Delcino Picinin Júnior

UMA METODOLOGIA E UM AMBIENTE MDE PARA A VERIFICAÇÃO DE APLICAÇÕES HIPERMÍDIA

Tese submetida ao Programa de Pós-Graduação
em Engenharia e Sistemas para a obtenção
do Grau de Doutor.
Orientador: Prof. Dr. Jean-Marie Farines
Coorientador: Prof. Dr. Cristian Koliver

Florianópolis

2016

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Picinin Júnior, Delcino
Uma metodologia e um ambiente MDE para a verificação de Aplicações Hipermídia / Delcino Picinin Júnior ; orientador, Jean-Marie Farines ; coorientador, Cristian Koliver. - Florianópolis, SC, 2016.
242 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de Automação e Sistemas. 2. Aplicações Hipermídia. 3. Verificação Formal. 4. Engenharia Dirigida a Modelos. I. Farines, Jean-Marie. II. Koliver, Cristian. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Delcino Picinin Júnior

UMA METODOLOGIA E UM AMBIENTE MDE PARA A VERIFICAÇÃO DE APLICAÇÕES HIPERMÍDIA

Esta Tese foi julgada adequada como **TESE DE DOUTORADO** no Curso de Doutorado em Engenharia de Automação e Sistemas e aprovada em sua forma final pela Banca Examinadora designada.

Dr. Rômulo Silva de Oliveira
Coordenador do PPGEAS

Dr. Jean-Marie Farines
Orientador

Dr. Cristian Koliver
Coorientador

Banca Examinadora:

Dr. Jean-Marie Farines
Orientador

Dra. Debora Christina Muchaluat Saade

Dr. Wanderley Lopes de Souza

Dr. Celso Alberto Saibel Santos

Dr. Roberto Willrich

Dr. Max Hering de Queiroz

Este trabalho é dedicado aos meus colegas de classe, orientadores e aos meus queridos pais.

AGRADECIMENTOS

Para realizar este trabalho, foi necessária a ajuda de diversas pessoas, para as quais quero registrar meus agradecimentos.

Agradeço, inicialmente, a Deus, sem o qual nada seria possível.

Agradeço aos meus pais e familiares pela ajuda, paciência e apoio.

Aos professores do PPGEAS, pela transmissão do conhecimento e pela dedicação.

Em especial, ao Prof. Dr. Jean-Marie Farines por ter me aceito como seu orientando e pela ótima orientação durante este período.

Ao meu co-orientador Prof. Dr Cristian Koliver por ter aceito me co-orientar neste trabalho, pela amizade e disposição em me ajudar sempre que necessário.

Ao Prof. Dr. Celso Alberto Saibel Santos pelo apoio e valiosas sugestões e ajudas durante a realização de todo o trabalho.

Em especial ao professor Luiz Fernando Gomes Soares (*in memoriam*), que participou do exame de qualificação e deu ótimas sugestões ao desenvolvimento do trabalho.

A todos os meus colegas e amigos do PPGEAS, por todos os momentos agradáveis de convívio e pelo apoio durante esta trajetória.

E, finalizando, aos meus colegas do IFSC, que sempre me deram apoio e um bom ambiente de trabalho.

RESUMO

No desenvolvimento de aplicações hipermídia, o projetista pode erroneamente inserir comportamentos indesejados. Metodologias baseadas em teste ou análise de linha temporal para verificar a corretude de aplicações são limitadas, por não serem exaustivas e serem consumidoras de tempo. Outra alternativa é a utilização de metodologias baseadas em verificação formal, que permitem uma análise exaustiva e mais rápida da aplicação. A verificação formal requer que a aplicação e os comportamentos a serem verificados estejam representados em linguagens formais, de difícil aprendizagem por um projetista de aplicação hipermídia. O presente trabalho propõe uma metodologia baseada no uso de verificação formal por *model-checking*, a partir de uma representação da aplicação, das propriedades a serem verificadas e do diagnostico de eventuais erros, ambos em linguagens e representações de fácil uso e entendimento para o projetista destas aplicações. Essa metodologia é dividida em quatro fases: Modelagem, Transformação, Verificação e Diagnóstico/Correção. Inicialmente, o projetista codifica sua aplicação em alguma linguagem de domínio específico (por exemplo, NCL ou SMIL), e especifica os comportamentos desejados a serem verificados numa linguagem de descrição simples, proposta neste trabalho. A seguir, essas descrições das aplicações e comportamentos são transformadas, seguindo a abordagem MDE (*Model Driven Engineering*), nos modelos formais utilizados na verificação. Em caso de algum comportamento desejado não ser satisfeito, a ferramenta de *model-checking* oferece um contraexemplo que, após transformação, é apresentado na forma de uma linha de tempo, permitindo diagnosticar a origem do erro e fornecer informações para a sua correção. Para apoiar a metodologia proposta, foi construído um protótipo de um ambiente de desenvolvimento, no qual o projetista pode verificar o comportamento de sua aplicação. As avaliações da metodologia e de seu ambiente, realizadas em diversas aplicações hipermídia mostram suas potencialidades de uso para aplicações mais complexas e no caso de edição "ao vivo" .

Palavras-chave: Aplicações Hipermídia. Verificação Formal. Engenharia Dirigida a Modelos.

ABSTRACT

In the development of hypermedia applications, the designer can mistakenly insert undesirable behaviors. Methodologies based on tests or timeline analysis to verify the correctness of applications are limited because they are not exhaustive and are time consuming. Another alternative is the use of methodologies based on formal verification, allowing an exhaustive and more fast analysis of the application. Formal verification requires that the application and behavior to be verified are represented in formal languages, which are difficult to learn by a hypermedia application designer. This work proposes a methodology based on the use of formal verification by model-checking, from an application representation, the properties to be verified and the diagnosis of errors, both in languages and representations of easy use and understanding by designer of these applications. This methodology is divided into four phases: Modeling, Transformation, Verification and Diagnosis/Correction. Initially, the designer encodes his application in any domain specific language (eg, NCL or SMIL), and specifies the desired behaviors to be checked in a simple description language proposed in this work. Then these descriptions of applications and behaviors are transformed, following the MDE approach (Model Driven Engineering), in formal models used for verification. If some desired behavior is not satisfied, the model-checking tool provides a counterexample that, after processing, is presented as a timeline, allowing to diagnose the source of the error and provide information for its correction. To support the proposed methodology, a prototype development environment was built, in which the designer can verify the behavior of your application. Evaluations of the methodology and its environment, performed in several hypermedia applications, showed their potential of use for more complex applications and in the case of editing "live".

Keywords: Hypermedia Applications. Formal Verification. Model Driven Engineering

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CSS	*Cascading Style Sheets*;
CTL	*Computation Tree Logic*;
DOM	*Document Object Model*;
DSML	*Domain-Specific Modeling Languages*;
EP	*Editor de Propriedades*;
GI	*Grafo Intermediário*;
HTML5	*Hypertext Markup Language*;
LP	*Linguagem de Propriedades*;
LTL	*Linear Temporal Logic*;
MDE	*Model-driven Engineering*;
NCL	*Nested Context Language* (Linguagem de Contextos Aninhados);
NCM	*Nested Context Model* (Modelo de Contextos Aninhados);
SMIL	*Synchronized Multimedia Integration Language* (Linguagem de Integração de Multimídia Sincronizada);
TTS	*Sistema de Transição Temporizado*;
TVDI	*Televisão Digital Interativa*;
W3C	*World Wide Web Consortium*;
WHATWG	*Web Hypertext Application Technology Working Group*; e
XML	*Extensible Markup Language*.

SUMÁRIO

1 INTRODUÇÃO

O progresso tecnológico em eletrônica e computação aumenta, dia após dia, a disponibilidade de diferentes tipos de informação. Essas informações podem ser acessadas por meio de vários dispositivos interconectados e distribuídos: computadores, *tablets*, celulares, TVs, navegadores GPS, entre outros. O uso de aplicações hipermídia é uma boa alternativa para permitir a uma gama de usuários com idades e perfis de conhecimentos distintos, o acesso a uma grande quantidade de informação, de maneira fácil.

Uma aplicação hipermídia é a união de diferentes tipos de mídia - vídeo, imagem, texto e áudio - com a possibilidade de interações do usuário. Essas aplicações, por possuírem interatividade, podem ter comportamentos não-lineares, uma vez que diferentes usuários podem tomar caminhos diferentes, de acordo com os seus interesses (BURTON; MOORE; HOLMES, 1995).

Aplicações hipermídia são codificadas comumente por alguma linguagem hipermídia de conhecimento do projetista. Essa linguagem segue, usualmente, o paradigma declarativo, e permite definir um conjunto de características da aplicação, tais como: a ordem e o tempo de apresentação das mídias (recursos temporais), os dispositivos de visualização e o posicionamento espacial das mídias nesses dispositivos (características espaciais) e as possíveis interações que o usuário pode gerar durante a apresentação da aplicação. Entre as linguagens hipermídia existentes, estão: SMIL (BULTERMAN, 2008), NCL (SOARES; BARBOSA, 2009), HTLM5 (PILGRIM, 2010).

No desenvolvimento de aplicações hipermídia, o projetista deve considerar algumas questões:

- *Hardware* - características do dispositivo onde a aplicação será exibida, podendo considerar, também, a possibilidade dela ser exibida em múltiplos dispositivos;
- *Design* - questões relativas à qualidade visual da aplicação, como, por exemplo o posicionamento das mídias no dispositivo de exibição e o tempo mínimo de exibição de determinadas mídias;
- Comportamento - as aplicações hipermídia devem comportar-se conforme o desejo do projetista, sem apresentarem comportamentos indesejados;
- Interatividade - a inclusão de interatividade torna a aplicação mais atrativa, mas também torna seu desenvolvimento mais complexo; e
- Edição “ao vivo” - aplicações apresentadas “ao vivo” são mais atrativas aos usuários, mas no processo de edição “ao vivo” o projetista tem limites de tempo para realizar a edição.

Neste processo de construção de aplicações hipermídia, o projetista pode, erroneamente, inserir comportamentos indesejados na aplicação. O desenvolvimento de aplicações totalmente corretas não é uma atividade trivial para cientistas da computação ou engenheiros, quanto mais para projetista de aplicações hipermídia, que normalmente são profissionais formados na área de jornalismo ou publicidade e propaganda.

Erros de sintaxe são facilmente apontados por boas ferramentas de autoria, mas erros de lógica que podem causar comportamentos indesejados são mais complexos de serem identificados. Quando a aplicação possui eventos não-determinísticos (tais como as interações do usuário), ela torna-se não-linear[1]. A não-linearidade em uma aplicação hipermídia aumenta sua complexidade e, consequentemente, a possibilidade de inserção de erros durante a fase de desenvolvimento, bem como a dificuldade de identificá-los.

Normalmente o projetista usa uma abordagem baseada em testes, onde, após editar a primeira versão da aplicação, ele efetua uma bateria de testes usando uma ferramenta de apresentação (*player*). Essa abordagem, por não ser exaustiva, não garante que a aplicação esteja livre de comportamentos indesejados. O resultado obtido em testes auxilia muito pouco o projetista no processo de correção dos comportamentos indesejados.

1.1 OBJETIVO

Com o intuito de auxiliar os projetistas, nesse trabalho propõe-se uma metodologia a ser adotada no desenvolvimento de aplicações hipermídia.

A metodologia proposta usa verificação formal, permitindo que todos os possíveis comportamentos da aplicação sejam verificados.

Visando facilitar o processo de correção de erro, para cada comportamento considerado indesejado, a metodologia apresenta ao projetista a sequência de ações que levaram a tal comportamento. De posse dessas informações, o projetista pode corrigir erros de projeto e implementação ainda na fase de desenvolvimento.

A conformidade entre a aplicação hipermídia e o modelo formal, usado na verificação formal, é obtida adotando-se uma abordagem MDE (*Engenharia Dirigida a Modelo*) baseada nos princípios de transformação de modelos.

A metodologia proposta não envolve questões de *hardware* e *design*, estando limitado a questões comportamentais, envolvendo interatividade.

Para possibilitar o uso dessa metodologia, foi construído um ambiente

[1] Compreende-se não-linearidade como sendo a possibilidade da aplicação possuir mais de uma sequência de apresentação, onde a sequência escolhida depende diretamente das interações do usuário, bem como do momento em que tais interações ocorrem.

de desenvolvimento. É pertinente ressaltar que o ambiente proposto requer que as aplicações estejas escritas na versão reduzida do NCL (NCL *Raw Profile*), ou em SMIL, usando estruturas de causalidade.

Como principais contribuições do trabalho, pode-se destacar a construção da metodologia e do ambiente desenvolvido, que permitem: (1) garantir que o projetista possa verificar formalmente o comportamento de sua aplicação, mantendo o uso das linguagens de projeto na qual ele está acostumado; (2) a verificação dos diferentes tipos de comportamentos (temporal, causal e espacial); (3) a ajuda para o projetista na detecção e correção de erros de projeto; e (4) a redução do tamanho do modelo formal, diminuindo assim o tempo necessário para o processo de verificação.

1.2 ORGANIZAÇÃO DA TESE

Esta tese está organizada em 7 capítulos. O Capítulo 2 inicialmente apresenta uma visão geral sobre aplicações hipermídia, o desenvolvimento dessas aplicações e as linguagens que podem ser adotadas. Finalizando, o capítulo apresenta os principais problemas encontrados quando se pretende desenvolver aplicações atrativas e sem comportamentos indesejados, bem como uma visão geral sobre os trabalhos existentes.

O Capítulo 3 tem por objetivo apresentar a metodologia de desenvolvimento proposta. Inicialmente, ele apresenta os principais requisitos a serem alcançados. Em um segundo momento, ele apresenta uma descrição geral da metodologia proposta. Na sequência, ele apresenta uma contextualização sobre as tecnologias adotadas no desenvolvimento do trabalho. O capítulo finaliza apresentando o ambiente desenvolvido que visa dar suporte à metodologia proposta.

O Capídulo 4 tem como foco apresentar as transformações de linguagem hipermídia e linguagem de especificação de propriedades para linguagens adotadas por ferramentas de verificação formal.

O Capítulo 5 apresenta, de maneira mais detalhada, os elementos que formam o ambiente de desenvolvimento implementado neste trabalho.

O Capítulo 6 tem por objetivo apresentar a avaliação da metodologia proposta adotando o ambiente de desenvolvimento implementado.

Finalmente, o Capítulo 7 apresenta as conclusões do trabalho e as perspectivas de trabalhos futuros.

No apêndice deste trabalho encontram-se: (1) informações relacionadas ao uso da metodologia e ao ambiente na verificação de aplicações codificadas na linguagem SMIL; (2) códigos NCL das aplicações usadas na avaliação da metodologia proposta; (3) uma apresentação geral sobre lógica

temporal; (4) representação de observadores, na forma de autômatos, usados pela metodologia proposta; e (5) um manual de instalação do ambiente proposto e desenvolvido.

2 APLICAÇÕES HIPERMÍDIA

Este capítulo fornece ao leitor uma visão geral sobre desenvolvimento de aplicações hipermídia. A parte inicial aborda algumas definições e classificações referentes a aplicações hipermídia, mostrando os diferentes tipos de aplicações e suas características. Em um segundo momento, são apresentadas algumas considerações sobre o desenvolvimento de aplicações hipermídia. Na sequência, apresentam-se as principais linguagens usadas na codificação dessas aplicações. Finalizando este capítulo, são apresentados os principais problemas a serem resolvidos, visando a construção de aplicações hipermídia corretas.

2.1 CARACTERÍSTICAS

Seguindo uma sequência cronológica, no início do desenvolvimento dos dispositivos eletrônicos e das redes de comunicação, as informações eram passadas aos usuário de maneira mais textual, sem muitas figuras ou vídeos. Tal limitação estava diretamente associada aos baixos recursos de processamento e comunicação existentes na época.

Com a evolução do poder de processamento dos dispositivos e da capacidade de comunicação nas redes, os projetistas passaram a inserir diferentes tipos de mídias junto com os textos, tais como áudios e vídeos, difundindo sistemas chamados "multimídia". Uma aplicação multimídia é composta pela união de diferentes tipos de mídia, dispostas de modo a propiciar ao usuário o acesso às mídias de maneira atraente e organizada. Uma aplicação multimídia tem sua apresentação linear, ou seja, as mídias são sempre apresentadas seguindo uma única ordem temporal.

Com o passar do tempo, as aplicações evoluíram passando a oferecer interatividade. Nesse momento, o usuário deixou de ser totalmente passivo, podendo interagir com a aplicação que está sendo apresentada e a apresentação deixou de ser linear. Para esse tipo de aplicação, adota-se o termo "hipermídia", representando aplicação multimídia interativa.

Dependendo do nível de abstração, o termo "aplicações hipermídia" pode ser muito amplo, abrangendo diferentes tipos de aplicações, como hipertexto, *site* WEB, TVDI (TV Digital Interativa) entre outros.

Um hipertexto apresenta informações escritas, organizadas de tal maneira que o leitor pode interativamente escolher vários caminhos, a partir de sequências possíveis (*hyperlinks*), sem estar preso a um encadeamento linear único.

Um *site* Web pode possuir todas as características de uma aplicação hipermídia, entretanto pode ser bem mais complexo. Adotando-se linguagens como PHP, voltadas ao desenvolvimento de *sites* dinâmicos, pode-se construir sistemas computacionais completos, transcendendo a ideia inicial de aplicações hipermídia.

Outro tipo de aplicação hipermídia são os programas para TVDI, desenvolvidos para serem apresentadas usando a tecnologia de TV Digital Interativa. A TVDI, diferentemente da tradicional televisão analógica, segue o conceito de comunicação bidirecional, criando a possibilidade de uma interação do usuário com a informação que está sendo exibida. A interação entre o usuário e a emissora é feita através de um canal de interatividade.

Embora a TVDI apresente melhora na qualidade da imagem, devido ao sinal ser digital, a interatividade é o grande diferencial diante da TV convencional. Dependendo da presença de canal de retorno, diferentes níveis de interatividade podem ser observados (WAISMAN, 2006):

- interatividade local - o canal de retorno não é utilizado. O usuário não pode enviar dados ao servidor, interagindo apenas com a aplicação que é carregada localmente;
- interatividade intermitente - o canal de retorno é usado apenas durante o envio de um determinado fluxo de dados, sendo a conexão fechada na sequência; e
- interatividade total - o usuário permanece todo o tempo conectado ao canal de retorno, podendo enviar dados ao servidor a qualquer momento.

Quanto maior o nível de interatividade, mais atrativa e funcional fica a aplicação TVDI.

Existem diferentes tipos de aplicações TVDI, podendo-se destacar:

- Programas Educativos - o aluno pode interagir acessando diferentes materiais e detalhando melhor o conteúdo estudado. Ele pode responder questionários, visando avaliar seu desempenho no processo de aprendizagem;
- Programas de Auditório - o usuário pode interagir com o apresentador, dando sua opinião e indicando a direção que deseja que o programa siga; e
- Programas de Esportes - o usuário pode acompanhar o programa, dando sua opinião e acessando mídias alternativas com informações complementares.

No que se refere à edição de aplicações TVDI, pode-se classificar as aplicações em duas categorias:

- programação pré-editada - o projetista codifica sua aplicação antes de disponibilizá-la para os usuários, tendo um determinado período de tempo para realizar testes e verificações; e
- programação “ao vivo” - o projetista codifica parte de sua aplicação durante a exibição. Esse tipo de desenvolvimento normalmente ocorre quando decisões de projeto são tomadas em decorrência do comportamento do usuário ou da informação que está sendo apresentada. Codificação durante a exibição pode ocorrer em programas de TVDI “ao vivo”. Nesse tipo de desenvolvimento existe uma restrição no tempo para realizar testes e verificações.

A contínua evolução tecnológica tem dado suporte ao desenvolvimento de novos meios de disponibilizar informações, bem como de comunicação e interação entre as pessoas e o mundo. Um desses meios consiste no uso de mídias multi-sensoriais, conhecido pelo termo “MulSeMedia”. Uma aplicação MulSeMedia combina as mídias tradicionais, interatividade e objetos que ativam os sentidos humanos, tais como: frio, calor, olfato e tato. Como exemplos de aplicações que podem usar MulSeMedia pode-se destacar: jogos, aplicações de realidade virtual e filmes (YUAN et al., 2015).

2.2 LINGUAGENS HIPERMÍDIA

Uma aplicação hipermídia é, em geral, codificada a partir de uma linguagem de domínio específico na área de hipermídia. Essa linguagem permite especificar o posicionamento das mídias na tela, bem como os momentos em que as mídias devem ser apresentadas. Esses momentos podem estar associados a diferentes fatores, como, por exemplo: relacionamentos entre mídias, tempo de duração de uma mídia, posicionamento da mídia ante dispositivos, ambiente onde a aplicação está sendo exibida e interações com o usuário.

As diferentes linguagens que podem ser adotadas no desenvolvimento de aplicações hipermídia podem ser classificadas segundo três paradigmas:

- Declarativo - a aplicação é codificada usando uma linguagem declarativa, tais como: NCL, SMIL e HTM5. Essas linguagens são mais fáceis de serem usadas por projetistas sem um amplo conhecimento de informática e lógica de programação;
- Imperativo - codifica-se a aplicação usando linguagens como: Java, Lua e JavaScript. Essas linguagens permitem construir aplicações mais sofisticadas, assemelhando-se mais a um programa de computador. Não são linguagens do domínio específico de aplicações hipermídia, embora

possam ser usadas para essa finalidade. Elas exigem do projetista um conhecimento aprofundado de lógica de programação; e

- Declarativo e Imperativo - é uma abordagem híbrida, onde adotam-se os dois paradigmas. Nessa abordagem, o projetista constrói algumas rotinas usando uma linguagem imperativa, e usa essas rotinas dentro de seu código declarativo como sendo um tipo de mídia. O projetista que constrói o código declarativo pode usar rotinas imperativas codificadas por outras pessoas, usando o princípio de caixa preta.

O presente trabalho tem como foco o paradigma declarativo, mais voltado para profissionais da área de jornalismo e demais profissionais da área de comunicação. Aplicações e rotinas em linguagens imperativas normalmente são codificadas por programadores de computador.

Nessa seção são apresentadas as linguagens declarativas: SMIL *(Synchronized Multimedia Integration Language)* e NCL *(Nested Context Language)*, bem como a linguagem imperativa Lua, usada em conjunto com a linguagem NCL, no desenvolvimento de aplicações para TVDI .

2.2.1 NCL *(Nested Context Language)*

A linguagem NCL (SOARES; BARBOSA, 2009) é uma linguagem declarativa baseada no modelo conceitual NCM (*Nested Context Model*). O NCM é um modelo criado no laboratório TeleMídia da PUC-Rio visando a codificação de aplicações hipermídia. Essa linguagem foi desenvolvida utilizando uma estrutura modular, seguindo os princípios adotados pelo W3C (*World Wide Web Consortium*).

A definição de documento no NCM é baseada no conceito de nós e elos, onde os nós são as mídias em NCL, ou um subnível hierárquico (nó de composição), e os elos são as ligações entre os nós. A Figura 1(a) apresenta uma aplicação NCL composta por nove nós de mídia e nove elos; já a Figura 1(b) apresenta uma aplicação com dois nós de composição.

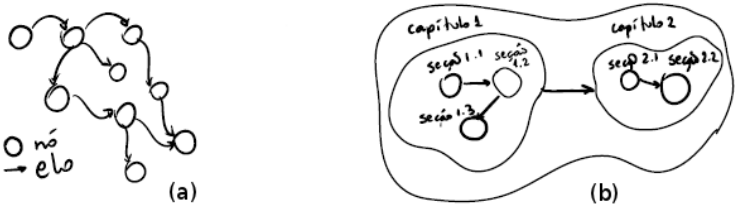


Figura 1 – (a) Nós de mídia; (b) Nó de composição (SOARES; BARBOSA, 2009)

Uma aplicação NCL é um arquivo escrito em XML *(Extensible Markup Language)* (FAWCETT; AYERS; QUIN, 2012) composto por um cabeçalho e um corpo (SOARES; BARBOSA, 2009). A Figura 2 apresenta os elementos que compõem um documento NCL, os quais serão detalhados no decorrer desta seção.

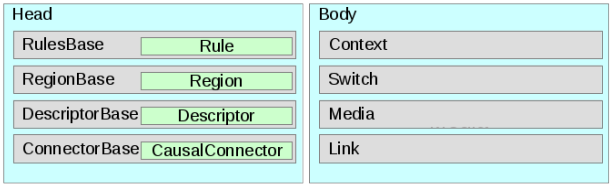


Figura 2 – Estrutura Documento NCL

NCL é atualmente um padrão ITU-T para serviços de TV usando o protocolo IP (IPTV) (ITU-T Recommendation H.761, 2009). Ele é usado também no padrão de transmissão de TV Digital Terrestre desenvolvido no Brasil (ISDB-T) (NBR15606-1, 2008).

Na Figura 3, apresenta-se uma versão simplificada do metamodelo da linguagem NCL, nesta simplificação não aparecem alguns atributos, tais como portas e propriedades. No trabalho será usada a versão NCL-RAW, que será descrita neste texto.

Neste metamodelo, constata-se que a aplicação hipermídia possui um componente chamado `NCL`, que contém dois sub-elementos: `Head` e `Body`.

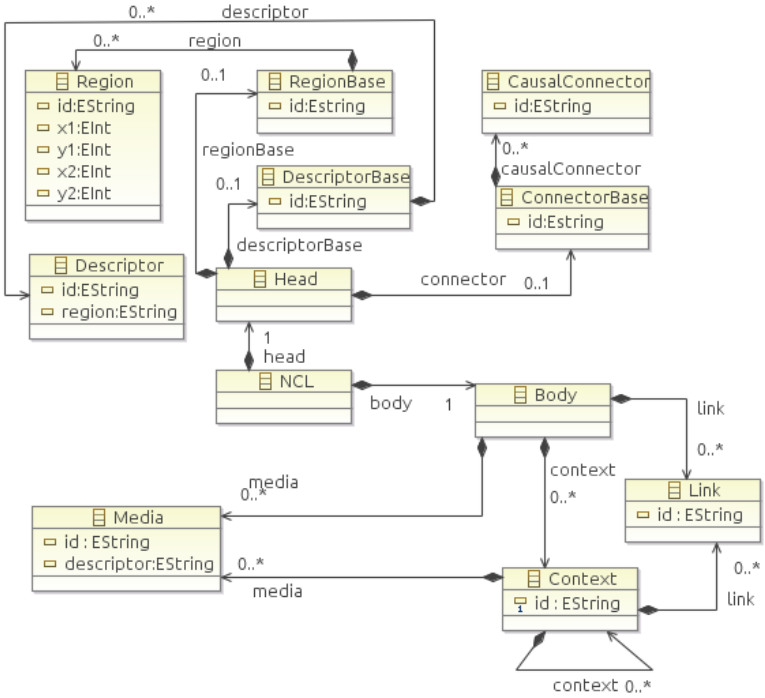


Figura 3 – Metamodelo NCL

`Head` - Cabeçalho da Aplicação

Dentro da área do cabeçalho de uma aplicação NCL existem algumas subdivisões:

- **`Region`** - área da tela onde uma mídia pode ser apresentada. Em NCL declaram-se várias regiões dentro da estrutura **`<RegionBase>`**;
- **`Descriptor`** - descritores contendo informações sobre uma mídia, tais como: a região onde essa mídia será apresentada, a duração da apresentação dessa mídia, o volume do som (se for uma mídia com áudio) entre outras. Declaram-se vários descritores dentro da estrutura **`<DescriptorBase>`**;
- **`CausalConnector`**- conectores que definem o comportamento dos elos da aplicação, como, por exemplo, existência de um determinado

atraso. O mesmo conector pode ser usado para vários elos. A declaração de conectores deve ser dentro da estrutura **<ConnectorBase>**;

- **Rule** - regras que definem um critério de escolha entre alternativas; as regras baseiam-se em informações do ambiente. Uma regra pode ser adotada, por exemplo, na escolha de qual descritor deve ser associado a uma mídia, ou na escolha entre a apresentação de duas mídias. As regras devem ser declaradas dentro da estrutura **<RuleBase>**.

Os conectores definem relações entre um ou mais nós de origem e um ou mais nós de destino. Cada relação tem uma ou mais condições de ativação e uma ou mais ações ativadas por essas condições. As condições são associados a conectores pelo parâmetro **role**. Entre as principais condições destacam-se:

- **onBegin** - condição satisfeita quando a apresentação da mídia ou âncora iniciar (momento da ativação);
- **onEnd** - condição satisfeita quando a apresentação da mídia ou âncora terminar (momento da desativação);
- **onKeySelection** - condição satisfeita quando a mídia ou âncora sofrer uma interação do usuário (momento da seleção);
- **onPause** - condição satisfeita quando a apresentação da mídia ou âncora sofrer uma pausa; e
- **onAbort** - condição satisfeita quando a apresentação da mídia ou âncora for abortada.

As ações, também associadas aos conectores pelo parâmetro **role**, podem ser:

- **set** - atribui um valor a um atributo da mídia (por exemplo, dimensão da região);
- **start** - inicia a apresentação do nó de mídia (ativação);
- **stop** - termina a apresentação do nó de mídia (desativação);
- **abort** - aborta a apresentação do nó de mídia;
- **pause** - pausa a apresentação do nó de mídia; e
- **resume** - retoma a apresentação do nó de mídia (caso esteja em pausa).

No Código 1, pode-se observar os componentes contidos no cabeçalho de uma aplicação NCL. Nesse cabeçalho foram declaradas regiões, descritores (cada um sendo associado a uma região) e conectores. O descritor possui o atributo **explicitDur**, que indica o tempo de exibição da mídia associada a esse descritor. O primeiro conector (linha 11), indica que a seleção de

uma determinada tecla (a ser futuramente identificada) ativará a exibição de uma nova mídia (a ser futuramente identificada). O segundo conector (linha 16), indica que o término de uma mídia (a ser futuramente identificada no corpo da aplicação) ativará a exibição de uma nova mídia (a ser futuramente identificada no corpo da aplicação) com atraso de 3 segundos.

Código 1 Exemplo NCL Head

```
1  <head>
2   <regionBase>
3    <region id="rg1"  left="408" top="56" width="400" height="500" />
4    ...
5   </regionBase>
6   <descriptorBase>
7    <descriptor id="dV" region="rg1" explicitDur="10s"/>
8    ...
9   </descriptorBase>
10  <connectorBase>
11   <causalConnector id="onSelectStart">
12    <connectorParam name="aTecla"/>
13    <simpleCondition role="onSelection" key="\$aTecla"/>
14    <simpleAction role="start"  qualifier="par"/>
15   </causalConnector>
16   <causalConnector id="onEndStart">
17    <simpleCondition role="onEnd"/>
18    <simpleAction role="start"  qualifier="par" delay="3s"/>
19   </causalConnector>
20  </connectorBase>
21 </head>
```

`Body` - Corpo da Aplicação

Já no corpo da aplicação NCL existem os seguintes componentes:

- **`media`:** em uma mídia ou nó de conteúdo, o comportamento está associado a descritores, conectores e elos. Ela pode possuir âncoras temporais ligadas a outras mídias por meio de elos. Uma âncora é uma posição temporal definida em uma mídia, na qual podem ser declarados um início e um final;
- **`link`:** é um elo que liga dois ou mais nós de mídia que fazem parte de um determinado relacionamento. Seus comportamentos são definidos nos conectores; e
- **`context` e `switch`:** são nós de composição, utilizados para estruturar uma aplicação hipermídia de maneira hierárquica. Dentro de um nó de composição são criados nós de mídias, os quais são acessados através de portas ou regras, dependendo do tipo de composição.

Os elos usam os mesmos atributos **role** existentes nos conectores, o conjunto de possíveis eventos e ações associados a uma ligação são os mesmos apresentados para conectores.

Além dos itens citados acima, uma aplicação NCL deve ter uma porta de entrada, que identifica qual a mídia que deve ter sua apresentação ativada pelo *player* no início da aplicação.

No Código 2, é apresentado o corpo da aplicação NCL cujo cabeçalho foi apresentado no Código 1. A porta de entrada (linha 2) indica que a aplicação deve começar pela exibição do *VideoAbertura*. Logo após a porta de entrada, são declaradas mídias (associadas a descritores) e elos (**links** associados a um conector). Pode-se observar que os elos estão associados a um conector através do parâmetro **xconnector**, possuindo os mesmos eventos e ações do conector, declarados através do parâmetro **role**. Outra informação contida nos elos é a identificação das mídias que fazem parte da ligação, através do parâmetro **component**. O primeiro elo (linha 5) está associado ao conector **onSelectStart**, e indica que a seleção da tecla RED sobre a mídia *Icone* ativa a exibição da mídia *ImagenFoto*. O segundo elo indica que o término da exibição da mídia *ImagenFoto* ativa a exibição da mídia *TextoFoto*.

Código 2 Exemplo NCL body

```
1  <body>
2   <port id="pInicio" component="VideoAbertura" />
3   <media type="image/mp4" id="VideoAbertura" src="a.mp4" descriptor="dV"/>
4   ...
5   <link id="vervelho" xconnector="onSelectStart">
6     <bind component="Icone" role="onSelection">
7       <bindParam name="aTecla" value="RED"/>
8     </bind>
9     <bind component="ImagenFoto" role="start" />
10  </link>
11  <link id="eV->sI" xconnector="onEndStart">
12     <bind component="ImagenFoto" role="onEnd" />
13     <bind component="TextoFoto" role="start" />
14  <link>
15 </body>
```

NCL RAW

O NCL RAW é uma versão reduzida do NCL, onde pode-se construir as mesmas aplicações construídas na versão completa. A Figura 4 apresenta os elementos que foram mantidos em um documento na versão NCL RAW.

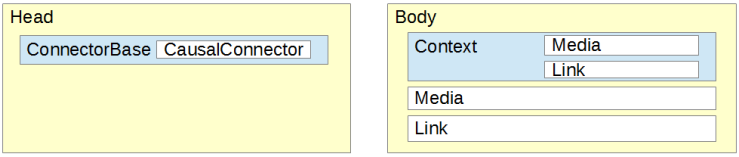


Figura 4 – Estrutura Documento NCL-RAW

No perfil RAW, as estruturas **`ncl`**, **`head`**, e **`body`** foram mantidas. Já as estruturas **`RegionBase`**, **`DescriptorBase`**, **`RuleBase`** e **`switch`** foram removidas.

Na versão RAW, usa-se o elemento **`property`** para definir propriedades e transições. Devido à eliminação do elemento **`switch`**, a única maneira de especificar alternativas é através das condições nos **`links`**. Com a eliminação das **`rules`**, o único elemento que restou no **`head`** foi o **`ConnectorBase`**, que define as relações usadas nos **`links`**.

