0.243693
abordagem	1	177413652	177413652	41.397	0.000469
replica:sujeito	10	123983123	12398312	2.893	0.024033
Residuals	18	77141436	4285635		

6.2.4 Segundo Experimento

O segundo experimento repetiu a configuração realizada no primeiro. A única diferença está na data, local e participantes do experimento, descritos a seguir.

6.2.4.1 Participantes

Os participantes deste experimento foram 6 pessoas que realizaram um curso sobre desenvolvimento de aplicações para TV Digital utilizando Ginga-NCL. Os participantes eram estudantes que cursavam entre o 3.o e 5.o período de uma graduação em análise e desenvolvimento de sistemas do IFBA. Nenhum deles conhecia a linguagem NCL ou Lua antes do curso e a maioria conhecia a linguagem HTML e/ou XML. O acompanhamento do experimento foi realizado pelo autor deste trabalho e um professor do IFBA.

6.2.4.2 Execução do Experimento

O experimento foi realizado no período de 27 de agosto de 2012 a 31 de agosto de 2012 num laboratório do IFBA durante a III Semana de Engenharia, Tecnologia e Inovação realizada na cidade de Salvador. Novamente, os três primeiros dias foram dedicados a ministrar um treinamento de duração de 24 horas (8 horas diárias) sobre as linguagens NCL e Lua no contexto de desenvolvimento para aplicações para TV Digital. Do mesmo modo, também foram utilizados para familiarização o NCL Eclipse e a ferramenta utilizada para Prototipação Rápida durante o treinamento. Do mesmo modo que no primeiro experimento, os últimos dois dias foram utilizados para a execução do experimento com o sorteio de uma aplicação-problema diferente por dia.

O sorteio para definição das 3 (três) réplicas dos quadrados latinos foi realizado com o mesmo procedimento do primeiro experimento.

6.2.4.3 Análise dos dados

Mais uma vez analisando o gráfico *box-plot* (Figura 6.41) gerado é possível observar que a abordagem *Com-Prototipação* tem melhor produtividade (em relação a tempo de desenvolvimento) do que a abordagem Sem Prototipação. Os valores das medianas de cada abordagem (aproximadamente 10.000 para a abordagem *Com-Prototipação* e 16.000 segundos para a abordagem *Sem-Prototipação)* encontram-se muito próximos dos valores gerados para o primeiro experimento. Porém, a distribuição do tempo de cada indivíduo é mais homogênea e próxima da mediana. Este ponto também pode ser confirmado pelos valores bem menores de desvio padrão em relação ao primeiro experimento (Tabela 6.3). A suposição é que tal resultado se deve ao fato de que os participantes possuem um perfil mais homogêneo. Em relação ao ganho de produtividade, foi calculado um índice com valor de 1,53 a favor da abordagem *Com-Prototipação*, representando um ganho um pouco maior em relação ao primeiro experimento. Também é interessante verificar que a média de tempo da abordagem Com-Prototipação

neste experimento ficou muito próxima do valor calculado no primeiro experimento (2 horas e 49 minutos e 2 horas e 51 minutos, respectivamente).

Tabela 6.3 Medidas de tempo médio e desvio padrão do segundo experimento

	Com-Prototipação	**Sem-Prototipação**
Tempo médio (em segundos)	10143	15555
Tempo médio (em horas e minutos)	2h:49m	4h:19m
Desvio padrão	841	1304

Mais uma vez, se analisarmos os resultados individuais (Figura 6.42) é fácil notar que a abordagem *Com*-Prototipação foi mais rápida em 100% dos casos, independente do tipo de aplicação-problema.

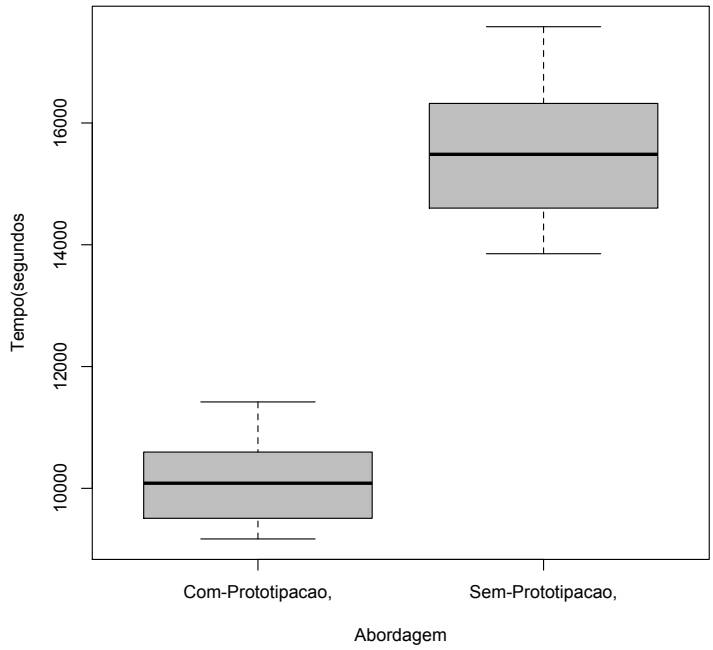


Figura 6.41: Gráfico *box-plot* do segundo experimento

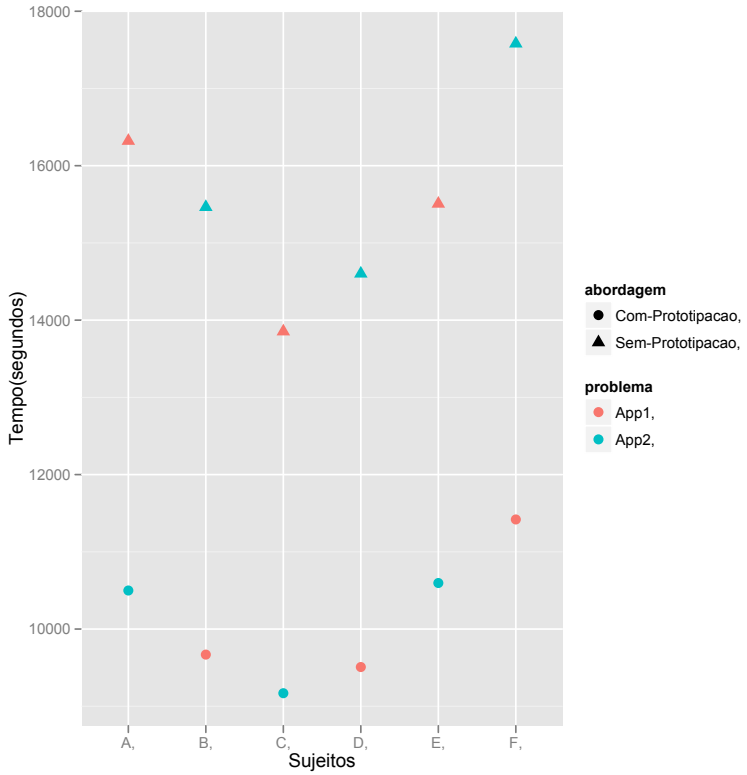


Figura 6.42: Resultados individuais para cada abordagem

O resultado da transformação *Box-Cox* (Figura 6.43) também comprovou a aditividade dos dados e o Teste de Aditividade de *Tukey* obteve valor de 0.094 também traz evidência da aditividade do modelo.

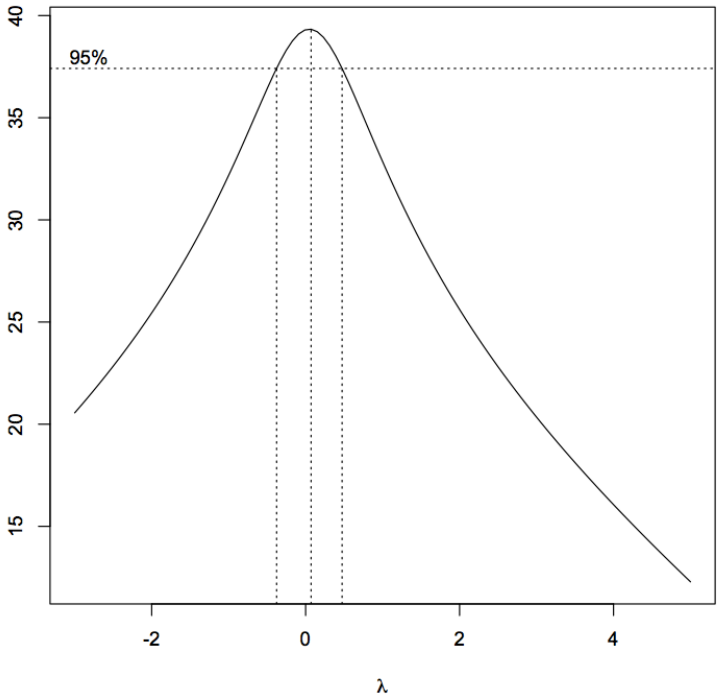


Figura 6.43: Método Box-Cox para o segundo experimento

A Tabela 6.4 exibe os resultados da análise estatística utilizando ANOVA e mais uma vez é possível verificar que hipótese $H_0 1$ é falsa. Consequentemente, quanto ao tipo de abordagem utilizada é possível concluir que o tipo de abordagem foi determinante para a diferença entre tempo de desenvolvimento.

Tabela 6.4 Resultado ANOVA do segundo experimento

	Df	Sum Sq	Mean Sq	F value	p-value
replica	2	8058449	4029224	24.524	0.00569
problema	1	222496	222496	1.354	0.30923
abordagem	1	87847585	87847585	534.677	0.00207
replica:sujeito	3	3105581	1035194	6.301	0.05375
Residuals	4	657201	164300		

6.2.5 *Terceiro Experimento*

O último experimento executado incluiu uma mudança na configuração utilizada anteriormente, pois, no momento da sua execução já era possível utilizar a extensão implementada na ferramenta NCL Composer, isto é, permitir a integração automática de código-fonte Lua como mídia imperativa na visão estrutural. Dessa forma, este experimento representa uma primeira avaliação externa da abordagem completa, que inclui as duas etapas: *Prototipação Rápida* e *Refinamento da Aplicação*. A Figura 6.44 ilustra a mudança. Apesar disso, as aplicações-problema (P1 e P2) permaneceram as mesmas (*TV Voting* e Slide Show). O principal motivo de não alterar as aplicações é permitir fazer comparações com os resultados já obtidos nos experimentos anteriores em relação ao uso apenas do NCL Eclipse e NCL Composer.

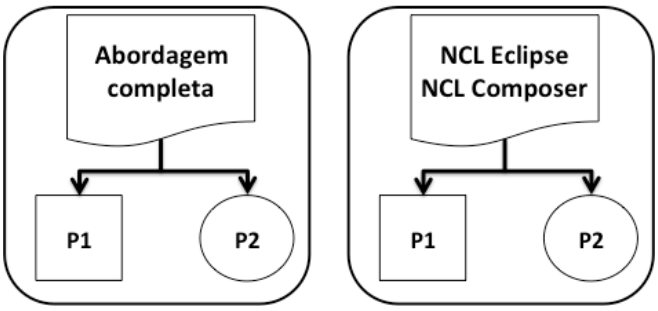


Figura 6.44: Configuração do terceiro experimento.

6.2.5.1 *Participantes*

Para este experimento foram selecionados 8 estudantes. Esses alunos estavam cursando uma disciplina que aborda o desenvolvimento de aplicações multimídia e a primeira parte do curso trata do desenvolvimento de aplicações para TV Digital com Ginga-NCL. Todos os participantes cursavam o 5.o período do curso de graduação em Sistemas de Informação da UFPB Campus IV localizado na cidade de Rio Tinto. Nenhum deles conhecia a linguagem NCL ou Lua anteriormente, a maioria co-

nhecia a linguagem HTML e/ou XML. Todos tinham entre 1 e 2 anos de experiência com programação. O acompanhamento do experimento foi realizado pelo autor deste trabalho e um aluno de pós-graduação da UFPB.

6.2.5.2 Execução do Experimento

Outra diferença desse experimento em relação aos dois anteriores, por se tratar de uma disciplina de graduação, é que o treinamento foi realizado durante um período longo de 6 semanas (duas aulas de 2 horas) entre o dias 07 de novembro de 2012 a 14 de dezembro de 2012. A carga horária total também foi de 24 horas e foi utilizado o mesmo material dos dois experimentos precedentes. O desenvolvimento das 2 (duas) aplicações foi realizada nos dias 19 e 21 de dezembro de 2012. Todas as atividades foram executadas num laboratório do Campus IV da UFPB e durante o treinamento. Do mesmo modo, também foram utilizados para familiarização, o NCL Eclipse e a ferramenta utilizada para Prototipação Rápida durante o treinamento. Como da outra vez, os dois últimos dias foram utilizados para a execução do experimento com o sorteio de uma aplicação-problema diferente por dia.

O sorteio para definição das 4 (quatro) réplicas dos quadrados latinos foi realizada do mesmo modo que os outros experimentos. Porém, dois alunos não conseguiram finalizar as aplicações em nenhum dos dois dias de modo que a análise só considerou 3 réplicas. Por sorte, eles tinham configurações diferentes no quadrado latino e não atrapalharam a combinação das outras réplicas. O critério estabelecido para eliminar os dados desses participantes foi ultrapassar o tempo de 5 horas (18.000 segundos) para implementar as aplicações-problema. Em outras palavras, o terceiro experimento considerou apenas a participação de 6 alunos.

6.2.5.3 Análise dos dados

A Figura 6.45 mostra o *box-plot* para o terceiro experimento e primeiro aspecto que chama a atenção é que a distância entre as medianas das duas abordagens aumentou consideravelmente. Em contrapartida, mesmo com um perfil de estudantes com mais experiência, a mediana dos valores do tempo de implementação da abordagem *Sem Prototipação* ficou novamente próximo dos 15.000 segundos. Também é válido afirmar que novamente a distribuição das medidas de tempo de implementação se manteve mais homogênea que o primeiro experimento. Tal resultado reforça a suposição do segundo experimento: o uso de perfis idênticos nos sujeitos gera uma distribuição dos dados mais próxima da mediana.

Tabela 6.5 Medidas de tempo médio e desvio padrão do terceiro experimento.

	Abordagem Completa	**Sem-Prototipação**
Tempo médio (em segundos)	4405	15135
Tempo médio (em horas e minutos)	1h:13m	4h:12m
Desvio padrão	751	1035

A permite uma observação dos dados com mais detalhes e fazendo a divisão do tempo médio da abordagem *Sem-Prototipação* pelo tempo médio da *Abordagem Completa,* obteve-se um fator de 3,43 de aumento de produtividade, mais que do que o dobro do resultado do segundo experimento.

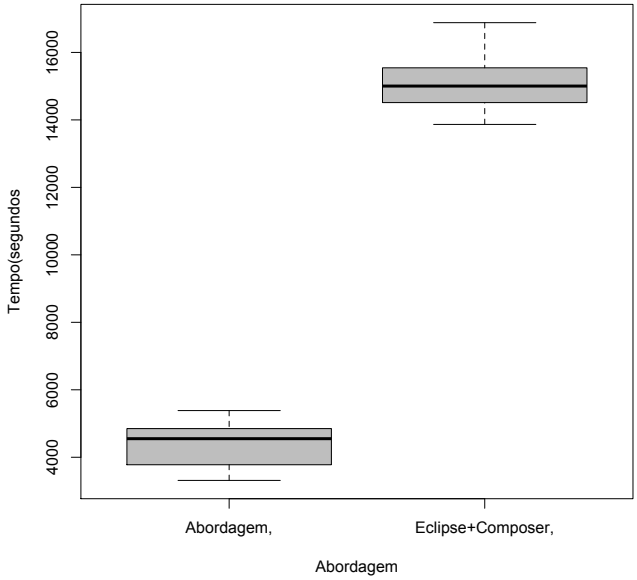


Figura 6.45: Gráfico *box-plot* do terceiro experimento

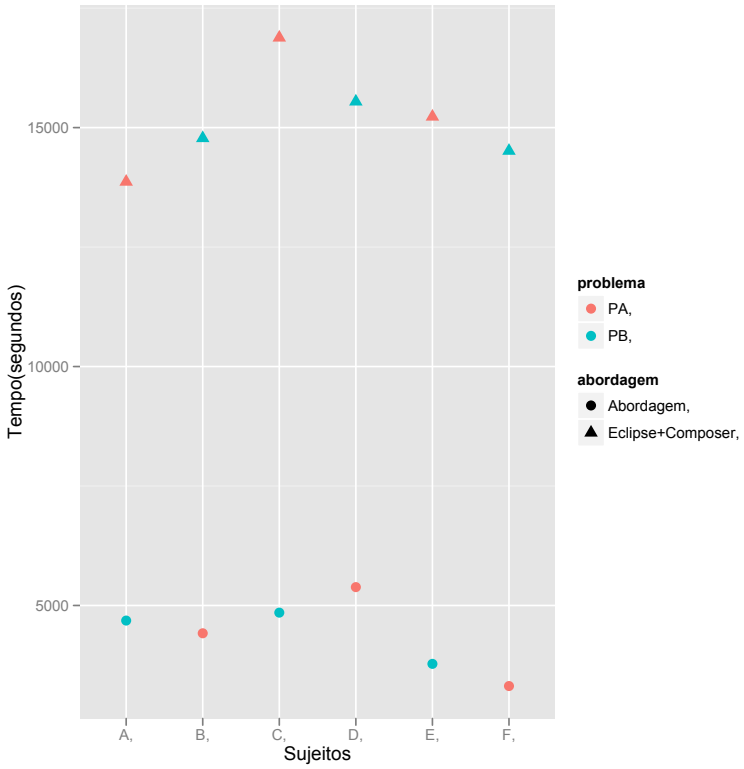


Figura 6.46: Resultados individuais para cada abordagem

Os resultados individuais na Figura 6.46 mostraram novamente que todos os sujeitos que utilizaram a Abordagem Completa terminam antes a implementação da aplicação-problema.

A curva formada pela transformação *Box-Cox* (Figura 6.47) teve o seu valor máximo muito próximo de 1, porém, como não alcançou, não foi necessário nenhuma transformação dos dados. Já o teste de Aditividade de *Tukey* retornou o valor de 0,596, garantindo novamente a aditividade entre os fatores de variância.

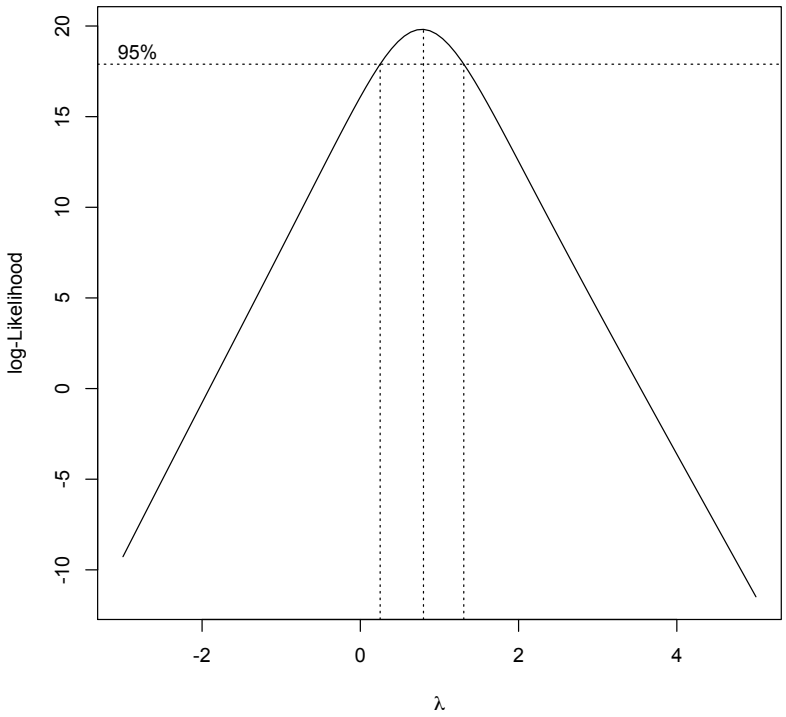


Figura 6.47: Método Box-Cox para o terceiro experimento

Por fim, a Tabela 6.6 sumariza a análise estatística ANOVA realizada. Nesse caso, podemos agora também negar a hipótese $H_0 2$ e afirmar a hipótese $H_1 2$ é verdadeira, ou seja, M1 e M3 não possuem diferença significativa no que diz respeito a tempo de desenvolvimento.