A versão RAW é a versão utilizada neste trabalho. Justifica-se seu uso por possuir menos elementos, consequentemente facilitando a construção de transformações necessárias no trabalho proposto e que serão apresentadas em um capítulo posterior.

NCLua

A linguagem NCL é uma linguagem declarativa, que não permite a implementação de algumas funcionalidades, como cálculos, ou operações que seguem uma sequência de passos para resolver um determinado problema. Visando permitir a construção de documentos hipermídia com funcionalidades além das proporcionadas pelas linguagens declarativas, pode-se adotar uma abordagem híbrida, onde se usa uma linguagem imperativa em conjunto com a declarativa. Para a linguagem declarativa NCL, adota-se a linguagem LUA como linguagem imperativa.

A integração entre as linguagens NCL e LUA é feita por meio de objetos imperativos NCLua, os quais são tradados em NCL como sendo mídias.

Para a construção de objetos NCLua, alguns módulos, além dos existentes na linguagem LUA, foram desenvolvidos (SANTA'ANNA et al., 2009):

- **event** - permite objetos NCLua se comunicarem com documento NCL, de maneira não intrusiva;
- **canvas** - possui funcionalidades para criação de desenhos gráficos na região do objeto;
- **settings** - oferece acesso a variáveis definidas no objeto **settings** do documento NCL;
- **persistent** - explora uma tabela com variáveis persistentes entre execuções de objetos imperativos.

A não intrusividade do objeto NCLua decorre do fato dele não alterar a estrutura do documento NCL, diferentemente do que ocorre com o JavaScript em relação ao código HTML.

Os objetos NCLua se relacionam bidirecionalmente com os demais elementos NCL por meio de suas âncoras, declaradas dentro do documento NCL.

Tanto o NCLua quanto o formatador NCL enviam e recebem eventos. Para que um objeto NCLua receba eventos vindos do formatador NCL, é necessário que ele registre uma função tratadora, através de uma chamada **event.register**. Já para emitir eventos, o objeto NCLua deve executar a função **event.post**.

No Código 3 pode-se ver a declaração de quatro mídias, onde: a primeira mídia é um objeto NCLua (contendo uma propriedade nomeada **inc** inicialmente valendo 1), a segunda e a terceira mídias são objetos imagem, a quarta e última é um objeto que representa um temporizador.

Código 3 Objeto NCLua

```
1  <media id="clicks" src="clicks.lua">
2        <property name="inc" value="1"/>
3  </media>
4  <media id="button" descriptor="dsButton" src="media/button.jpg"/>
5  <media id="win" descriptor="dsButton" src="media/win.jpg"/>
6  <media id="timer" type="application/x-ginga-time">
7        <area id="areaTotal" begin="35s" end="40s"/>
8  </media>
```

Seguindo no mesmo exemplo, o Código 4 apresenta três diferentes relacionamentos entre objetos. O primeiro indica que ao iniciar o temporizador, o objeto NCLua também é iniciado. O segundo relacionamento indica que ao ser selecionada a imagem **button**, um evento **set** é gerado para o objeto NCLua através da âncora **inc** (tal evento passa um atributo com valor 1). Por fim, o terceiro relacionamento ocorre quando o temporizador alcança o tempo 35s. Quando isso ocorre, é testado se o valor contido na propriedade **inc** do objeto NCLua é maior ou igual a 3; se for, inicia a mídia imagem **win**.

Código 4 Links NCLua

```
<link xconnector="onBeginStart">
   <bind role="onBegin" component="timer"/>
   <bind role="start"   component="clicks"/>
</link>
...
<link xconnector="onSelectionSet">
   <bind role="onSelection" component="button"/>
   <bind role="set" component="clicks" interface="inc">
        <bindParam name="var" value="1"/>
   </bind>
</link>
...
<link xconnector="onCondGteBeginStart">
   <linkParam name="var" value="3"/>
   <bind role="onBegin" component="timer" interface="areaTotal"/>
   <bind role="attNodeTest" component="clicks" interface="inc"/>
   <bind role="start" component="win"/>
</link>
```

O evento **set** gerado para o objeto NCLua, no Código 4 é tratado pela função apresentada no Código 5.

Código 5 Função Lua

```
local cnt = 0
function handler (evt)
 if evt.class=='ncl' and  evt.type=='attribution'
    and evt.name=='inc' then
       cnt = cnt + evt.value
       event.post { class = 'ncl',
           type = 'attribution',
           name = 'inc',
           action = 'stop',
           value  = cnt }
 else
    return
 end
end
event.register(handler)
```

Nessa função, pode-se observar que ao chegar o evento, seus atributos são testados. Caso o evento seja associado ao atributo **inc**, o valor passado como parâmetro é somado ao valor contido no objeto NCLua, e este é retornado ao parâmetro **inc** através do evento **post**. É válido destacar que a cada chegada de evento neste objeto o valor da variável **cnt** é incrementado com 1, e o valor resultante do incremento é repassado através do evento **post** para

o parâmetro **`inc`** do objeto NCLua, declarado no código NCL.

Neste trabalho não são analisados comportamentos de rotinas escritas em NCLua.

2.2.2 SMIL (*Synchronized Multimedia Integration Language*)

A linguagem SMIL (*Synchronized Multimedia Integration Language*) (BULTERMAN, 2008) é uma linguagem baseada em XML. Uma aplicação SMIL é composta por um cabeçalho e um corpo. A Figura 5 apresenta os elementos que compõem um documento SMIL.

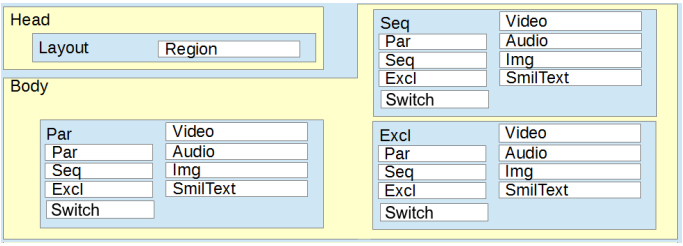


Figura 5 – Estrutura de um documento SMIL

A linguagem SMIL também é uma linguagem declarativa. A Figura 6 apresenta uma versão simplificada do seu metamodelo. Nessa simplificação não aparecem alguns elementos, como, por exemplo, o **`switch`**.

O metamodelo SMIL possui um elemento principal, chamado “SMIL”, que possui dois sub-elementos: **`Head`** e **`Body`**.

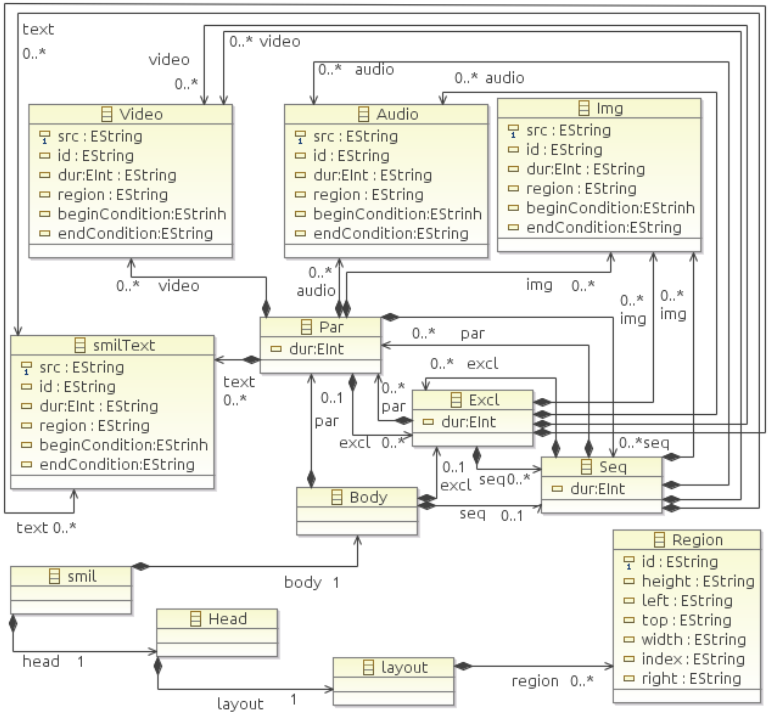


Figura 6 – Metamodelo SMIL

`Head` - Cabeçalho da Aplicação

O cabeçalho possui informações sobre título e autoria da aplicação, bem como a definição do posicionamento das mídias, usando o elemento **`layout`**. Internamente ao **`layout`**, são criados elementos **`region`** indicando as posições onde as mídias podem ser apresentadas. Dentre outros atributos, uma **`region`** possui um posicionamento e um tamanho. O Código 6, apresenta o exemplo de um cabeçalho SMIL onde é declarada uma **`region`** associada ao identificador *RG*1.

Código 6 Cabeçalho em SMIL

```
<head>
...
<layout>
<region xml:id="RG1" top="5px" left="10%" width="80%" height="30px"/>
</layout>
</head>
```

`Body` - Corpo da Aplicação

O corpo possui informações sobre o comportamento temporal da aplicação e informações das mídias que compõem a aplicação.

Na definição do comportamento temporal podem ser usadas as estruturas de composição: **`seq`**, **`par`**, **`excl`** e **`switch`**.

Uma estrutura **`seq`** define um grupo de elementos que são apresentados em sequência, um após o outro; a estrutura **`par`** define um grupo de elementos que devem ser executados paralelamente. No Código 7, há duas imagens sendo apresentadas em paralelo com uma estrutura **`par`**. Dentro dessa estrutura há mais duas imagens sendo apresentadas em sequência.

Código 7 Par e Seq em SMIL

```
 <par>
    <img id="vim" src="dt/a0.png" region="RG1" begin="0s" dur="7s"/>
    <img src="dt/a1.png" begin="vim.endEvent+3s" dur="8s"/>
     <seq>
         <img src="dt/a2.png" begin="0s" dur="18s"/>
         <img src="dt/a3.png" begin="0s"/>
     </seq>
 </par>
```

A estrutura **`excl`** é semelhante à estrutura **`par`**, mas com a restrição adicional de que apenas um elemento pode ser apresentado em um dado momento. Se qualquer elemento começa a ser apresentado durante a apresentação de outro, o primeiro elemento termina. Essa estrutura é usada para definir uma exclusão mútua de mídias. No Código 8, há duas imagens dentro de uma estrutura **`excl`**. Quando a segunda começar a ser apresentada, a primeira irá parar, pois apenas uma pode ser apresentada por vez.

Código 8 Excl em SMIL

```
1  <excl>
2          <img src="dt/a1.png" begin="0s" region="r2"/>
3          <img src="dt/a2.png" begin="5s" region="r3"/>
4  </excl>
```

A estrutura **switch** escolhe, em uma lista de elementos, qual deverá ser apresentado. Essa escolha é baseada em um critério definido dentro do **switch**. Na apresentação do **switch**, o *player* começa a análise de qual mídia satisfaz o critério, começando pela primeira mídia da lista. Ao encontrar uma que satisfaça, ela será apresentada, não levando em consideração o restante das mídias contidas na lista. No Código 9, um **switch** é usado para selecionar entre três idiomas de uma mídia de áudio.

Código 9 Switch em SMIL

```
1  <switch>
2   <audio src="eng.wav"  system-language="en"/>
3   <audio src="fra.wav"  system-language="fr"/>
4   <audio src="port.wav" system-language="pt"/>
5  </switch>
```

É possível criar aplicações contendo vários níveis de estruturas de diferentes tipos, ou seja, é possível criar uma estrutura **par** que contenha dentro de si uma estrutura **seq** e uma estrutura **switch**.

Um elemento ou uma estrutura pode conter atributos que definem seus comportamentos temporais. Entre os atributos possíveis estão:

- **delay** - define um atraso que deve ocorrer no início da mídia. Se estiver dentro de uma estrutura **seq**, esse valor representará um tempo em que nenhuma mídia será apresentada dentro do **seq**. Se estiver dentro de um **par**, representa o atraso do início desta mídia em relação ao início das outras mídias dentro do **par**;
- **begin** - indica o tempo em que a mídia deve ser iniciar;
- **end** - representa o tempo em que a mídia deve terminar;
- **dur** - representa a duração da mídia ou estrutura;
- **repeatCount** - representa o número de repetições da apresentação;
- **endsync** - usada em estruturas para sincronizar o final das mídias ou elementos internos. Pode ter as seguintes alternativas:
  - **first** - termina quando o primeiro elemento ou sub-estrutura terminar;

- **last** - termina quando o último elemento ou sub-estrutura terminar; e
- **id(*ID*)** - termina quando o elemento indicado por *ID* terminar.

Os atributos **begin** e **end** podem estar associados com eventos de outras mídias, como, por exemplo, o início ou término de outra mídia. Além dos eventos **begin** e **end**, existe o evento de seleção **click** que ocorre quando uma mídia é selecionada. No Código 10, há uma estrutura **par** contendo uma mídia e uma estrutura **par** interna. A estrutura **par** interna tem sua apresentação iniciada no momento que a mídia sofre uma seleção. Ela possui duas mídias. Embora ela seja uma estrutura **par**, a sua segunda mídia interna iniciará três segundos após o término da primeira mídia interna. Isso ocorre devido à associação do evento de início da segunda mídia com o de término da primeira.

Código 10 Begin End em SMIL

```
1     <par>
2       <img id="B" src="dt/bck.png" dur="10s"/>
3       <par begin="B.click">
4               <img id="vim" src="dt/a1.png" begin="0s" dur="7s"/>
5         <img src="dt/a2.png" begin="vim.endEvent+3s" dur="8s"/>
6       </par>
7     </par>
```

A metodologia proposta a ser apresentada neste trabalho restringe-se a analisar o comportamento de aplicações SMIL que adotam estruturas de causalidade, definidas nos atributos **begin** e **end**.

2.3 METODOLOGIA DE DESENVOLVIMENTO SEM VERIFICAÇÃO FORMAL

Uma aplicação hipermídia é formada pela junção de um conjunto de diferentes mídias, sendo que algumas dessas oferecem a possibilidade de interação do usuário. Na etapa de projeto, os desenvolvedores desses sistemas devem sempre considerar requisitos, tais como:

- Qualidade visual e do conteúdo - a fim de fazer uma aplicação atraente em relação à forma como deve ser apresentado o conteúdo com inclusão de interatividade;
- Comportamento correto - respeitar as restrições temporais e espaciais, bem como o não-determinismo introduzido pela interatividade do usuário; e

- Tempo de verificação adequado - o tempo necessário no processo de verificação não deve ser elevado, evitando, assim, tornar o uso da abordagem inviável.

A inclusão de interatividade em uma aplicação tende a elevar seu nível de complexidade, e quanto mais complexa uma aplicação, maior o risco do projetista inserir erroneamente comportamentos indesejados.

No processo de desenvolvimento o projetista normalmente adota uma ferramenta de autoria. As ferramentas de autoria para NCL e SMIL com interface gráfica, tais como Composer (GUIMARAES; COSTA; SOARES, 2008), Eclipse e GRiNS (BULTERMAN et al., 1998), ajudam os projetistas no desenvolvimento de aplicações hipermídia. Como principais características dessas ferramentas pode-se destacar:

- análise sintática do código, indicando erros de sintaxe;
- uma linha de tempo, que permite identificar visualmente os pontos de início e fim da exibição de cada mídia;
- uma interface visual que permite indicar o posicionamento de cada mídia na tela de exibição; e
- ligação com um *player* que permite a visualização da aplicação em exibição.

A Figura 7 mostra um exemplo da linha de tempo da ferramenta de autoria GRiNS, usada na edição de aplicações em SMIL. Nessa representação da aplicação, cada linha corresponde a uma mídia identificada com o símbolo de diamante. Períodos de exposição da mídia são representadas por retângulos, posicionados na linha de tempo.

O uso de uma análise da linha temporal permite ao projetista identificar alguns erros, como posicionamento temporal errado de uma mídia.

Além de analisar a linha do tempo, o projetista pode realizar testes na aplicação usando um *player*. Esses testes auxiliam na identificação de comportamentos indesejáveis antes da aplicação ser disponibilizada.

A linha temporal apresentada por ferramentas de autoria permite ao projetista identificar algumas erros de projeto e comportamentos indesejados antes da exibição da aplicação. Infelizmente, essa linha não permite representar e verificar *a priori* todas as situações possíveis, devido à interatividade que pode haver em aplicações hipermídia.

A metodologia baseada em testes no *player*, além de consumir um tempo elevado, não é exaustiva.

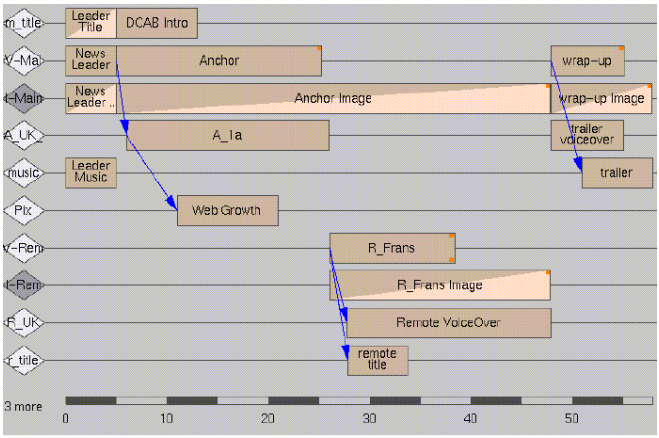


Figura 7 – Linha Temporal do GRiNS (BULTERMAN et al., 1998)

2.4 O PROBLEMA A RESOLVER

Pode-se dizer que um dos problemas no desenvolvimento de aplicações hipermídia consiste na construção de aplicações interativas, com a garantia de não possuírem comportamentos indesejados.

Visando garantir a corretude de aplicações hipermídia, o projetista deve adotar uma metodologia de desenvolvimento.

Metodologias baseadas em linha temporal e testes podem diminuir a quantidade de comportamentos indesejados, mas não garantem sua inexistência, além de consumirem um demasiado tempo na etapa de testes, tornando-se inviáveis para aplicações que possuem um alto grau de interatividade, principalmente quando o tempo disponível para verificação é limitado.

Metodologias usando Verificação Formal requerem que a aplicação e os comportamentos a serem verificados estejam modelados em linguagens formais, usadas por ferramentas de verificação formal. A codificação usando uma linguagem formal é uma atividade demorada, e seu uso não é uma atividade trivial para engenheiros ou cientistas da computação, quanto mais para projetistas de aplicações hipermídia, que normalmente são da área de jornalismo. Dependendo do tamanho da aplicação, o processo de verificação também pode ser demorado.

2.5 ESTUDO DE PROPOSTAS EXISTENTES

Nos últimos anos foram propostas algumas abordagens, técnicas e ferramentas para apoiar o desenvolvimento de aplicações hipermídia. Nessa seção são descritas algumas destas propostas. Essa descrição não é exaustiva, mas oferece uma boa visão do estado da arte sobre a validação temporal de aplicações hipermídia.

A Tabela 1 apresenta a lista de trabalhos que serão abordados, indicando o ano de sua publicação.

Tabela 1 – Trabalhos Relacionados

1998	2001	2002	2004	2005	2006	2010	2011	2012	2015
Santos, Soares, Souza et al.	Oliveira, Turine e Masieiro Na e Furuta	Felix, Haeusler e Soares Yu, He, Gao et al.	Sampaio e Courtiat	Bertino, Ferrari, Perego et al.	Elias, Easwarakumar e Chbeir	Yovine, Oliveiro, Monteverde et al.	Gaggi e Bossi Bouyakoub e Belkhir	Belouer e Maris	Santos, Braga, Muchaluat-Saade et al.

2.5.1 Descrição dos Trabalhos Existentes

Nesta seção são apresentadas individualmente alguns trabalhos relacionados, previamente listados na Tabela 1.

Santos, Soares, Souza et al.

No trabalho de Santos et al. (1998) efetuam-se verificações de aplicações codificadas na Linguagem NCL. Para realizar a verificação, a aplicação NCL é traduzida para a linguagem usada no *framework* RT-LOTOS. Esta tradução é realizada de maneira automática, seguindo um conjunto de regras de transformação.

Em RT-LOTOS o comportamento da aplicação é modelado usando-se para isso uma estrutura de processos, e uma biblioteca de processos. O uso da biblioteca de processos permite o reúso de partes da aplicação.

Um grafo de alcançabilidade é obtido a partir da especificação formal em RT-LOTOS, onde cada vértice neste grafo representa um estado alcançável, e cada aresta representa a ocorrência de um evento, ou progressão do tempo.

Com o grafo de alcançabilidade, diferentes comportamentos podem ser verificados, como, por exemplo, se uma mídia sempre que exibida alcança o estado final, ou se uma mídia sempre alcança um estado que indica sua exibição.

O trabalho classifica os comportamentos das mídias em duas categorias:

- qualitativo - quando não depende da duração da mídia; e
- quantitativo - quando depende da duração da mídia.

O trabalho também pode considerar fatores relacionados à plataforma de execução. Nesse caso, ele efetua uma segunda classificação dos comportamentos:

- intrínsecos - se não dependem da plataforma; e
- extrínsecos - se dependem da plataforma.

A abordagem usada na especificação formal permite modelar comportamentos não-determinísticos, como interações com o usuário, por exemplo.

Na e Furuta

Na abordagem proposta por Furuta et al. (2001), chamada caT, a aplicação hipermídia é codificada adotando o formalismo redes de Petri, onde cada lugar representa uma mídia e cada ligação representa as relações entre as mídias.

Devido ao fato das redes de Petri não possuírem identificação para os *tokens*, nesse trabalho foi proposto uma rede de Petri que supre essa limitação, onde é possível identificar os diferentes *tokens*. Essa rede de Petri proposta foi chamada de *High-Level Petri Net*.

O trabalho incorpora redes de Petri Hierárquicas, permitindo diminuir a complexidade do grafo de autoria e melhorar o reúso.

É possível o uso de dois tipos de ferramentas para a análise da rede de Petri. A primeira é uma ferramenta de depuração interativa, a segunda é uma ferramenta de análise que faz uso de uma árvore de alcançabilidade.

Atualmente, o trabalho permite analisar: existência de estado terminal, *boundness* (se há um lugar com número ilimitado de *tokens*), e segurança

(se em todo lugar há somente um *token*). Tais análises são importantes em sistemas onde *tokens* podem representam limites de recursos do sistema.

Oliveira, Turine e Masieiro

Em HMBS (OLIVEIRA; TURINE; MASIEIRO, 2001) (*Hypermedia Model Based on Statechart*) as aplicações hipermídia são escritas adotando-se gráficos de estado, que são uma extensão de máquinas de estados finitos. Nesse formalismo, os estados representam as páginas (informações apresentadas para os usuários) e as transições representam as ativações dos possíveis links.

Segundo os autores, modelos baseados em gráficos de estado adotam formalismos matemáticos ricos que podem ser explorados para melhorar a qualidade da aplicação.

A verificação dos comportamentos da aplicação é feito através de uma árvore de alcançabilidade, obtida a partir do gráfico de estados. Com a árvore, é possível verificar diferentes comportamentos, como por exemplo:

- se uma determinada página é alcançada, tendo como ponto inicial um determinado estado; e

- se um conjunto de páginas podem ser apresentadas simultaneamente.

Segundo o autor, o modelo proposto é adequado para aplicações que possuem uma estrutura hierárquica, tais como livros, artigos científicos, manuais e tutoriais *on-line*. O modelo proposto não inclui mecanismos que permitam especificar sincronismos requeridos por aplicações com conteúdo multimídia.

Yu, He, Gao et al.

Yu et al. (2002) propõe uma abordagem formal para modelagem e análise dos aspectos temporais de aplicações SMIL, utilizando o modelo SAM (Modelo de Arquitetura de Software). Segundo o autor, o SAM baseia-se em dois formalismos, combinando redes de transição de predicados (PrTNs) e lógica temporal.

No que diz respeito ao aspecto em tempo real, uma rede PrTNs é similar a uma rede de Petri.

Elementos de sincronização do SMIL são sistematicamente modelados em PrTNs, e propriedades tais como segurança e vivacidade, são especificadas usando fórmulas LTL (*Linear Temporal Logic*) e verificadas por ferramentas automáticas.

O trabalho também usa árvore de alcançabilidade para verificar os estados das mídias existentes na aplicação. Provas por dedução, indução estrutural, bem como técnicas de verificação do modelo são aplicadas para verificar requisitos de sincronização da aplicação SMIL.

Na abordagem apresentada o projetista modelo os aspectos da aplicação SMIL em PrTNs, não existindo um processo de tradução de SMIL para PrTNs. Após a modelagem em PrTNs ocorre a tradução desse modelo para um formato CTS (*clocked transition systems*), entrada para a ferramenta de verificação STeP. Essa ferramenta verifica se o modelo satisfaz determinadas propriedades escritas em LTL.

Embora o processo de modelagem em PrTNs seja manual, o autor apresenta o conjunto de regras de modelagem que o usuário do SAM deve seguir.

Felix e Soares

No trabalho de Felix, Haeusler e Soares (2002), usa-se verificação formal para analisar os comportamentos de aplicações hipermídia.

Nesse trabalho adota-se um formalismo intermediário, no qual o usuário deve escrever sua aplicação, este formalismo é chamado ORL (*Objects Representation Language*). As especificações de ORL foram escritas usando os conceitos de modelagem estabelecidos em NCM (*Nested Context Model*), um modelo conceitual para documentos hipermídia.

Após a obtenção do modelo em ORL, ele é transformado em um modelo em BTA (*Broadcaster Timed Automaton*) em um segundo passo. Após a transformação, usando a ferramenta de verificação UPPAAL, são verificadas propriedades, tais como a alcançabilidade.

Para que possa efetuar a verificação, o usuário deve escrever manualmente suas propriedades usando a linguagem CTL (*Computational Tree Logic*), e então passá-las para a ferramenta UPPAAL juntamente com o modelo formal da aplicação.

Segundo o autor, um documento é consistente se é possível apresentá-lo de forma que todas as restrições de sincronização de eventos possam ser cumpridas.

Sampaio e Courtiat

No trabalho desenvolvido em (SAMPAIO; COURTIAT, 2004) é proposta uma metodologia de desenvolvimento para aplicações escritas na linguagem SMIL 2.0. Essa metodologia é composta pelas seguintes fases:

1. edição da aplicação usando SMIL 2.0;

2. tradução automática de estrutura lógica e temporal do documento em um formalismo usando especificações em RT-LOTOS;

3. criação do grafo de alcançabilidade mínimo a partir da especificação RT-LOTOS através da ferramenta RTL; e

4. agendamento da apresentação do documento se for constatada consistência.

As aplicações são consideradas consistentes quando o estado final é sempre alcançado por todas as mídias.

A fim de evitar qualquer inconsistência durante a apresentação de uma aplicação, a abordagem permite aplicar uma técnica de correção através da qual todos os caminhos inconsistentes gerados a partir do grafo de alcançabilidade são removidos. Essa abordagem permite que o autor de uma aplicação possa apresentar apenas as partes da aplicação consistentes. Essa abordagem também considera eventos não-determinísticos, como interatividade.

Bertino, Ferrari, Perego et al.

No trabalho desenvolvido em (BERTINO et al., 2005) foi proposta uma abordagem para desenvolvimento de aplicações multimídia baseada na especificação de restrição.

Nessa abordagem a aplicação é descrita usando uma representação em um paradigma mais elevado, que ao final é traduzido para a linguagem hipermídia.

O paradigma a ser usado fornece um conjunto de restrições, que descrevem as relações temporais e espaciais entre os objetos, independentemente do modelo utilizado para implementar a aplicação final.

Um exemplo de restrição descrita no trabalho é: o objeto “A” inicia antes do objeto “B”, sem especificar quanto tempo antes. Segundo os autores, a principal desvantagem da abordagem baseada em restrições é o menor grau de controle sobre o resultado, uma vez que o processo de especificação não oferece uma projeção clara da aplicação final.

Elias, Easwarakumar e Chbeir

No trabalho de Elias, Easwarakumar e Chbeir (2006) é proposto um algoritmo para avaliação da consistência de relações temporais e espaciais, baseando-se no uso de restrições.

Como exemplo de restrição o autor exemplifica: o objeto “C” inicia 10 segundo antes do objeto “B”.

Na abordagem proposta, além das restrições, são definidas prioridades. Estas prioridades são usadas em casos onde duas restrições se contraponham, e a restrição com maior prioridade deve ser mantida.

No que se refere à sobreposição espacial, a abordagem permite verificar a sobreposição na apresentação de dois objetos. A trabalho não considera aplicações com interatividade, onde o usuário pode influenciar na ordem do fluxo de exibição da aplicação.

Yovine, Oliveiro, Monteverde et al.

Em Yovine et al. (2010) foi proposta uma abordagem em que a aplicação hipermídia é modelada como uma rede de Petri temporal (*Temporal Petri Net* - TPN).

Neste trabalho as propriedades são modeladas em um modelo VTS (*Visual Timed Scenarios*), que vem a ser uma linguagem gráfica para descrever eventos. As propriedades especificadas em VTS são traduzidas em observadores, representados também na forma de redes de Petri temporais.

A abordagem usa o modelo de verificação para verificar propriedades que incluem: ineditismo (*freshness*), tempo de resposta limitada (*bounded response*) e correlação de eventos.

Bouyakoub e Belkhir

No trabalho de Bouyakoub e Belkhir (2011), uma aplicação SMIL é construída usando a ferramenta de autoria SMIL Builder, que usa um modelo baseado em uma extensão de redes de Petri hierárquica.

Na abordagem proposta o projetista descreve a aplicação em SMIL e a ferramenta efetua automaticamente a transformação para redes de Petri. Essa transformação é realizada dinamicamente durante o processo de edição do documento SMIL: na interface gráfica da ferramenta, o projetista pode visualizar a criação da rede de Petri que representa os comportamentos da aplicação SMIL que ele está editando.

O SMIL Builder não adota ferramentas de verificação de modelo; a análise da consistência verifica conflitos nos parâmetros **`begin`**, **`end`** e **`duration`**.

A realização da análise ocorre durante o processo de edição, de forma incremental. Entretanto, nos parâmetros **`begin`**, **`end`** e **`duration`**, não pode haver valores não-determinísticos.

A abordagem proposta considera apenas relações temporais, não abordando questões espaciais. Por não permitir avaliação de eventos não-determinísticos, a ferramenta não pode ser usada da edição de aplicações SMIL

que possuem interatividades.

Gaggi e Bossi

Na abordagem proposta por Gaggi e Bossi (2011), as aplicações hipermídia são escritas em uma semântica formal que engloba os elementos da linguagem SMIL. Nessa semântica, o projetista codifica as aplicações usando a linguagem SMIL e codifica propriedades temporais adotando regras de inferência baseadas na lógica de Hoare.

As propriedades temporais dentro do código SMIL definem como deve estar o estado da aplicação antes e depois da execução de um determinado trecho de código SMIL.

O processo de verificação ocorre durante o processo de edição, ou seja, o trabalho adota uma abordagem incremental. Se alguma inconsistência é encontrada, o sistema retorna indicando a parte do código onde ocorre o erro, dando possíveis sugestões de como efetuar a correção.

Nesse trabalho, a análise da consistência identifica conflitos nos parâmetros **`begin`**, **`end`** e **`duration`**, onde todos os valores devem ser definidos de maneira prévia e determinística.

A abordagem apresentada considera apenas relações temporais, não sendo possível verificar relações espaciais.

Por realizar a verificação de valores definidos de modo determinístico, a abordagem não pode ser usada da edição de aplicações SMIL que possuem interatividades.

Belouaer e Maris

No trabalho de Belouaer e Maris (2012), é apresentada uma abordagem SMT (*Sat Module Theory*) para resolver problemas de planejamento espaço-temporal.

Segundo os autores, um problema de planejamento espaço-temporal consiste de um conjunto de ações, um estado inicial e um estado que deseja-se alcançar. Há três categorias de restrições a serem analisadas: (1) restrições inerentes ao conjunto de ações, (2) restrições de efeito contraditório ao conjunto de ações, e (3) restrições de movimento contraditório ao conjunto de ações.

Ao resolver o problema, considerando as restrições que representam as ações, é possível verificar se o objetivo pode ser atingido ou não.

Santos, Braga, Muchaluat-Saade et al.

No trabalho apresentado em (SANTOS et al., 2015), verificam-se comportamentos espaciais e temporais, considerando que o posicionamento espacial de uma determinada mídia pode mudar dinamicamente.

A abordagem proposta usa o formalismo SHM (*Simple Hypermedia Model*) e um tradutor usando o sistema Maude (CLAVEL et al., 2007). Segundo os autores, o modelo SHM foi concebido para abordar os seguintes requisitos:

- lidar com diferentes paradigmas de autoria, baseados em eventos, baseados em restrições, etc;
- permitir descrever os *layouts* espacial e temporal;
- permitir descrever atributos espaciais em função do tempo; e
- ser capaz de realizar a validação de documentos em etapas diferentes do ciclo de vida do documento.

Nesse trabalho, as aplicações podem ser escritas nas linguagens NCL e SMIL. Verificam-se fórmulas LTL usando o *model-checker* Maude.

2.5.2 Comparação dos Trabalhos Existentes

A Tabela 2 apresenta um quadro comparativo indicando as questões abordadas por cada um dos trabalhos relacionados previamente apresentados.

Dos trabalhos apresentados, nem todos são voltados para aplicações hipermídia nas linguagens NCL e SMIL, como por exemplo os trabalhos de (OLIVEIRA; TURINE; MASIEIRO, 2001), (ELIAS; EASWARAKUMAR; CHBEIR, 2006) e (BELOUAER; MARIS, 2012).

Propostas Existentes	Edição em NCL	Edição em SMIL	Tradução Manual	Tradução Automática	Relacionamento Temporal	Relacionamento Espacial	Interatividade	Resultado Representado em Linha Temporal
Santos et al. 1998	✓			✓	✓		✓	
Na e Furuta 2001		✓	✓				✓	
Oliveira et al. 2001					✓		✓	
Yu 2002		✓	✓		✓		✓	
Felix 2002	✓		✓		✓		✓	
Sampaio et al. 2004		✓		✓	✓		✓	
Bertino et al. 2005		✓		✓	✓	✓	✓	
Elias et al. 2006					✓	✓		
Yovine et al. 2010	✓			✓	✓		✓	
Bouyakoub e Belkhir 2011		✓		✓	✓			
Gaggi e Bossi 2011		✓		✓	✓			
Belouer e Maris 2012					✓	✓	✓	
Santos 2015	✓	✓		✓	✓	✓	✓	
Presente Trabalho	✓	✓		✓	✓	✓	✓	✓

Tabela 2 – Questões abordadas nas propostas

Em muitos trabalhos, o projetista deve aprender novas linguagens, como naquele desenvolvido em (BERTINO et al., 2005) voltado para a linguagem SMIL. Neste trabalho, a aplicação é descrita usando para isso uma representação em um paradigma mais elevado, que ao final é traduzido para a linguagem SMIL.