Tabela 6.6 Resultado ANOVA do terceiro experimento

	Df	Sum Sq	Mean Sq	F value	p-value
replica	2	4911241	2455621	3.820	0.11800
problema	1	74419	74419	0.116	0.75100
abordagem	1	345430891	345430891	537.340	0.00205
replica:sujeito	3	613443	204481	0.318	0.81300
Residuals	4	2571415	642854		

6.2.6 Ameaças à Validade

Para identificar as ameaças à validade do estudo aqui proposto foi utilizado um guia definido por Cook (COOK; CAMPBELL, 1979) em que as ameaças são identificadas de acordo com o tipo da validade (interna, externa, de conclusão ou construção).

6.2.6.1 Validade Interna

A principal ameaça nesse aspecto está relacionada ao controle do experimento. Dessa forma, é importante que o uso das ferramentas (que representam a adoção de uma metodologia) não seja influenciado pelo seu ambiente de execução. Para evitar essa ameaça, foram utilizadas máquinas pré-formatadas e com instalação mínima de outros softwares, de forma a evitar a influência de outros processos/software na execução das ferramentas. Outros aspectos que foram controlados para não prejudicar a validade interna foram os sujeitos e o objeto do experimento. A atuação dos sujeitos foi controlada por meio do uso de uma documentação que guiou cada etapa da execução do experimento, adicionalmente, foi realizada uma atividade de familiarização (por exemplo, foi explicado como fazer as medições de tempo) com o ambiente de execução do experimento, antes da sua execução final. Já o objeto do experimento foi controlado através de documentação detalhada da sua especificação, bem como a disponibilização dos seus elementos de mídia.

6.2.6.2 Validade Externa

A validade externa deste experimento está relacionada aos perfis dos sujeitos, o objeto do experimento utilizado, a instrumentação e o projeto experimental elaborados no estudo. Tais elementos em conjunto podem prejudicar a generalização dos resultados. Nesse sentido, acredita-se que as principais ameaças são os sujeitos e o objeto de estudo. No caso dos sujeitos, apesar de não representar precisamente profissionais da área de desenvolvimento de aplicações multimídia, acreditamos que o treinamento oferecido aproximou o perfil desses sujeitos ao de profissionais do mercado. Já em relação ao uso de apenas um objeto, tal característica pode prejudicar a generalidade do experimento, pois é possível que outras categorias de aplicações com requisitos diferentes apresentem um melhor ou pior resultado para as metodologias. Porém, por ser um dos primeiros estudos primários realizados nessa área, acredita-se que o objetivo principal deste trabalho foi definir um guia inicial para experimentos na área, ao invés de uma avaliação com várias categorias de aplicações. Por outro lado, o conjunto de categorias de aplicações para TVD não é tão amplo e acredita-se que as duas aplicações-problema representem características comuns à maioria das aplicações neste domínio.

6.2.6.3 Validade de Conclusão

A principal ameaça de validade de conclusão está relacionada ao aspecto social do experimento, ou seja, os sujeitos. Por exemplo, seus comportamentos podem ser guiados para aumentar a chance

de rejeitar (ou não) a hipótese nula de acordo com sua preferência pessoal por uma ou outra metodologia. Para evitar tal ameaça, os sujeitos não conheciam os objetivos e as hipóteses do experimento.

6.2.6.4 Validade de Construção

A principal ameaça desse tipo de validade está relacionada aos elementos do projeto experimental, como exemplo: o tamanho das amostras e os testes estatísticos utilizados. Em relação ao primeiro item, foi considerado que a quantidade de réplicas de experimentos foi maior que a quantidade definida no tamanho das amostras do projeto experimental, de forma a obter significância estatística dos dados. Adicionalmente, as análises estudadas consideraram um nível de confiança de 95%, possibilitando uma probabilidade satisfatória relacionada às zonas de rejeição das hipóteses nulas (JAIN, 1991). Em relação ao segundo item, o autor do experimento se aprofundou em assuntos de aspectos estatísticos, além de ter procurado ajuda de outros pesquisadores na área de experimentação em engenharia de software para revisar o projeto do experimento.

6.3 Considerações sobre o capítulo

Neste capítulo foi descrita a avaliação da tese defendida neste trabalho. Foram realizadas 3 (três) provas de conceitos exploratórias e 3 (três) estudos empíricos, todos em ambiente acadêmico. O objetivo para os três primeiros estudos foi avaliar a sistematização da aplicação da abordagem em diferentes categorias de aplicação (família de aplicação). Já os 3 (três) últimos tiveram como meta assinalar os efeitos da utilização da abordagem no desenvolvimento de aplicações de TVD em relação à outras abordagens. Embora as duas técnicas de avaliação utilizadas representem apenas um subconjunto do que pode ser avaliado na abordagem proposta, acredita-se que os resultados são um ponto inicial no sentido de definir uma metodologia para avaliar e melhorar o desenvolvimento de aplicações para TVD.

Em relação aos estudos experimentais, apesar de existirem ameaças à validade citadas na Seção 6.2.6, os estudos possuem resultados e indicações que permitiram conclusões relevantes no contexto deste trabalho. Também foi possível observar um crescimento no fator de produtividade (considerando apenas a execução da Engenharia de Aplicação da abordagem) na medida em que mais atividades foram incluídas na abordagem. Por exemplo, os valores na execução do primeiro, segundo e terceiro experimento, respectivamente foram: 1,34, 1,53 e 3,43. Um aspecto muito importante é que o valor obtido no último experimento está no intervalo de 3 a 10, obtido em outros estudos que avaliaram o uso de MDD (ver Introdução). Acredita-se que a partir de outras melhorias esse fator possa aumentar ainda mais.

Algumas falhas também foram identificadas após a execução dos estudos experimentais. Por exemplo, no processo de definição da posição dos sujeitos no plano experimental, a distribuição não foi totalmente aleatória, uma vez que a ordem de chegada influenciava. O correto seria aguardar a pre-

sença de todos os participantes e a partir daí realizar o sorteio da posição nos quadrados latinos. Outro ponto que pode ser melhorado é a coleta de métricas. Atualmente, cada sujeito possui um software de medição instalado em sua máquina e é necessária sua intervenção para iniciar, interromper, retornar e parar a coleta do tempo. Tal fato pode gerar situações em que o sujeito pode esquecer ou agir de má fé na execução de algum passo importante para a coleta como, por exemplo, não iniciar a coleta no início do experimento ou realizar interrupções sem autorização ou controle. Como sugestão para contornar esse problema, sugere-se a utilização de um sistema automatizado para coleta de métricas das estações remotas, mas que possua um gerenciamento centralizado para o responsável pelo experimento.

Vale a pena lembrar que apesar dos estudos estarem restritos ao uso do Eclipse EMF e da geração de código para a plataforma Ginga-NCL, os artefatos gerados (*Wizards*, *Modelo de Features*, *Modelo Template Modelo Instanciado*) nas três primeiras etapas da *Prototipação Rápida* podem ser reaproveitados em outros contextos. Isto aumenta a independência da solução com relação às tecnologias e domínios envolvidos.

7 CONCLUSÃO

Este trabalho fez uma apresentação da motivação, dos seus objetivos e escopo no capítulo inicial. O segundo capítulo desta tese apresentou uma revisão da literatura sobre o desenvolvimento orientada a modelos explicando quais as principais abordagens da indústria e seus principais elementos.

O Capítulo 2 apresentou os principais trabalhos relacionados e elaborou uma análise comparativa de modo a contextualizar a proposta em relação a outras pesquisas, bem como apontar algumas limitações das ferramentas existentes e duas grandes questões abertas para a área de desenvolvimento de aplicações para TV Digital: (1) definição de abordagens de desenvolvimento que integrem de modo sistemático as diferentes disciplinas (software, mídia e usabilidade) envolvidas na construção desse tipo de aplicação; e (2) estabelecimento de ambientes de desenvolvimento que suportem concomitantemente a autoria criativa e melhor estruturação e manutenção da lógica de aplicação desse tipo de software.

No Capítulo 3 foram apresentados os conceitos básicos sobre a plataforma de TV Digital, exemplos de aplicações e linguagens e ambientes de desenvolvimento que utilizam a linguagem declarativa NCL e a linguagem imperativa Lua.

No Capítulo 4 foi apresentada uma solução que procura atacar as duas questões apresentadas no Capítulo 2. De modo geral foi definida uma metodologia sistemática de desenvolvimento de software que utiliza conceitos e práticas do MDD, GSD e LPS no domínio de aplicações para TV Digital. Além disso, a abordagem propõe dividir o desenvolvimento de aplicação em duas etapas: *Prototipação Rápida e Refinamento da Aplicação*. A primeira etapa tem como objetivo a geração automática de uma estrutura parcial de uma aplicação de TV Digital, ao mesmo tempo em que permite a integração de profissionais da equipe de mídia (responsável pelo projeto de mídia) e a equipe de software (responsável pelo projeto de software) durante todas as atividades da etapa. O objetivo da segunda etapa é permitir que durante a fase de implementação da aplicação também aconteça uma melhor integração entre as equipes de mídia e software.

Já o Capítulo 5 inicialmente fez a avaliação do uso da abordagem por meio do desenvolvimento de três tipos de aplicações. A ideia foi demonstrar a viabilidade da abordagem apresentando passos importantes para sua execução. Posteriormente, foi definido um projeto de experimento controlado e apresentado os resultados de três execuções, onde foi possível definir quantitativamente e com nível de significância alto o ganho de produtividade da aplicação da abordagem quando comparada com outras soluções existentes.

7.1 Contribuições

Nesta tese almejou-se resolver parte do problema da integração de projeto de mídia e software no domínio de aplicações de TV Digital utilizando o desenvolvimento orientado a modelos. Após pes-

quisas nas áreas relacionadas, foi definida uma abordagem MDD para desenvolvimento de aplicações Ginga-NCL, visando integrar profissionais da equipe de mídia e da equipe de software desde as atividades de análise e projeto até a implementação do código-fonte final da aplicação, visando elevar a produtividade (em relação ao tempo de implementação) no que diz respeito à autoria criativa de aplicações NCL. Neste sentido, as seguintes contribuições foram alcançadas:

- Uma abordagem sistemática, detalhando atividades e artefatos que explicam o que é necessário para realizar cada tarefa de modo geral;
- Detalhamento das atividades de análise e projeto no sentido de definir um método concreto para utilização da linguagem MML em conjunto com um Modelo de *Features* e técnicas de transformação (Modelo *Template* para Modelo Instanciado) e geração (Modelo Instanciado de código utilizando ferramentas padronizadas pela indústria. Particularmente, foram mapeadas duas transformações: (1) Modelo de *Features* para Modelo *Template* e (2) Modelo *Template* para Modelo Instanciado e um algoritmo de geração de código: Modelo Instanciado para código-fonte NCL. Este último considera a geração de uma aplicação que é compatível com uma ferramenta de autoria que permite o projeto criativo final da implementação da aplicação;
- Detalhamento das atividades de implementação no que diz respeito à integração do projeto de mídia e de software utilizando a ferramenta Composer. Em particular, também foram definidas duas extensões para uma linguagem visual dessa ferramenta de modo a facilitar integração de código imperativo num documento declarativo;
- Modelagem e implementação de três famílias de aplicações de TV Digital que exploram novos cenários de uso (serviços Web) que ainda são objeto de pesquisa por apresentar cenários que podem ser utilizados em: TV Social, TV Conectada, TV Híbrida. Por exemplo, as aplicações *SlideShow* e *Weather* TV que acessam dados de softwares que se encontram na Web. Os exemplos de uso também podem ser aproveitados em cursos de formação na área de desenvolvimento de aplicações para TV Digital, visto que modelam diversas questões importantes desse tipo de aplicação, por exemplo, integração com código Lua.
- Planejamento e realização de estudos empíricos. A literatura apresenta poucos estudos que comparam metodologias de desenvolvimento de aplicações multimídia ou mesmo metodologias MDD. Dessa forma, o projeto do experimento e os resultados dos estudos foram relevantes não apenas para avaliar a abordagem, mas também para a comunidade científica e profissional da área de engenharia de software e TV Digital.

Além das contribuições documentadas nesta tese, o autor teve 5 trabalhos publicados em conferências (2 nacionais e 3 internacionais) como primeiro autor (KULESZA et al. 2012a) (KULESZA et al. 2012b), (KULESZA, R.; MEIRA, S. R. L.; SOUZA FILHO, G. L., 2011) (KULESZA et al.,

2011) e 2 trabalhos publicados em conferência como co-autor (FERREIRA, T. P.; KULESZA, R. ; SOUZA FILHO, G. L, 2011) (FERREIRA, T. P.; KULESZA, R.; SOUZA FILHO, G. L, 2012) e 1 capítulo de livro publicado como primeiro autor (KULESZA, R. et al., 2010).

7.2 Trabalhos futuros

Existem várias oportunidades de trabalhos futuros a partir desta tese, seja para explorar pontos que foram identificados como deficientes e abertos a contribuições, ou para complementar este estudo atacando limitações e/ou pontos não tratados. Dessa forma, os seguintes itens podem ser explorados como pesquisa posteriormente:

- **Incluir outras etapas no processo de desenvolvimento**: por questão de foco e escopo, este trabalho não inclui todas as atividades que podem ocorrer em um processo de desenvolvimento. Desse modo, podem-se incluir na abordagem outras fases e atividades relacionadas a disciplinas não estudadas nesta tese, por exemplo, integrar uma etapa de especificação de requisitos de modo a extrair automaticamente as características a partir de artefatos específicos (roteiros, estórias do usuário etc.). Também pode estudar como tratar questões do gerenciamento de requisitos no que diz respeito a gerenciamento de mudanças durante as fases de implementação da aplicação. Por fim, pode também incluir atividades de validação e verificação de software para garantir como resultado um software de maior qualidade.
- **Integração com o projeto da interface gráfica**: o desenvolvimento de aplicações multimídia envolve além do projeto de mídia e projeto de software, o projeto de interface gráfica com o usuário. Nesta tese este último aspecto não foi considerado. Portanto, existe uma oportunidade de pesquisa que inclui atividades que consideram as metodologias e práticas da comunidade de interface homem-computador. O criador da linguagem MML sugere o uso de diagramas de tarefas e diagramas de interfaces gráficas abstratas. Outra sugestão é incluir nas atividades já existentes do processo atual, mecanismos de assistência contextualizada automatizada, para guiar os engenheiros de domínio na elaboração dos elementos de interface gráficos de acordo com assistentes automáticos que surgem de acordo com contexto da atividade em execução. Por exemplo, no momento da criação de um elemento de interface gráfica, poderiam surgir guias para orientar o engenheiro na elaboração de um modelo ou geração de código que atenda um conjunto de requisitos de usabilidade. Alguns exemplos de tecnologia que permitem o uso de mecanismos dessa natureza em abordagens MDD são: *Microsoft Blueprints*, receitas do *openArchitectureWare* e quadro de tarefas do Eclipse GMF.

- **Desenvolvimento de uma ferramenta de autoria de *templates*:** uma das grandes dificuldades de se aplicar a abordagem são as tarefas associadas a Engenharia de Domínio que envolvem a elaboração dos modelos e seus configurações de similaridades e variabilidades. Tal tarefa exige conhecimento da plataforma EMF, das linguagens MML e UML. Uma contribuição seria criar um editor de *templates* que facilitasse essa tarefa. Nessa direção, é possível apenas desenvolver um editor mais amigável para UML ou utilizar conceitos de *templates* de composição para permitir a edição de *templates* a partir de reuso de elementos de granularidade mais baixa. Para isso, seria necessário definir um modelo de integração entre a MML e uma linguagem como TAL ou Luar, por exemplo.
- **Especialização da abordagem para plataformas específicas**: uma das bandeiras que o MDD gosta de defender é a possibilidade de partir de modelos PIM para a transformação de modelos PSM de diferentes plataformas. Na abordagem proposta, os *Modelos Templates* poderiam ser reutilizados para geração de código para outras plataformas, por exemplo, Ginga-J, HTML5, Android Google TV. Esta extensão é interessante, pois pode explorar outras plataformas de TV Digital (TV Conectada, TV Híbrida etc.). Porém, um grande desafio e uma boa oportunidade para estudos exploratórios é garantir que as atividades se mantenham com a mesmo nível de sistematização, principalmente para a definição da estrutura geral da aplicação e geração do código do metamodelo MML para outro metamodelo que não o NCL.
- **Especialização da abordagem para subdomínios**: outra possibilidade é aplicar a abordagem num subdomínio específico de TV Digital, de modo a refinar as definições de atividades para o domínio mais geral e também fazer estudos para definir mecanismos de integração com frameworks (por exemplo, na área de jogos ou redes sociais) ou DSLs mais especializadas (aplicações de livros eletrônicos). Nesse mesmo sentido, a partir da especialização da abordagem para subdomínios, poderiam ser exploradas metodologias de geração de projeto arquitetural e código-fonte que consideram requisitos não-funcionais avançados. Por exemplo, no caso da TV Digital, poderiam ser tratadas questões de integração com serviços Web, sensibilidade ao contexto, escalabilidade do uso do canal de retorno, entre outros. Nesses cenários poderiam ser definidas ou utilizadas DSL que tratam requisitos não-funcionais, também chamadas de linguagens de descrição de arquiteturas (do inglês, ADL – *Architecture Description Language*), por exemplo, SMADL (NASCIMENTO et al., 2012). Nesse caso, poderia estender a visão estrutural do NCL Composer para conseguir extrair informações da ADL, permitir a configuração de parâmetros de modo visual e assim realizar a integração com objetos de mídia imperativos que representam máquinas sociais.

- **Replicação dos estudos empíricos**: uma das limitações deste trabalho são os estudos empíricos. É notório que para realizar os experimentos em primeiro lugar é necessário ter o conhecimento específico, pouco ainda difundido na área de Engenharia de Software. Outro problema é conseguir um ambiente, os sujeitos adequados e a disponibilidade de tempo deles para executar o estudo. No contexto dos estudos deste trabalho, eram necessárias no mínimo 34 horas para cada experimento. Considerando esses dois fatores, podem-se realizar adequações ao projeto do experimento para aumentar a validade dos resultados, por exemplo, adicionando mais métricas de produtividade de desenvolvimento. Outro ponto é elaborar aplicações-problema que tratam pontos mais específicos e demande um tempo menor para desenvolvimento, dessa forma, pode-se viabilizar uma quantidade maior de execução de experimentos. Recomenda-se também pesquisar um método de coleta de tempo automática e centralizado, controlado pelo executor, de modo a evitar trapaças ou configuração errada na coleta do tempo por parte dos sujeitos.

7.3 Considerações Finais

Nesta pesquisa apresentou-se a tese de que o MDD combinado com técnicas de desenvolvimento de família de sistemas pode melhorar a produtividade do desenvolvimento de aplicações para TV Digital, detalhando uma abordagem para esta finalidade e sua avaliação por meio de exemplos de uso e estudos empíricos. A principal contribuição está em facilitar a integração do projeto de mídia e projeto de software durante as fases de análise, projeto e implementação de aplicações que seguem o paradigma declarativo da linguagem NCL.

Dentre as vantagens de aplicar as Etapas de Prototipação e Refinamento da Aplicação é possível citar: (i) consistência entre as aplicações (*branding*), onde é possível definir e seguir um mesmo padrão para a estrutura, comportamento e identidade visual. Tal característica é muito importante quando se imagina uma organização (emissora, agência de produção de conteúdo, etc.) com muitas filiais e que se deseja manter um padrão para o desenvolvimento das aplicações; (ii) a definição de um conjunto de família de aplicações permite facilmente a indexação e agrupamento das aplicações; (iii) como consequência de (i) e (ii), existe também a possibilidade de aumentar o reuso por meio da reutilização dos modelos PSM, criados para um tipo de aplicação e que é comum para outro tipo ou mesmo para outra plataforma; (iv) a inclusão de atividades de análise e projeto podem trazer benefícios durante a implementação, uma vez que comunicação entre o cliente pode ser mais efetiva para identificação dos requisitos; (v) apesar de demandar um esforço inicial para o estabelecimento de um conjunto comum de aplicações, uma vez realizado este trabalho, o desenvolvimento de aplicações a partir dos *templates* se torna mais rápido, pois permite reduzir a quantidade de código que os programadores precisam produzir através da geração automática de várias partes da implementação da aplicação.