O modo como é realizada a verificação, e os tipos de comportamentos verificados são diferente em cada trabalho, alguns sendo mais abrangentes que outros. Alguns trabalhos realizam uma verificação estática, não usado ferramentas de *model-checking*, como é o caso de (BOUYAKOUB; BELKHIR, 2011), (BOSSI; GAGGI, 2011) e (ELIAS; EASWARAKUMAR; CHBEIR, 2006). Tais abordagens também não consideram a possibilidade de interatividade.

Alguns trabalhos verificam a alcançabilidade de estados pretendidos, como ocorre em (SANTOS et al., 1998) e (SAMPAIO; COURTIAT, 2004); já em outros, o projetista pode verificar diferentes tipos de propriedades, como

por exemplo nos trabalhos de (FELIX; HAEUSLER; SOARES, 2002), (YU et al., 2002), (YOVINE et al., 2010) e (SANTOS et al., 2015). Dos trabalhos que adotam especificação de propriedades, a maioria requer que o projetista saiba escrever tais propriedades em linguagens específicas. Os trabalhos (SANTOS et al., 2015) e (YOVINE et al., 2010) adotam editores de uso mais simples para projetistas sem conhecimento de especificação formal.

Alguns dos trabalhos realizam verificação incremental, como, por exemplo, os trabalhos de (BOUYAKOUB; BELKHIR, 2011) e (BOSSI; GAGGI, 2011). Tais abordagens se adaptam melhor para a linguagem SMIL. Nenhum dos trabalhos apresentados possui uma abordagem de redução que vise diminuir o tamanho do modelo formal, para, assim, acelerar o processo de verificação.

3 METODOLOGIA DE DESENVOLVIMENTO PROPOSTA

Conforme anteriormente mencionado, para garantir a corretude de aplicações hipermídia, o projetista deve adotar uma metodologia que guie o processo de desenvolvimento.

O uso de metodologias baseadas em análise da linha temporal ou testes pode diminuir a quantidade de comportamentos indesejados, mas, por não serem exaustivas, elas não garantem que todos os possíveis comportamentos tenham sido testados. O uso de metodologias baseadas em Verificação Formal, por sua vez, permitem uma verificação exaustiva. Entretanto, tais metodologias requerem que a aplicação e os comportamentos a serem verificados estejam modelados em linguagens formais, usadas por ferramentas de verificação formal.

O uso de ferramentas e linguagens de verificação formal não é uma atividade trivial para engenheiros ou profissionais de informática, quanto mais para projetistas, que normalmente são da área de jornalismo ou comunicação. O processo de transformação manual de uma aplicação escrita em uma linguagem hipermídia para uma linguagem formal além de demorado, está propício a erros de codificação.

Para aplicações editadas "ao vivo", um empecilho adicional é o tempo disponível para verificar a corretude da aplicação, já que ele não pode ser superior ao período que antecede a exibição.

Visando resolver os problemas identificados, neste trabalho propõe-se uma metodologia de desenvolvimento baseada no uso de verificação formal adotando-se uma abordagem MDE (*Model Driven Engineering*). A metodologia proposta tem as seguintes características:

- efetuar verificação exaustiva dos comportamentos identificados pelo projetista;
- não exigir do projetista conhecimento sobre linguagens ou ferramentas de verificação formal;
- não consumir tempo na codificação do modelo formal;
- garantir a conformidade entre o modelo formal e o modelo hipermídia; e
- realizar a verificação em um tempo adequado para seu uso no desenvolvimento de aplicações hipermídia.

A abordagem MDE permite o uso da verificação formal sem exigir do projetista conhecimentos de tecnologias e linguagens que não são de seu domínio de conhecimento, sem gastar tempo no processo de construção do

modelo formal e garantindo a conformidade entre os modelos hipermídia e formal.

Este capítulo inicialmente apresenta os principais requisitos referentes ao desenvolvimento de aplicações hipermídia para o problema a ser tratado neste trabalho e satisfeitos pela metodologia proposta. Em um segundo momento, ele apresenta a descrição geral dessa metodologia. Na sequência, são apresentados os principais elementos técnicos da metodologia proposta:

- MDE - Engenharia Dirigida a Modelos;
- Linguagem formal FIACRE; e
- Verificação Formal.

Após apresentar a base tecnológica da metodologia proposta, o capítulo finaliza apresentando uma visão geral do ambiente desenvolvido que visa dar suporte ao uso dessa metodologia.

3.1 PRINCIPAIS RELAÇÕES EM APLICAÇÕES HIPERMÍDIA

No projeto de uma aplicação hipermídia, é necessário que o projetista verifique as relações existentes na aplicação. Tais relações tendem a agregar complexidade na aplicação e no processo de codificação.

A Figura 8 mostra as relações a serem verificadas visando a obtenção de uma aplicação hipermídia atraente e que não apresente comportamentos indesejados. No restante desta seção, essas relações serão detalhadas.

$$\textit{Relacionamentos} \begin{cases} \text{Relações Temporais} \begin{cases} \text{Intra-mídia} \\ \text{Inter-mídia} \end{cases} \\ \text{Relações Espaciais} \end{cases}$$

Figura 8 – Tipos de relacionamentos de uma aplicação hipermídia

Além das relações, esta seção aborda questões relativas a interatividade e edição “ao vivo”, que podem tornar uma aplicação mais atrativa e seu projeto mais complexo.

3.1.1 Relações Temporais

Em uma aplicação hipermídia, o projetista pode especificar atributos da mídia (como o tempo de exibição) e as relações entre as diferentes mí-

dias (JANSEN; CESAR; BULTERMAN, 2010).

Seguindo a definição apresentada em (YANG, 2000), tais atributos e relações classificam-se em duas categorias:

- **Intra-mídia** - relacionamentos que envolvem somente atributos internos de uma mídia, como começo, término e duração; e
- **Inter-mídia** - relacionamentos que envolvem mais de uma mídia, como o conjunto definido por Allen (1983) ilustrado na Figura 9.

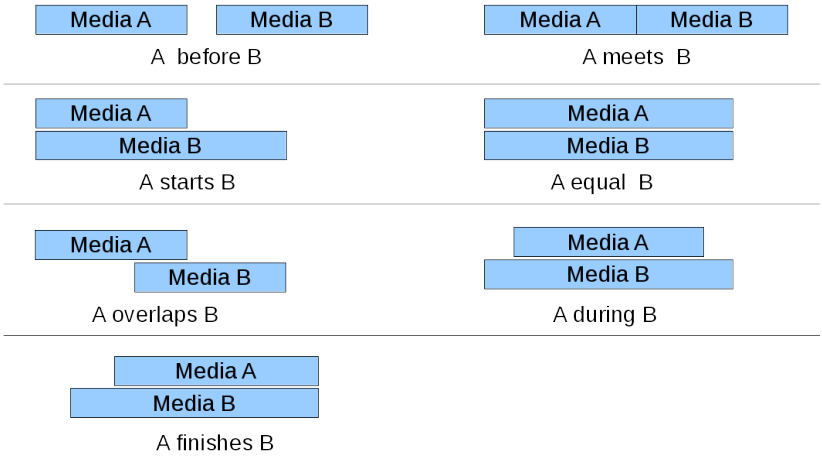


Figura 9 – Relacionamentos Inter-Mídia (ALLEN, 1983)

Em ambos os tipos de relacionamentos temporais, o projetista pode, erroneamente, introduzir inconsistências temporais, como conflitos nos atributos internos de uma mídia ou erros de especificação nas relações temporais entre duas ou mais mídias (YANG, 2000).

3.1.2 Relações Espaciais

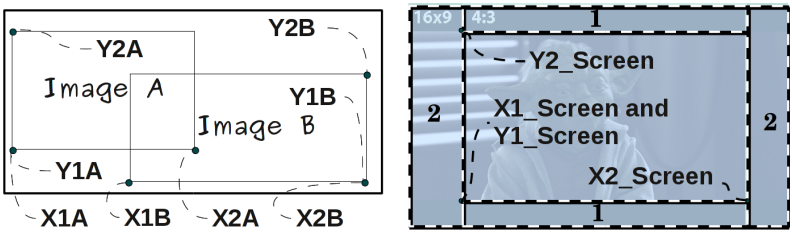
Um objeto visual (imagem, vídeo, texto) é apresentado em uma região pré-definida da tela, posicionada em relação às demais regiões de tela reservadas para outros objetos. Pode ser considerado como comportamento espacial indesejado a sobreposição espacial entre dois (ou mais) objetos visuais exibidos no mesmo tempo (sobreposição temporal). Em outras palavras, a sobreposição espacial de dois ou mais objetos na tela pode ser considerada

indesejada quando ocorre em conjunto com uma sobreposição temporal dos mesmos objetos.

A Figura 10(a) ilustra uma situação na qual existem duas regiões na tela, ocupadas pelas imagens “A” e “B”. Observa-se que a parte inferior direita da imagem “A” se sobrepõe parcialmente com a parte superior esquerda da imagem “B”. Se na apresentação dessas duas mídias ocorrer sobreposição temporal, parte de uma dessas duas imagens não ficará visível.

Além da sobreposição espacial de objetos, pode-se considerar como um comportamento espacial não desejado o uso de regiões possivelmente inadequadas. Entre tais regiões destacam-se as bordas da tela, que podem sofrer cortes ou serem distorcidas, dependendo do formato da tela. Além de cortes e distorções, a borda horizontal inferior pode ser sobreposta por uma legenda inserida via *closed caption* (também conhecida pela sigla CC), bem como outros tipos de legenda normalmente inseridos nessa posição.

A Figura 10(b), ilustra os dois principais formatos de tela: 16:9 e 4:3. Nessa figura também são mapeadas as regiões considerados inadequadas (identificadas pelos números 1 e 2).



(a) Regiões com objetos sobrepostos (b) Diferentes tamanhos de tela

Figura 10 – Regiões Espaciais

A metodologia proposta não abrange relações espaço-temporais, onde podem ocorrer alterações do tamanho das regiões usadas na aplicação.

3.1.3 Interatividade

Em uma aplicação hipermídia que possui mídias interativas, o usuário pode interagir durante o intervalo de exibição dessas mídias. A ocorrência ou ausência de uma interação em uma mídia, bem como o momento exato de sua ocorrência, dependem do desejo do usuário e geram múltiplas possibilidades de comportamentos, assim aumentando o número de situações a serem verificadas. Conforme mencionado anteriormente, estas aplicações são

consideradas não-lineares. O usuário, acostumado com a Web, onde ele pode interagir com o conteúdo e com outras pessoas, tende a querer mais interatividade também em aplicações para TVDI, não se satisfazendo apenas com conteúdos totalmente lineares.

3.1.4 Edição "Ao Vivo"

No universo de aplicações TVDI, existem diferentes categorias de aplicações diretamente relacionadas ao tipo de informação que se deseja apresentar. Algumas aplicações TVDI, como aquelas voltadas a noticiários, esportes e programas de auditório, podem necessitar modificações durante a sua exibição. Esse tipo de edição será doravante chamado de *edição "ao vivo"*. O tempo concedido para editar modificações geralmente é curto, exigindo um processo de validação muito ágil durante a exibição da aplicação como um todo, mas antes da exibição da parte da aplicação que sofreu a modificação.

Testes via execuções em *players* de mídia são poucos adequados para aplicações editadas "ao vivo", devido ao tempo disponível ser demasiado limitado. A verificação de todas as possíveis possibilidades da aplicação também é algo muito dispendioso e não se aplica em situações de edição "ao vivo". Uma abordagem de detecção de erros num tempo adequado é um requisito essencial para uma metodologia de construção de aplicações hipermídia que possam ser editadas "ao vivo".

De acordo com a BBC Academy[1], aplicações "ao vivo" podem ser divertidas e emocionantes, mas sem um planejamento cuidadoso podem ter resultados catastróficos. Assim os cuidados com imprevistos são fundamentais.

3.2 DESCRIÇÃO GERAL DA METODOLOGIA PROPOSTA

A metodologia proposta neste trabalho combina os pontos fortes de linguagens hipermídia, facilmente compreensíveis por projetistas, para descrever a aplicação, e da linguagem formal, útil para verificar a satisfação dos requisitos do sistema.

A metodologia proposta é dividida em quatro fases: Modelagem, Transformação, Verificação e Diagnóstico/Correção, conforme ilustrado na Figura 11.

[1] http://www.bbc.co.uk/academy/production/television/live-broadcast

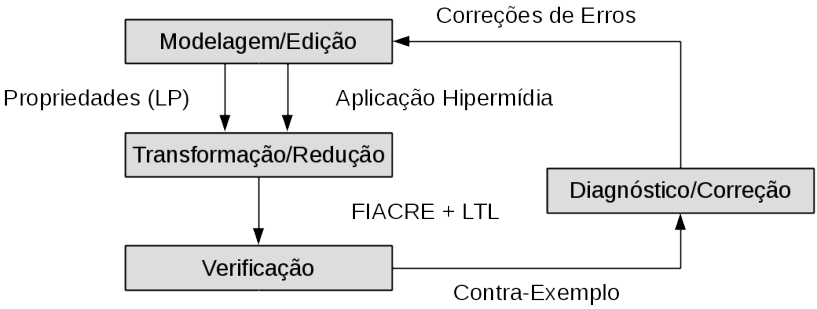


Figura 11 – Estrutura em Quatro Fases

A modelagem é a fase em que o projetista descreve a aplicação usando uma linguagem hipermídia (no presente caso, NCL ou SMIL), em geral apoiado por alguma ferramenta de autoria. Nessa fase, o projetista também deve especificar um conjunto de comportamentos a serem verificados por meio da Linguagem de Propriedades (LP), a ser proposta neste trabalho.

O objetivo do nível seguinte é a transformação automática do modelo na linguagem hipermídia e das propriedades na linguagem LP para um modelo formal na linguagem de verificação FIACRE. Essa transformação deve garantir a coerência entre os modelos de entrada e saída, ou seja, o modelo formal gerado deve representar fielmente os comportamentos da aplicação hipermídia e as propriedades a serem verificadas. Visando atingir uma tradução fiel, usa-se a abordagem MDE (SCHMIDT, 2006).

Após a verificação, caso alguma propriedade não seja satisfeita, um módulo apresenta o contraexemplo em uma linguagem compreensível pelo projetista, com a sequência de ações que levou à não satisfação da propriedade. De posse dessas informações, o projetista pode diagnosticar o problema ocorrido e fazer as correções que julgar adequadas.

3.3 ENGENHARIA DIRIGIDA O MODELO - MDE

A modelagem é uma etapa fundamental na engenharia. Engenheiros rotineiramente criam modelos para analisar e conceber sistemas complexos a serem desenvolvidos (STAHL; VOELTER; CZARNECKI, 2006).

Um modelo é uma representação abstrata de um sistema, contendo sua estrutura, funções e comportamentos, codificado em uma linguagem de domínio específico (DSML). DSML são projetadas para serem usadas em propósitos específicos, tais como: SQL, HTML, VHDL e Logo, respectivamente

para banco de dados, web, circuitos integrados e educação (BRAMBILLA; CABOT; WIMMER, 2012).

3.3.1 Princípios de Base da Engenharia de Modelos

A engenharia de modelos MDE é uma metodologia de desenvolvimento que permite a construção e alteração de sistemas, completos ou parciais, por meio da transformação de modelos (STAHL; VOELTER; CZARNECKI, 2006).

Em uma abordagem MDE, um modelo escrito usando uma DSML, é transformado em outro modelo, seguindo rigorosas regras de transformação. Essas regras realizam o processo de transformação automaticamente, garantindo que o modelo gerado será transformado rigorosamente e não terá erros introduzidos (STAHL; VOELTER; CZARNECKI, 2006).

Várias DSMLs podem ser utilizadas para representar aspectos diferentes durante o desenvolvimento de um sistema. Por exemplo, os aspectos relacionados com desenvolvimento, documentação, simulação e verificação.

Usando-se MDE, constrói-se uma aplicação em uma única linguagem de modelagem de alto nível, e diferentes modelos que representam diferentes aspectos dessa aplicação podem ser gerados automaticamente. A Figura 12 ilustra uma estrutura representando a adoção de MDE. Nessa estrutura, há três regras de mapeamento adotadas para gerar três diferentes modelos, tendo como entrada apenas um modelo.

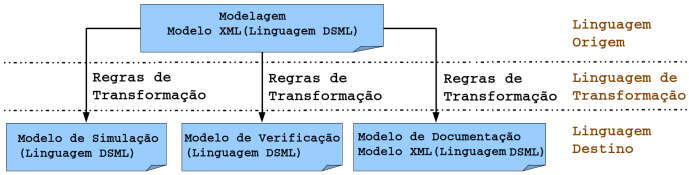


Figura 12 – Engenharia Dirigida a Modelos: Princípios

O uso de MDE garante a transformação rigorosa usando uma hierarquia de metamodelos representando as linguagens DSML. Um metamodelo é um esquema que define a estrutura semântica e sintática do modelo.

Segundo (STAHL; VOELTER; CZARNECKI, 2006), o metamodelo é um dos aspectos mais importantes da abordagem MDE, tendo as seguintes funções:

- Descrever a sintaxe abstrata das DSMLs;
- Validar a sintaxe dos modelos de entrada e de destino das DSMLs; e

- Validar a sintaxe das regras de transformação em relação ao metamodelo da linguagem de transformação.

A Figura 13 ilustra a estrutura hierárquica da metodologia MDE, a qual é composta por três níveis (COMBEMALE, 2008):

1. No nível mais alto encontra-se o metametamodelo (MMM), que se autodefine;
2. No nível intermediário encontram-se os metamodelos, que devem estar em conformidade com o MMM. Em uma transformação de modelos, há três tipos de metamodelos: Fonte (MMa), Transformação (MMt) e Destino (MMb); e
3. No nível inferior, encontram-se os modelos fonte (Ma) e destino (Mb) (respectivamente, em conformidade com seus metamodelos MMa e MMb). A transformação de modelos ocorre seguindo um conjunto de regras rígidas de acordo com o metamodelo de transformação MMt.

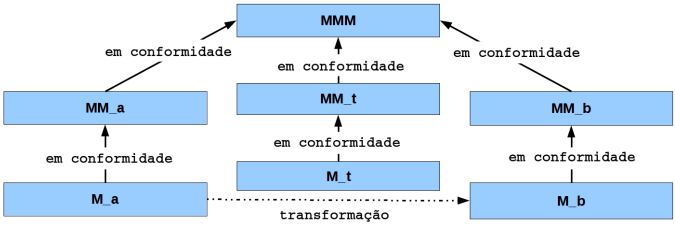


Figura 13 – Conformidade em MDE (JOUAULT; KURTEV, 2006)

Para realizar as transformações na metodologia proposta, adotou-se um subconjunto de ferramentas do projeto TOPCASED (FARAIL et al., 2006), ambiente que abrange várias ferramentas de verificação e transformação de modelos MDE. A transformação de modelos MDE segue a estrutura hierárquica proposta pela OMG [2]. Este ambiente fornece a base para transformar os modelos codificados em uma linguagem hipermídia para modelos codificados em linguagens adotadas por ferramentas de verificação formal.

3.3.2 Linguagens de Transformação MDE

Para realizar a transformação de um modelo de origem para um modelo de destino, o programador deve escrever as regras de transformação utilizando uma linguagem de transformação. Na sequência desta seção serão

[2] http://www.omg.org/

apresentados os dois principais tipos de linguagens de transformação de modelos: Modelo para Modelo (M2M) e Modelo para Texto (M2T).

Modelo para Modelo (M2M)

Como exemplo de uma linguagem de transformação M2M, pode-se destacar a ATL (JOUAULT; KURTEV, 2006), desenvolvida pelo grupo ATLAS (INRIA e LINA), que possui um ambiente de desenvolvimento para plataforma Eclipse.

A Figura 14 apresenta em uma forma visual os elementos que compõem uma transformação na linguagem ATL:

- modulo contendo um conjunto de regras que irão guiar o processo de transformação;
- modelo origem, que será transformado em um modelo destino, seguindo as regras de transformação;
- metamodelo origem, com o qual o modelo origem deve estar em conformidade;
- metamodelo destino, com o qual o modelo destino deve estar em conformidade; e
- motor de transformação, que verifica as conformidades dos modelos com seus respectivos metamodelos e realiza o processo de transformação tendo com base as regras de transformação.

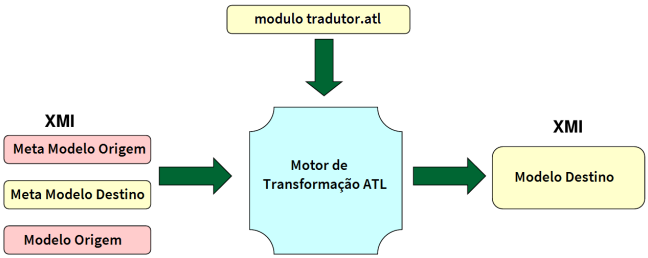


Figura 14 – Motor de Transformação ATL

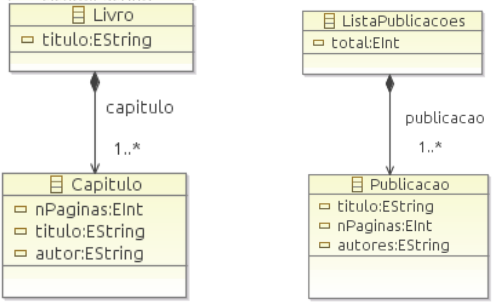
Em ATL, as regras de transformação podem ser decompostas em três partes: **`Header`**, **`Rules`** e **`Helpers`** (JOUAULT; KURTEV, 2006).

O **`Header`** é usado para declarar informações gerais como o nome do módulo, e os metamodelos origem e destino da transformação.

As **Rules** são a principal parte de uma transformação ATL. Elas descrevem as regras de transformação que definirão como o modelo origem é transformado para o modelo destino.

Por fim, os **Helpers** são sub-rotinas usadas para evitar a construção de códigos redundantes na implementação das regras de transformação. Os **Helpers** são usados por outros **Helpers** e por **Rules**.

A codificação de regras de transformação em ATL será apresentada adotando-se um estudo de caso, onde deseja-se criar um modelo chamado **Bibliografia** tendo como entrada um modelo chamado **Livraria**. Tanto os modelos origem quanto o destino devem estar em conformidade com seus metamodelos, apresentados respectivamente nas Figuras 15(a) e 15(b).



(a) MM Livraria (b) MM Bibliografia

Figura 15 – Metamodelos MDE

O Código 11 apresenta o exemplo de um modelo origem, escrito em conformidade com o metamodelo previamente apresentado na Figura 15(a).

Código 11 Modelo Livraria

```
1 <Livro titulo="Livros Introdução à Construção de Algoritmos">
2 <Capitulo nPaginas="10" titulo="Introducao" autor="Cristian Koliver"/>
3 <Capitulo nPaginas="20" titulo="Estruturas" autor="Ricardo Dorneles"/>
4 </Livro>
```

O primeiro passo para codificar as regras de transformação em ATL é criar o **Header** ATL. O Código 12 apresenta o **Header** ATL da transformação **Livraria2Bibliografia**. Na linha 1 deste código é declarado o nome do módulo, que deve ser o mesmo nome do arquivo; nas linhas 3 e 4 são declarados respectivamente os metamodelos destino e origem.

Código 12 `Header` ATL

```
1  module Livraria2Bibliografia;
2
3  create OUT : Bibliografia
4     from IN : Livraria;
```

Após declarar o **`Header`**, deve-se declarar os **`Helpers`** e **`Rules`**. O Código 13 apresenta um **`Helper`** ATL , onde o termo **`Livraria!Livro`** (linha 1) define o contexto de entrada da rotina, nesse exemplo o contexto é o elemento **`Livro`**, pertencente ao metamodelo **`Livraria`**. Nesse **`Helper`**, o termo **`getAutores():String =`** (linha 1) representa o nome da rotina e o tipo de retorno. O apontador **`self`** (linha 2) representa o acesso ao elemento indicado no contexto, que nesse exemplo é o elemento **`Livro`**. O código contido na linha 2 cria uma coleção com todos os autores de capítulos de um livro. O comando **`asSet()->`** (linha 3) recebe a coleção e a transforma essa coleção em um conjunto, eliminando os duplicados. O código contido entre as linhas 3 e 9 apresenta uma estrutura de iteração, que recebe como entrada o conjunto de nomes, e apresenta como saída uma *string* contendo todos os nomes concatenados.

Código 13 `Helper` ATL - Retorna *String*

```
1  helper context Livraria!Livro def : getAutores() : String =
2         self.capitulos->collect(e | e.autor)->
3                asSet()-> iterate(autorNome; acc : String = '' |
4                                  acc +
5                                  if acc = '' then
6                                     autorNome
7                                  else
8                                     ' e ' + autorNome
9                                  endif
10                              );
```

O Código 14 apresenta outro **`Helper`**. Essa segunda rotina tem a função de somar o total de páginas de um determinado livro.

Código 14 `Helper` ATL - Retorna *int*

```
1  helper context Livraria!Livro def : getNPaginas() : Integer =
2         self.capitulos->collect(f|f.nPaginas)->
3                         iterate(paginas; acc : Integer = 0 |
4                                   acc + paginas
5                                );
```

Por fim, o Código 15 apresenta a **`Rule`** que efetua a transformação de um modelo em conformidade com o metamodelo **`Livraria`** para um modelo em conformidade com o metamodelo **`Bibliografia`**. O código contido entre as linhas 2 e 5 acessa o modelo de entrada, restringindo apenas livros que possuem mais de 2 páginas. Nesse código, entre as linhas 6 e 11 ocorre a escrita no modelo destino. É válido destacar o uso de **`Helpers`** nas linhas 4,9 e 10.

Código 15 `Rules` ATL

```
1  rule Livraria2Bibliografia {
2     from
3           b : Livraria!Livro (
4               b.getNPaginas() > 2
5           )
6     to
7           out : Bibliografia!Publicacao (
8                titulo <- b.titulo,
9                autores <- b.getAutores(),
10               nPaginas <- b.getNPaginas()
11          )
12 }
```

O Código 16 apresenta o modelo destino, em conformidade com o metamodelo **`Bibliografia`** previamente apresentado na Figura 15(b). Esse modelo foi gerado pela aplicação das regras de transformação, tendo como entrada o modelo previamente apresentado no Código 11. No processo de transformação, a **`Rule Livraria2Bibliografia`** realiza a geração do modelo destino, obtendo as informações do modelo origem de forma direta (como, por exemplo, o título do livro), ou fazendo uso de um **`Helpers`**. O **`Helper getAutores`** foi usado para obter e concatenar o nome dos autores, já o **`Helper getNPaginas`** foi usado para obter o total de páginas do livro, bem como para limitar a transformação de modo a desconsiderar livros com menos de 3 páginas.

Código 16 Modelo Bibliografia

```
1 <ListaPublicacoes total="1">
2 <Publicacao titulo="Livros Introdução à Construção de Algoritmos"
3    nPaginas="30" autores="Cristian Koliver e Ricardo Dorneles"/>
4 </ListaPublicacoes>
```

Modelo para Texto (M2T)

Em uma linguagem M2T apenas o modelo origem deve estar em conformidade com seu metamodelo, pois em M2T o modelo de destino é codificado em texto plano (ou seja, sem metamodelo).

Como exemplo de uma linguagem de transformação M2T, pode-se destacar a Acceleo, desenvolvido pela empresa francesa Obeo [3]. Assim como a ATL, Acceleo também possui um ambiente de desenvolvimento para plataforma Eclipse.

Em Acceleo as regras de transformação são escritas como rotinas, chamadas **`Template`**. Várias **`Templates`** podem ser criadas em código Acceleo, sendo que uma dessas deve ser indicada como sendo a principal. Em Acceleo o metamodelo de entrada deve ser indicado no início do arquivo contendo as regras de transformação (Obeo, 2008).

A codificação de regras de transformação em Acceleo será apresentada adotando-se um estudo de caso, onde deseja-se criar um arquivo texto contendo as informações existentes no modelo **`Bibliografia`**. O modelos de entrada deve estar em conformidade com seu metamodelo, apresentado previamente na Figura 15(b).

O Código 17 apresenta parte do tradutor Acceleo, onde na linha 2 encontra-se a declaração do metamodelo usado como entrada do transformador. Entre as linhas 4 e 11 encontra-se a **`Template`** principal.

A **`Template`** principal, nomeado por **`tradutor`**, possui como parâmetro de entrada o elemento **`ListaPublicacoes`** pertencente ao metamodelo **`Bibliografia`**. Entre as linhas 7 e 10 foi criada uma área de escrita, todas as saídas de texto nessa área serão escritas no arquivo texto. A linha 8 imprime o texto "Total de Livros:" seguido pelo conteúdo do atributo **`lisPub.total`**. A linha 9 faz uma chamada para outra **`Template`**, passando o elemento **`ListaPublicacoes`** como parâmetro.

[3] http://www.obeo.fr/en/

Código 17 `Main Template` Acceleo

```
1  [comment encoding= ISO-8859-1 /]
2  [module generate('bibliografia.ecore')]
3
4  [template public tradutor(lisPub : ListaPublicacoes)]
5     [comment @main/]
6
7     [file ('ListaBibliografia.txt', false, 'ISO-8859-1')]
8       Total de Livros: [lisPub.total)/]
9       [listaLivros(lisPub)/]
10    [/file]
11 [/template]
```

O Código 18 apresenta uma **`Template`** Acceleo, cuja chamada foi declarada na linha 9 do Código 17. Entre as linhas 3 e 10 encontra-se uma iteração que percorre por todas as publicações existentes no modelo de entrada. Entre as linhas 4 e 9 encontra-se um teste, que verifica apenas publicações que possuem mais de 10 páginas. As linhas 6,7 e 8 imprimem informações referentes a essas publicações com mais de 10 páginas.

Código 18 `Template` Acceleo

```
1  [template public listaLivros(lisPub : ListaPublicacoes)]
2
3   [for(pb : Publicacao |  lisPub.publicacao)]
4    [if (pb.nPaginas > 10)]
5     ****LIVRO****
6     Titulo: [pb.titulo/]
7     Lista de Autores: [pb.autores/]
8     Total de Páginas: [pb.nPaginas/]
9    [/if]
10  [/for]
11
12 [/template]
```

O Código 19 apresenta o modelo destino, gerado pela aplicação das regras de transformação, tendo como entrada o modelo previamente apresentado no Código 16. No processo de transformação, a **`Template tradutor`** escreva as informações referentes à linha 1 do Código 19, e chama a **`Template listaLivros`**, que iterativamente imprime as informações dos livros cujo número de página seja maior que 10.

Código 19 `Template` Acceleo

```
1  Total de Livros: 1
2   ****LIVRO****
3   Titulo: Livros Introdução à Construção de Algoritmos
4   Lista de Autores: Cristian Koliver e Ricardo Dorneles
5   Total de Páginas: 30
```

3.4 METAMODELOS

Conforme anteriormente mencionado, a metodologia proposta é dividida em quatro fases: Modelagem, Transformação, Verificação e Diagnóstico/Correção. As fases de Modelagem e Transformação fazem uso de dois metamodelos, que serão apresentados nessa seção.

O primeiro a ser apresentado é o metamodelo da linguagem de especificação de propriedades LP, desenvolvida neste trabalho, para que o projetista possa especificar as propriedades desejadas na fase de Modelagem.

Na sequência é apresentando o metamodelo chamado Grafo Intermediário (GI), usado na fase de Transformação. O uso de GI tem por objetivo facilitar a inclusão de novas linguagens hipermídia na metodologia proposta, e dar suporte ao uso de uma abordagem baseada em redução de grafos, a ser apresentada. Mais detalhes sobre o uso do GI serão apresentados no Capítulo 4.

Metamodelo LP - (Linguagem de Propriedades)

Visando ter uma representação de propriedades em uma linguagem simplificada, nesse trabalho definiu-se a linguagem para especificação de propriedades LP. Essa linguagem é usada pelo Editor de Propriedades (EP), para representar os comportamentos definidos pelo projetista.

A Figura 16 apresenta a representação do metamodelo da linguagem LP, onde pode-se observar a existência de dois elementos: **`PropertyList`** e **`Property`**.

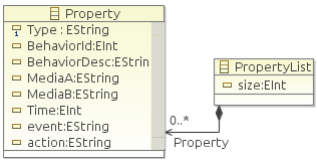


Figura 16 – Metamodelo LP

No metamodelo LP, o elemento **PropertyList** possui como atributo um contador de propriedades **Size** e uma referência para uma lista de elementos do tipo **Property**. O elemento **Property** possui uma lista de atributos das propriedades especificadas pelo projetista usando o EP:

- **Type –** identifica o tipo de propriedade, que pode ser: Intra-mídia, Inter-mídia, Causal e Espacial;
- **BehaviorId –** identificador numérico da propriedade;
- **BehaviorDesc –** descrição textual da finalidade da propriedade;
- **Media1 –** identificador textual de uma mídia cujo comportamento será verificado;
- **Media2 –** identificador textual de uma segunda mídia cujo comportamento será verificado, caso envolva duas mídias;
- **MediaN –** identificador textual de uma segunda mídia cujo comportamento será verificado, caso envolva N mídias;
- **Event –** evento observado, caso seja uma propriedade causal;
- **Action –** ação observada, caso seja uma propriedade causal; e
- **Time –** tempo explícito a ser observado.

Na versão atual, o editor EP permite especificar apenas propriedades entre duas mídias. O Código 20 apresenta o exemplo de um arquivo de propriedades em conformidade com o metamodelo apresentado na Figura 16.

Código 20 Propriedades em LP

```
<PropertyList size="1">
 <Property Type="Intra"   BehaviorId="3"
  BehaviorDesc="sempre termina"
  media1="Shoes"/>
 <Property Type="Spatial" BehaviorId="30"
  BehaviorDesc="sobreposicao temporal e espacial"
  media1="Icone" media2="Background" time="5"/>
</PropertyList>
```

Metamodelo GI - (Grafo Intermediário)

O GI é um grafo dirigido $G = (V,A)$, onde V representa uma lista de vértices de mídias (vídeo, áudio, imagem ...). O conjunto de arestas A é um subconjunto de produtos cartesianos $Condition \times Action$ onde $Condition = \{'onBegin', 'onEnd', 'onAbort', 'onResume', 'onPause', 'onSelection'\}$ e $Action = \{'Start', 'Stop', 'Abort', 'Pause', 'NaturalEnd', 'Resume'\}$.