REFERÊNCIAS

ACCIOLY, P. R. G. *Comparing Different Testing Strategies for Software Product Lines*. 103f. Dissertação (Mestrado), Universidade Federal de Pernambuco, Recife, 2012.

ALFONSI, B. I Want My IPTV: Internet Protocol Television Predicted a Winner. *IEEE Distributed Systems Online*, 2005.

ALMEIDA, R. B., *Modeling Software Product Line Variability in Use Case Scenarios— An Approach Based on Crosscutting Mechanisms*. 2010. 150f. Tese (Doutorado), Universidade Federal de Pernambuco, Recife, Brasil.

AMBLER, S. W. Agile model driven development is good enough. *IEEE Software*, v. 20, n. 5, p. 71–73, 2003.

AMOR, M.; FUENTES, L.; PINTO, M. A survey of multimedia software engineering. *Journal of Universal Computer Science*. 2004. v.10, n. 4. p. 473–498.

AZEVEDO, R. G. A. et al. Textual Authoring of Interactive Digital TV Applications. *In: Proceedings of the IX European Conference on Interactive Television*. 2011. p.23-243. ACM. Lisboa.

BABBIE, E. R. *Survey research methods*. 1990. Wadsworth.

BACHMAYER, S.; LUGMAYR, A.; KOTSIS, G. Convergence of Collaborative Web Approaches And Interactive TV Program Formats. Int. *Journal of Web Information Systems*, 2010.

BELTRÃO FILHO, M. F. de H. *GingaWay – Uma Ferramenta para Criaçao de Aplicaçoes Ginga NCL*. Trabalho de Graduação (Monografia), Universidade Federal de Pernambuco. Recife, 2008.

BERNARDI, M. L.; DI LUCCA, G. A.; DISTANTE, D. A Model-Driven Approach for the Fast Prototyping of Web Applications. In: *XXIII IEEE International Symposium on Web Systems Evolution (WSE 2011)*. 2011. P.65-74. Benevento

BOOCH, G et al. An MDA manifesto. *The MDA Journal:* Model Driven Architecture Straight from the Masters, 2004, pp. 133–143.

BOX G. E. P., HUNTER, S. J., HUNTER, W. G. *Statistics for experimenters: design, innovation, and discovery*. Wiley-Interscience, 2005.

BROOKS, F. P. *The Mythical Man-Month: Essays on Software Engineering (Anniversary Edition)*. Addison-Wesley, Boston, Massachusetts, 1995.

BROWN, A. W. Model driven architecture: Principles and practice. *Software and System Modeling*, 2004, v.3(4) pp. 314-327.

BULTERMAN, D. C. A.; HARDMAN, L. Structured multimedia authoring. *ACM Transaction on Multimedia Computation and Communication.* Appl. 2005, 1(1), p89–109.

CALVARY, G. et al. *Plasticity of user interfaces: A revised reference framework*. In: Pribeanu, C., Vanderdonckt, J. (eds.) TAMODIA. 2002. p. 127–134. Publishing House, Budapeste.

CASSOU, D. et al. A Generative Programming Approach to Developing Pervasive Computing Systems. In: *Proceedings of the 8th international conference on Generative programming and component engineering*, 2009. p.137-146,

CAZENAVE, F; QUINT, V; ROISIN, C. Timesheets.js: tools for web multimedia. *ACM Multimedia*, p.699-702, 2011.

CAZENAVE, F; QUINT, V; ROISIN, C. Timesheets.js: when SMIL meets HTML5 and CSS3.In: *Proceedings of the 11th ACM symposium on Document engineering*, p.699-702, 2011.

CESAR, P. *et al*. Fragment, tag, enrich, and send: Enhancing social sharing of video. *ACM Trans. Multimedia Comput. Commun. Appl.*, ACM, Nova Yorque, v. 5, p. 19:1–19:27, 2009.

CHORIANOPOULOS, K. *Virtual Television Channels: Conceptual Model, User Interface Design and Affective Usability Evaluation*. Tese (Doutorado), Universidade de Economia e Negócios de Atenas, 2004.

CLEMENTS, P., NORTHROP, L.: *Software Product Lines: Practices and Patterns*, Addison-Wesley Professional, 2001.

COOK, T. D.; CAMPBELL, D. T. *Quasi-Experimentation: Design and Analysis Issues for Field Set-*

tings. 1979. Houghton Mifflin Company.

COPPENS, T.; TRAPPENIERS, L; GODON, M. AmigoTV: A Social TV experience through triple-play convergence, In *Proceedings of the II European Conference on Interactive Television*, 2004.

CRUZ, V. M.; MORENO, M. F.; SOARES, L. F. G. *TV Digital Para Dispositivos Portáteis- Middlewares*. 2008. 74f. Monografias em Ciência da Computação (Monografia). Departamento de Informática, PUC-Rio, Rio de Janeiro.

CZARNECKI, K.; EISENECKER, U. W. G*enerative Programming: Methods, Tools, and Applications*. Boston: Addison-Wesley, 2000.

CZARNECKI, K. Overview of generative software development. In: BANÂTRE, J.-P. et al. (Ed.). *Unconventional Programming Paradigms, International Workshop* (UPP 2004). Springer, (Lecture Notes in Computer Science, v. 3566), p. 326–341, 2004.

CZARNECKI, K.; HELSEN, S.; EISENECKER, U. Staged configuration using feature models. In: NORD, R. L. (Ed.). In: *Proceedings of the Third Software Product Line Conference*. Boston, MA: Springer, 2004. (LNCS 3154), p. 266–283.

CZARNECKI, K.; ANTKIEWICZ M. Mapping features to models: A template approach based on superimposed variants. In: *4th International Conference on Generative Programming and Component Enginering (GPCE 2005)*. 2005. Springer. p. 422-437. Tallinn.

CZARNECKI, K.; HELSEN, S. Feature-based survey of model transformation approaches. *IBM Systems Journal*, v. 45, n. 3, p. 621–645, 2006.

DAMASCENO, J.; SANTOS, J.; MUCHALUAT-SAADE, D. C. EDITEC: Editor Gráfico de Templates de Composição para Facilitar a Autoria de Programas para TV Digital Interativa. In*: Simpósio Brasileiro de Sistemas Multimídia e Web* (WebMedia), 2010.

DEELSTRA S.; SINNEMA, M.; BOSCH J. Product derivation in software product families: a case study. *The Journal of Systems and Software*. 2005. 74(2): p.173–194.

DELTOUR, R.; ROISIN, C. The Limsee3 Multimedia Authoring Model. In: *Proceedings of the 2006 ACM symposium on Document engineering (DocEng '06)*. Nova Iorque, ACM, 2006. p. 173–175.

DEMARCO, T. *Structured Analysis and System Specification*, Englewood Cliffs, Nova Jersey, 1979.

D'SOUZA, D. F.; WILLS, A. C., *Objects, Components, and Frameworks with UML - The Catalysis Approach*. Addison-Wesley, 1999, p. 785.

ENCARNAÇÃO, H. T. da. *NCLite: Explorando o Conceito de Cenas Interativas em Ferramentas de Autoria para TV Digital*, Dissertaçao (Mestrado em Informática), Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro, 2010

ENGELS G.; SAUER, S. Object-oriented Modeling of Multimedia Applications, *Handbook of Software Engineering and Knowledge Engineering*, Ed. Singapore: World Scientific. 2002. v. 2. p. 21–53.

FERREIRA, T. P.; KULESZA, R. ; SOUZA FILHO, G. L. iTV Visual Design: Uma Ferramenta para Desenvolvimento Visual de Aplicações Java compatíveis com o Padrão Brasileiro de TVD. In: *X Simpósio Brasileiro de Qualidade de Software (SBQS) / V Workshop de Desenvolvimento Rápido de Aplicações* (WDRA). Curitiba, 2011.

FERREIRA, T.P.; KULESZA, R.; SOUZA FILHO, G.L. Uma Ferramenta de Autoria para Aplicações NCL e NCLua: uma Abordagem Orientada a Templates e com Suporte a Serviços Web. In: *XXVIII Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia). Workshop de Teses e Dissertações*, São Paulo, 2012.

FOWLER, M. *Language Workbenches: The Killer-App for Domain Specific Languages?*, Disponível em: <http://martinfowler.com/articles/languageWorkbench.html>. Acesso em: 15 de jan. 2013

FRATERNALI, P.; COMAI, S.; BOZZON, A. Engineering Rich Internet Applications with a Model-Driven Approach. *ACM Transactions on the Web,* 2010. v.4,(2). ACM. Nova Iorque.

FURTADO, A.W.B. *Domain-Specific Game Development.* 2012. 259f. Tese (Doutorado). Universidade Federal de Pernambuco, Recife. Brasil.

GAUFFRE, G.; DUBOIS, E.; BASTIDE, R. Domain-Specific Methods and Tools for the Design of Advanced Interactive Techniques. *Models in Software Engineering Lecture Notes in Computer Science*, v.5002, 2008, p. 65-76.

GAWLINSKI, M. *Interactive Television Production*. Focal Press. 2003.

GOLDMAN, D. B *et al*. Schematic storyboarding for video visualization and editing. *ACM Transactions Graph*, ACM. Nova Iorque. 2006. v. 25. p. 862–871.

GOMAA, H. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*, Addison-Wesley, 2005, p. 701.

GREENFIELD, J. et al. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley & Sons, 2004.

HANNINGTON, A.; REED, K. Factors in multimedia project and process management– australian survey findings. *In: ASWEC 2007: Proceedings of the 2007 Australian Software Engineering Conference*, 2007. p. 379–388. IEEE Computer Society, Washington.

HARDMAN, L *et al*. Canonical processes of semantically annotated media production. *Multimedia Systems*, 2008. Springer Berlin, v. 14, n. 6, p. 327–340.

HOOGEVEEN, M. Towards a theory of the effectiveness of multimedia systems. *International Journal of Human Computer Interaction* 9(2). 1997.151–168.

ITU-T H.761. *Nested Context Language (NCL) and Ginga-NCL for IPTV Services*. Recommendation H.761, Genebra. 2011.

JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. 1991. John Wiley.

KANG K. C., COHEN S., HESS J., Novak W., A. Peterson. *Feature-Oriented Domain Analysis (FODA) – Feasibility Study*, Technical Report CMU/SEI-90-TR-21, SEI/CMU, 1990.

KANG, K. et al. FORM: A Feature-Oriented Reuse Method with domain-specific reference architectures. In: *Annals of Software Engineering Notes*, v.05, 1998, p.143–168.

KÄRNÄ, J.; TOLVANEN, J.P.; KELLY, S. Evaluating the use of DSM in embedded UI application development, In: *Proceedings of DSM'09 at OOPSLA*, 2009.

KEMPTHORNE, O.; HINKELMANN, K. *Design and Analysis of Experiments. Vol. I: Introduction to Experimental Design*. Wiley-Interscience. 1994.

KLEPPE, A.; WARMER, J.; BAST, W. MDA Explained - *The Model Driven Architecture: Practice and Promise.* [S.l.]: Addison-Wesley. 2003. (Object Technology Series).

KORDON F. L. An Introduction to Rapid System Prototyping, *IEEE Transactions on Software Engineering*, vol. 28, no. 9, 2002, p.817-821.

KULESZA, R. et al. Towards a generative software development approach for rapid prototyping iDTV applications. In: *XVII Simpósio Brasileiro de Sistemas Web e Multimídia (Webmedia 2012)*, São Paulo. 2012.

KULESZA, R. et al. A Model Driven Approach for Integration of Interactive Applications and Web Services: A Case Study in Interactive Digital TV Platform. In: *IEEE International Conference on Multimedia & Expo Workshops (ICMEW 2012)*. Austrália, 2012.

KULESZA, R.; MEIRA, S. R. L.; SOUZA FILHO, G. L. Integration of Interactive Multimedia Applications and Web Services: Moving towards a model-driven approach. In: *CLEI 2011. Simposio Latinoamericano sobre Ingeniería de Software*, Quito, 2011.

KULESZA, R. et al. Uma Abordagem Dirigida por Modelos para Integração de Aplicações Interativas e Serviços Web: Estudo de caso na Plataforma de TV Digital. In: *XVII Simpósio Brasileiro de Sistemas Web e Multimídia (Webmedia 2011)*, Florianópolis. 2011.

KULESZA, R. et al. *Ginga-J* - An Open Java-based Application Environment for Digital Interactive Television Services. In: *IFIP Advances in Information and Communication Technology*. New York : Springer Verlag, 2011. v. 365. p. 36-48.

KULESZA, R. et al. *Desenvolvimento de Aplicações Imperativas para TV Digital no Middleware Ginga com Java*. In: Jussara Marques. (Org.). Mini-Cursos do XVI Simpósio Brasileiro de Sistemas Multimídia e Web. 1ed. Porto Alegre: Sociedade Brasileira de Computação, 2010, v. 1, p. 30-71.

LANG, M.; FITZGERALD, B. Hypermedia Systems Development Practices: A Survey, *IEEE Software*, vol. 22, no. 2, p. 68–75, 2005.

LANG, M.; FITZGERALD, B. New branches, old roots: A study of methods and techniques in web/hypermedia systems design. In: *IS Management*, v. 23, n. 3, p. 62–74, 2006.

LEE, C. J. et al. Emotionally reactive television, In: *Proceedings of the XXI International Conference on Intelligent User Interfaces*, ACM Press, 2007. p. 329-322.

LEE, J.; MUTHIG, D. Feature-oriented variability management in product line engineering. *Communications of the ACM*, v. 49, n. 12, p. 55–59, 2006.

LIDDLE, S. W. Model-Driven Software Development. *Handbook of Conceptual Modeling*: *Theory, Practice, and Research Challenges.* Springer-Verlag. 2011. p. 17-54.

LIMA, B. S. et al. Composer 3: Ambiente de autoria extensível, adaptável e multiplataforma. In: WebMedia 2010 - Workshop de TV Digital Interativa. 2010. Belo Horizonte.

LIMA, B. S; MORENO, M. F.; SOARES, L. F. G. Considering Non-functional Aspects in the Design of Hypermedia Authoring Tools. In: *ACM Symposium on Applied Computing* 2011. 2011.Taiwan.

LOWE, D. *Hypermedia and the Web: An Engineering Approach*. Nova Iorque. John Wiley & Sons, Inc. 2009.

LUCRÉDIO, D. *Uma abordagem orientada a modelos para reutilizacão de software*. Tese de Doutorado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação 2009.

LULA, M. M. M. *Um Processo Ágil para Especificação de Requisitos em Programas Interativos com foco em Roteiros de TV*. 97f. Dissertação (Mestrado), Universidade Federal da Paraíba, João Pessoa, 2011.

MAC WILLIAMS, C.; WAGES, R. Video conducting the Olympic Games 2008: The iTV field trial of the EU-IST project live. *In Proceedings of the 3th International Conference on Digital Interactive Media in Entertainment and Arts*. 2008. p. 436-40.

MAHONEY, M. S. Finding a history for software engineering. *IEEE Annals of the History of Compu-*

ting, 2004, v.26 (1), p.8–19.

MARQUES NETO, M. C. *Contribuições para a Modelagem de Aplicações Multimídia em TV digital interativa*. 2011. 170f. Tese (Doutorado). Universidade Federal da Bahia, Salvador, Brasil.

MARQUES NETO, M. C.; SANTOS, C. A. S. StoryToCode: A new model for specification of convergent interactive digital TV applications. *Journal of the Brazilian Computer Society*. 2010. p3-21.

MELLOR, S.J. *MDA Distilled* – Principles of Model Driven Architecture [S.l.]: Addison-Wesley, 2004.

MUCHALUAT-SAADE, D. C. *Relações em Linguagens de Autoria Hipermídia: Aumentando Reuso e Expressividade.* 2003.215f. Tese (Doutorado). Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

MUSBURGER, R.; KINDEM, G. *Introduction to Media Production: The Path to Digital Media Production*. Elsevier Science. 2009.

NACK, F. Capture and transfer of metadata during video production. In: *Proceedings of the ACM workshop on Multimedia for human communication: from capture to convey*. Nova Iorque, ACM. 2005. p. 17–20,

NASCIMENTO, L. M., GARCIA, V. C., MEIRA, S. R L. *SMADL: The Social Machines Architecture Description Language*. Proceedings of the Doctoral Symposium of the 5th International Conference on Software Language Engineering, 2012, p.35-52

OBJECT MANAGEMENT GROUP (OMG). *MDA Guide Version 1.0.1.* [S.l.], 2003. Disponível em: <http://www.omg.org>. Acesso em: 15 janeiro de 2013.

PATERNO, F.: *Model-Based Design and Evaluation of Interactive Applications*. Springer, Londres. 1999.

PLEEGER, S. Design and Analysis in Software Engineering, Part 1: The Language of Case Studies and Formal Experiments. *Software Engineering Notes*. 1994. 19(4): p.16–20.

PLEUSS, A.; VITZTHUM, A.; HUSSMANN, H. Integrating heterogeneous tools into model-centric development of interactive applications. In: *MODELS 2007*. LNCS, 2007. vol. 4735, p. 241–255. Springer, Heidelberg.

PLEUSS, A. *Model-driven development of interactive multimedia applications*. 2009. 287f. Tese (Doutorado), Universidade de Munique, Munique, Alemanha.

PLEUSS, A.; HUSSMANN, H. Model-Driven Development of Interactive Multimedia Applications with MML. *Studies in Computation Intelligence*. Springer. 2011. v. 340, p.199-218.

PLEUSS, A.; HAUPTMANN, B.; DHUNGANA D.; BOTTERWECK G. User interface engineering for software product lines: the dilemma between automation and usability. In: *ACM Symposium on Engineering Interactive Computing Systems*, 2012. p.25-34.

PONS, C; GIADINI, R; PEREZ, G. *Desarrollo de Software Dirigido por Modelos – Conceptos teóricos y su aplicación práctica*. EDULP. 2010.

PROTA, T. M. MoonDo-Eclipse: *Um Ambiente para Desenvolvimento de Aplicações Declarativas para o SBTVD*. 2012. 167f. Dissertação (Mestrado), Universidade Federal de Pernambuco, Recife.

RIBEIRO, N. *Multimédia: Tecnologias Interativas*. 2007. FCA. 3.a edição.

SAFA, L. The Making Of User-Interface Designer: A Proprietary DSM Tool, In: *Proceedings of DSM'07,* OOPSLA, 2007.

SAKIA R. M. The box-cox transformation technique: A review. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 41(2), 1992.

SÁNCHEZ, I. F. H. *Quadrados latinos com aplicações em engenharia de software*. 100f. Dissertação (Mestrado), Universidade Federal de Pernambuco, Recife, 2011

SARINHO, V. T; APOLINARIO, A. L. A Generative Programming Approach for Game Development. In: *VIII Simpósio Brasileiro de Jogos e Entretenimento Digital* (SBGames 2009). 2009. p.9-18. Rio de Janeiro.

SANTOS, J. A.; MUCHALUAT-SAADE, D. C. XTemPlate 3.0 language: easing the authoring of NCL programs for interactive digital TV. In: *Proceedings of the XV Brazilian Symposium on Multimedia and the Web (WebMedia)*. p.171-178, 2009.

SCHMIDT, D. C. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, v. 39, n. 2, p. 25–31, 2006.

SCHWALB, E. *iTV Handbook: Technologies & Standards Computers in Entertainment (CIE) - Theoretical and Practical Computer Applications in Entertainment* , ACM, New York, NY, USA, v. 2, 2004.

SELIC, B. MDA manifestations. *The European Journal for the Informatics Professional*, IX(2). 2008. p.12–16.

SHAW M. The coming-of-age of software architecture research. In: *Proceedings of the 23rd International Conference on Software Engineering*. Washington (ICSE 2001), DC, USA: IEEE Computer Society, 2001, p. 656.

SOARES, L.F.G.; RODRIGUES, R.F. *Nested Context Model 3.0 Part 1 — NCM Core.* 2005. Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, Rio de Janeiro.

SOARES NETO, C. S. *Autoria de Documentos Hipermídia Orientada a Templates*. 2010. Tese (Doutorado). Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

SOUZA JÚNIOR, P. J. DE. *LuaComp: Ferramenta de Autoria de Aplicaçoes para TV Digital*. Dissertação (Mestrado em Engenharia Elétrica). Universidade de Brasília, 2009.