No GI âncoras são representadas por arestas, não há representação de eventos do tipo atribuição e eventos associados a mais de um elo, do tipo $m\chi n$.

No que tange à estrutura de armazenamento de GI, optou-se pelo uso de uma lista de adjacências, onde cada vértice $v_i \in V(i = 1,2,...,k)$ possui um conjunto de arestas de entrada $a_{in}(v_i) \subset A$ e um conjunto de arestas de saída $a_{out}(v_i) \subset A$.

Em um grafo, uma aresta de entrada em um vértice é uma aresta de saída em outro vértice, conforme ilustrado na Figura 17.

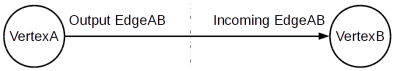


Figura 17 – Aresta GI

Visando armazenar em uma mesma estrutura o modelo da aplicação e as propriedades a serem verificadas, optou-se por criar uma versão estendida de GI. Nessa versão estendida, as propriedades a serem verificadas são adicionadas na estrutura do grafo. A Figura 18 apresenta o metamodelo referente à versão GI+LP.

Nesse metamodelo, o elemento raiz é chamado **`IntermediateGraph`** e possui três elementos filhos:

- **`VertexList`** - é uma lista de elementos do tipo **`Vertex`**, ou seja, armazena todos os vértices existentes no grafo;
- **`NestedContext`** - armazena informações referentes a aninhamentos existentes na aplicação hipermídia; e
- **`PropertyList`** - ponteiro para o metamodelo LP.

Cada elemento **`Vertex`** possui um conjunto de arestas de entrada e saída, representadas, respectivamente, pelos elementos **`IncomingEdgeList`** e **`OutputEdgeList`**. Os vértices de saída são classificados de acordo com as condições causais que levam à sua ativação. Para tal, adotou-se a estrutura **`OutputEdgeListType`**. Algo semelhante ocorre nos vértices de entrada, que são classificados de acordo com as ações geradas por sua ativação. Para tal, adotou-se a estrutura **`IncomingEdgeListType`**. Por fim, os elementos **`OutputEdge`** e **`IncomingEdge`** possuem informações referentes às ligações causais entre os vértices.

É importante destacar que o metamodelo do GI apresentado é uma versão estendida do GI, pois adiciona o metamodelo LP. Em MDE, versões estendidas que unem dois metamodelos são conhecidas como *supermetamodelos*.

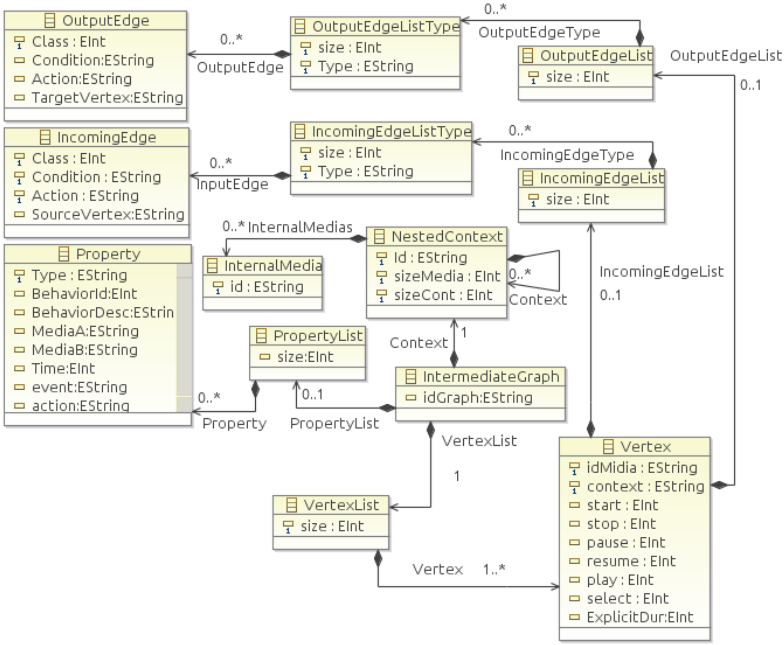


Figura 18 – GI+LP Metamodelo

O *player* Ginga utiliza um grafo chamado HTG (*Hypermedia Temporal Graph)* em sua estrutura interna. Embora GI e HTG (COSTA, 2010) sejam grafos, eles não possuem a mesma estrutura. No IG, os vértices representam mídias, e as arestas representam condições e ações. No HTG cada vértice representa uma ação em uma determinada mídia, e as arestas representam as condições que quando satisfeitas ativam as ações.

Existem outros trabalhos voltados a edição e apresentação de aplicações hipermídia que também fazem uso de grafos.

O trabalho de (YU; XIANG, 1997) apresenta um ambiente de apresentação e autoria para sistemas hipermídia. Este sistema associa dinamicamente questões espaciais com questões temporais. Para representar as relações ele faz uso de um grafo dirigido chamado MRG (*Media Relation Graph*), onde os nodos representam as mídias, e as ligações representam as relações entre as mídias.

Em (ROSSUM et al., 1993) é apresentado um editor e ambiente de apresentação chamado CMIFed. Este ambiente permite ao autor especificar

quando e o que é apresentado, utilizando vários canais de saída simultâneos. A ferramenta de apresentação usa um algoritmo simples, mas eficaz para satisfazer restrições de tempo. O algoritmo começa pela construção de um grafo dirigido de dependências. Os nós no grafo correspondem aos pontos iniciais e finais das mídias, as arestas representam relações de temporização entre dois nós.

3.5 LINGUAGEM DE VERIFICAÇÃO FIACRE

FIACRE (*Format Intermédiaire pour les Architectures de Composants Répartis Embarqués*) é uma linguagem de verificação, desenvolvida no projeto TOPCASED [4], sendo usada, principalmente, para o desenvolvimento de sistemas embarcados. FIACRE permite representar aspectos temporais e comportamentais de sistemas (BERTHOMIEU et al., 2008).

FIACRE é uma linguagem fortemente tipada, tendo seus modelos compostos por dois tipos de construções:

- **`Component`** - elemento que permite construir estruturas hierárquicas em modelos FIACRE. Dentro de um **`Component`** pode-se instanciar paralelamente outros **`Components`** e **`Process`**. Todo modelo FIACRE possui, no mínimo, um **`Component`**, chamado **`Component main`**;
- **`Process`** - máquina de estados usada para modelar um determinado comportamento, adotando transições determinísticas ou não determinísticas.

Em FIACRE, dois ou mais **`Process`** podem comunicar-se de duas maneiras:

- Portas Temporizadas - mecanismo de comunicação baseado em envio e recebimento de mensagens; e
- Memória Compartilhada - área de memória onde dois ou mais processos podem ler e escrever.

Portas podem ser utilizadas para emitir um sinal avisando uma mudança de estado (instrução "`!`" ao lado do nome da porta), ou receber um sinal sendo avisado de uma mudança de estado de outro processo (instrução "`?`" ao lado do nome da porta).

O Código 21 apresenta o exemplo de um componente FIACRE, onde são instanciados paralelamente um **`Process`** (linha 6), e hierarquicamente, um segundo **`Component`** (linha 7).

[4] http://www.topcased.org/

Código 21 Componente FIACRE

```
1 component  Component_Exemplo1 [...] is
2
3  ...
4
5  par * in
6         processo_Exemplo1[...]
7  ||     Component_Exemplo2[...]
8  end
```

O Código 22 apresenta o exemplo de um **`Process`** composto por quatro estados. Esse **`Process`** inicia no estado *estado*1 (linha 6), onde espera pela chegada de um sinal pela porta *porta*1. Na chegada desse sinal, o **`Process`** passa para o estado *estado*2 (linha 7). Desse novo estado, o **`Process`** pode, não deterministicamente, escolher dois caminhos: enviar um sinal pela porta *porta*2 e ir para o estado final *estado*3 ou enviar um sinal pela porta *porta*3 e ir para o estado final *estado*4.

Código 22 Processo FIACRE

```
1  process nome_processo  [...] is
2          states estado1, estado2, estado3, estado4
3
4          init to estado1
5
6          from estado1    porta1?        to estado2
7          from estado2    select
8                            porta2!      to estado3
9          []               porta3!      to estado4
10                         end
```

Portas FIACRE são síncronas, ou seja, o remetente de uma mensagem bloqueia até que o receptor possa recebê-la. Uma comunicação assíncrona, como em dispositivos TVDI para apresentação de aplicações hipermídia, pode ser representada em FIACRE usando memória compartilhada ou um processo chamado **`Glue`**. Um **`Glue`** é um **`Process`** intermediário que conecta um **`Process`** emissor com um **`Process`** receptor, gerando uma comunicação assíncrona entre eles. Quando o emissor tenta enviar uma mensagem para um receptor que não está apto a recebê-la, o **`Glue`** descartará tal mensagem, não bloqueando o emissor.

A linguagem FIACRE possui o comando *wait* que permite especificar intervalos temporais de espera utilizados em **`Process`** para representar períodos de tempo consumidos por uma determinada tarefa (BERTHOMIEU et al., 2011).

Além do modelo formal representando a aplicação, pode-se usar FIACRE para representar as propriedades a serem verificadas por meio de processos observadores ou fórmulas de lógica temporal.

3.6 VERIFICAÇÃO FORMAL DE MODELOS (*MODEL CHECKING*)

Verificação formal de modelos é uma técnica que permite detectar erros e outros problemas de especificação em sistemas computacionais de maneira automática (DRECHSLER, 2004).

Pode-se classificar os tipos de verificação formal em três categorias:

- Verificação de Equivalência - garante a equivalência entre dois modelos, que podem ser representados em diferentes níveis de abstração (DRECHSLER, 2004);
- *Model Checking* - verifica a satisfação de determinadas propriedades por um modelo; e
- Observadores e Alcançabilidade - observa o comportamento de um modelo e dos estados que podem ou não ser alcançados.

No presente trabalho, adotam-se *Model Checking* e Observadores para verificar os comportamentos do modelo.

Model Checking

Dentre os formalismos mais utilizados para especificação e avaliação de propriedades em um modelo, destacam-se as lógicas temporais. Na verificação de um modelo, uma fórmula lógica define um comportamento que o sistema deve ter quando executado.

O modelo a ser verificado deve estar representado na forma de uma máquina de estados (autômato), composta por estados e transições. Um estado é uma descrição do sistema em um instante de tempo, isto é, os valores associados as suas variáveis naquele instante. Transições são relações entre estados que expressam as mudanças de estados.

Uma computação é uma sequência infinita de estados, onde cada estado é obtido a partir do estado anterior usando uma relação de transição. Todas as possíveis execuções do autômato geram uma árvore chamada Estrutura de Kripke. Computações são representadas como caminhos numa Estrutura de Kripke.

Nessa estrutura, as proposições que rotulam um estado são mais importantes do que as ações que rotulam as transições de um autômato, permitindo que os rótulos das transições sejam omitidos. Dessa forma, um autômato pode ser visto como uma 4-tupla $A = (Q;T;Q_0;l)$ onde Q é o conjunto de estados, $T \subseteq Q \times Q$ é o conjunto de transições, Q_0 é o estado inicial e l é o conjunto de rótulos associados às transições.

Um rótulo do estado deve associar a cada estado $q \in Q$ o conjunto de $l(q)$ de proposições atômicas verificadas por q. Tal rotulação é essencial e deve ser feita juntamente com a construção da estrutura.

Pode-se considerar verificação formal como sendo o seguinte problema de decisão:"Dada uma estrutura Kripke K e fórmulas temporais A, as fórmulas temporais em A são válidas para K?".

Formulas de Lógica Temporal

Lógica temporal é adequada para especificar sistemas concorrentes. Ela expressa a ordem dos eventos que ocorrem durante a execução do sistema, sem tempo explícito. Os modelos mais utilizados são:

1. *Linear Temporal Logic* (LTL) - define o comportamento como um conjunto de sequências infinitas, sem examinar as execuções alternativas; e
2. *Computation Tree Logic* (CTL) - expressa o comportamento como uma "árvore de caminhos", onde cada ramo representa uma sequência de alternativa de estados.

A LTL se interessa por uma sequência de estados ao longo de um caminho único, enquanto CTL quantifica os caminhos possíveis de cada estado. Em ambas as lógicas, o estado inicial é a raiz da árvore de computação (CLARKE et al., 2001); (BERARD et al., 2010).

Neste trabalho será adotada a linguagem LTL para especificar os comportamentos a serem verificados. A escolha por LTL deve-se ao fato de ser

uma linguagem mais expressiva e intuitiva, dando mais suporte a criação de fórmulas de maneira composicional (NAIN; VARDI, 2007).

As linguagens CTL e LTL possuem operadores lógicos como:

- *true* e *false* - constantes;
- $\neg$ - negação;
- $\vee$ - conjunção;
- $\wedge$ - disjunção;
- $\Rightarrow$ - implicação; e
- $\Leftrightarrow$ - dupla implicação.

Ambas linguagens possuem os seguintes operadores temporais:

- *Next* (**()**) - A propriedade é satisfeita no próximo estado;
- *Future* ($\Diamond$) - A propriedade é satisfeita em algum estado futuro;
- *Always* ($\Box$) - A propriedade é satisfeita em todos os estados futuros; e
- *Until* (**U**) - A propriedade *P1* é satisfeita até que a *P2* seja satisfeita.

A verificação de modelos corresponde ao seguinte problema de decisão: "Considerando uma estrutura de Kripke K e uma fórmula de lógica temporal Φ, K satisfaz a fórmula Φ?".

Mais detalhes de lógica temporal pode ser encontrado no Anexo (G) deste documento.

Observadores

Adotando-se apenas fórmulas LTL ou CTL, não se consegue representar a passagem do tempo de forma explícita. Quando é necessário verificar comportamentos temporais que envolvem a passagem do tempo, relacionamentos espaciais ou causalidades, pode-se fazer uso de observadores. A inclusão de observadores permite detectar os momentos correspondentes ao início ou ao fim de uma exibição bem como medir o tempo e a duração de uma determinada situação.

Um observador pode ser representado por um autômato que escuta algumas informações pré-definidas por meio de mensagens recebidas da aplicação observada. Observadores não influenciam o funcionamento interno da aplicação.

Observador Básico

Visando verificar comportamentos que envolvem medir a passagem do tempo, além de fórmula LTL, no presente trabalho foram usados observadores. Os observadores adotados foram construídos tendo como estrutura geral um observador básico, apresentado na Figura 19.

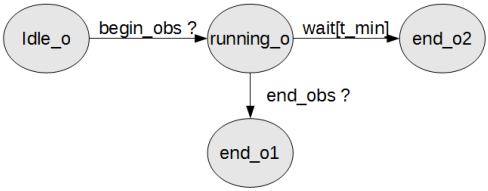


Figura 19 – Observador Básico

O Código 23 ilustra um processo FIACRE representando o observador ilustrado na Figura 19. Esse observador verifica o tempo de exposição de uma mídia. A partir do estado *idle_o*, ele se move para o estado *running_o* após receber a mensagem *begin_obs*, proveniente da aplicação. Do estado *running_o*, dois eventos podem ocorrer: (1) a passagem do tempo ($wait[t_{min}]$) e a alteração para o estado *end_o*2, o que corresponde a $observation\_time > t_{min}$; (2) o recebimento da mensagem *end_obs* que leva ao estado *end_o*1, que corresponde a $observation\_time \leq t_{min}$.

Código 23 Observador básico FIACRE

```
1 process basic_obs  [begin_obs:bool,end_obs:bool] is
2         states idle_o, running_o, end_o1, end_o2
3         init to idle
4         from idle_o     begin_obs?           to running_o
5         from running_o  select
6                           end_obs?           to end_o1
7         []                wait[t_min];  to end_o2
8                         end
```

Com base no observador apresentado no Código 23, foram construídos observadores para verificar todos os comportamentos de Allen definidos em (ALLEN, 1983). O observador apresentado na Figura 20 refere-se ao comportamento de Allen que verifica se “A sobrepõe B” medindo o tempo entre o início de “A” e o início de “B”. Este observador não mede o tempo entre o final de “A” e o final de “B”.

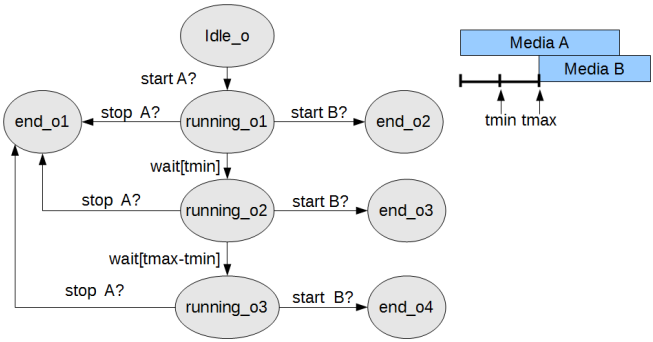


Figura 20 – Observador A sobrepõe B

Observador de Tempo Global

Existem situações onde faz-se necessário uma medida quantitativa e explícita do tempo: a Figura 21 mostra a estrutura de um “Observador de Tempo Global”, onde com cada mudança de estado representa a passagem de 1 (uma) unidade de tempo.

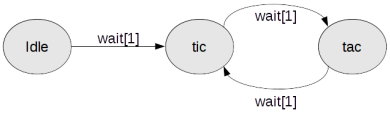


Figura 21 – Observador de Tempo Global

O “Observador de Tempo Global” é útil para analisar e identificar, no contraexemplo, o momento exato da ocorrência dos eventos que levaram à não satisfação de uma determinada propriedade.

3.7 DESCRIÇÃO GERAL DO AMBIENTE DE DESENVOLVIMENTO

Visando dar suporte ao uso da metodologia proposta, foi construído um ambiente de desenvolvimento que integra ferramentas de autoria com ferramentas de verificação formal. Nesta seção serão descritos os componentes que integram esse ambiente.

Conforme mencionado previamente, a metodologia é composta por um conjunto de fases, que vão da modelagem da aplicação hipermídia até análise dos resultados. No ambiente proposto, tais fases são divididas em quatro, as quais são ilustradas na Figura 22.

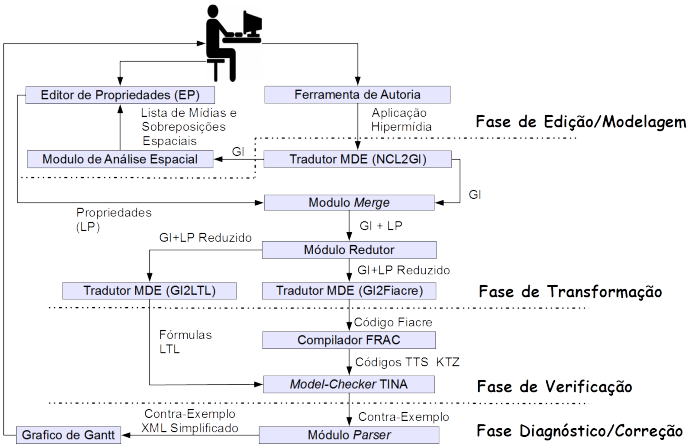


Figura 22 – Estrutura Geral do Ambiente

1 - Fase de Edição

Nessa fase o projetista codifica sua aplicação hipermídia e um conjunto de propriedades a serem verificadas.

Para codificar sua aplicação, o projetista pode fazer uso da ferramenta de autoria de sua preferência. As aplicações hipermídia geradas são gravadas no formato XML.

Para especificar o conjunto de propriedades a serem verificadas, o projetista deve adotar a linguagem de propriedades LP, definida para este propósito. A especificação das propriedades em LP é realizada por meio do EP, desenvolvido neste trabalho, que será apresentado no capítulo 5.

2 - Fase de Transformação

As linguagens hipermídia têm como principal foco proporcionar facilidades para o projetista no processo de codificação, como, por exemplo, permitir o reúso de partes do código. Visando esse objetivo, a estrutura das linguagens hipermídia separam o código em seções, de acordo com sua finalidade.

A representação das relações temporais entre as mídias adotando *links* não se mostrou muito adequada para implementação do ambiente de desenvolvimento pretendido. Assim, conforme anteriormente mencionado, optou-se por fazer uso de uma estrutura intermediária na forma do grafo dirigido GI. O GI contém as mesmas informações existentes no modelo original, mas estruturadas na forma de uma lista de adjacências.

Para obter-se a aplicação hipermídia estruturada na forma do grafo GI, adotou-se uma transformação automática baseada na abordagem MDE. Essa transformação segue um conjunto de regras que serão detalhadas e exemplificadas no Capítulo 4.

O uso do GI, além de facilitar a implementação dos demais elementos do ambiente proposto, permite a inclusão de novas linguagens hipermídia no ambiente, sem a necessidade de reconstruir todos seus módulos, apenas o módulo de tradução dessa nova linguagem para o formato do GI tem que ser feito.

Então, após a obtenção do GI e do conjunto de propriedades codificadas na linguagem LP, efetua-se a junção de GI+LP e, na sequência, o processo de redução. Nesse processo, para cada propriedade, é gerado um GI+LP reduzido, onde a redução é guiada pela propriedade. Esta redução segue um conjunto de regras, descritas no Capítulo 4.

Após o processo de redução, ocorre uma segunda tradução MDE. Nessa tradução, cada modelo na forma de GI+LP é automaticamente traduzido em um modelo formal FIACRE. As regras que guiam essa tradução

também são detalhadas e exemplificadas no Capítulo 4.

O modelo em FIACRE contém a representação formal reduzida da aplicação hipermídia, integrada com o conjunto com observadores e propriedades LTL que verificam os comportamentos desejados.

3 - Fase de Verificação

Terminada a etapa de tradução, a cadeia possui o modelo formal em FIACRE reduzido. A seguir, ocorre a etapa de verificação. Essa etapa usa o compilador Frac (BERTHOMIEU et al., 2008) e o verificador Tina (BERTHOMIEU; RIBET; VERNADAT, 2004) para análise da satisfação das propriedades verificadas. Para cada modelo em FIACRE reduzido, é verificada a propriedade que guiou sua redução. Um conjunto de modelos e propriedades podem ser verificados de forma paralela.

4 - Diagnóstico/Correção de Erros

Após o término do processo de verificação, normalmente algumas propriedades são satisfeitas e outras não. A ferramenta de verificação apresenta seus resultados na forma de texto plano e, para as propriedades não satisfeitas, apresenta também um contraexemplo, que vem a ser a sequência de eventos que levou à não satisfação da propriedade (vide Capítulo 5).

4 A FASE DE TRANSFORMAÇÃO NA METODOLOGIA PROPOSTA

Essa fase visa a transformação de uma aplicação codificada em uma linguagem hipermídia (NCL ou SMIL) e de um conjunto de propriedades desejadas codificadas na linguagem LP, respectivamente para o modelo intermediário de verificação (FIACRE) e para as fórmulas LTL, permitindo, assim, a verificação da aplicação.

Para realizar a transformação, essa fase adota dois passos, utilizando como modelo intermediário o grafo GI. Dois módulos auxiliares, conforme ilustrado na Figura 23, permitem mesclar comportamentos e propriedades e reduzir GI.

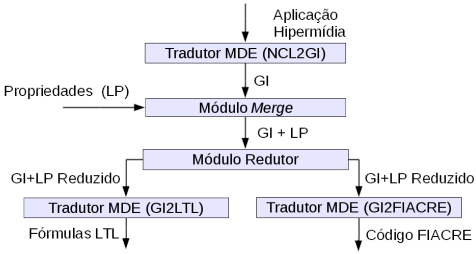


Figura 23 – Tradução em Dois Passos

No primeiro passo, a aplicação escrita em linguagem hipermídia, é traduzida para o formato GI, seguindo uma transformação M2M.

No segundo passo de transformação, a aplicação (representada por GI) e um conjunto de propriedades são traduzidos para um código na linguagem FIACRE e fórmulas LTL, usando duas transformações M2T.

O uso de transformações em dois passos e do GI facilita a integração de novas linguagens hipermídia na cadeia de verificação, requerendo, para tal, a construção de um novo transformador específico SMIL2GI para o primeiro passo. Outra vantagem do uso do GI é a possibilidade de reduzir o tamanho da aplicação usando regras de redução baseadas em algoritmos e estruturas de dados consolidados na Teoria dos Grafos. Mais detalhes dessa redução serão apresentados adiante neste capítulo.

No que se refere à consistência sintática, na primeira transformação, os modelos de entrada e saída são validados em relação a seus metamodelos, seguindo a abordagem MDE para transformações M2M. Na segunda transformação, que segue a abordagem MDE para transformações M2T, a consis-

tência sintática é validada apenas para o modelo de entrada em relação ao seu metamodelo. A consistência sintática do modelo FIACRE (no formato de texto plano) é validada durante a fase de avaliação, quando ocorre sua compilação para um código básico orientado à verificação (TTS (HENZINGER; MANNA; PNUELI, 1992) no caso).

Este capítulo apresenta os elementos da fase de transformação na metodologia proposta, envolvendo metamodelos, redução e transformações com suas regras.

Visando facilitar a compreensão do leitor, este capítulo apresenta cada parte da fase de transformação em uma subseção, seguindo a ordem de dependência entre as partes:

1. Transformação da Linguagem Hipermídia (NCL) para GI;
2. Redução de GI;
3. Transformação de GI+LP em um modelo na Linguagens FIACRE; e
4. Transformação de GI+LP em conjunto com propriedades nas Linguagens FIACRE e LTL.

4.1 TRANSFORMAÇÃO DE NCL PARA GRAFO INTERMEDIÁRIO

Nesta seção serão apresentadas e exemplificadas as regras de transformação que guiam o processo de tradução de uma aplicação escrita na linguagem hipermídia NCL para o formato intermediário GI.

4.1.1 Regras de Transformação de NCL para GI

Na transformação NCL para GI, o modelo de entrada deve estar em conformidade com o metamodelo NCL, apresentado no Capítulo 2 (Figura 3 página 34). O modelo GI gerado pela transformação deve estar, por sua vez, em conformidade com seu metamodelo, apresentado no Capítulo 3 (Figura 18 página 76).

O processo de tradução MDE é guiado por um conjunto de regras de tradução, ilustradas na Tabela 3 e listadas abaixo:

Regra 1 - cada `Media` NCL é traduzida para um `Vertex` em GI, que é um elemento interno de `VertexList`;

Regra 2 - os `Links` e `CausalConnectors` NCL que têm uma determinada mídia como origem, são traduzidos para `OutputEdge` internos ao `Vertex` obtido na aplicação da **Regra-1** para essa mídia origem. Já os `Links` e

`CausalConnectors` NCL que têm uma determinada mídia como destino, são traduzidos para `IncomingEdge` internos ao `Vertex` obtido na aplicação da **Regra 1** para essa mídia destino;

Regra 3 - cada `Region` do NCL é traduzida num atributo no `Vertex` de GI;

Regra 4 - os `Contexts` são traduzidos como um atributo no `Vertex` e um elemento de `NestedContext` do GI; e

Regra 5 - o atributo `ExplicitDur` NCL é traduzido para um atributo no `Vertex` de GI.

No caso de um elemento `Media` estar associado a mais de um contexto (reúso), no processo de tradução este resultará em mais de um `Vertex`.

Tabela 3 – Correspondência NCL - GI a partir das regras

NCL	**Grafo Intermediário**	
Media	Vertex	
Links e CausalConnectors	OutputEdge	Vertex
	IncomingEdge	Vertex
Region	Region Attribute	Vertex
Context	Context Attribute	Vertex
	Context	NestedContext
ExplicitDur Attribute	ExplicitDur Attribute	Vertex

4.1.2 Aplicação das Regras

O Código 24 apresenta parte de uma aplicação NCL onde são declaradas duas mídias (iniciando nas linhas 5 e 7), um `link` (iniciando na linha 10). O `link` indicam que a seleção do botão “RED” na mídia “video1” causará o início da mídia “video3”.

Código 24 Documento NCL

```
1  <causalConnector id="onKeySelectionStart">
2     <simpleCondition role="onSelection"/> <simpleAction role="start"/>
3  </causalConnector>
4  ...
5  ...
6  ...
7  <media id="video1" src="video1.mpg"/>
8  <context id="contV2">
9    <media id="video2" src="video2.mpg" explicitDur="20s"/>
10 </context>
11 ...
12 <link id="l1" xconnector="onKeySelectionStart">
13    <bind role="onSelection" component="video1" value="RED"/>
14    <bind role="start" component="video2"/>
15  </link>
```

O Código 25 apresenta o modelo GI, obtido pela transformação do NCL para GI, aplicando as regras descritas abaixo:

- **Regra 1** - os elementos `Media` "Video1" e "Video2" (linhas 5 e 7 do código NCL) são traduzidos para elementos `Vertex` em GI (linhas 1 e 8 no código GI);
- **Regra 2** - o `Link` NCL (linhas 10) é traduzido para um `IncomingEdge` no `Vertex` "Video2" do GI (linhas 11) e para um `OutputEdge` no `Vertex` "Video1" do GI (linhas 4).
- **Regra 4** - o `Context` em NCL (linha 6) é traduzido para um atributo no `Vertex` "Video2" (linha 8 no código GI); e
- **Regra 5** - o atributo `ExplicitDur="20"` em NCL (linha 7 do código NCL) é traduzido para o atributo `ExplicitDur="20"` no `Vertex` "Video2" (linha 8 no código GI).

Neste exemplo a **Regra 3** não foi aplicada pois não foram definidas `Regions` na aplicação NCL apresentada no Código 24.

Código 25 Documento GI traduzido de NCL

```
<Vertex idMedia="video1" context="main"/>
<OutputEdgeList size="1">
 <OutputEdgeType size="1" Type="onSelection">
  <outputEdge Condition="onSelection" key="RED" target="video2" id="L1"/>
 </OutputEdgeType>
</OutputEdgeList>
</Vertex>
<Vertex idMedia="video2" context="contV2" explicitDur="20s"/>
<IncomingEdgeList size="1">
 <IncomingEdgeType size="1" Type="start">
  <incomingEdge Action="start" sourceVertex="video1" id="L1"/>
 </IncomingEdgeType>
</IncomingEdgeList>
</Vertex>
```

4.2 REDUÇÃO DO GRAFO INTERMEDIÁRIO

Quando adotada para verificação de comportamentos envolvendo mídias ou propriedades específicas, uma metodologia que gere o modelo, tendo como entrada a aplicação completa, pode ter um custo de verificação elevado. Visando resolver esse problema de custo computacional, na metodologia proposta adotou-se uma abordagem baseada em redução da aplicação, já representada na forma de GI.

Na abordagem proposta, para cada propriedade a ser verificada é gerado um grafo reduzido $GI_k = (V_{GI_k}, A_{GI_k})$. O processo de redução proposto é guiado por um conjunto de critérios de preservação e regras de redução. Critérios de preservação restringem a redução de alguns elementos da aplicação. Já as regras de redução definem como a redução deve ocorrer. Ambos estão relacionados com a semântica de aplicações. Note que $V_{GI_k} \subset V_{GI}$, exceto para $k = 0$, $(GI_0 = GI)$ e GI_k *não é* um sub-grafo de GI.

Critérios de Preservação - definiram-se dois critérios de preservação de vértices que devem ser considerados no processo de redução:

Critério 1 - preservar os vértices que representam mídias interativas;

Critério 2 - preservar os vértices $v_i \in V_{GI_k}$ contendo mídias utilizadas como argumentos de propriedade durante a verificação.

Regras de Redução - definem os casos nos quais pode-se reduzir GI:

Regra 1 - eliminar vértices que não influenciam direta ou indiretamente os vértices mantidos nos critérios de preservação;

Regra 2 - eliminar vértices devido a homomorfismo (processo de mapeamento de um grafos de entrada para um novo grafo, respeitando a estrutura do grafo de entrada). Todos os vértices intermediários de grau 2 (ou seja, que têm apenas 2 vizinhos) entre a raiz r e o vértice v usado na propriedade a ser verificada. No entanto, após a aplicação da regra, GI deve conter uma nova aresta $e_{r\to v}$ que soma os tempos de ativação dos vértices no caminho entre r e v: $(r, v, e_{r\to v}) \mid l_{r\to v} = < condition_{r\to x}, action_{x\to v} >$ como em (BOUYAKOUB; BELKHIR, 2011); e

Regra 3 - ao eliminar vértices de um ciclo, devem ser mantidos, no mínimo, dois vértices.

Aplicação das Regras

A aplicação das regras de redução e preservação é exemplificada considerando o gráfico mostrado na Figura 24. A propriedade a ser verificada analisa se as mídias representadas pelos vértices E e C sempre terminam juntas.

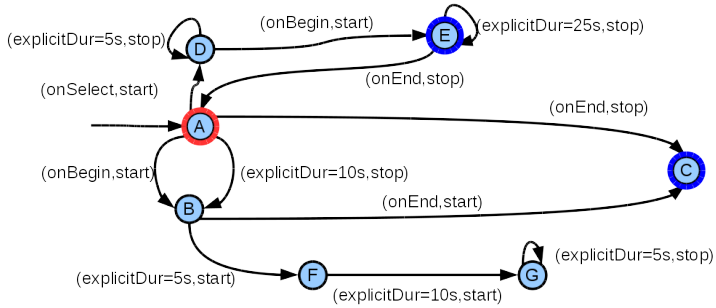


Figura 24 – Exemplo de Grafo Intermediário GI_k

- **Regra 1** - para o grafo GI_k na Figura 24, os vértice F e G não influenciam os vértices A,E e C (Figura 25).

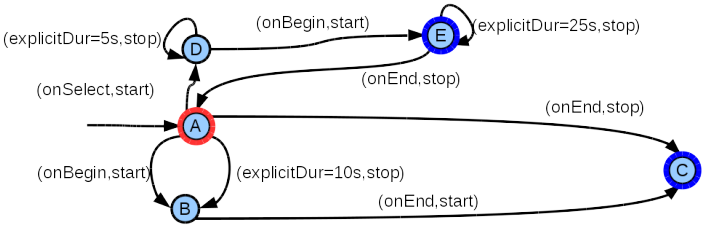


Figura 25 – GI_k aplicando Regra R1

- **Regra 2** - a Figura 26 apresenta GI_k após a aplicação da Regra 2 no grafo da Figura 25. Ela mostra a redução do grafo eliminando-se o vértice intermediário B de grau 2, que tem uma saída pela aresta $l_{B\to C} =< onEnd, start >$ e duas entradas pela aresta $l_{A\to B} =< onBegin, start >$ e pela aresta $l'_{A\to B} =< explicitDur = 10s, stop >$. No exemplo, uma nova aresta é criada para cada *action* da aresta de entrada de B, se essa *action* puder tornar verdadeira a *condition* da aresta de saída de B (neste caso, a condição é *onEnd* e a ação possível é $explicitDur = 10s$). Essa nova aresta liga diretamente os vértices A e C (criada pela junção da *condition* de entrada com a *action* de saída).