STEINBERG D et al. *EMF: Eclipse Modeling Framework*. 2.a Edição, *Addison-Wesley Professional*, 2008.

SUNDAR, S. S. Theorizing interactivity's effects. *The Information Society*. 2004. v. 20, n. 5, p. 385–389.

SZEKELY, P. A. Retrospective and challenges for model-based interface development. In: *Bodart, F., Vanderdonckt, J. (eds.) DSV-IS, 1996*. p. 1–27. Springer, Heidelberg.

SZYPERSKI, C. *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley, 2002.

THOMAS, D. MDA: Revenge of the modelers or UML utopia? *IEEE Software*, v. 21, n. 3, p. 15–17, 2004.

TOLVANEN, J. P.; KELLY, S. Defining domain-specific modeling languages to automate product derivation: Collected experiences, In: *SPLC-Europe*, ser. LNCS, vol. 3714. Springer, 2005, p. 198–209.

URSU, M. F. et al. Interactive TV narratives: opportunities, progress, and challenges, In: *ACM Transactions on Multimedia Computing, Communications, and Applications* (TOMCCAP). 2008. v.4(4).

VANOIRBEEK, C et al. A Lightweight Framework for Authoring XML Multimedia Content on the Web. *Multimedia Tools and Applications (MTAP)*, Springer Netherlands, 2012.

VEIGA, E. G. *Um Modelo de Processo para o Desenvolvimento de Programas para TV Digital e Interativa baseado em Metodologias Ágeis*. Dissertação (Mestrado em Computação), Universidade de Salvador, Salvador, 2005.

WEISS, D.; LAI, C. *Software Product-Line Engineering: A Family-Based Software Development Process*, Addison-Wesley Professional, 1999.

WOHLIN C. et al. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.

W3C. World Wide Web Consortium, *XHTML 1.0 The Extensible HyperText Markup Language*, W3C Recommendation xhtml1-20020801, 2002

YIN, R. K. *CaseStudy Research: Design and Methods*. 1994. SAGE Publications.

YU, L.: Prototyping, Domain Specific Language, and Testing. *Journal of Engineering Letters*, Vol. 16, 2008.

APÊNDICE A

Modelo conceitual NCM

O NCM (Modelo de Contexto Aninhado, do inglês *Nested Contex Model*) (SOARES e RODRIGUES, 2005) é baseado nos conceitos usuais de nós e elos. Os nós representam os fragmentos de informação e os elos (links) são usados para definir relacionamento entre os nós.

No NCM, os grafos podem ser "aninhados", organizados de forma hierárquica, permitindo segmentar um documento multimídia, conforme desejado. Isso é feito através de "nós de composição" (também denominado de contexto). Dessa forma, no NCM, existem duas classes básicas de nós:

- **Nós de conteúdo ou de mídia**: associado a um elemento de mídia, como áudio, vídeo, texto, imagem, aplicação, tempo, etc.
- **Nós de composição ou contexto**: aninhamento de um ou mais nós de conteúdo e/ou composição.

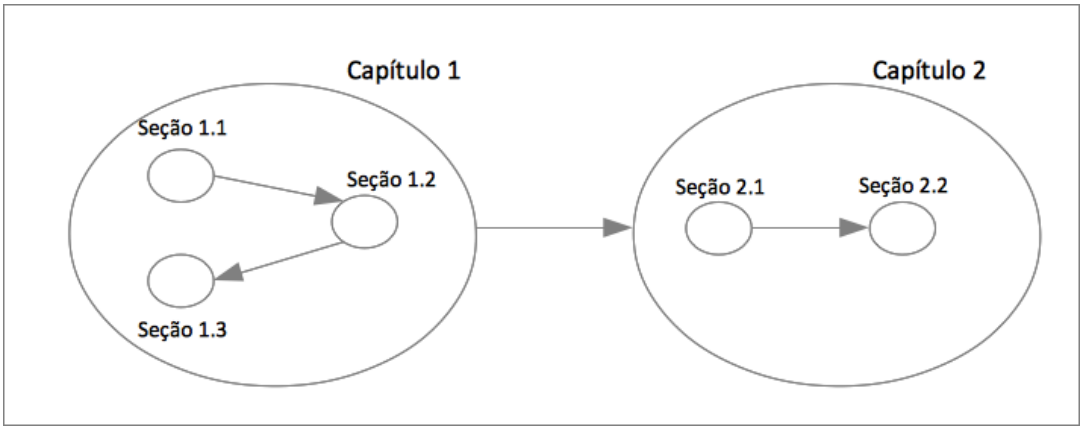


Figura A.1: Exemplo de estrutura no NCM composta de nós de composição, de conteúdo e elos Baseado em (SOARES e RODRIGUES, 2005).

A Figura A.1 ilustra esses conceitos no NCM. Nessa figura, capítulo 1 e capítulo 2, representam nós de composição, enquanto que cada Seção é um nó de conteúdo. A seguir uma descrição dos principais entidades e conceitos do NCM básico.

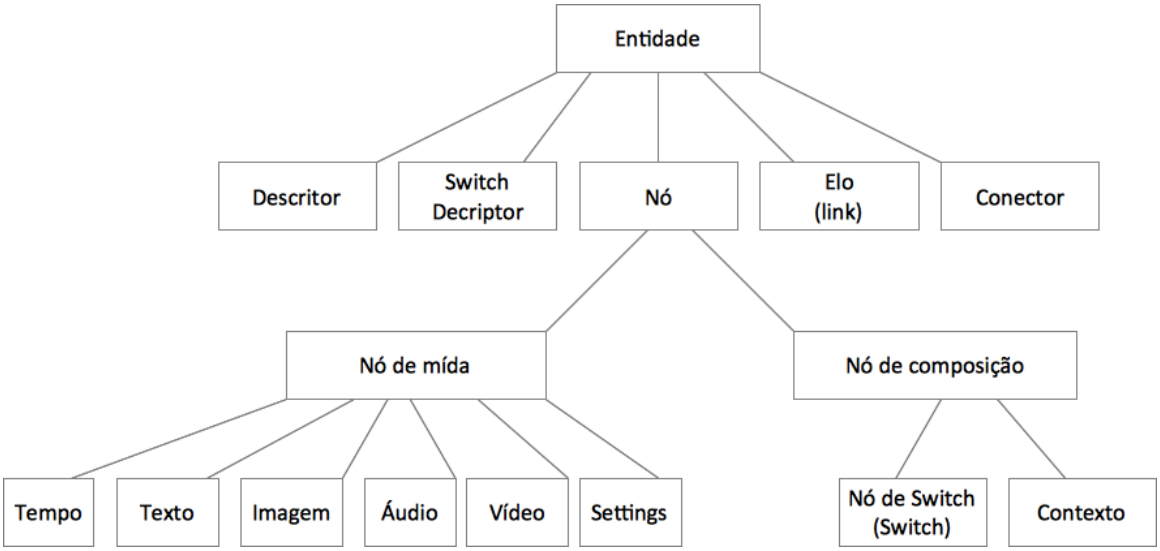


Figura A.2: Modelo conceitual NCL básico

Nós e ancoras - um nó (Figura A.2), também chamado objeto NCM, possui um identificador único, um conteúdo e um conjunto de âncoras. O conteúdo de um nó é formado por um conjunto de unidades de informação (UI). A noção exata do que constitui uma unidade de informação é parte da definição do nó e depende de sua especialização, como veremos adiante. Por exemplo, as UIs de um nó de vídeo podem ser frames de vídeo. Uma âncora é um trecho (subconjunto de UIs) de um nó. Uma âncora pode ser do tipo âncora de conteúdo ou âncora de propriedade. Uma âncora de conteúdo define um segmento de mídia (intervalo de tempo e/ou região do espaço) que poderá ser utilizado como ponto de ativação dos elos. Já as âncoras de propriedade se referem atributos de um nó que podem ser manipulados por elos. Exemplos de atributos são: volume em nó de áudio, coordenadas e dimensões de exibição em um nó de áudio, vídeo ou texto.

Nós de conteúdo (ou de mídia) - representam os usuais objetos de mídia. Eles devem ser especializados em subclasses para melhor definir a interpretação do conteúdo (texto, imagem, áudio, vídeo, objetos com código Java, Lua, etc.). Por exemplo, em um nó de vídeo, o conjunto de quadros em sequência compõem seu conteúdo. Existem dois tipos especiais de nó de conteúdo: (1) nó de tempo: representa um tempo absoluto (por exemplo, o horário de *Greenwich*) ou relativo (baseado em algum evento, como o início de apresentação do documento). Cada instante de tempo e uma UI desse nó. Dessa forma, intervalos de tempo podem ser definidos, como âncoras de conteúdo desse nó, o que permite que eventos sejam acionados em um tempo específico; (2) nó de settings: representa todas as variáveis que, são controladas pelo formatador (ferramenta responsável por apresentar o documento). Essas variáveis são representadas por atributos desses nós. Exemplo: tabelas SI em um ambiente de TV Digital.

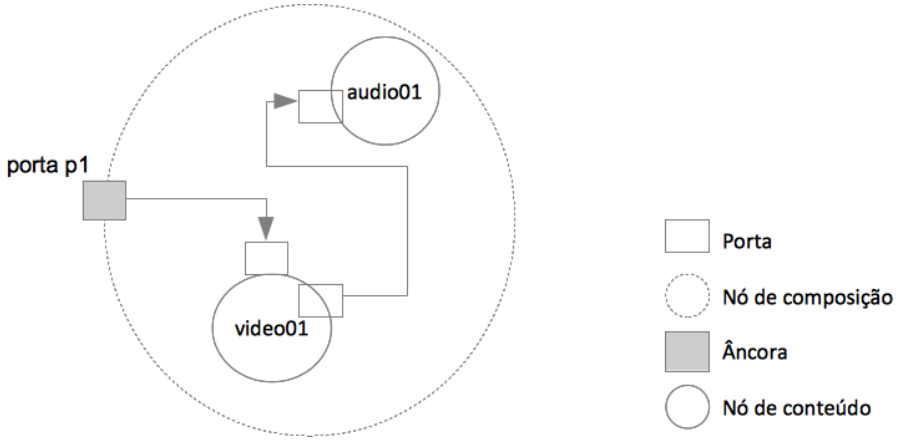


Figura A.3: Porta como ponto de entrada ao nó video01 (interno) ao contexto.

Nós de composição (ou contexto) - são utilizados para estruturar o documento. Seu conteúdo inclui um conjunto de nós de mídia e de composição, recursivamente. Dessa forma, os contextos podem ser "aninhados" para refletir a estrutura do documento (ver Figura A.1) e ajudar o autor do documento a organizar o mesmo (daí o nome, modelo de contexto aninhado). Um nó de contexto possui como atributos adicionais um conjunto de elos relacionando seus objetos. Além da lista de âncoras, um nó de composição (ou contexto) possui nas suas interfaces uma lista de portas. Uma porta oferece acesso externo ao conteúdo interno de um nó de contexto. Em outras palavras, para um elo (*link*) apontar para um nó interno ao nó contexto, esse nó de contexto deve possuir uma porta que leve ao nó interno. Como pode ser observado na Figura A.3. o nó interno *video01* só pode ser acessado através da interface *portap1*. Como não existe porta associada ao nó *audio01*, ele não pode ser acessado externamente. Em uma aplicação multimídia pode ser necessária a escolha entre nós (objetos). Por exemplo, dependendo da escolha do usuário fosse escolhido a legenda do vídeo em português ou inglês. Com o objetivo de permitir a especificação de alternativas a serem escolhidas, o NCM define uma entidade chamada de nó switch, um tipo de nó de composição.

Nó de *Switch* - Em uma aplicação multimídia pode ser necessária a escolha entre nós (objetos). Por exemplo, dependendo da escolha do usuário em que fosse escolhido a legenda do vídeo em português ou inglês. Com o objetivo de permitir a especificação de alternativas a serem escolhidas, o NCM define uma entidade chamada de nó switch, um tipo de nó de composição. O conteúdo de um nó switch é um conjunto de nós de conteúdo ou de contexto. Ele possui um atributo adicional, que define para cada nó interno, uma regra (*rule*) associada. As regras são definidas em uma lista ordenada. O primeiro nó que tenha sua regra avaliada como verdadeira deve ser eleito a alternativa relacionada. Nós Switch também possuem uma lista ordenada de portas-switch. Portas *swicth* constituem mais um ponto de interface. Elas definem um conjunto de mapeamentos para nós contidos no nó switch. A definição de portas-*swicth* permite que elos sejam ancorados em nó switch, independentemente do filho que será selecionado. O formatador deve decidir quando avaliar regras e analisar nós switch. Um nó

switch, como podemos ver, é o suporte de NCM para definição de alternativas de conteúdos. Como podemos ver até então, a Figura A.4 apresenta os tipos de interfaces de um nó NCM.

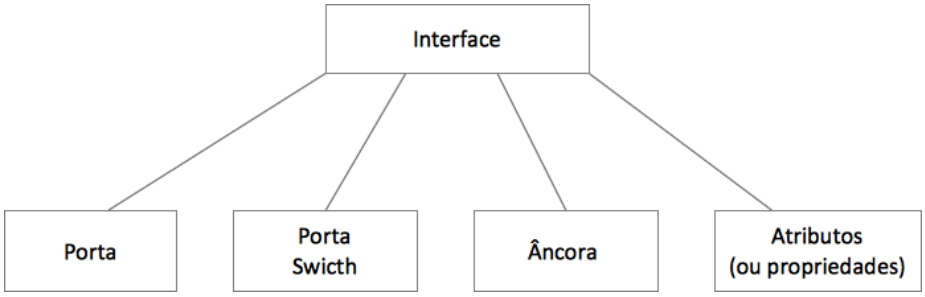


Figura A.4: Interfaces de nós NCM

Descritores - A forma como um objeto de mídia deve ser exibido, onde e por quem, é especificada na entidade descritor. É nessa entidade que são dadas as características iniciais da apresentação. Iniciais porque elas podem mudar com o tempo. Por exemplo, o posicionamento de uma imagem pode mudar. O descritor também define onde o objeto deve ser apresentado, incluindo o dispositivo de exibição. A forma como nó será exibido pode depender da situação. Por exemplo, a resolução de um vídeo de ser reduzida se o dispositivo de exibição for um celular ao invés de uma televisão de alta definição. Assim, podemos associar vários descritores a um mesmo nó de mídia, que serão escolhidos de acordo com uma regra (*rule*). O conjunto de descritores alternativos deve ser definido dentro da entidade switch de descritores, que permite o conteúdo deve ser apresentado de forma adaptada ao contexto de exibição.

Elos (links) - Os elos (*links*) associam nós através de conectores (*connectors*), que definem a semântica da associação entre os nós, independentemente dos nós que irão interagir. Um elo é composto por um conector e por ligações (*binds*), que indicam as interfaces dos componentes (nós de conteúdo ou de contexto) envolvidos no elo e o papel (role) de cada nó de acordo com a semântica do conector.A Figura A.5 mostra um exemplo de dois elos utilizando o mesmo conector R.

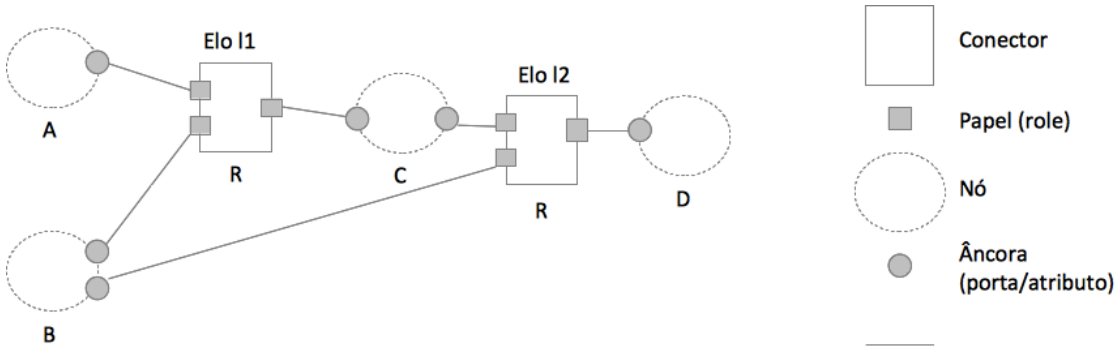


Figura A.5: Exemplo de Elo usando o conector R

Conectores: conforme comentado anteriormente, os conectores definem a semântica da associação (relação) entre os nós, independentemente dos nós que irão interagir. A versão 3.0 do NCM concentra esforços na especificação de relações de sincronização espacial e temporal. Um conector define

uma relação através de seus papéis (*roles*) e sua cola (*glue*), que especifica como os papéis interagem. No NCM, os conectores podem ser causais e de restrição. Numa relação (*conector*) causal uma condição deve ser satisfeita para disparar uma ação. Um exemplo de relação causal é uma relação que inicia um nó quando outro nó termina sua apresentação. Na relação (*conector*) de restrição, por outro lado, nenhuma causalidade é envolvida. Por exemplo, suponha uma restrição que defina que um nó deve terminar sua apresentação no mesmo tempo em que outro nó começa a dele. A ocorrência de uma apresentação sem a ocorrência de outra também satisfaz a restrição, que define que se e somente se esses dois nós forem apresentados, seus tempos de fim e início devem coincidir. Para TV Digital, a linguagem NCL utiliza apenas conectores causais (a cola é sempre de causalidade).

Eventos - um evento é uma ocorrência no tempo que pode ser instantânea ou ter uma duração mensurável. O NCM define algumas classes básicas de eventos que podem ser estendidas: eventos de exibição, composição, seleção, superposição, arraste, foco e atribuição. Um evento de exibição (ou apresentação): representa a exibição de uma âncora, de um nó de conteúdo (segmento de mídia) em uma dada perspectiva (histórico de navegação) e seguindo as diretrizes de um descritor (perspectivas e descritores diferentes podem gerar diferentes eventos de exibição).

- Eventos de composição: apresentam a estrutura de um nó de composição (ou mapa da composição).
- Eventos de seleção: representa a seleção de uma âncora de um nó de conteúdo. Forma mais comum de selecionar é através de dispositivos de entrada como mouse, teclado, controle remoto, etc.
- Eventos de superposição, arraste e foco: também representam a ação de interação sobre uma âncora de um nó de conteúdo.
- Evento de atribuição: refere-se a mudança de um atributo de um nó ou de seu comportamento (definido no descritor).

No NCM, cada evento define uma máquina de estados que deve ser mantida pelo formatador. A Figura A.6 apresenta a máquina de estados dos eventos NCM.

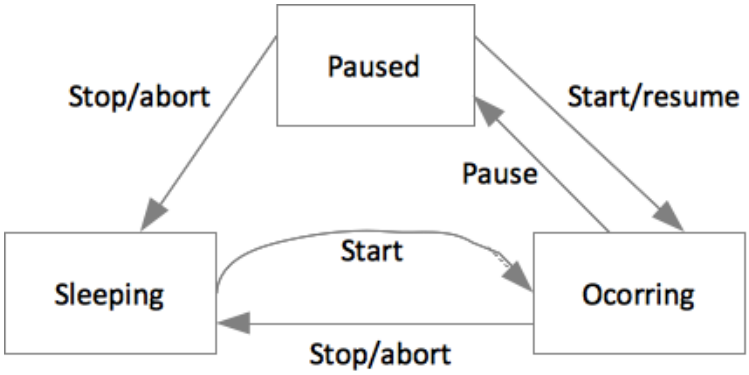


Figura A.6: Máquina de estados NCM

Um evento NCM pode estar em um dos seguintes estados: dormindo (*sleeping*), ocorrendo (*ocurring*) e pausado (*paused*). Todo evento possui um atributo ocorrências que conta o número de vezes que o evento muda de ocorrendo para dormindo durante a apresentação do documento. Eventos de exibição e atribuição também possuem um atributo repetições que determina o número de vezes seguidas que o evento deve ocorrer. Um evento inicia no estado dormindo. Ao iniciar, o evento passa para o estado ocorrendo. Se o evento for temporariamente suspenso, ele vai para o estado pausado e permanece enquanto a situação durar. Ao final do evento, ele retorna para o estado dormindo. A transição de estados de um evento é comumente usada para definir conectores. A sincronização espacial/temporal no NCM é baseada em eventos e não em marcas de tempos (*timestamp*) ou outros modelos de sincronização (redes Petri, hierárquica, etc.).