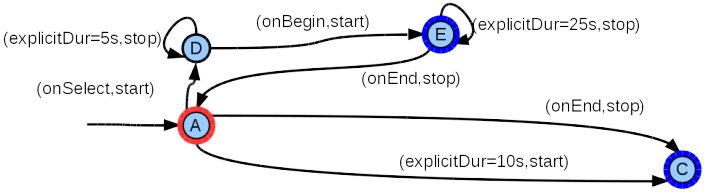


Figura 26 – Aplicando Regra R2

- **Regra 3** - na eliminação por homomorfismo em um circuito, devem ser mantidos no mínimo dois vértices no circuito, como mostrado na Figura 27, onde foi eliminado o vértice D.

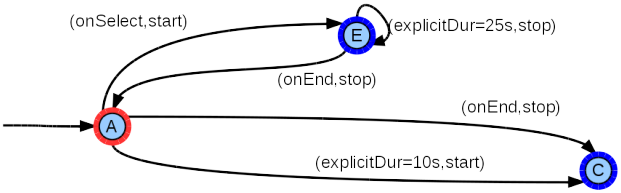


Figura 27 – Aplicando Regra R3

Conclusão

O uso da abordagem baseada em redução permite diminuir consideravelmente o tamanho do modelo GI a ser transformado em um modelo FIACRE e, na sequência, no modelo usado pelas ferramentas de verificação formal. Para cada propriedade a ser verificada, é gerado um modelo reduzido no qual todas as partes do modelo inicial que são desnecessárias ao processo de verificação foram eliminadas. Essa diminuição acelera o processo de verificação formal. Dependendo das restrições existentes na edição "ao vivo", a abordagem baseada em redução pode ser uma alternativa viável. No Capítulo 6 será apresentada uma tabela ilustrando, em números, os ganhos obtidos com a redução na metodologia e ambiente propostos.

4.3 TRANSFORMAÇÃO DO GRAFO INTERMEDIÁRIO PARA LINGUAGEM FIACRE

Conforme mencionado anteriormente na Seção 3.7, o GI possui uma versão estendida, o qual armazena as informações da aplicação e as informações das propriedades a serem verificadas.

Nesta seção, apresenta-se a transformação da aplicação, armazenada no GI estendido, para a linguagem FIACRE.

4.3.1 Regras de Transformação de GI para FIACRE

Na transformação de GI para FIACRE, o modelo de entrada deve estar em conformidade com o metamodelo de GI. Conforme mencionado anteriormente, o modelo resultante da tradução não possui metamodelo por ser uma tradução M2T.

A transformação de GI para FIACRE faz-se por meio do conjunto de

regras descritas abaixo:

Regra 1 - cada `Vertex` no GI é traduzido para um `Media Component` FIACRE, contendo a instanciação de um `Media Process` e de um `Glue Process`;

Regra 2 - os `IncomingEdge` são traduzidos para comandos de recebimento de mensagens dentro do `Glue Process`. Os `IncomingEdge` também são traduzidos em portas declaradas no `Media Component` FIACRE, adicionando-se a letra "g" ao identificador da `IncomingEdge`. Essa porta permite o repasse das mensagens que chegam ao `Glue Process` para o `Media Process`. Os `OutputEdge` são traduzidos para comandos de envio de mensagens dentro do `Media Process`. Cada `OutputEdge` interativo (contendo a condição `onSelection`) gera um evento dentro de um processo `Remote Control`, responsável por gerar interações do usuário;

Regra 3 - o parâmetro `explicitDur`, definido no `Vertex` do GI, cria um atraso `wait[]` interno ao `Media Process`. O parâmetro `explicitDur` também cria uma porta que permite ao `Media Process` o envio de uma mensagem avisando ao seu `Glue Process` o término do atraso definido pelo comando `wait[]`;

Regra 4 - os `Media Component` obtidos na transformação dos `Vertex` são instanciados no `Main Component` FIACRE, respeitando possíveis aninhamentos definidos no atributo `Context`; e

Regra 5 - os `IncomingEdge` são traduzidos em portas declaradas no `Main Component` FIACRE, ou declarados em componentes aninhados, definidos no atributo `Context`. Essas portas são usadas como parâmetros nas instanciações do `Media Component`, `Glue Process` e `Media Process`.

A Tabela 4 ilustra o mapeamento de algumas das regras de transformação, no que tange à criação de processos, componentes, portas e parâmetros.

Tabela 4 – Correspondência GI - FIACRE a partir das regras

Grafo Intermediário		**Fiacre**	
Vextex		Media Component	Media Process
			Glue Process
Vertex	IncomingEdge(Edges)	Port	Main Component
			Media Component
		Input Parameter	Media Component
			Glue Process
			Media Process
		Output Parameter	Glue Process
	OutputEdge(Edges)	Output Parameter	Media Component
			Media Process
	ExplicitDur Attribute	Port	Media Component
		Input Parameter	Media Process
		Output Parameter	Glue Process

4.3.2 Aplicação das Regras

Na exemplificação das traduções a serem apresentadas, considera-se como entrada o fragmento da aplicação na forma de GI apresentado no Código 25.

Componente Media - o Código 26 apresenta o componente FIACRE, obtido na tradução de GI para FIACRE aplicando-se as seguintes regras:

- **Regra 1** - o `Vertex` "Video2" (linha 8 do código GI) é traduzido para o "component_video2" (linha 1 do código FIACRE). Dentro deste componente são instanciados um `Glue Process` e um `Media Process` (linhas 6 e 7 do código FIACRE);
- **Regra 2** - a `IncomingEdge` do `Vertex` "Video2" (linha 11 do código GI) é traduzida para uma porta dentro do `Media Component` (linha 3 do código FIACRE); e
- **Regra 3** - o `explicitDur` do `Vertex` "Video2" (linha 8 do código GI) é traduzido para uma porta dentro do `Media Component` (linha 3 do código FIACRE).

Código 26 Componente FIACRE Video2

```
1  component  Component_Video2 [L1:bool] is
2
3   port          explicitDur:bool in [0,0], gL1:bool in [0,0]
4
5   par * in
6          Glue_Video2[L1,gL1,explicitDur]
7   ||     Media_Video2[gL1,explicitDur]
8   end
```

Cada `Media Component` sempre possui um `Glue Process` e um `Media Process`, conforme descrito na **Regra 1** e apresentado na Figura 28. O `Glue Process` recebe mensagens enviadas por outros processos e, dependendo do tipo de mensagem e do estado atual do `Media Process`, repassa a mensagem para o `Media Process` ou a descarta, assim garantindo a comunicação assíncrona sem *buffer* e com descarte. Os ambientes de exibição de aplicações hipermídia normalmente trocam mensagens de maneira assíncrona, sem *buffer* e com descarte. O uso de uma estrutura que adote um `Glue Process` permite representar tal assincronia dentro de uma linguagem síncrona, como é o caso da FIACRE.

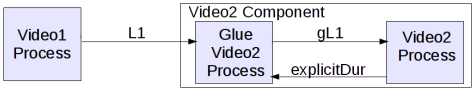


Figura 28 – Comunicação via Glue

Processo Media - o comportamento de cada mídia é mapeado em um `Media Process`, representado por uma máquina de estados. Todas as máquinas de estado que representam um `Media Process` devem ter, no mínimo, três estados: `idle_m` (representando o estado antes da exibição da mídia), `running_m` (representando um estado onde a mídia está ativa, sendo exibida) e `end_m` (representando um estado após a exibição da mídia), além de estados intermediários para receber e enviar mensagens. O Código 27 apresenta o processo `Media Process` para a mídia "Video2", e foi construído pela aplicação das seguintes regras:

- **Regra 1** - o `Vertex` (linha 8 do código GI) é traduzido para um `Media Process` (linha 1 do código FIACRE);
- **Regra 2** - o `IncomingEdge` (linha 11 do código GI) é traduzido para um comando de recebimento de mensagens dentro do `Media Process` (linha 7 do código FIACRE); e

- **Regra 3** - o parâmetro `explicitDur` definido no `Vertex` (linha 8 do código GI) cria um atraso `wait[]` interno ao `Media Process` (linha 8 do código FIACRE), e cria o envio de uma mensagem (linha 9 do código FIACRE).

Código 27 Processo FIACRE Mídia Video2

```
1 process Media_Video2  [L1:bool,explicitDur:bool] is
2         states idle_m, running_m, end_m
3         var result:bool:=false
4
5         init to  idle_m
6
7         from idle_m     L1?result;             to running_m
8         from running_m  wait[20];              to end_m
9         from end_m      explicitDur!result;    to  idle_m
```

Processo Glue - todas as máquinas de estado que representam um `Glue Process` devem ter, no mínimo, dois estados: `idle_g` (representando o estado onde o `Media Process` não está ativo e as mensagens devem ser descartadas) e `running_m` (representando o estado onde o `Media Process` está ativo e as mensagens devem ser repassadas para ele), além de estados intermediários para receber e reenviar mensagens. O Código 28 apresenta o processo `Glue process` para a mídia "Video2", construído seguindo as seguintes regras:

- **Regra 1** - o `Vertex` (linha 8 do código GI) é traduzido para um `Glue Process` (linha 1 do código FIACRE);
- **Regra 2** - o `IncomingEdge` (linha 11 do código GI) é traduzido para um comando de recebimento de mensagens dentro do `Glue Process` (linhas 8 e 13 do código FIACRE), e repasse dessa mensagem (linha 17 do código FIACRE); e
- **Regra 3** - o parâmetro `explicitDur` definido no `Vertex` (linha 8 do código GI) resulta no recebimento de mensagem (linhas 9 e 14 do código FIACRE).

Código 28 Glue FIACRE Video2

```
1  process fiacre_Glue_Video02  [L1:bool,gL1:bool,explicitDur:bool] is
2         states idle_g, running_g, start_g
3          var result:bool:=false
4
5          init to idle_g
6
7          from idle_g     select
8                           L1? result;              to start_g
9          []               explicitDur? result;     to idle_g
10                         end
11
12         from running_g  select
13                          L1? result;              to running_g
14         []               explicitDur? result;     to idle_g
15                         end
16
17         from start_g    gL1! result;             to running_g
```

Processo Remote Control - em aplicações que possuem interação, o processo chamado `Remote Control` é responsável por gerar interações da aplicação.

Na máquina de estados do `Remote Control`, o tempo é discretizado em alguma unidade de tempo, semelhante aos tempos representados nas aplicações hipermídia. Quanto maior a granularidade, maior a possibilidade de ocorrência de explosão combinatória.

O Código 29 apresenta o processo `Remote Control`, criado com base na seguinte regra:

- **Regra 2** - o `outputEdge` do `Vertex Video1` (linha 4 do código GI) é traduzido em um parâmetro (linha 1 do código FIACRE) e é também traduzido para um envio de mensagem representando uma interação (linha 8 do código FIACRE).

Código 29 Processo FIACRE "Remote Control"

```
1  process Remote_Control  [L1:bool] (status) is
2          states running
3          var result:bool:=false
4
5          init to running
6          from running   select
7                              if (status='running') then
8                                 L1!result;             to running
9                              else
10                                wait[1];               to running
11         []                  wait[1];                  to running
12                         end
```

O uso de testes condicionais no processo `Remote Control` (linha 7 do Código 29) permite verificar o estado de cada mídia interativa antes da emissão de mensagens interativas, desse modo diminuindo o número de mensagens. Para verificar o estado das mídias interativas, adotou-se memória compartilhada.

Componente Main - é o componente responsável pela composição paralela dos demais componentes e processos que representam as mídias e os contextos da aplicação.

O Código 30 apresenta parte do componente *main*, que foi criado conforme o seguinte conjunto de regras:

- **Regra 4** - os elementos do tipo `Vertex` (linhas 1 e 8 do código GI) são instanciados como componentes (linhas 8 e 9 do código FIACRE) no `Main Component` FIACRE, respeitando possíveis aninhamentos definidos no atributo `Context`; e
- **Regra 5** - os `IncomingEdge` e `OutputEdge` (linhas 4 e 11) são traduzidos em portas (linha 5 do código FIACRE) usadas na instanciação dos componente (linha 8 do código FIACRE).

Código 30 FIACRE *main* Component

```
component  main is

var        status:flag := [idle]

port         L1:bool in [0,0]

 par * in
 || Component_Video1 [L1] (status) || Remote_Control [L1](status)
 end
main
```

4.4 TRANSFORMAÇÃO DE LP PARA LINGUAGEM FIACRE E LTL

Por estarem representadas em uma linguagem de alto nível, as propriedades definidas em LP devem ser transformadas em fórmulas de lógica temporal codificadas na linguagem LTL, ou em observadores, codificados na linguagem FIACRE, dependendo do tipo de propriedade.

Esta seção inicialmente apresenta a representação das propriedades nas linguagens LTL e FIACRE. Em um segundo momento, ela apresenta o processo de transformação da representação LP para tais linguagens.

4.4.1 Propriedades em LTL e FIACRE

A LP permite especificar quatro categorias de propriedades: (1) Intra-Mídia; (2) Inter-Mídia; (3) Causal; e (4) Espacial, onde, para cada categoria, definiu-se um tipo de representação nas linguagens LTL e FIACRE. Propriedades que verificam diferentes tipos de comportamento sem considerar a contagem do tempo podem ser representadas apenas com fórmulas em LTL. Propriedades que consideram a contagem do tempo necessitam, também, de observadores codificados em FIACRE.

Propriedades Intra-Mídia são usadas para verificar comportamentos internos de uma mídia. Na metodologia proposta definiu-se um conjunto de propriedades Intra-Mídia que utilizam fórmulas LTL e observadores. Propriedades Intra-mídia que não envolvem a contagem do tempo são transformadas em fórmulas LTL:

- sempre que a aplicação tem seu início ativado, no futuro a mídia terá sua exibição iniciada:

$\Box(\Diamond system\_started \Rightarrow \Diamond(media\_started))$

- a mídia nunca tem sua exibição iniciada: $\Box(\neg(media\_started))$
- sempre que a mídia tiver sua exibição iniciada, no futuro terá sua exibição terminada: $\Box(\Diamond media\_started \Rightarrow \Diamond(media\_stopped))$

Propriedades intra-mídia que envolve a contagem do tempo:

- quando a mídia é apresentada, ela permanece por um tempo mínimo/-máximo em exibição. Verifica-se essa propriedade por meio da junção do observador de tempo ilustrado na Figura 29 e da fórmula LTL de não-alcançabilidade $\neg(\Box(observer\_end1))$. A satisfação dessa fórmula indica que a mídia permaneceu em exibição pelo tempo mínimo *tmin*.

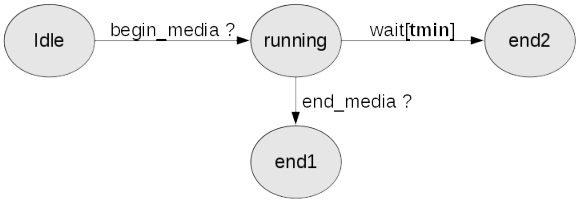


Figura 29 – Observador de Tempo Mínimo

O observador apresentado na Figura 29 captura a mensagem de início de exibição de uma determinada mídia por meio da porta `begin_media`, e a mensagem de término da exibição dessa mídia por meio da porta `end_media`.

Propriedades Inter-Mídia são usadas para verificar comportamentos que envolvem o relacionamento entre um conjunto de mídias.

Para verificar as sete propriedades de Allen (ALLEN, 1983), previamente mostradas no Capítulo 3 (Figura 9 página 61), adota-se um conjunto de observadores e fórmulas de não-alcançabilidade. Nesta seção exemplifica-se apenas uma propriedade de Allen:

- a exibição da mídia "A" inicia após o início da mídia "B" e termina antes do término da exibição da mídia "B". Verifica-se essa propriedade através da junção do observador ilustrado na Figura 30 e da fórmula LTL de não-alcançabilidade $\neg(\Box(observer\_end1))$.

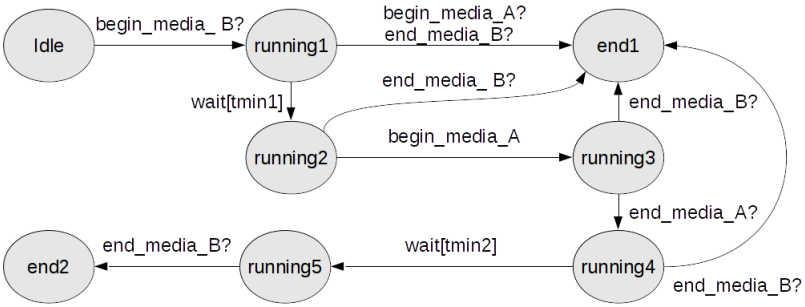


Figura 30 – “A durante B”

O observador apresentado na Figura 30 captura as mensagens de início de exibição das mídias “A” e “B” por meio das porta `begin_media_A` e `begin_media_B` respectivamente. Esse observador captura as mensagens de término de exibição dessas duas mídias por meio das portas `end_media_A` e `end_media_B`. A alcançabilidade do estado `end2` indica a satisfação da propriedade verificada. A não-alcançabilidade do estado `end1` associada ao término das mídias permite constatar a alcançabilidade de `end2`.

Para verificar a sobreposição temporal por um tempo mínimo, adota-se um observador e uma fórmula de não-alcançabilidade:

- sempre que apresentadas, as mídias “A” e “B” possuem sobreposição temporal. Verifica-se essa propriedade por meio da junção do observador ilustrado na Figura 31 e da fórmula LTL de não-alcançabilidade $\neg(\Box(observer\_end1))$.

No observador apresentado na Figura 31, a alcançabilidade do estado `end2` indica a satisfação da propriedade verificada. As mensagens de início de exibição das mídias “A” e “B” são capturadas por meio das portas `begin_media_A` e `begin_media_B`. Esse observador captura as mensagens de término de exibição dessas duas mídias por meio das portas `end_media_A` e `end_media_B`. Nesse observador, a alcançabilidade do estado `end2` indica a ocorrência de sobreposição temporal. Por ser um observador com apenas dois estados finais, a não-alcançabilidade do estado `end1` indica a alcançabilidade de `end2`.

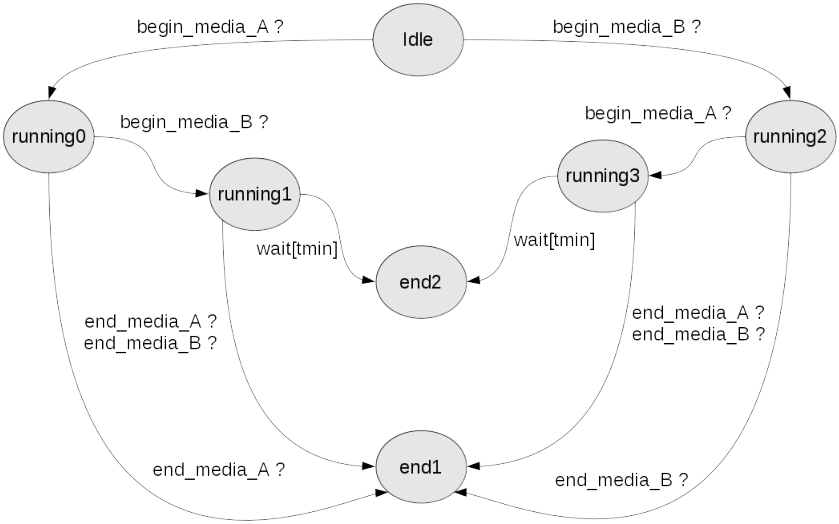


Figura 31 – Observador de Sobreposição Temporal

Propriedades Causais verificam relacionamentos causais, onde um evento (`begin, end, select`) em uma mídia pode causar uma ação (`start,stop`) em outra mídia. As ações (`pause,abort`) não são tratadas.

Para exemplificar uma propriedade causal, a Figura 32 apresenta parte de uma aplicação NCL representada na forma do grafo, onde a ativação da mídia "C" pode ter duas causas, que são: (1) término da exibição da mídia "A", ou (2) término da exibição da mídia "B".

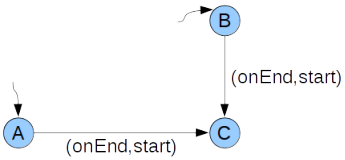


Figura 32 – Parte da Aplicação Hipermídia - Grafo NCL

Para verificar a Causalidade, adota-se um observador e uma fórmula de não-alcançabilidade:

- sempre que a mídia "C" for ativada, a causa de sua ativação é o término da exibição da mídia "A". Verifica-se essa propriedade por meio da junção do observador ilustrado na Figura 33 e da fórmula LTL de

não-alcançabilidade $\Diamond(C\_started \wedge (\neg(observer\_end1)))$.

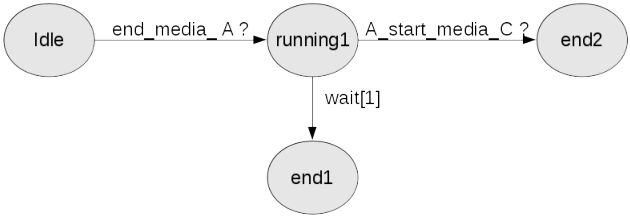


Figura 33 – Observador de Causalidade

Considerando os processos FIACRE representando o comportamento da mídia "C", o observador apresentado na Figura 33 captura as mensagens que chegam ao `Glue Process`, bem como aquelas repassadas do `Glue Process` para o seu `Media Process`. A porta `end_media_A` captura a mensagem que chega ao `Glue Process` avisando o término da exibição da mídia "A". A porta `A_start_media_C` captura a mensagem emitida do `Glue Process` para o `Media Process` ativando a exibição da mídia "C". A alcançabilidade do estado `end1` indica que a mensagem recebida pela porta `end_media_A` é descartada pelo `Glue Process`. Já o estado `end2` indica que tal mensagem é repassada ao `Media Process` adotando a porta `A_start_media_C`. O início da exibição da mídia "C" e a não alcançabilidade do estado `end1` no observador indicam que o término de exibição da mídia "A" foi a causa da ativação da exibição da mídia "C".

É válido destacar que uma fórmula LTL, sem um observador, permitiria apenas provar que "A" é executado após "C", mas não garante que foi "A" quem causou o início de "C".

Propriedades Espaciais permitem verificar a ocorrência de uma sobreposição espacial, total ou parcial, em conjunto com uma sobreposição temporal. Também permitem verificar o uso de uma região da tela considerada inapropriada. Em ambos os casos, é possível verificar o período de tempo no qual tal comportamento permaneceu ativo.

A verificação de sobreposição apenas espacial é realizada por um módulo estático, ainda na fase de especificação de propriedades. Ele é melhor detalhado no Capítulo 5. A ocorrência da sobreposição espacial e temporal no mesmo momento é realizada pela associação do módulo estático de análise de sobreposição espacial com a propriedade Inter-Mídia que verifica sobreposição temporal.

A abordagem proposta não considera a mudança de posição das mídias durante a exibição.

4.4.2 Transformação de LP para FIACRE e LTL

A Tabela 5 ilustra de maneira resumida quais os tipos de observadores e fórmulas LTL usados na verificação dos diferentes tipos de propriedades.

Tabela 5 – Correspondência LP - Observadores e Fórmulas LTL

LP	Observadores e Fórmulas LTL
Intra-Mídia (sem contagem de tempo)	Fórmula LTL
Intra-Mídia (com contagem de tempo)	Observador de Tempo (mínimo/máximo)
	Fórmula LTL de não alcançabilidade
Inter-Mídia	Observadores de Allen e Sobreposição
	Fórmula LTL de não alcançabilidade
Causal	Observador de Causalidade
	Fórmula LTL de não alcançabilidade
Espacial	Observador de Sobreposição
	Fórmula LTL de não alcançabilidade

O Código 31 representa uma propriedade verificada por meio de um observador e de uma fórmula LTL. Essa propriedade verifica a sobreposição espacial e temporal entre as mídias “Icone”e “Background”, por um tempo mínimo de 5 segundos.

Código 31 Propriedade em LP

```
1 <PropertyList size="1">
2  <Property Type="Inter-media"
3            BehaviorDesc="sobreposicao espacial" BehaviorId="15"
4            mediaA="Icone" mediaB="Background" time="5"/>
5 </PropertyList>
```

O Código 32 apresenta o observador FIACRE necessário para verificar a propriedade previamente ilustrada no formato LP (linha 2 do Código 31). Esse observador é semelhante àquele apresentado na Figura 31. Entretanto, o tempo mínimo no observador ilustrado no Código 32 é de 5 segundos. Nesse observador, a alcançabilidade do estado *end_o*2 indica que ocorreu a sobreposição temporal de, no mínimo, 5 segundos. Todos os estados possuem transições não determinísticas, permitindo, assim, representar todas as possíveis situações a serem observadas.

O Código 33 apresenta o “*main component*” de um modelo FIACRE composto pela instanciação de dois componentes representando duas mídias

Código 32 Observador Sobreposição Temporal 5s FIACRE

```
process Ob_sob  [begin_media_A:bool, begin_media_B:bool,
                 end_media_A:bool, end_media_B:bool] is

  states            idle_o, running_o0, running_o1, running_o2,
                    running_o3, end_o1, end_o2
  init to idle

  from idle_o     select
                    begin_media_A?            to running_o0
  []                begin_media_B?            to running_o2
                  end
  from running_o0 select
                    end_media_A?              to end_o1
  []                begin_media_B?            to running_o1
                  end
  from running_o1 select
                    end_media_A?              to end_o1
                    end_media_B?              to end_o1
  []                wait[5,5];                to end_o2
                  end
  from running_o2 select
                    end_media_B?              to end_o1
  []                begin_media_A?            to running_o3
                  end
  from running_o3 select
                    end_media_A?              to end_o1
                    end_media_B?              to end_o1
  []                wait[5,5];                to end_o2
                  end
```

(Linhas 8 e 9 do código FIACRE) bem como a instanciação do processo observador (Linha 10 do código FIACRE) (Esse observador foi previamente apresentado no Código 32). Abaixo da região de instanciação, encontra-se a declaração em FIACRE da fórmula LTL (Linha 15 do código FIACRE) que verifica a não alcançabilidade do estado “end1”, interno ao observador.

4.5 CONCLUSÃO

Este capítulo objetivou apresentar os elementos que compõem a fase de transformação na metodologia proposta.

Code 33 Componente *main* FIACRE com observador e fórmulas LTL

```
component  main is

var  status:flag := [idle]
port  begin_media_A,end_media_A,
      begin_media_B,end_media_B:bool in [0,0]

par * in
 || Component_Background [begin_media_A,end_media_A]
 || Component_Icone [begin_media_B,end_media_B] (status)
 || Ob_sob [begin_media_A,begin_media_B,end_media_A,end_media_B]
 end
main

/*Mutual exclusion */
property mutex is ltl [] not (Ob_sob/state end1)
assert mutex
```

No primeiro momento foi descrito e exemplificado o processo de transformação da aplicação escrita na Linguagem Hipermídia (NCL) para uma representação na forma d o GI.

Visando a diminuição do tempo necessário para o processo de verificação, na sequência apresentou-se uma abordagem baseada em redução do modelo, onde o GI é reduzido tendo como base as propriedades a serem verificadas.

O capítulo conclui descrevendo e exemplificando as transformações de GI e do conjunto de propriedades (GI+LP), respectivamente, em um modelo da aplicação na linguagem FIACRE e em propriedades nas linguagens FIACRE e LTL. É válido destacar que a metodologia e ambiente propostos poderão ser estendidos para outras linguagens. Dependendo das características dessas novas linguagens, pode ser necessário criar novas regras de transformação e estender a abrangência dos modelos GI e FIACRE, ou apenas criar novas regras de transformação destas linguagens para GI.

5 O AMBIENTE DE DESENVOLVIMENTO

O ambiente de desenvolvimento implementado neste trabalho tem por objetivo dar suporte à metodologia de desenvolvimento proposta. Este capítulo inicialmente apresenta uma descrição detalhada da implementação dos elementos que compõem o ambiente desenvolvido. Em um segundo momento, ele apresenta um guia de uso do ambiente com as principais funcionalidades.

5.1 IMPLEMENTAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

O ambiente desenvolvido é composto por um conjunto de módulos, ou ferramentas, alguns já existentes e outros desenvolvidos no contexto do trabalho.

Esta seção tem por objetivo apresentar cada elemento que compõe o ambiente de desenvolvimento, descrevendo sua tarefa, dados de entrada e de saída. Para os elementos implementados neste trabalho serão apresentados algumas características de sua implementação. A Figura 34 apresenta a estrutura geral do ambiente de desenvolvimento, considerando sua divisão em quatro fases.

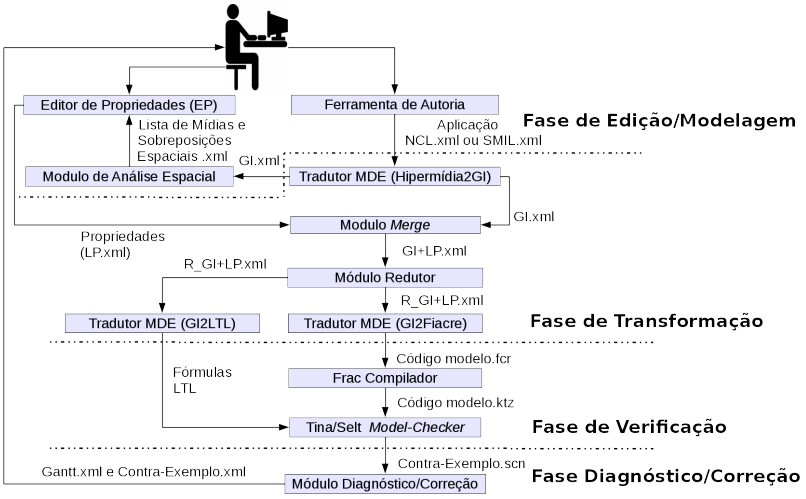


Figura 34 – Estrutura do Ambiente de Desenvolvimento

As informações passadas de um módulo para outro são armazenadas na forma de arquivos. Os arquivos com extensão *fcr* representam modelos formais escritos na linguagem FIACRE. A extensão *ktz* indica um modelo representado na forma de uma estrutura Kripke. Por fim, o arquivo com extensão *scn* indica um contraexemplo na forma textual, gerado na saída do verificador Tina/Selt.

5.1.1 Fases do Ambiente de Desenvolvimento

Assim como a metodologia proposta, o ambiente é composto por quatro fases que se interligam, comunicando-se através da leitura e escrita em arquivos de dados.

1-Fase de Edição

A fase de Edição, primeira das quatro fases, é aquela na qual ocorre o maior nível de interação com o projetista. Esta fase é composta por três elementos: Ferramenta de Autoria, Editor de Propriedades e Módulo de Análise Espacial.

Ferramenta de Autoria

No ambiente proposto neste trabalho não foi desenvolvida uma ferramenta de autoria. O projetista tem a liberdade de usar aquelas com as quais está familiarizado. Como exemplo de ferramentas de autoria, pode-se destacar o NCL-Composer e o Eclipse-NCL para a linguagem NCL e o GRiNS para a linguagem SMIL. Na versão atual, para que possa ser usada no ambiente desenvolvido, a saída da ferramenta de autoria NCL deve ser uma aplicação codificada no formato NCL-Raw (vide Seção 2.2.1), sem erros de sintaxe.

Editor de Propriedades (EP)

O EP, interface para edição de propriedades, foi desenvolvido usando a biblioteca *Java Swing* [1]. O EP é uma interface gráfica amigável onde o projetista pode especificar o conjunto de comportamentos que deseja verificar. Na Figura 35 os retângulos indicam os principais elementos que formam o EP, a saber:

[1] http://docs.oracle.com/javase/tutorial/uiswing/

- Lista da Categoria de Propriedades;
- Lista de Mídias Disponíveis;
- Lista de Propriedades Disponíveis (na categoria selecionada);
- Campo de Tempo Mínimo, onde o projetista indica o tempo mínimo a ser observado pela propriedade selecionada; e
- Lista de Propriedades e Mídias Criadas (irão gerar uma propriedade na linguagem LP).

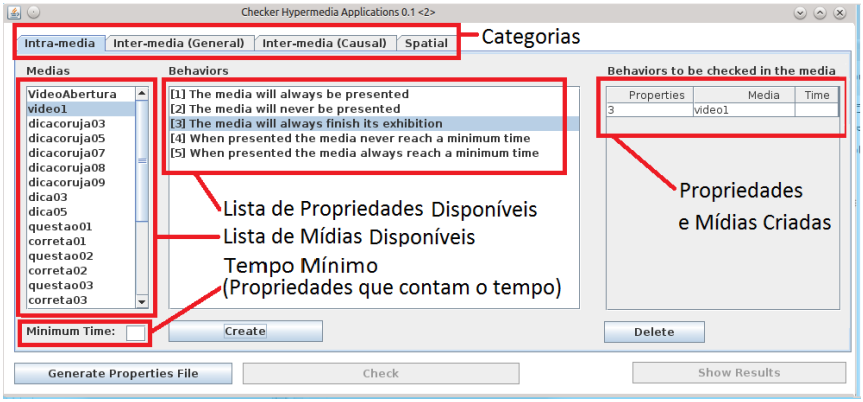


Figura 35 – Editor de Propriedades

O EP recebe como entrada uma lista contendo todas as mídias usadas na aplicação, bem como a relação de todas as possíveis sobreposições espaciais que podem ocorrer durante a apresentação dessa aplicação. Essas informações são obtidas a partir de um módulo chamado Análise Espacial, apresentado no decorrer desta seção.

A saída do EP é uma lista de propriedades escritas na linguagem LP, previamente apresentada na Seção 3.4. Para a mídia e a propriedade selecionada no exemplo da Figura 35, a propriedade a ser criada na linguagem LP é apresentada no Código 34. Essa propriedade verifica se, quando apresentada, a mídia "Video1" sempre tem sua exibição terminada. Por ser uma propriedade Intra-Midia e não verificar tempo explícito, os parâmetros `mediaB` e `time` não possuem valor.

Tanto as informações de entrada quando as de saída são armazenadas em arquivos no formato XML, sendo usada a biblioteca JDOM para criá-los e manipulá-los.

Código 34 Propriedade em LP Referente a Figura 35

```
1 <PropertyList size="1">
2  <Property Type="Intra-media"
3            BehaviorDesc="sempre termina sua exibição"
4            BehaviorId="3"
5            mediaA="Video1"
6            mediaB=""
7            time=""/>
8 </PropertyList>
```

Módulo de Análise Espacial

Objetos visuais (imagem, vídeo, texto) são apresentados em uma região pré-definida da tela, posicionada em relação às demais regiões que podem ser reservadas para outros objetos. O Módulo de Análise Espacial foi desenvolvido em Java e tem por objetivo: (i) obter a lista dos objetos visuais usados na aplicação; (ii) obter os posicionamentos espaciais desses objetos; e (iii) verificar possíveis sobreposições espaciais entre diferentes objetos visuais.

Esse módulo recebe como entrada a aplicação na forma do grafo GI, obtido a partir do tradutor "Hipermidia2IG", apresentado a seguir. Usando a biblioteca JDOM, o módulo carrega as informações sobre as regiões a serem ocupadas pelas mídias de um arquivo GI+PL.xml. Adotando uma estrutura de testes condicionais, verifica se existe alguma região sobreposta, ou o uso de regiões consideradas inadequadas, conforme ilustrado previamente na Figura 10, página 62.

O Código 35 apresenta o exemplo de um possível código gerado pelo módulo de Análise Espacial. De modo mais detalhado, nesse exemplo pode-se observar, as coordenadas cartesianas (`X1,Y1`) e (`X2,Y2`) relativas às regiões espaciais usadas pelas mídias. Nesse exemplo, as coordenadas apresentadas na linha 2 indicam possíveis sobreposições espaciais envolvendo as mídias: *icone* e *bg*. A linha 8 indica que a mídia *photo* pode estar posicionada em uma região de borda que pode ser considerada inadequada. Por fim, entre as linhas 12 e 17, está declarada a lista de todos os objetos visuais.

2-Fase de Transformação

A fase de transformação é composta por três tradutores (Hipermidia2IG, R_IG+PL2FIACRE e R_IG+PL2LTL), um módulo de composição (*merge*) e um módulo redutor.

Código 35 Resultado do Módulo de Análise Espacial

```
<Sobreposicoes quantia="3">
  <Elemento midA="icon"  midB="bg" tipo="2"
      x1A="67.5" x1B="0.0" x2A="75.95" x2B="100.0"
      y1A="81.6" y1B="0.0" y2A="88.29" y2B="100.0"/>
  <Elemento midA="shoes" midB="bg" tipo="2"
      x1A="15.0" x1B="0.0" x2A="40.0"  x2B="100.0"
      y1A="15.0" y1B="0.0" y2A="40.0"  y2B="100.0"/>
  <Elemento midA="photo" midB="[8-1.2]" tipo="20"
      x1A="5.0"  x1B="0"   x2A="23.5"  x2B="0"
      y1A="74.8" y1B="0"  y2A="93.3"   y2B="0"/>
</Sobreposicoes>
<Midias quantia="4">
   <Midia nome="photo"/>
   <Midia nome="icon"/>
   <Midia nome="shoes"/>
   <Midia nome="bg"/>
</Midias>
```

Tradutor Hipermidia2IG

Considerando a linguagem hipermídia NCL, este tradutor recebe como entrada uma aplicação escrita em NCL-Raw e apresenta, como saída, essa aplicação representada na forma do GI.

Para construir o tradutor Hipermidia2IG, adotou-se a linguagem de transformação de modelos ATL (JOUAULT; KURTEV, 2006), que é uma linguagem M2M, bem como o conjunto de regras de transformação baseadas nos metamodelos previamente apresentados nos Capítulos 2 e 3.

No processo de codificação, optou-se por criar pequenas rotinas chamadas `helper`, visando o seu reúso em diferentes partes do código.

Por ser ATL uma linguagem intrinsecamente recursiva, pode ocorrer estouro de pilha na execução de um tradutor implementado nessa linguagem. Esse fator foi considerado na etapa de implementação: não foram criadas recursões profundas, sempre propiciando o desempilhamento quando possível.

Módulo *Merge* - Adiciona Propriedades ao Grafo Intermediário

Conforme ilustrado na Figura 34, esse módulo recebe como entrada dois arquivos, um contendo a aplicação escrita na forma de um grafo `GI.xml` (saída do tradutor Hipermidia2IG), e o outro contendo a lista de propriedades especificadas `LP.xml` (saída do editor EP). Sua saída é um único arquivo

`GI+LP.xml`, em conformidade com o metamodelo GI+LP.

As linguagens MDE (ATL e Acceleo) não permitem o uso de mais de um modelo como entrada de uma transformação. Para suprir essa limitação, a literatura sugere o uso de um super-modelo, que é a união de dois ou mais modelos. Para gerar essa união, criou-se um módulo escrito em Java, usando a biblioteca JDOM.

Módulo Redutor

Este módulo recebe, como entrada, um arquivo `GI+LP.xml` completo; cada propriedade contida em LP orienta o processo de redução gerando uma versão reduzida do modelo GI. Essa versão reduzida, juntamente com a propriedade que a orientou, são armazenadas no arquivo `R_GI+LP.xml`.

O módulo redutor foi construído tendo como base o conjunto de regras de redução e preservação previamente apresentadas na Seção 4.2.

A construção desse módulo foi realizada adotando-se a linguagem Java. Sua estrutura está dividida em três partes:

Carga - os dados do grafo GI, aplicação e propriedades, são carregados em uma estrutura de dados definida a partir do metamodelo do GI+LP;

Redução - adotam-se algoritmos de Teoria dos Grafos, regras de redução e critérios de preservação, para reduzir o tamanho do modelo GI+LP; e

Escrita - grava o GI+LP reduzido em um arquivo no formato XMI, o qual deve estar em conformidade com o metamodelo do GI.

Tradutor GI2Fiacre

Este tradutor recebe o arquivo `R_GI+LP.xml` como entrada e apresenta, como saída, um único modelo FIACRE contendo: (i) os comportamentos da aplicação modelados em FIACRE; e (ii) um observador (caso a propriedade LP necessite o uso de um observador).

Para criar o tradutor GI2Fiacre adotou-se a linguagem de transformação de modelos Acceleo, que é uma linguagem M2T. A escolha de uma linguagem M2T tem como justificativa o fato da próxima fase do ambiente proposto requerer, como entrada, o arquivo `modelo.fcr`, que é apresentado em texto plano.

Na codificação em Acceleo, optou-se por criar pequenas rotinas chamadas `template`, visando o seu reúso em diferentes partes do código.

O código GI2Fiacre ficou com, aproximadamente, 3000 linhas. Essa quantidade se deve a dois fatores: (i) ao gerar o modelo em FIACRE, o código Acceleo precisa respeitar a endentação, quebras de linha e alinhamentos

desejados no modelo FIACRE a ser gerado; e (ii) são implementados geradores para todos os diferentes tipos de `component`, `media process`, `glue process` e observadores.

Tradutor GI2LTL

Este tradutor recebe como entrada o arquivo `R_GI+LP.xml`, fazendo uso apenas da propriedade LP para gerar uma fórmula LTL que será usada pela ferramenta de *model-checking*.

Assim como ocorreu no tradutor GI2Fiacre, na construção deste tradutor optou-se também pela linguagem M2T Acceleo.

A propriedade escrita em LP interna ao arquivo `R_GI+LP.xml` é usada como entrada nos dois tradutores M2T: no GI2Fiacre para gerar um observador em FIACRE, e no GI2LTL para gerar uma fórmula em LTL.

3-Fase de Verificação

A fase de verificação adota um conjunto de ferramentas de verificação desenvolvidas no projeto TOPCASED [2], originalmente usadas na verificação de sistemas embarcados.

A Figura 36 mostra a estrutura de processo de verificação. Conforme esta figura, o compilador Frac recebe, como entrada, um modelo codificado na linguagem FIACRE e apresenta, como saída, um código TTS (*Time Transition System*) que, na sequência, é transformado em uma estrutura Kripke. O verificador Selt/Tina verifica se a propriedade representada por uma fórmula na linguagem LTL é satisfeita pelo modelo representado na estrutura Kripke. Caso a propriedade não seja satisfeita, o verificador Selt/Tina gera um contraexemplo no formato SCN (*shortest counter-example*).

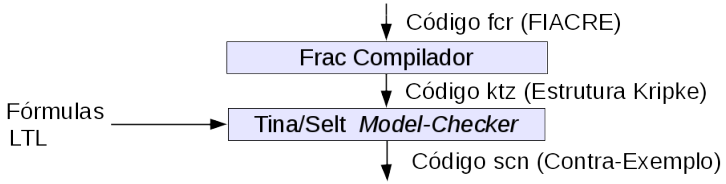


Figura 36 – Cadeia de Verificação

[2] http://www.topcased.org/

Compilador Frac

O compilador Frac pertence ao conjunto de ferramentas da linguagem FIACRE.

O Código 36 (linhas 1 e 3) apresenta o trecho de código do ambiente desenvolvido (na forma de um arquivo `shell script`), onde ocorre a chamada do compilador Frac. Neste código pode-se verificar a entrada do arquivo `modelo.fcr` e a saída dos arquivos `modelo.tts` e `modelo.ktz`. As linhas 2 e 3 apresentam o trecho que código onde o modelo representado em TTS é transformado em uma estrutura Kripke, resultado no arquivo `modelo.ktz`.

Código 36 Shell Script - Chamada Compilador Frac

```
1 frac modelo.fcr  modelo.tts
2 make -f Makefile modelo
3 tina modelo.tts modelo.ktz
```

Verificador TINA/SELT

O Tina/Selt é um *toolbox* para editar e analisar Redes de Petri. Entre as ferramentas pertencentes ao Tina/Selt encontra-se o *model-checker* Selt que verifica se uma dada fórmula LTL é satisfeita por um modelo formal. Caso não seja satisfeita, o Selt retorna como resultado um contraexemplo no formato SCN indicando a sequência de passos que leva a não satisfação da fórmula.

4-Fase de Diagnóstico/Correção

A fase de Diagnóstico/Correção é dividida em duas etapas. A primeira é responsável por fazer a análise dos resultados obtidos; a segunda é responsável por apresentar esses resultados adotando-se uma linha temporal.

Módulo *Parser*

O módulo *Parser* recebe, como entrada, um contraexemplo no formato SCN. Ele analisa quais as partes do contraexemplo são pertinentes ao projetista e identifica o momento em que cada mudança de estado ocorreu.

O Código 37 apresenta parte de um contraexemplo escrito no formato SCN. Nesse exemplo, para fins de simplicidade, são omitidos alguns estados. Para identificar a passagem do tempo em segundos, observa-se a troca de

estados do processo identificado por `p__relogio_1`. Comparando as linhas 1 e 3, rotuladas por `state 0` e `state 1`, observa-se a mudança do processo `p__relogio_1` do estado `tic` para o estado `tac`, representando a passagem de 1 segundo no tempo. Nas linhas 6 e 8 (`state 9` e `state 10`), o processo `p__video1_1` passou do estado `sstopped` para o estado `sstarted`.

Código 37 Contraexemplo em SCN file

```
1  state 0:  g__video1_1_sidle  g__video2_1_sidle
2      p__video1_1_sstopped  p__video2_1_sstopped p__relogio_1_stic
3  state 1:  g__video1_1_srunning1  g__video2_1_sidle
4      p__video1_1_sstopped  p__video2_1_sstopped p__relogio_1_stac
5  ...
6  state 9:  g__video1_1_srunning  g__video2_1_srunning
7      p__video1_1_sstopped  p__video2_1_sstopped p__relogio_1_stac
8  state 10:  g__video1_1_srunning  g__video2_1_srunning
9      p__video1_1_sstarted  p__video2_1_sstopped p__relogio_1_stic
```

Após o processo de análise, esse módulo apresenta para o projetista um contraexemplo no formato XML contendo a lista de troca de estados de cada mídia, e indicando os momentos em que cada troca ocorreu. O Código 38 exemplifica um contraexemplo no formato XML, gerado a partir da versão completa do arquivo SCN apresentado no Código 37.

No código XML, cada mídia gera um elemento `Media` que possui a lista de todos os estados pelos quais a mídia passou, identificados pelo elemento `Line`. No exemplo ilustrado no Código 38, as linhas 2 e 3 indicam a troca de estado da mídia "Video1", do estado `stopped` para o estado `started`. Essa informação foi obtida pelas linhas 6 e 8 (`state 9` e `state 10`) no Código 37.

Código 38 Contraexemplo em XML file

```
1  <Media id='Video1'>
2  <Line id='0' time='0'  state='stopped'>
3  <Line id='1' time='5'  state='started'>
4  <Line id='2' time='10'  state='selected'>
5  <Line id='3' time='12'  state='stopped'>
6  <\Media>
7  <Media id='Video2'>
8  <Line id='0' time='0'  state='stopped'>
9  <Line id='1' time='11'  state='started'>
10 <Line id='2' time='18'  state='stopped'>
11 <\Media>
```

Módulo Linha Temporal

A partir do arquivo XML, a fase de Diagnóstico/Correção apresenta o contraexemplo na forma de linha temporal, construído usando a biblioteca de gráficos JFreeChart [3].

Tendo como base as informações do Código 38, a Figura 37 ilustra o mesmo contraexemplo gerado pelo ambiente na forma de linha temporal.

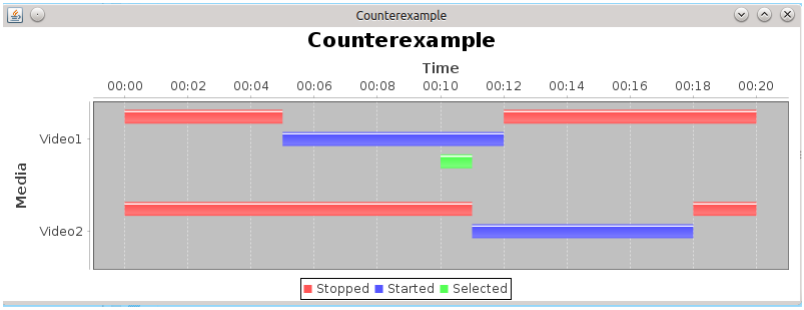


Figura 37 – Contraexemplo em Linha Temporal

5.1.2 Integração dos Módulos, Ferramentas e Tradutores

Para implementar o ambiente proposto, utilizou-se o Eclipse. Ele possui suporte para o desenvolvimento adotando uma gama de linguagens, entre as quais destacam-se as utilizadas neste trabalho:

- ATL e Acceleo - linguagens de transformação de modelos MDE adotadas para codificar os tradutores de modelos, M2M e M2T, respectivamente;
- Java - utilizada para implementar os demais módulos e interfaces gráficas; e
- ShellScript - adotada para escrever *scripts* responsáveis pela execução das ferramentas de verificação formal.

Os tradutores de modelos codificados em ATL e Acceleo foram exportados para a forma de uma biblioteca JAR, podendo ser ativados por qualquer aplicação escrita em Java, ou até mesmo de um terminal que possua instalado a máquina virtual Java.

[3] http://www.jfree.org/jfreechart/

Os *scripts* responsáveis pela execução das ferramentas de verificação formal também podem ser ativados por qualquer aplicação escrita em Java, ou até mesmo de um terminal, se no referido computador estiver instalado o interpretador ShellScript.

A passagem de dados entre os módulos, ferramentas e tradutores foi realizada através da escrita e leitura em arquivos XML, leitura de arquivo texto e escrita em arquivo XML.

As chamadas das rotinas de cada módulos, ferramentas e tradutores foram realizadas por meio de uma chamada codificada em Java, integrada na interface gráfica do EP.

5.2 GUIA DE USO

No uso da metodologia e ambiente propostos, o projetista deve seguir uma sequência de cinco passos.

Passo 1 - Edição da Aplicação

Inicialmente, o projetista escreve sua aplicação hipermídia usando a ferramenta de autoria de sua preferência. Para aplicações codificadas na linguagem NCL, é necessário que a ferramenta de autoria, ou algum *plugin* associado a essa ferramenta, converta a aplicação para o formato "*RAW Profile*". A Figura 38 apresenta uma ilustração da ferramenta de autoria Composer, composta por um conjunto de visões.

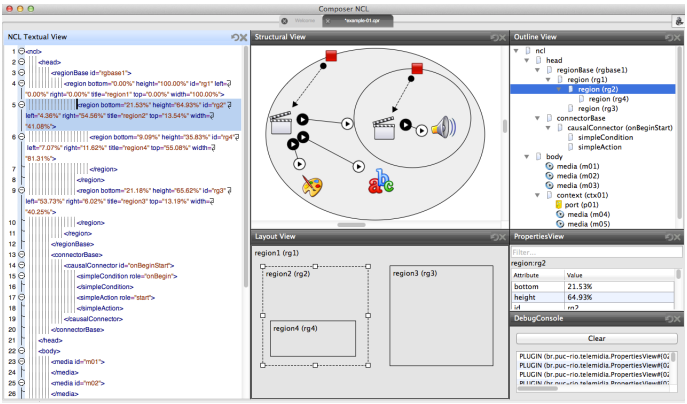


Figura 38 – Ferramenta de Autoria Composer

Entre as visões existentes no Composer, destaca-se a visão “Estrutural”, onde o projetista pode visualizar, de maneira gráfica, as diferentes relações causais entre as mídias declaradas na aplicação. Outra importante visão é a “Textual”, onde o projetista pode visualizar e editar o seu código NCL, tendo seus elementos sintaticamente identificados por diferentes cores.

Passo 2 - Carga da Aplicação

Após a edição da primeira versão da aplicação, o projetista deve efetuar um processo chamado carga, onde ele seleciona o nome do arquivo onde sua aplicação está salva. A interface que permite essa seleção é ilustrada na Figura 39.

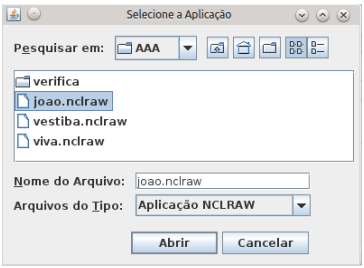


Figura 39 – Interface de Seleção da Aplicação

Passo 3 - Especificação das Propriedades

Após a carga da aplicação, ocorre o passo de especificação de propriedades. Para esse passo, o projetista deve adotar o EP, desenvolvido neste trabalho, e ilustrado na Figura 40. Nesse editor, o projetista pode especificar quatro diferentes tipos de comportamentos a serem verificados: intra-mídia, inter-mídia geral, inter-mídia causal e espacial.

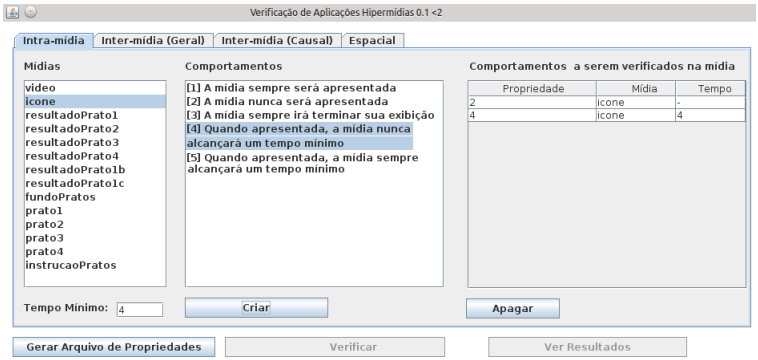


Figura 40 – Interface para definir propriedades

Para especificar cada propriedade, inicialmente o projetista deve efetuar as seguintes seleções:

- uma ou duas mídias, dependendo do tipo de propriedade; e
- a propriedade que deseja verificar para essas mídias.

Tendo efetuado a seleção das mídias e da propriedade, o projetista deve pressionar o botão `[Criar]`, para que a propriedade envolvendo essas mídias seja criada.

Passo 4 - Ativação das Transformações e Verificações

Após a criação das propriedades no **Passo 3**, o projetista deve ativar a transformação de modelos e o processo de verificação.

Para ativar a transformação de modelos o projetista deve pressionar o botão `[Gerar Arquivo de Propriedades]`. Após o processo de transformação, o projetista deve ativar o processo de verificação, pressionando o botão `[Verificar]`.

Passo 5 - Análise e Interpretação dos Resultados

Após o término da verificação, o botão `[Ver Resultados]`, previamente ilustrado na Figura 40, permite ao projetista efetuar a análise e a interpretação dos resultados obtidos no processo de verificação.

No processo de verificação, algumas propriedades são satisfeitas e outras não. A informação de quais propriedades foram satisfeitas é apresentada na interface ilustrada na Figura 41. Nessa interface, os resultados têm sua exi-

bição organizada por categoria de propriedade. As propriedades em vermelho não foram satisfeitas pelo modelo.

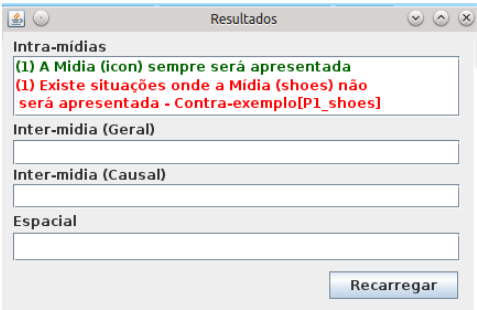


Figura 41 – Resultado da Verificação

Para que o projetista possa ver o contraexemplo dessas propriedades, ele seleciona com um *click* do *mouse*. A Figura 42 ilustra o contraexemplo gerado pelo ambiente na forma de linha temporal. A linha temporal mostra que, aproximadamente aos 10 segundos, a mídia interativa "Video1" foi selecionada, ativando a exibição da mídia "Video2". Caso esse não fosse o comportamento desejado, o projetista deveria voltar ao **Passo 1**, efetuar as correções e repetir novamente todos os passos seguintes.

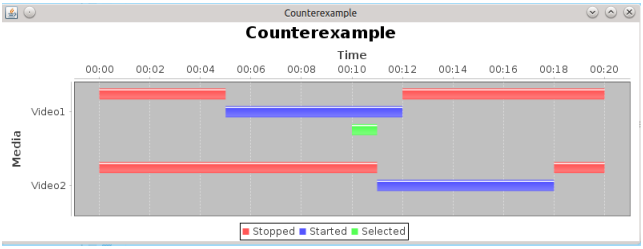


Figura 42 – Estrutura Contraexemplo

5.3 CONCLUSÃO

Durante a implementação do ambiente de desenvolvimento algumas decisões de projeto tiveram que ser tomadas, como: (i) linguagens a serem adotadas, (ii) implementação de mais de um tradutor MDE, (iii) uso de um grafo intermediário, e (iv) implementação de mais de um módulo em Java.

A opção por dividir as tarefas em mais de um tradutor (e mais de um módulo), fazendo uso de diferentes linguagens, teve como objetivo simplificar o processo de implementação, diminuir o tamanho dos módulos e dos tradutores, e usar sempre a linguagem mais adequada para determinada tarefa.

As linguagens MDE usadas possuem suporte para Java. Desse modo, pode-se realizar a junção de todos os elementos da estrutura implementados e apresentados na Figura 34, assim criando uma única ferramenta, integrada às ferramentas de autoria e de verificação.

6 AVALIAÇÃO DA METODOLOGIA E DO AMBIENTE

Os capítulos anteriores apresentaram e detalharam a proposta de uma metodologia e ambiente a serem adotados no desenvolvimento de aplicações hipermídia.

Neste capítulo, apresenta-se a avaliação da metodologia e do ambiente propostos. Para alcançar isso, em um primeiro momento são apresentadas e detalhadas as aplicações hipermídia que serão adotadas como estudo de caso na avaliação e no uso da metodologia e ambiente. Na sequência, utilizam-se a metodologia e o ambiente propostos para verificar alguns dos comportamentos dessas aplicações, descrevendo as etapas de uso e os resultados obtidos. Após apresentar o uso da metodologia e ambiente, são apresentadas considerações relativas ao tempo necessário para realizar o processo de verificação. O capítulo termina por uma discussão dos resultados obtidos e da avaliação da metodologia e do ambiente.

6.1 APLICAÇÕES HIPERMÍDIA

Visando validar a metodologia e ambiente propostos, selecionaram-se três aplicações hipermídia a serem usadas como estudos de caso. Todas as aplicações selecionadas possuem mídias interativas.

Os códigos NCL-Raw das aplicações apresentadas como estudos de caso encontram-se nos Anexos B,C e D deste documento.

6.1.1 O Primeiro João

A aplicação "O primeiro João" [1] é uma animação desenvolvida no Núcleo de Artes, Design e Animação do Departamento de Artes e Design da Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio).

De modo resumido, essa aplicação inicia com a exibição de um vídeo principal (*VideoPrincipal*), de um som (*Som*) e de uma imagem de fundo (*ImagemFundo*). Esse vídeo apresenta um garoto jogando bola em um campo de futebol amador, conforme ilustrado na Figura 43(a). Aos 15 segundos de exibição desse vídeo surge um ícone interativo (*Icone*), ilustrado na Figura 43(b), que será desativado aos 21 segundos. Caso esse ícone seja selecionado (botão RED) num período de até 6 segundos, ocorrerá a desativação do ícone, e a ativação de duas novas mídias: um segundo vídeo (*Chuteira*)

[1] http://clube.ncl.org.br/node/46

e um formulário (*FormularioIngles* ou *FormularioPortugues*) onde o usuário poderá efetuar a compra de uma chuteira, ilustrado na Figura 43(c). Esse segundo vídeo se auto desativará após 14 segundos de exibição, e desativará também o formulário. O término do vídeo principal desativará a exibição das demais mídias.

(a) Vídeo Principal (b) Icone Interativo RED (c) Após Seleção do Icone

Figura 43 – Aplicação “O Primeiro João”

A Figura 44 apresenta um diagrama representando a aplicação “O Primeiro João”. Na representação com esse diagrama, o retângulo com cantos arredondados e bordas espessas representa o Controle Remoto. O retângulo contendo bordas tracejadas representa um nó de contexto, usado no aninhamento de aplicações NCL. Os demais retângulos representam as mídias que fazem parte da aplicação. As ligações entre os retângulos são usadas para indicar as relações existentes entre as mídias. As ligações tracejadas representam eventos interativos, tendo como origem o Controle Remoto e como destino a mídia de abertura da aplicação, e as mídias interativas.

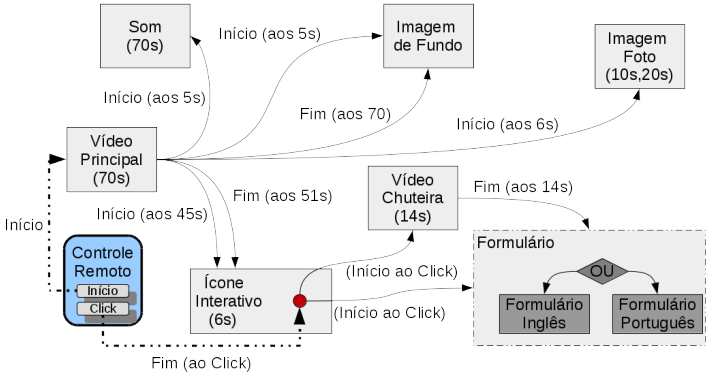


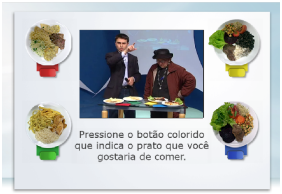
Figura 44 – Diagrama da Aplicação - “O Primeiro João”

6.1.2 Viva Mais

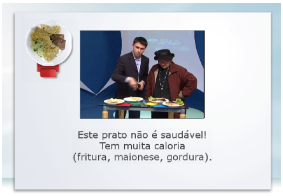
A aplicação "Viva Mais" [2] foi desenvolvida no centro de jornalismo da Universidade Federal de Santa Catarina (UFSC). Essa aplicação tem por objetivo alertar sobre a importância de uma alimentação saudável, visando uma boa qualidade de vida.

A aplicação "Viva Mais" inicia com a exibição de um vídeo principal (*VideoPrincipal*). Após 63 segundos, esse vídeo ativa a exibição de um ícone interativo (*Icone*), ilustrado na Figura 45(a). Caso esse ícone interativo seja selecionado (botão RED), ele se auto desativará, e ativará uma nova mídia (*FundoPratos*). A ativação da mídia *FundoPratos* causa a ativação de outras quatro mídias interativas (*Prato*1,*Prato*2,*Prato*3 e *Prato*4), ilustradas na Figura 45(b). Essas quatro mídias interativas representam quatro diferentes tipos de pratos identificados pelas cores: RED, GREEN, BLUE e YELLOW. A seleção do botão equivalente à cor do prato desativará todos os pratos, e ativará um texto detalhando a qualidade nutricional do tipo de prato selecionado (*ResultadoPrato*$_{(n)}$). A Figura 45(c) ilustra uma situação na qual foi selecionado o tipo de prato associado à cor RED.

(a) Icone Interativo RED



(b) Quatro Pratos Interativos



(c) Após Seleção do Prato RED

Figura 45 – Aplicação "Viva Mais"

A Figura 46 apresenta o diagrama de parte da aplicação "Viva Mais". Para facilitar a visualização da aplicação nessa figura, foram apresentados

[2] http://clube.ncl.org.br/node/29

quatro mídias correspondentes aos pratos, sendo que na figura, são mostradas as ligações de saída apenas para a mídia *Prato*1; as demais mídias *PratoN* possuem ligações equivalentes àquelas da mídia *Prato*1.

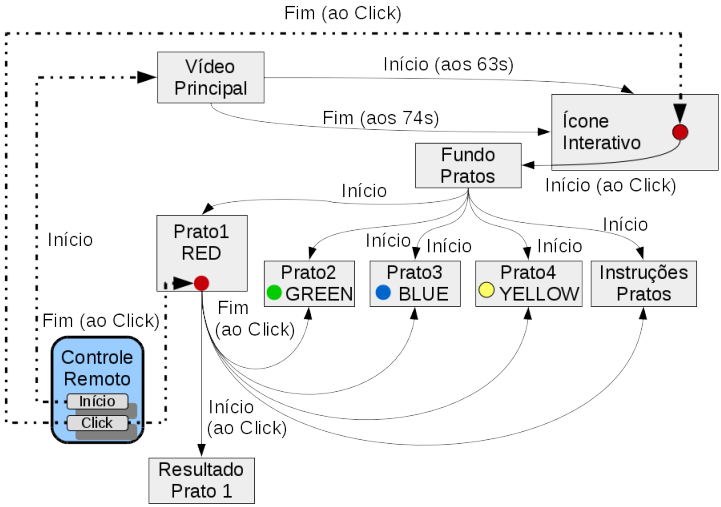


Figura 46 – Diagrama de Parte da Aplicação - “Viva Mais”

6.1.3 VestibaTV

A aplicação “VestibaTV” [3] tem como propósito ajudar estudantes a se prepararem para provas de vestibular. Essa aplicação foi desenvolvida no Laboratório de Aplicações de Vídeo Digital (LAVID) da Universidade Federal do Paraíba (UFPB).

A aplicação inicia com um vídeo de abertura (*VideoAbertura*) que, quando termina, ativa um vídeo principal (*Video*1) com duração de três minutos. Durante a apresentação desse segundo vídeo, aparecem 5 mídias interativas (*DicaCoruja*5*s*, *DicaCoruja*30*s*, *DicaCoruja*60*s*, *DicaCoruja*90*s* e *DicaCoruja*12*s*), cujos momentos de exibição são ilustrados na Figura 47.

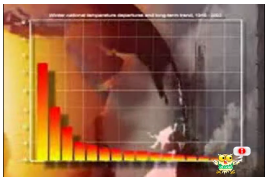
A ativação de qualquer uma das cinco mídias interativas apresentadas na Figura 47 deve ser feita pela seleção do botão RED.

[3] http://www.academia.edu/4511561/VestibaTV

(a) *DicaCoruja5s* aos 5s

(b) *DicaCoruja30s* aos 30s

(c) *DicaCoruja60s* aos 60s



(d) *DicaCoruja90s* aos 90s

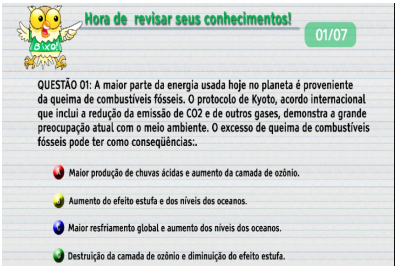
(e) *DicaCoruja120s* aos 120s

Figura 47 – Aplicação “VestibaTV”

A seleção do ícone interativo que é exibido aos 5 segundos (*DicaCoruja5s*) se desativa, e dispara a exibição da mídia *Text*1, ilustrada na Figura 48(a). Após ativada, a mídia *Text*1 só poderá ser desativada pelo botão RED, não existindo outra condição de desativação.

O segundo ícone interativo, que aparece aos 30 segundos (*DicaCoruja30s*), quando selecionado se desativa e ativa a exibição da mídia *Question*1. Essa mídia, ilustrada na Figura 48(b), consiste de uma questão de múltipla escolha.

(a) Texto ativado pela *DicaCoruja5s*



(b) Questionário ativado pela *DicaCoruja30s*

Figura 48 – Aplicação “VestibaTV”

A Figura 49 apresenta um diagrama representando parte da aplicação “VestibaTV”. (Não foi apresentada a aplicação completa, mas apenas as partes necessárias para os experimentos de verificação.) Essa aplicação possui uma grande quantidade de ligações e mídias, um diagrama representando a

aplicação completa tornaria sua apresentação e visualização muito complexas.

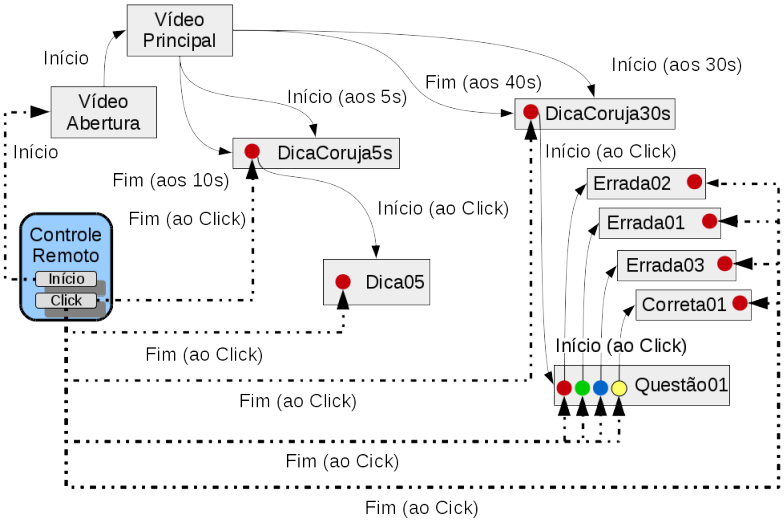


Figura 49 – Diagrama de Parte da Aplicação - “VestibaTV”

6.2 USO DA METODOLOGIA E AMBIENTE

A metodologia e ambiente propostos foram validados por meio da verificação de um conjunto de comportamentos nas aplicações previamente apresentadas.

6.2.1 Propriedades Intra-Mídias

Para ilustrar o uso da metodologia e ambiente propostos na verificação de propriedades Intra-Mídias, adotou-se, como estudo de caso, a aplicação “O primeiro João”, cujo código NCL é apresentado no Anexo (B).

Analisou-se os comportamentos internos da mídia interativa *Icone* que, segundo o projeto da aplicação “O Primeiro João”, deve aparecer aos 15 segundos de exibição do vídeo principal e ser desativada aos 21 segundos, ou após uma interação.

Para essa aplicação foram definidas as seguintes propriedades:

Propriedade 1 Quando a aplicação for ativada, a mídia *Icone* interativo será sempre apresentada;

Propriedade 2 Quando iniciada, a mídia *Icone* interativo terminará sempre sua apresentação; e

Propriedade 3 Quando apresentada, a mídia *Icone* nunca permanecerá por mais de 6 segundos.

Estas propriedades permitiram verificar se os comportamentos pretendidos no projeto se confirmam.

O primeiro passo do projetista na metodologia proposta consiste em selecionar o código NCL-Raw da aplicação "O Primeiro João". O código NCL é automaticamente transformado em um código FIACRE, representando o comportamento da aplicação. A seguir, as propriedades são especificadas usando o editor visual de propriedades EP, previamente apresentado no Capítulo 5. As propriedades introduzidas no editor seguem a sintaxe definida na linguagem LP (conforme apresentado no Capítulo 4), e depois automaticamente transformadas em fórmulas LTL e observadores, seguindo os casos.

As propriedades 1 e 2 introduzidas por meio do EP são transformadas automaticamente nas seguintes fórmulas LTL:

Propriedade 1 $\Box(Application\_started \Rightarrow \Diamond(Icone\_started))$

- Sempre que a aplicação tiver sua apresentação ativada, no futuro a mídia *icone* será apresentada.

Propriedade 2 $\Box(Icone\_started \Rightarrow \Diamond(Icone\_stopped))$

- Sempre que for iniciada, a mídia *icone* terminará sua apresentação.

Nessas fórmulas LTL, o elemento `Application_started` representa o estado onde a aplicação teve sua apresentação iniciada. Os elementos `Icone_started` e `Icone_stopped` representam, respectivamente, os estados onde a mídia *icone* teve sua apresentação iniciada e terminada.

Após efetuadas as transformações NCL-Fiacre e LP-LTL, ocorre o processo de verificação formal. A Tabela 6 apresenta os resultados da verificação das propriedades 1 e 2. Esses resultados confirmam que quando a aplicação for ativada, a mídia *Icone* sempre é apresentada, e sua apresentação sempre termina.

Tabela 6 – “O Primeiro João” - Propriedade 1 e 2

Propriedade	Resultado	Estados	Transições	Tempo de Resposta
1	verdade	469	902	0.6 segundos
2	verdade	469	902	0.6 segundos

Se o projetista erroneamente eliminar a ligação que ocorre aos 21 segundos desativando a mídia *Icone*, será inserido um comportamento indesejado, e neste caso o resultado da propriedade 2 irá mudar, conforme apresentado na Tabela 7.

Tabela 7 – “O Primeiro João - com erro do projetista” - Propriedade 2

Propriedade	Resultado	Estados	Transições	Tempo de Resposta
2	falso	296	581	0.6 segundos

A Figura 50 apresenta o contraexemplo (referente à propriedade 2) na forma de linha temporal. Nesse contraexemplo observa-se que a mídia *icone*, após ser ativada aos 15 segundos, permanece sendo apresentada até o término da aplicação. Por meio desse diagrama, conhecendo a estrutura da aplicação, na qual o *Icone* é desativado aos 21 segundos, é possível ao projetista identificar a falta de um mecanismo para desativar a mídia aos 21 segundos, e corrigir o erro incluindo tal mecanismo no código NCL.

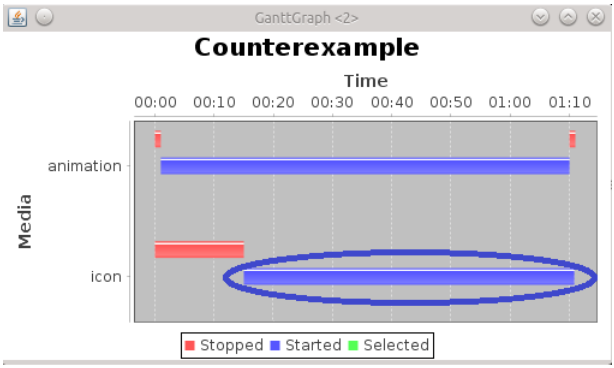


Figura 50 – Propriedade 2 - Linha temporal “O Primeiro João” com erro

Para verificar a propriedade 3, que envolve explicitamente o tempo, é gerado pelo ambiente um processo observador em FIACRE que conte a

passagem do tempo, e, na sequência, verificar a alcançabilidade desse observador (por exemplo a partir de uma fórmula LTL). Tanto o observador quanto a fórmula LTL usada para verificar sua alcançabilidade são construídos automaticamente pelo ambiente proposto a partir da declaração LP feita no editor EP. A Figura 51 mostra o observador (`ObsTempo`), que conta se o tempo de apresentação da mídia *icone* pode alcançar mais de 6 segundos. Esse processo observador do comportamento da mídia *icone* está ligado ao processo FIACRE que o representa através das portas *begin_icone* e *end_icone*.

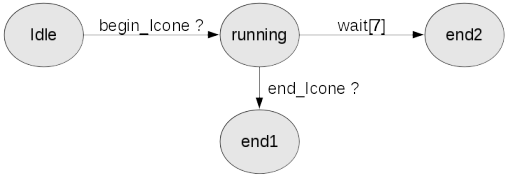


Figura 51 – Observador de Tempo Mínimo `ObsTempo`

Adota-se, então, a seguinte fórmula para verificar o comportamento do observador `ObsTempo`:

Propriedade 3 $\neg(\Box(ObsTempo\_end2))$

- O observador nunca alcançará o estado *end*2, correspondente à ultrapassagem do tempo máximo definido.

O observador `ObsTempo` tem apenas dois estados finais. Logo, a não alcançabilidade do estado *end*2 indica que *end*1 sempre é alcançado. Neste caso, a mensagem *end_icone* sempre ocorre antes de passarem 7 segundos.

A verificação realizada é apresentada na Tabela 8, indicando que o estado `end2` não é alcançado. Logo, a mídia *icone* pode permanecer por, no máximo, 6 segundos, confirmando o comportamento desejado na aplicação e descrito anteriormente.

Tabela 8 – "O Primeiro João" - Propriedade 3

Propriedade	Resultado	Estados	Transições	Tempo de Resposta
3	verdade	469	902	0.60 segundos

Caso o projetista cometa um erro de programação colocando a mensagem de término da mídia *Icone* para ocorrer aos 22 segundos, e não aos 21 segundos como descrito anteriormente, a resposta da propriedade 3 mudará, conforme apresentado na Tabela 9.

Tabela 9 – "O Primeiro João" - Propriedade 3 com erro

Propriedade	Resultado	Estados	Transições	Tempo de Resposta
3	falso	756	1476	0.65 segundos

A Figura 52 apresenta o contraexemplo (referente à propriedade 3) na forma de linha temporal, onde pode-se observar que o tempo de apresentação inicia aos 15 segundos e termina aos 22 segundos, correspondendo a um tempo superior a 6 segundos. Neste caso, esta constatação ajuda no processo de correção do erro no código NCL.

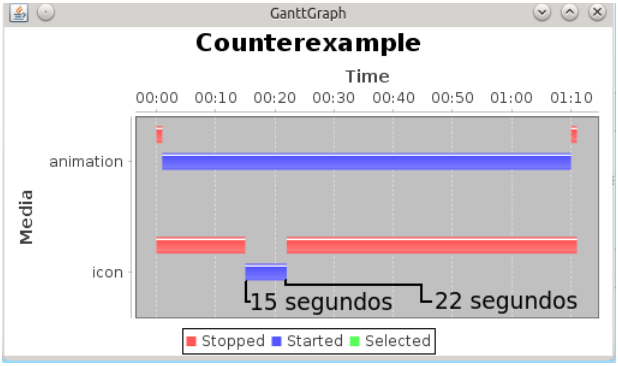


Figura 52 – Propriedade 3 - Linha temporal "O Primeiro João"

6.2.2 Propriedades Inter-Mídias

Para ilustrar a verificação deste tipo de propriedades, usa-se a aplicação "Viva Mais", cujo código NCL está no Anexo C.

Para ilustrar essa aplicação, verificou-se as propriedades "`Meets`" e "`Before`" definidas por Allen, conforme apresentado no Capítulo 3, e ilustradas na Figura 53.

Na aplicação "Viva Mais" essas propriedades têm por objetivo verificarem os seguintes comportamentos:

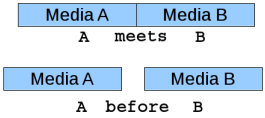


Figura 53 – Propriedade de Allen - “A meets B” e “A before B”

Propriedade 4 Na apresentação das mídias *Prato*1 e *ResultadoPrato*1, a mídia *ResultadoPrato*1 pode iniciar imediatamente após o término de *Prato*1.

Propriedade 5 Na apresentação das mídias *Prato*1 e *ResultadoPrato*1, a mídia *ResultadoPrato*1 pode iniciar um tempo após o término de *Prato*1.

Para verificar a propriedade 4, foi construído um processo observador em FIACRE que verifica se não existe um tempo entre o término da exibição da mídia *Prato*1 e o início da exibição da mídia *ResultadoPrato*1. A Figura 54 ilustra esse observador que foi chamado `ObsMeets`. Ele está ligado ao processo FIACRE que representa os comportamentos das mídias *Prato*1 e *ResultadoPrato*1 através das portas *begin_Prato*1, *begin_ResultadoPrato*1 e *end_Prato*1.

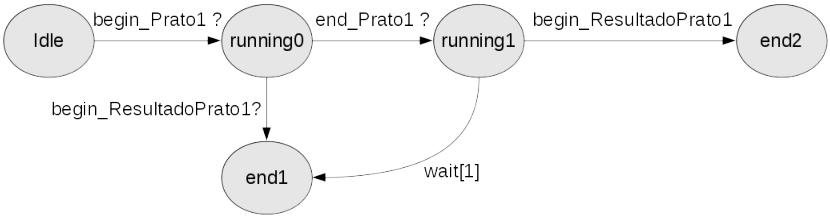


Figura 54 – Observador da propriedade `Meets` - `ObsMeets`

Adota-se a fórmula de não alcançabilidade no comportamento do observador `ObsMeets`:

Propriedade 4 $\neg(\Box(ObsMeets\_end2))$

- O observador não alcança o estado *end*2.

A Tabela 10 mostra que a propriedade 4 não foi satisfeita pelo modelo. Logo, o estado `end2` pode ser alcançado. Este resultado indica que a mensagem `begin_ResultadoPrato1` (responsável por ativar a mídia *ResultadoPrato*1) pode chegar imediatamente após a mensagem `end_Prato1` (responsável pelo término da apresentação da mídia *Prato*1).

Tabela 10 – “Viva Mais” - Propriedade 5

Propriedade	Resultado	Estados	Transições	Tempo de Resposta
4	falso	40753	160396	13.84 segundos

A Figura 55 apresenta o contraexemplo obtido na verificação, onde pode-se observar que a mídia *ResultadoPrato*1 inicia sua apresentação imediatamente após o término da apresentação da mídia *Prato*1.

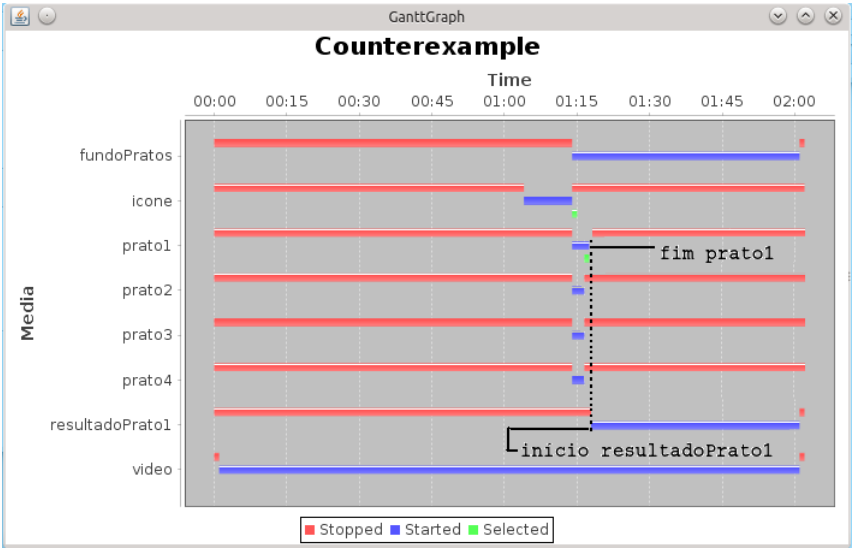


Figura 55 – Propriedades 4 - Linha temporal “Viva Mais”

Para verificar a propriedade 5, foi construído um processo observador em FIACRE que verifica se existe um tempo entre o término da exibição da mídia *Prato*1 e o início da exibição da mídia *ResultadoPrato*1. A Figura 56 ilustra esse observador, chamado `ObsBefore`.

Adota-se a fórmula de não alcançabilidade no comportamento de `ObsBefore`:

Propriedade 5 $\neg(\Box(ObsMeets\_end1))$

- O observador não alcança o estado *end*1.

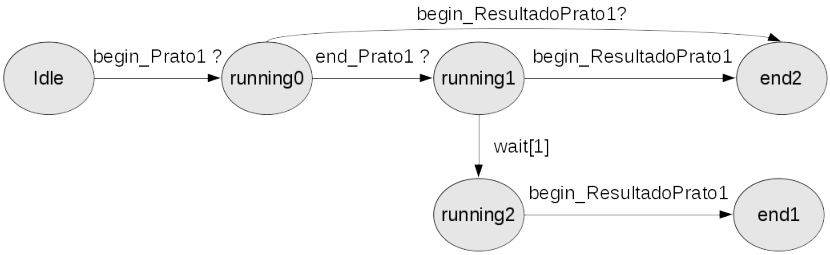


Figura 56 – Observador da propriedade `Before` - `ObsBefore`

A Tabela 11 mostra que a propriedade 5 foi satisfeita pelo modelo. Logo, o estado `end1` não pode ser alcançado. Esse resultado indica que entre as mensagens `end_Prato1` e `begin_ResultadoPrato1` não existe um intervalo de tempo.

Tabela 11 – "Viva Mais" - Propriedade 5

Propriedade	Resultado	Estados	Transições	Tempo de Resposta
5	verdade	40357	159641	13.71 segundos

As verificações comprovam, então, que o comportamento está de acordo com a especificação pretendida: *ResultadoPrato*1 aparece imediatamente após a exibição de *Prato*1.

6.2.3 Propriedades Causais

Para demonstrar a verificação das propriedades causais, adotou-se como estudo de caso a aplicação "Viva Mais", já citada. A seguinte propriedade demonstra a verificação da causalidade entre as mídias *Icone* e *FundoPratos*:

Propriedade 6 A seleção da mídia interativa *Icone* ativa de modo direto a exibição da mídia *FundoPratos*.

O observador apresentado na Figura 57 permite verificar essa propriedade. Esse observador foi chamado `ObsCausalidade` e está ligado a processos FIACRE que representam os comportamentos das mídias *Icone* e *FundoPratos* através das portas *Select_Icone* e *Begin_FundoPratos*.

Nos casos onde quer se verificar a causalidade não imediata, basta remover o estado *end*1 do observador da Figura 57.

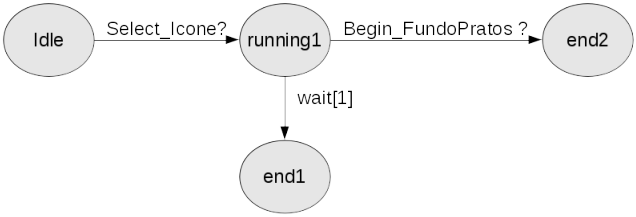


Figura 57 – Observador de Causalidade - `ObsCausalidade`

A verificação da causalidade pode ser feita também utilizando apenas fórmulas LTL, exceto quando uma mídia tem múltiplas causas que a levem para um determinado estado. Para verificar a causalidade neste caso, é necessário utilizar um observador que identifique a ativação do *link* causal existente na aplicação, e não apenas a troca de estados dos processos que representam as mídias.

Adota-se a seguinte fórmula de não alcançabilidade na verificação do comportamento do observador `ObsCausalidade`:

Propriedade 6 $\neg(\Box(ObsCausalidade\_end2))$

- O observador não alcança o estado *end*2.

A Tabela 12, indica que o estado `end2` será sempre alcançado, o que significa que a mídia $FundoPratos$ pode ter sua apresentação ativada pela seleção da mídia $Icone$, conforme especificado no projeto da aplicação.

Tabela 12 – "Viva Mais" - Propriedade 5

Propriedade	Resultado	Estados	Transições	Tempo de Resposta
6	falso	1085	2039	0.61 segundos

A Figura 58 apresenta o contraexemplo na forma de linha temporal, indicando a existência de um caminho onde a seleção da mídia $Icone$ inicia a apresentação da mídia $Fundo\_Pratos$.

6.2.4 Propriedades Inter-Mídias Espaciais

Para ilustrar a verificação das propriedades espaciais, adotou-se como estudo de caso a aplicação "VestibaTV", cujo código NCL se encontra no Anexo (D).

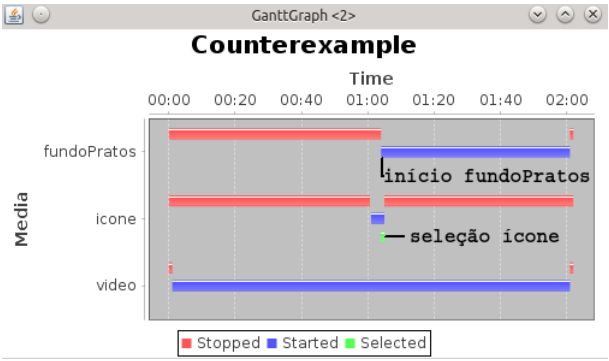


Figura 58 – Propriedade 6 - Linha temporal “Viva Mais”

As mídias existentes na aplicação “VestibaTV” ocupam diferentes regiões da tela, que podem se sobrepor espacialmente de modo parcial ou total. No momento da especificação das propriedades pelo projetista, o editor visual de propriedades EP indica possíveis sobreposições espaciais fornecidas pelo módulo de análise espacial, cabendo ao projetista decidir se deseja verificar se tais sobreposições vão acontecer também no tempo.

Considerando a possibilidade de existirem sobreposições espaciais e temporais conjuntamente, verificam-se propriedades espacial/temporal do tipo:

Propriedade 7 Nunca ocorrerá sobreposição espacial na exibição das mídias *Dica*05 e *Questao*1, por um tempo igual ou superior a 2 segundos.

A existência de sobreposição espacial nas regiões usadas pelas mídias *Dica*05 e *Questao*1 foi confirmada pelo Módulo de Análise Espacial, apresentado no Capítulo 5. A propriedade 7 é verificada a partir de um observador que analisa a existência de sobreposição temporal, e determine o tempo que tal sobreposição permanece ativa. Esse observador chamado `ObsSobreTemporal` é mostrado na Figura 59. O processo observador está ligado aos processos FIACRE que representam os comportamentos das mídias *Dica*05 e *Questao*1 através das portas *Begin_Dica*05, *End_Dica*05, *Begin_Questao*1, e *End_Questao*1.

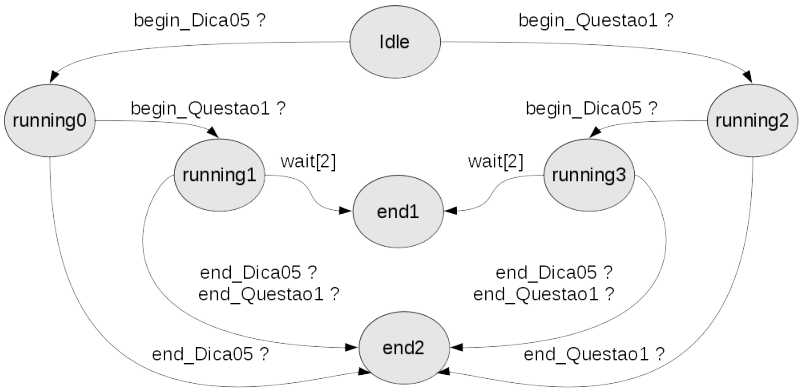


Figura 59 – Observador de sobreposição temporal `ObsSobreTemporal`

O observador `ObsSobreTemporal` possui caminhos que representam todas as possíveis combinações de chegadas de mensagens ativando e desativando a exibição das mídias *Dica*05 e *Questao*1. O estado *end*1 indica a ocorrência de uma sobreposição com tempo igual ou superior a 2 segundos. Adota-se a fórmula de não alcançabilidade para verificação do comportamento desse observador:

Propriedade 7 $\neg(\Box(ObsSobreTemporal\_end1))$

- O observador não alcança o estado *end*1

O resultado **falso**, conforme apresentado na Tabela 13, indica que a sobreposição espacial e temporal na aplicação "VestibaTV" pode ocorrer por 2 segundos ou mais.

Tabela 13 – "Viva Mais" - Propriedade 5

Propriedade	Resultado	Estados	Transições	Tempo de Resposta
7	falso	10550	29579	1.98 segundos

O comportamento em que ocorre a sobreposição temporal e espacial por mais de 2 segundos é indesejado. A Figura 60 ilustra a linha temporal representando a sequência de ações que resultaram nesse comportamento. Usando a linha temporal pode-se observar que as mídias *Dica*05 e *Questao*1 estão ambas em estado de execução (representado pela linha azul) no período entre 30 e 32 segundos.

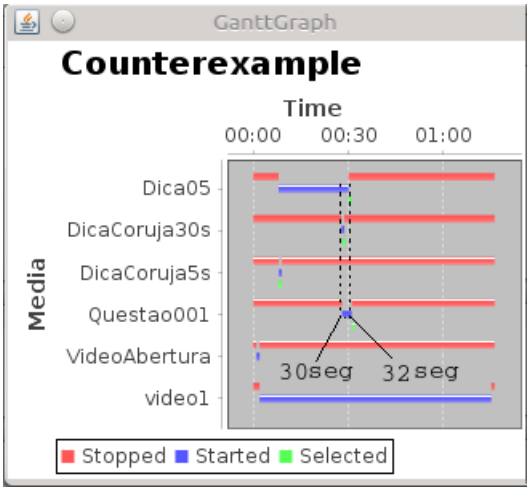


Figura 60 – Propriedade 7 - Linha temporal “VestibaTV”

A propriedade 7 identifica um comportamento indesejado na aplicação “VestibaTV”. Como possível correção o projetista poderá incluir um parâmetro `explicitDur` no código NCL da mídia *Dica*05, para desativar a exibição da mídia após a passagem de um determinado período de tempo. O Código 39 apresenta a declaração NCL da mídia *Dica*05, na aplicação “VestibaTV” onde foi incluído o parâmetro `explicitDur` (linha 2).

Código 39 “VestibaTV” Após Correção

```
1 <media  id="Dica05" src="media/Dica05.png">
2 <property name="explicitDur" value="15s"/>
3 <property name="left" value="8%"/>
4 <property name="top" value="72%"/>
5 <property name="width" value="85%"/>
6 <property name="height" value="23%"/>
7 </media>%</PropertyList>
```

A mídia *Dica*05 é ativada pela seleção da mídia interativa *DicaCoruja*5*s*. Seguindo a especificação da aplicação, essa interação pode ocorrer, no máximo aos 10 segundos. Colocando-se na mídia *Dica*05 o parâmetro `explicitDur` com 15 segundos, no pior caso ela será desativada aos 25 segundos (10+15), sendo que a mídia *Questao*1 é ativada pela seleção da mídia interativa *DicaCoruja*30*s*, que pode ocorrer no tempo mínimo de 30 segundos, momento no qual a mídia *DicaCoruja*30*s* é ativada. Desta forma, é garantido

que não ocorra sobreposição na exibição das mídias. Se o projetista optar, como correção para o erro, a inclusão de `explicitDur` com 15 segundos na mídia *Dica*05, ele garantirá um intervalo mínimo de 5 segundos entre a exibição da mídia *Dica*05 e da mídia *Questao*1. Nesse caso, o resultado da propriedade 7 é apresentado na Tabela 14, sendo que a situação descrita por *end*1 não será alcançada.

Tabela 14 – "VestibaTV Modelo - modificado" - Propriedade 7

Propriedade	Resultado	Estados	Transições	Tempo de Resposta
7	verdadeiro	6001	15118	1.70 segundos

Esta seção teve por objetivo exemplificar o uso da metodologia e ambiente desenvolvidos, onde foram realizados experimentos com algumas aplicações e propriedades. Outras propriedades foram verificadas neste mesmo exemplo com sucesso. Essas aplicações foram também executadas no *player* Ginga, comprovando os resultados obtidos no uso da metodologia e ambiente propostos.

6.3 AVALIAÇÃO DO OBSERVADOR DE TEMPO GLOBAL

O observador de tempo global permite identificar o momento em que cada ação ocorre, facilitando a identificação das causas de erros e a correção da aplicação. Entretanto, a inclusão deste observador eleva o consumo de tempo no processo de verificação, conforme ilustrado na Tabela 15. Os valores apresentados nesta tabela foram obtidos na verificação da seguinte propriedade na aplicação "Viva Mais":

Propriedade Nunca "Ícone" é exibido, e imediatamente após "Resultado-Prato1" é exibido.

Tabela 15 – Uso do Observador de Tempo Global

Resultado	Estados	Transições	Tempo de Resposta
Com Observador Global de Tempo			
verdade	29099	117582	7,66 segundos
Sem Observador Global de Tempo			
verdade	14354	51426	4.05 segundos

6.4 AVALIAÇÃO DA REDUÇÃO

Os experimentos não visaram apenas avaliar a metodologia e ambiente de verificação desenvolvidos, mas também avaliar o ganho na diminuição do tempo de execução quando adotada a abordagem de redução de modelos. A abordagem baseada em redução diminui o tamanho do modelo formal, usando como guia do processo de redução, informações referentes a propriedade a ser verificada. Dependendo das restrições definidas pelo projetista para aplicações editadas "ao vivo", essa abordagem pode se tornar de suma importância.

No que refere ao tempo necessário para a verificação e o tamanho do modelo, foram comparados os resultados obtidos com os modelos completos e reduzidos; a Tabela 16 apresenta os resultados dessas comparações.

Tabela 16 – Modelo Reduzido X Modelo Completo

Modelo	**Estados**	**Transições**	**Tempo de Resposta**
Primeiro João - Propriedade Intra-Mídia: A mídia *Icone* sempre será apresentada			
Modelo Completo	2278	4880	1.09 segundos
Modelo Reduzido	469	902	0.6 segundos
Viva Mais - Propriedade Intra-Mídia: A mídia *Prato*1 sempre permanece por um tempo mínimo de 2s			
Modelo Completo	59636	240956	16.0 segundos
Modelo Reduzido	22830	79928	3.85 segundos
VestibaTV - Propriedade Espacial: Nunca ocorre sobreposição espacial por um tempo de até 2s entre as mídias *Dica*05 e *Questao*1			
Modelo Completo	1433563	5183755	240.45 segundos
Modelo Reduzido	10550	29579	1.98 segundos

O ganho obtido pela redução em "O Primeiro João" foi baixo: com o modelo reduzido o tempo de resposta ficou 1.8 vezes menor, comparado com o modelo completo. Esse resultado deve-se à simplicidade do modelo e à pouca interatividade na aplicação.

A aplicação "Viva Mais", em comparação com "O Primeiro João", tem mais mídias interativas, resultando em um aumento do não-determinismo. Para essa aplicação, o ganho na redução foi mais significativo: o resultado com o modelo reduzido foi obtido com tempo 4.1 vezes menor, comparando com o tempo necessário com o modelo completo.

Finalmente, a aplicação "VestibaTV" obteve um ganho mais elevado

usando a abordagem de redução, diminuindo o tempo de resposta em aproximadamente 121 vezes. Esse ganho é atribuível ao maior tamanho e à maior interatividade dessa aplicação.

Outros experimentos com outras propriedades apresentaram resultados semelhantes. As medidas de tempo obtidas mostram que a abordagem proposta, usando redução de modelos, diminui consideravelmente o tempo necessário para a verificação.

Todos os experimentos foram realizados adotando-se o Linux como sistema operacional, distribuição Kubuntu 14.04 LTS, executando em um Sony Vaio com processador AMD Phenom II, possuindo 4GBytes de memória RAM.

6.5 CONSIDERAÇÕES

O ambiente desenvolvido permitiu o uso da metodologia proposta, escondendo do projetista as dificuldades referentes à verificação formal de modelos, tais como construção de propriedades LTL e de observadores, e a construção do modelo formal para descrição do sistema a verificar.

Desse modo, o uso da metodologia tornou-se simples, sem exigir do projetista quaisquer conhecimentos sobre verificação formal, bastando para ele escrever a aplicação na linguagem que está acostumado (NCL por exemplo) e especificar as propriedades a serem verificadas, usando uma interface gráfica baseada nas seleções de mídias e comportamentos.

Os resultados obtidos foram apresentados ao projetista indicando as propriedades que foram satisfeitas ou não pelo modelo. No caso da não satisfação, um contraexemplo é apresentado na forma de linha temporal, com a indicação do tempo onde cada troca de estado de uma mídia ocorre, semelhante à linha temporal comumente usada por ferramentas de autoria.

Os experimentos mostraram a efetividade na identificação de comportamentos existentes na aplicação, e a detecção, e posterior correção, de eventuais erros.

Adotando-se o processo de redução, o tempo necessário para a verificação formal teve uma diminuição considerável, obtendo em um dos casos a diminuição em 121 vezes.

7 CONCLUSÕES E TRABALHOS FUTUROS

Conforme referido na introdução deste trabalho, a disponibilidade de diferentes tipos de informações aumenta dia a dia. O uso de aplicações hipermídia é uma alternativa atraente como um meio de fornecer essas informações a usuários com diferentes perfis de conhecimento. Em um mundo conectado, as pessoas se tornam cada vez mais exigentes no que se refere à atualidade das informações e à possibilidade de interagirem com essas informações.

A interatividade permite ao usuário da aplicação interagir não somente recebendo informações, mas também enviando informações referentes a seus desejos, que podem ser, por exemplo, o recebimento de mais informações, o desejo de agregar suas opiniões na aplicação para que outros possam vê-las, ou até mesmo o desejo de contratar um serviço.

A exigência de informações sempre atuais, supridas pela edição "ao vivo", faz com que o tempo de desenvolvimento de uma aplicação hipermídia torne-se um fator crítico. As aplicações com níveis mais elevados de interatividade, normalmente possuem um código mais complexo. Logo, nesse tipo de aplicação é mais provável a existência de erros de especificação.

Com o objetivo de auxiliar os projetistas, neste trabalho foi proposta uma metodologia para o desenvolvimento de aplicações hipermédia. Essa metodologia visa identificar possíveis comportamentos indesejáveis na fase de desenvolvimento. Para apoiar essa metodologia, foi construído um ambiente onde o projetista, ainda usando suas linguagens e ferramentas de autoria preferidas, pode efetuar a verificação dos comportamentos da aplicação. O ambiente desenvolvido usa uma abordagem baseada em redução, que diminui o tempo necessário no processo de verificação. Dependendo das restrições definidas pelo projetista na edição "ao vivo", essa abordagem pode se tornar de suma importância.

Este capítulo tem por objetivo apresentar as conclusões obtidas durante o trabalho, bem como apresentar trabalhos futuros que possam complementar o realizado até o presente momento.

7.1 CONCLUSÕES

A metodologia e o ambiente desenvolvidos integram, de modo transparente, ferramentas de autoria com ferramentas para verificação formal. Para essa integração, adota-se uma abordagem MDE baseada em princípios de transformação de modelos, com objetivo de garantir a coerência entre modelos.

A realização de experimentos constatou que a metodologia e o ambiente apresentados neste trabalho cumprem os principais requisitos referentes à edição e à verificação de aplicações hipermídia : (1) integração com linguagens e ferramentas de autoria existentes (o projetista ainda usa a linguagem hipermídia e ferramentas de autoria comumente adotadas); (2) a facilidade de especificação de comportamentos a serem verificados (interface de alto nível gráfico para a especificação de comportamentos); (3) verificação de diferentes tipos de comportamentos (temporal, espacial e causal); (4) fácil integração do ambiente com ferramentas formais consolidadas de verificação; e (5) auxílio para o projetista na correção de erros (o ambiente apresenta a sequência de ações que levaram ao erro comportamental na forma de linha temporal).

O presente trabalho mostrou que é possível o uso de ferramentas de verificação formal na avaliação dos comportamentos de aplicações hipermídia, sem exigir do projetista quaisquer conhecimentos prévios de tais ferramentas. O uso foi possível devido ao ambiente seguir a abordagem MDE, que permitiu traduzir de maneira fiel e automática de aplicações escritas com linguagens hipermídia em modelos formais com representação de estados, bem como traduzir propriedades de uma linguagem de alto nível para fórmulas LTL e observadores. Além dos elementos já citados, as transformações MDE agregaram ao modelo formal, características relativas à plataforma de execução, pois a verificação formal considera o documento hipermídia sendo apresentado em uma plataforma de exibição.

Os resultados obtidos durante o desenvolvimento da metodologia e do ambiente foram apresentados como artigo completo nos seguintes eventos:

- WebMedia 2011 - Uma abordagem MDE para Modelagem e Verificação de Documentos Multimídia Interativos. In: XVII Simpósio Brasileiro de Sistemas Multimídia e Web - WEBMIDIA, 2011, Florianópolis. Anais do XVII Simpósio Brasileiro de Sistemas Multimídia e Web. Porto Alegre: Sociedade Brasileira de Computação, 2011. v. 1. p. 91-98.
- WebMedia 2012 - An approach to verify live NCL applications. In: XVIII Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia 2012, 2012, São Paulo. Proceedings of the 18th Brazilian Symposium on Multimedia and the Web. New York, NY, USA: ACM, 2012. p. 223-232.
- SAM 2014 - Verifying Hypermedia Applications by Using a MDE Approach. In: System Analysis and Modelling Conference - SAM 2014, 2014, Valencia - Espanha. Proceedings of the 8th System Analysis and Modelling Conference - SAM 2014. Valencia - Espanha, 2014. p. 1.

Um artigo completo contendo a descrição de todo o trabalho está

sendo escrito para submissão para a revista “Multimedia Tools and Applications - Springer”.

7.2 TRABALHOS FUTUROS

Como trabalhos futuros que venham a complementar e melhorar o trabalho aqui apresentado destacam-se:

- melhorar a interface de especificação de propriedade e o conjunto de propriedades oferecidas ao projetista. Uma nova interface apresentará as mídias e as relações de maneira gráfica, semelhante ao que ocorre na visão estrutural apresentada pela ferramenta de autoria Composer. Nessa interface o projetista inicialmente fará a seleção das mídias, e na sequência definirá as propriedades que deseja verificar;
- permitir a verificação considerando o uso de múltiplos dispositivos. Usando múltiplos dispositivos, o projetista tem maior liberdade em apresentar relações interativas que permitam complementar o conteúdo que está sendo apresentado, sem precisar se preocupar com sobreposições espaciais. O uso de múltiplos dispositivos é explorado no trabalho de Costa et al. (COSTA; MORENO; SOARES, 2009);
- construção de um módulo usando a linguagem LUA, na forma de um escalonador de eventos, que permita a simulação da execução dos contraexemplos;
- integração de novas linguagens na metodologia e ambiente propostos. Pretende-se complementar a metodologia e o ambiente para permitirem a verificação de comportamentos em aplicações MPEG-V (YOON et al., 2015);
- converter os *scripts* que interligam o ambiente desenvolvido com as ferramentas de verificação, da linguagem Shell-Script para a linguagem Python, o que tornaria o ambiente portável para outras plataformas;
- implementar uma versão distribuída, onde cada modelo possa ser verificado em um núcleo diferente de uma mesma máquina, ou em máquinas diferentes em um cluster de computadores. Em redes locais, as atuais tecnologias de interconexão possuem latência bem abaixo de 1 segundo. O uso de paralelismo associado a verificação formal já vem sendo adotado, como, por exemplo, no trabalho de Saad et al. (SAAD; DAL-ZILIO; BERTHOMIEU, 2013);
- associar o Observador de Tempo Global e o Controle Remoto em um mesmo processo, e permitir discretizar o tempo apenas nos períodos onde possam ocorrer interações; e

- permitir que o usuário possa definir o período de tempo mínimo entre cada interação no Controle Remoto.

REFERÊNCIAS

ALLEN, J. F. Maintaining knowledge about temporal intervals. **Commun. ACM**, ACM, New York, NY, USA, v. 26, p. 832–843, November 1983. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/182.358434>.

ARDILA, L.; PéREZ-LLOPIS, I.; ESTEVE, M.; PALAU, C. E. Interoperable architecture for joint real/virtual training in emergency management using the mpeg-v standard. **Computer Standards & Interfaces**, v. 41, p. 39 – 55, 2015. ISSN 0920-5489. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0920548915000239>.

BELOUAER, L.; MARIS, F. SMT Spatio-Temporal Planning (ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS), Atibaia, São Paulo, Brazil, 25/06/2012). 2012.

BERARD, B.; BIDOIT, M.; FINKEL, A.; LAROUSSINIE, F.; PETIT, A.; PETRUCCI, L.; SCHNOEBELEN, P. **Systems and Software Verification: Model-Checking Techniques and Tools**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 3642074782, 9783642074783.

BERTHOMIEU, B.; BODEVEIX, J.-P.; FARAIL, P.; FILALI, M.; GARAVEL, H.; GAUFILLET, P.; LANG, F.; VERNADAT, F. Fiacre: an Intermediate Language for Model Verification in the Topcased Environment. In: **ERTS 2008**. Toulouse France: [s.n.], 2008.

BERTHOMIEU, B.; BODEVEIX, J.-P.; FARAIL, P.; FILALI, M.; GARAVEL, H.; GAUFILLET, P.; LANG, F.; VERNADAT, F. **The Syntax and Semantics of FIACRE**. [S.l.], 2011. Disponível em: <http://homepages.laas.fr/bernard/fiacre/ acessado em fevereiro de 2011>.

BERTHOMIEU, B.; RIBET, P.-O.; VERNADAT, F. The tool TINA - Construction of Abstract State Spaces for Petri Nets e Time Petri Nets. **International Journal of Production Research**, v. 14, n. 42, 2004.

BERTINO, E.; FERRARI, E.; PEREGO, A.; SANTI, D. A constraint-based approach for the authoring of multi-topic multimedia presentations. In: **IEEE International Conference on Multimedia and Expo, 2005. ICME 2005**. [S.l.]: IEEE CS Press, 2005. p. 578–581. ISBN 0-7803-9331-7.

BOSSI, A.; GAGGI, O. Analysis and verification of SMIL documents. **Multimedia Systems**, Springer Berlin / Heidelberg, p. 1–20, abr. 2011. ISSN 0942-4962. Disponível em: <http://dx.doi.org/10.1007/s00530-011-0233-1>.

BOUYAKOUB, S.; BELKHIR, A. Smil builder: An incremental authoring tool for smil documents. **ACM Trans. Multimedia Comput. Commun. Appl.**, ACM, New York, NY, USA, v. 7, p. 2:1–2:30, February 2011. ISSN 1551-6857. Disponível em: <http://doi.acm.org/10.1145/1870121.1870123>.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. **Model-Driven Software Engineering in Practice**. 1st. ed. [S.l.]: Morgan & Claypool Publishers, 2012. ISBN 1608458822, 9781608458820.

BULTERMAN, D. **Synchronized Multimedia Integration Language (SMIL 3.0)**. [S.l.], dec 2008. Http://www.w3.org/TR/2008/REC-SMIL3-20081201/.

BULTERMAN, D.; HARDMAN, L.; MULLENDER, S.; RUTLEDGE, L.; PRIME, M.; WILSON, M. Grins: A graphical interface for creating and playing smil documents. In: **in Proceedings of the 7th International World Wide Web Conference**. [S.l.: s.n.], 1998. p. 519–529.

BURTON, J. K.; MOORE, D. M.; HOLMES, G. A. Hypermedia concepts and research: an overview. **Computers in Human Behavior**, v. 11, n. 3-4, p. 345–369, 1995. Disponível em: <http://www.sciencedirect.com/science/article/pii/074756329580004R>.

CLARKE, E.; BIERE, A.; RAIMI, R.; ZHU, Y. Bounded model checking using satisfiability solving. **Form. Methods Syst. Des.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 19, n. 1, p. 7–34, jul. 2001. ISSN 0925-9856. Disponível em: <http://dx.doi.org/10.1023/A:1011276507260>.

CLAVEL, M.; DURáN, F.; EKER, S.; LINCOLN, P.; MARTÍ-OLIET, N.; MESEGUER, J.; TALCOTT, C. **All About Maude - a High-performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic**. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN 3-540-71940-7, 978-3-540-71940-3.

COMBEMALE, B. **Approche de métamodélisation pour la simulation et la vérification de modèle – Application à l'ingénierie des procédés**. Tese (Doutorado) — Institut National Polytechnique, Université de Toulouse, jul.

2008. In french. Disponível em:
<http://ethesis.inp-toulouse.fr/archive/00000666/>.

COSTA, R. M. d. R.; MORENO, M. F.; SOARES, L. F. G. Ginga-ncl: supporting multiple devices. In: **Proc. of the XV Brazilian Symp. on MM and the Web**. New York, NY, USA: ACM, 2009. (WebMedia '09), p. 6:1–6:8. ISBN 978-1-60558-880-3. Disponível em:
<http://doi.acm.org/10.1145/1858477.1858483>.

COSTA, R. M. R. **Controle do Sincronismo Temporal de Aplicações Hipermídia**. Tese (Doutorado) — PUC Rio, 2010.

DRECHSLER, R. **Advanced Formal Verification**. Norwell, MA, USA: Kluwer Academic Publishers, 2004. ISBN 1402077211.

ELIAS, S.; EASWARAKUMAR, K. S.; CHBEIR, R. Dynamic consistency checking for temporal and spatial relations in multimedia presentations. In: **Proceedings of the 2006 ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2006. (SAC '06), p. 1380–1384. ISBN 1-59593-108-2. Disponível em:
<http://doi.acm.org/10.1145/1141277.1141596>.

FARAIL, P.; GAUFILLET, P.; CANALS, A.; CAMUS, C. L.; SCIAMMA, D.; MICHEL, P.; CREGUT, X.; PANTEL, M.; VERNADAT, F. The TOPCASED project: Toolkit in OPen-source for Critical Aplications e SystEms development. **3th European Congress on Embedded Real Time Software - ERTS**, 2006.

FAWCETT, J.; AYERS, D.; QUIN, L. R. E. **Beginning XML**. 5th. ed. Birmingham, UK, UK: Wrox Press Ltd., 2012. ISBN 1118162137, 9781118162132.

FELIX, M.; HAEUSLER, E.; SOARES, L. **Validating hypermedia documents: a timed automata approach**. PUC, 2002. Disponível em:
<http://books.google.com/books?id=6Cj1GwAACAAJ>.

GUIMARAES, R. L.; COSTA, R. M. D. R.; SOARES, L. F. G. Composer: Authoring tool for itv programs. In: **Proceedings of the 6th European conference on Changing Television Environments**. Berlin, Heidelberg: Springer-Verlag, 2008. (EUROITV '08), p. 61–71. ISBN 978-3-540-69477-9.

HENZINGER, T.; MANNA, Z.; PNUELI, A. Timed transition systems. In: **Proc. of the REX Workshop 1991 (REX'91)**. New York, NY, USA: Springer, 1992. (LNCS), p. 226–251.

ITU-T Recommendation H.761. Nested Context Language (NCL) and Ginga-NCL for IPTV Services. In: . Genova: [s.n.], 2009.

JANSEN, J.; CESAR, P.; BULTERMAN, D. C. A. A model for editing operations on active temporal multimedia documents. In: **ACM Symposium on Document Engineering'10**. [S.l.: s.n.], 2010. p. 87–96.

JOUAULT, F.; KURTEV, I. Transforming models with atl. In: **Proceedings of the 2005 International Conference on Satellite Events at the MoDELS**. Berlin, Heidelberg: Springer-Verlag, 2006. (MoDELS'05), p. 128–138. ISBN 3-540-31780-5, 978-3-540-31780-7.

NA, J.-C.; FURUTA, R. Dynamic documents: Authoring, browsing, and analysis using a high-level petri net-based hypermedia system. In: **Proceedings of the 2001 ACM Symposium on Document Engineering**. New York, NY, USA: ACM, 2001. (DocEng '01), p. 38–47. ISBN 1-58113-432-0. Disponível em: <http://doi.acm.org/10.1145/502187.502194>.

NAIN, S.; VARDI, M. Y. Automated technology for verification and analysis: 5th international symposium, atva 2007 tokyo, japan, october 22–25, 2007 proceedings. In: ____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. cap. Branching vs. Linear Time: Semantical Perspective, p. 19–34. ISBN 978-3-540-75596-8. Disponível em: <http://dx.doi.org/10.1007/978-3-540-75596-8>.

NBR15606-1, A. **Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital**. [S.l.], 2008.

Obeo. **Acceleo user guide**. [S.l.], 2008.

OLIVEIRA, M. C. F. de; TURINE, M. A. S.; MASIEIRO, P. C. A statechart-based model for modeling hypermedia applications. **ACM Transactions on Information Systems**, v. 1, 2001.

PILGRIM, M. **HTML5: Up and Running**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2010. ISBN 0596806027, 9780596806026.

ROSSUM, G. van; JANSEN, J.; MULLENDER, K. S.; BULTERMAN, D. C. A. Cmifed: A presentation environment for portable hypermedia documents. In: **Proceedings of the First ACM International Conference on Multimedia '93, Anaheim, CA, USA, August 1-6, 1993.** [s.n.], 1993. p. 183–188. Disponível em: <http://doi.acm.org/10.1145/166266.166287>.

SAAD, R. T.; DAL-ZILIO, S.; BERTHOMIEU, B. An experiment on parallel model checking of a CTL fragment. **CoRR**, abs/1301.7533, 2013. Disponível em: <http://arxiv.org/abs/1301.7533>.

SAMPAIO, P. N. M.; COURTIAT, J.-P. An approach for the automatic generation of rt-lotos specifications from smil 2.0 documents. **J. Braz. Comp. Soc.**, v. 9, n. 3, p. 39–51, 2004.

SANTA'ANNA, F.; NETO, C. S. S.; AZEVEDO, R. G. A.; BARBOSA, S. D. J. Desenvolvimento de aplicações declarativas para tv digital no middleware ginga com objetos imperativos lua. In: . [s.n.], 2009. Disponível em: <http://www.telemidia.puc-rio.br/sites/telemidia.puc-rio.br/files/MCNCLua.pdf>.

SANTOS, C. A. S.; SOARES, L. F. G.; SOUZA, G. L.; COURTIAT, J.-P. Design methodology and formal validation of hypermedia documents. In: **Proc. of the 6th ACM Intl. Conf. on MM**. New York, NY, USA: ACM, 1998. (MM '98), p. 39–48. ISBN 0-201-30990-4. Disponível em: <http://doi.acm.org/10.1145/290747.290753>.

SANTOS, J. A. dos; BRAGA, C.; MUCHALUAT-SAADE, D. C.; ROISIN, C.; LAYAÏDA, N. Spatio-temporal validation of multimedia documents. In: **Proceedings of the 2015 ACM Symposium on Document Engineering**. New York, NY, USA: ACM, 2015. (DocEng '15), p. 133–142. ISBN 978-1-4503-3307-8. Disponível em: <http://doi.acm.org/10.1145/2682571.2797060>.

SCHMIDT, D. C. Model-driven engineering. **IEEE Computer**, v. 39, n. 2, February 2006. Disponível em: <http://www.truststc.org/pubs/30.html>.

SOARES, L. F. G.; BARBOSA, S. D. J. **Programando em NCL 3.0: desenvolvimento de aplicações para o middleware Ginga, TV Digital e Web**. Rio de Janeiro, Brazil: Elsevier, 2009. ISBN 978-85-352-3457-2.

STAHL, T.; VOELTER, M.; CZARNECKI, K. **Model-Driven Software Development: Technology, Engineering, Management**. [S.l.]: John Wiley & Sons, 2006. ISBN 0470025700.

WAISMAN, T. **Usabilidade em Serviços Educacionais em Ambiente de TV Digital**. Tese (Doutorado) — Escola de Comunicação e Artes - USP, São Paulo, Brasil, 2006.

YANG, C.-C. Detection of the time conflicts for smil-based multimedia presentations. **Workshop on Computer Networks, Internet, e Multimedia**, p. 57– 63, 2000.

YOON, K.; KIM, S.; HAN, J.; HAN, S.; PREDA, M. **Mpeg-V: Bridigng the Virtual and Real World**. Elsevier Limited, Oxford, 2015. (Academic Press). ISBN 9780124201408. Disponível em: <https://books.google.com.br/books?id=UZCzngEACAAJ>.

YOVINE, S.; OLIVERO, A.; MONTEVERDE, D.; CORDOBA, L.; REITER, G. An approach for the verification of the temporal consistency of ncl applications. In: **Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia)**. Belo Horizonte, Brasil: [s.n.], 2010.

YU, H.; HE, X.; GAO, S.; DENG, Y. Modeling and analyzing smil documents in sam. In: **ISMSE**. [S.l.: s.n.], 2002. p. 132–139.

YU, J.; XIANG, Y. Hypermedia presentation and authoring system. **Computer Networks and {ISDN} Systems**, v. 29, n. 8-13, p. 875 – 886, 1997. ISSN 0169-7552. Papers from the Sixth International World Wide Web Conference. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0169755297000615>.

YUAN, Z.; BI, T.; MUNTEAN, G.; GHINEA, G. Perceived synchronization of mulsemedia services. **IEEE Transactions on Multimedia**, v. 17, n. 7, p. 957–966, 2015. Disponível em: <http://dx.doi.org/10.1109/TMM.2015.2431915>.

APÊNDICE A – Extensão da Metodologia para a Linguagem SMIL

A.1 LINGUAGEM SMIL

A metodologia proposta também se aplica à verificação de um subconjunto de aplicações hipermídia codificadas na linguagem SMIL, restringindo-se apenas a verificar aplicações codificadas utilizando estruturas de causalidade. Nesta seção apresenta-se o mecanismo de tradução de SMIL para o formato GI.

A.1.1 Transformação De SMIL Para Grafo Intermediário

Na transformação SMIL para GI, os modelos devem estar em conformidade com seus respectivos metamodelos, seguindo o conjunto de regras descrito abaixo e ilustrado na Tabela A.1.1:

Regra 1 - cada `Media` SMIL - smilText, video, áudio e img - é traduzido para um `Vertex` em GI, que é um elemento interno do `VertexList`;

Regra 2 - os elementos `Region` em SMIL são traduzidos para atributos no `Vertex`, considerando a relação entre o elemento `Media` que originou o `Vertex` e o elemento `Region`;

Regra 3 - o atributo `dur` SMIL é traduzido para um atributo `ExpliticDur` no `Vertex` de GI;

Regra 4 - as relações *endCondition* e *beginCondition* identificadas pelos atributos `end` e `begin` em SMIL são traduzidos para `Edges` no GI. Na criação dos `Edges`, a regra analisa a origem e o destino da ligação; e

Regra 5 - os elementos `par`, `seq` e `excl` em SMIL são traduzidos para `Edges` no GI. Na criação dos `Edges`, a regra analisa o tipo de estrutura e suas condições de ativação e desativação.

A.1.2 Aplicação das Regras

O Código 40 apresenta parte de uma aplicação SMIL, onde são declaradas duas mídias (linhas 2 e 3). A atribuição `begin=0s` na mídia Video1 (linha 2) indica que sua exibição começará junto com a ativação da estrutura `par`. Já a atribuição na mídia "Video2" `begin=video1.click` (linha 3) indica que essa mídia começará quando a mídia "Video1" for selecionada. Ambas as mídias têm seu término definido pelo atributo `dur`.

O Código 41 apresenta o modelo GI, obtido pela transformação do SMIL para GI, aplicando-se as regras descritas abaixo:

Tabela A.1.1 – Correspondência SMIL - GI a partir das regras

SMIL	**Grafo Intermediário**	
Media	Vertex	
Region	Region Attribute	Vertex
beginCondition	OutputEdge	Vertex
	IncomingEdge	Vertex
endCondition	OutputEdge	Vertex
	IncomingEdge	Vertex
par	OutputEdge	Vertex
	IncomingEdge	Vertex
seq	OutputEdge	Vertex
	IncomingEdge	Vertex
excl	OutputEdge	Vertex
	IncomingEdge	Vertex

Código 40 Documento SMIL

```
1 <par dur="indefinite">
2          <video id="video1" begin="0s" dur="10s"/>
3          <video id="video2" begin="video1.click" dur="20s" />
4 </par>
```

- **Regra 1** - os elementos `Media` “Video1” e “Video2” (linhas 2 e 3 do código SMIL) são traduzidos para o `Vertex` em GI (linhas 1 e 8 no código GI), que é um elemento interno do `VertexList`;
- **Regra 3** - os atributos `dur` SMIL (linhas 2 e 3 do código SMIL) são traduzidos respectivamente para os atributos `ExpliticDur` no `Vertex` de GI (linhas 1 e 8); e
- **Regra 4** - o atributo `begin` da mídia “Video2” SMIL (linha 3 do código SMIL) é traduzido para `IncomingEdge` no `Vertex` “Video2” (linha 11 do código GI) e para `OutputEdge` no `Vertex` “Video1” (linha 4 do código GI).

Código 41 Documento GI traduzido de SMIL

```
<Vertex idMedia="video1" context="par1" explicitDur="10s"/>
<OutputEdgeList size="1">
 <OutputEdgeType size="1" Type="onSelection">
  <outputEdge Condition="onSelection" key="click1" target="video2" id="L1"/>
 </OutputEdgeType>
</OutputEdgeList>
</Vertex>
<Vertex idMedia="video2" context="par1" explicitDur="20s"/>
<IncomingEdgeList size="1">
 <IncomingEdgeType size="1" Type="start">
  <incomingEdge Action="start" sourceVertex="video1" id="L1"/>
 </IncomingEdgeType>
</IncomingEdgeList>
</Vertex>
```

APÊNDICE B – Código NCL - O Primeiro João

Code 42 Codigo NCL O Primeiro João

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nclRaw.ecore:NCL xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
        xmlns:nclRaw.ecore="/modelo/nclRaw.ecore" id="joao">
 <head>
    <connectorBase>
      <causalConnector id="onKeySelectionStopSetStart">
        <connectorParam name="varSet"/>
        <connectorParam name="keyCode"/>
        <connectorParam name="vIdioma"/>
        <compoundCondition operator="and">
        <simpleCondition role="onSelection" key="$keyCode"/>
        <assessmentStatement comparator="eq">
        <attributeAssessment role="systemLanguage"
              attributeType="nodeProperty" eventType="presentation"/>
                <valueAssessment value="$vIdioma"/>
        </assessmentStatement>
        </compoundCondition>
        <compoundAction operator="seq">
          <simpleAction role="start" max="unbounded" qualifier="par"/>
          <simpleAction role="set" value="$varSet"/>
          <simpleAction role="stop" max="unbounded" qualifier="par"/>
      </compoundAction>
      </causalConnector>

      <causalConnector id="onEndSetStop">
        <connectorParam name="varSet"/>
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
          <simpleAction role="set" value="$varSet"/>
          <simpleAction role="stop" max="unbounded" qualifier="par"/>
        </compoundAction>
      </causalConnector>
```

Code 43 Codigo NCL O Primeiro João

```
  <causalConnector id="onBeginStart">
        <simpleCondition role="onBegin"/>
        <simpleAction role="start"/>
      </causalConnector>

      <causalConnector id="onEndStop">
        <simpleCondition role="onEnd"/>
        <simpleAction role="stop" max="unbounded" qualifier="par"/>
      </causalConnector>

      <causalConnector id="onEndStart">
        <simpleCondition role="onEnd"/>
        <simpleAction role="start"/>
      </causalConnector>
    </connectorBase>
  </head>
   <body>
    <port id="entry" component="animation"/>
    <context id="form">
      <port id="spForm" component="ptForm"/>
      <port id="seForm" component="enForm"/>
      <media id="ptForm" src="ptForm.htm">
        <property name="left" value="56.25%"/>
        <property name="top" value="8.33%"/>
        <property name="width" value="38.75%"/>
        <property name="height" value="71.7%"/>
        <property name="zIndex" value="3"/>
      </media>
      <media id="enForm" src="enForm.htm">
        <property name="left" value="56.25%"/>
        <property name="top" value="8.33%"/>
        <property name="width" value="38.75%"/>
        <property name="height" value="71.7%"/>
        <property name="zIndex" value="3"/>
      </media>
    </context>
```

Code 44 Codigo NCL O Primeiro João

```
<media id="animation" src="animGar.mp4">
     <property name="width" value="100%"/>
     <property name="height" value="100%"/>
     <property name="zIndex" value="2"/>
     <property name="bounds"/>
     <area id="segPhoto" begin="6s" end="40s"/>
     <area id="segIcon" begin="15s" end="21s"/>
     <property name="explicitDur" value="70s"/>
   </media>
   <media id="music" src="choro.mp3">
      <property name="explicitDur" value="65s"/>
   </media>
   <media id="photo" src="photo.png">
     <property name="left" value="5%"/>
     <property name="top" value="6.7%"/>
     <property name="width" value="18.5%"/>
     <property name="height" value="18.5%"/>
     <property name="zIndex" value="3"/>
     <property name="explicitDur" value="10s"/>
   </media>
   <media id="icon" src="icon.png">
     <property name="left" value="67.5%"/>
     <property name="top" value="11.7%"/>
     <property name="width" value="8.45%"/>
     <property name="height" value="6.7%"/>
     <property name="zIndex" value="4"/>
   </media>
   <media id="shoes" src="shoes.mp4">
     <property name="left" value="15%"/>
     <property name="top" value="60%"/>
     <property name="width" value="25%"/>
     <property name="height" value="25%"/>
     <property name="zIndex" value="3"/>
     <property name="explicitDur" value="14s"/>
   </media>
   <media id="background" src="background.png">
     <property name="width" value="100%"/>
     <property name="height" value="100%"/>
     <property name="zIndex" value="1"/>
   </media>
```

Code 45 Codigo NCL O Primeiro João

```
<link id="lIcon" xconnector="onBeginStart">
      <bind role="onBegin" component="animation" interface="segIcon"/>
      <bind role="start" component="icon"/>
    </link>
    <link id="lEndIcon" xconnector="onEndStop">
      <bind role="onEnd" component="animation" interface="segIcon"/>
      <bind role="stop" component="icon"/>
    </link>

    <link id="lBeginShoes1" xconnector="onKeySelectionStopSetStart">
      <bind role="onSelection" component="icon">
        <bindParam name="keyCode" value="RED"/>
        <bindParam name="vIdioma" value="pt"/>
      </bind>
      <bind role="start" component="shoes"/>
      <bind role="start" component="form" interface="spForm"/>
      <bind role="set" component="animation" interface="bounds">
         <bindParam name="varSet" value="5%,6.67%,45%,45%"/>
      </bind>
      <bind role="stop" component="icon"/>
    </link>

    <link id="lBeginShoes2" xconnector="onKeySelectionStopSetStart">
      <bind role="onSelection" component="icon">
        <bindParam name="keyCode" value="RED"/>
        <bindParam name="vIdioma" value="en"/>
      </bind>
      <bind role="start" component="shoes"/>
      <bind role="start" component="form" interface="seForm"/>
      <bind role="set" component="animation" interface="bounds">
         <bindParam name="varSet" value="5%,6.67%,45%,45%"/>
      </bind>
      <bind role="stop" component="icon"/>
    </link>

```

Code 46 Código NCL O Primeiro João

```
<link id="lEndShoes" xconnector="onEndSetStop">
      <bind role="onEnd" component="shoes"/>
      <bind role="set" component="animation" interface="bounds">
        <bindParam name="varSet" value="0,0,854,480"/>
      </bind>
      <bind role="stop" component="form"/>
    </link>

    <link id="lMusic" xconnector="onBeginStart">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="music"/>
    </link>
    <link id="lPhoto" xconnector="onBeginStart">
      <bind role="onBegin" component="animation" interface="segPhoto"/>
      <bind role="start" component="photo"/>
    </link>
    <link id="lBackground" xconnector="onBeginStart">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="background"/>
    </link>
    <link id="lEndBackground" xconnector="onEndStop">
      <bind role="onEnd" component="animation"/>
      <bind role="stop" component="background"/>
    </link>
  </body>
</nclRaw.ecore:NCL>
```

APÊNDICE C – Código NCL - Viva Mais

Code 47 Codigo NCL Viva Mais

```
<<?xml version="1.0" encoding="ISO-8859-1"?>
<nclRaw.ecore:NCL xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:nclRaw.ecore="/modelo/nclRaw.ecore" id="vm">

<head>
  <connectorBase>

    <causalConnector id="onBeginStart">
      <simpleCondition role="onBegin"/>
      <simpleAction role="start" max="unbounded" qualifier="seq"/>
    </causalConnector>

    <causalConnector id="onEndStart">
      <simpleCondition role="onEnd"/>
      <simpleAction role="start" max="unbounded" qualifier="seq"/>
    </causalConnector>

    <causalConnector id="onEndStop">
      <simpleCondition role="onEnd"/>
      <simpleAction role="stop" max="unbounded" qualifier="par"/>
    </causalConnector>

    <causalConnector id="onEndAbort">
      <simpleCondition role="onEnd"/>
      <simpleAction role="abort" max="unbounded" qualifier="par"/>
    </causalConnector>

    <causalConnector id="onKeySelectionStartStop">
      <connectorParam name="keyCode"/>
      <simpleCondition role="onSelection" key="$keyCode" />
      <compoundAction operator="seq">
        <simpleAction role="stop" max="unbounded" qualifier="seq"/>
        <simpleAction role="start" max="unbounded" qualifier="seq"/>
      </compoundAction>
    </causalConnector>
```

Code 48 Codigo NCL Viva Mais

```
  <causalConnector id="onKeySelectionSetResizeStartStop">
      <connectorParam name="keyCode" />
      <connectorParam name="var" />
      <simpleCondition role="onSelection" key="$keyCode"/>
      <compoundAction operator="seq">
        <simpleAction role="stop" max="unbounded" qualifier="seq"/>
        <simpleAction role="set" value="$var" max="unbounded" qualifier="seq"/>
        <simpleAction role="start" max="unbounded" qualifier="seq"/>
      </compoundAction>
    </causalConnector>
  </connectorBase>
</head>

<body>

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

  <media  id="video" src="video/vivamais.mp4" >
    <area id="mainEvent" begin="0s"/>
    <area id="arIcone"  begin="63s" end="74s" />
    <property name="bounds"/>
    <property name="height" value="100%"/>
    <property name="width" value="100%"/>
    <property name="explicitDur" value="120s"/>
  </media>

  <media id="icone" src="imagens/interativo.png">
      <property name="height" value="7.29%"/>
      <property name="right" value="2%"/>
      <property name="top" value="2%"/>
      <property name="width" value="33.59%"/>
    </media>

```

Code 49 Codigo NCL Viva Mais

```
    <media id="resultadoPrato1" src="imagens/InstPratoVermelhoFundo.png">
      <property name="height" value="16.67%"/>
      <property name="left" value="25%"/>
      <property name="top" value="60%"/>
      <property name="width" value="50%"/>
    </media>

    <media id="resultadoPrato2" src="imagens/InstPratoAmareloFundo.png">
      <property name="height" value="16.67%"/>
      <property name="left" value="25%"/>
      <property name="top" value="60%"/>
      <property name="width" value="50%"/>
    </media>

    <media id="resultadoPrato3" src="imagens/InstPratoVerdeFundo.png">
      <property name="height" value="16.67%"/>
      <property name="left" value="25%"/>
      <property name="top" value="60%"/>
      <property name="width" value="50%"/>
    </media>

    <media id="resultadoPrato4" src="imagens/InstPratoAzulFundo.png">
      <property name="height" value="16.67%"/>
      <property name="left" value="25%"/>
      <property name="top" value="60%"/>
      <property name="width" value="50%"/>
    </media>
```

Code 50 Codigo NCL Viva Mais

```
<context id="ctxPratos">

    <port id="poEntPratos" component="fundoPratos"/>

    <media id="fundoPratos" src="imagens/fundo_pratos.png">
        <property name="width" value="100%"/>
        <property name="height" value="100%"/>
    </media>

    <media id="prato1" src="imagens/PratoVermelhoFundo.png">
      <property name="height" value="29.58%"/>
      <property name="left" value="7.5%"/>
      <property name="top" value="7.6667%"/>
      <property name="width" value="19.53%"/>
    </media>

    <media id="prato2" src="imagens/PratoAmareloFundo.png">
      <property name="height" value="29.58%"/>
      <property name="left" value="73.75%"/>
      <property name="top" value="7.6667%"/>
      <property name="width" value="19.53%"/>
    </media>

    <media id="prato3" src="imagens/PratoVerdeFundo.png">
      <property name="height" value="29.58%"/>
      <property name="left" value="7.5%"/>
      <property name="top" value="46.6667%"/>
      <property name="width" value="19.53%"/>
    </media>

    <media id="prato4" src="imagens/PratoAzulFundo.png">
      <property name="height" value="29.58%"/>
      <property name="left" value="73.75%"/>
      <property name="top" value="46.6667%"/>
      <property name="width" value="19.53%"/>
    </media>
```

Code 51 Codigo NCL Viva Mais

```
    <media id="instrucaoPratos" src="imagens/InstPratosFundo.png">
       <property name="height" value="16.67%"/>
       <property name="left" value="25%"/>
       <property name="top" value="60%"/>
       <property name="width" value="50%"/>
     </media>

     <link id="l1" xconnector="onBeginStart">
       <bind component="fundoPratos" role="onBegin"/>
       <bind component="prato1" role="start"/>
       <bind component="prato2" role="start"/>
       <bind component="prato3" role="start"/>
       <bind component="prato4" role="start"/>
       <bind component="instrucaoPratos" role="start"/>
     </link>

    </context>

     <link id="l2" xconnector="onKeySelectionStartStop">
       <bind component="prato1" role="onSelection">
         <bindParam name="keyCode" value="RED"/>
       </bind>
       <bind component="prato2" role="stop"/>
       <bind component="prato3" role="stop"/>
       <bind component="prato4" role="stop"/>
       <bind component="instrucaoPratos" role="stop"/>
       <bind component="resultadoPrato1" role="start"/>
     </link>

     <link  id="l3" xconnector="onKeySelectionStartStop">
       <bind component="prato2" role="onSelection">
         <bindParam name="keyCode" value="YELLOW"/>
       </bind>
       <bind component="prato1" role="stop"/>
       <bind component="prato3" role="stop"/>
       <bind component="prato4" role="stop"/>
       <bind component="instrucaoPratos" role="stop"/>
       <bind component="resultadoPrato2" role="start"/>
     </link>
```

Code 52 Codigo NCL Viva Mais

```
  <link  id="l4" xconnector="onKeySelectionStartStop">
      <bind component="prato3" role="onSelection">
        <bindParam name="keyCode" value="GREEN"/>
      </bind>
      <bind component="prato1" role="stop"/>
      <bind component="prato2" role="stop"/>
      <bind component="prato4" role="stop"/>
      <bind component="instrucaoPratos" role="stop"/>
      <bind component="resultadoPrato3" role="start"/>
    </link>

  <link  id="l5" xconnector="onKeySelectionStartStop">
      <bind component="prato4" role="onSelection">
        <bindParam name="keyCode" value="BLUE"/>
      </bind>
      <bind component="prato1" role="stop"/>
      <bind component="prato2" role="stop"/>
      <bind component="prato3" role="stop"/>
      <bind component="instrucaoPratos" role="stop"/>
      <bind component="resultadoPrato4" role="start"/>
  </link>

  <link  id="l6" xconnector="onBeginStart">
    <bind component="video" interface="arIcone" role="onBegin"/>
    <bind component="icone" role="start"/>
  </link>

  <link  id="l7" xconnector="onEndStop">
    <bind component="video" interface="arIcone" role="onEnd"/>
    <bind component="icone" role="stop"/>
  </link>
```

Code 53 Codigo NCL Viva Mais

```
 <link  id="l8" xconnector="onKeySelectionSetResizeStartStop">
    <bind component="icone" role="onSelection">
      <bindParam name="keyCode" value="RED"/>
    </bind>
    <bind component="video" interface="bounds" role="set">
      <bindParam name="var" value="27.97%, 13.33%, 44.22%, 41.87%"/>
    </bind>
    <bind component="fundoPratos" role="start"/>
    <bind component="icone" role="stop"/>
  </link>

  <link  id="l9" xconnector="onEndStop">
    <bind component="video" role="onEnd"/>
    <bind component="prato1" role="stop"/>
    <bind component="prato2" role="stop"/>
    <bind component="prato3" role="stop"/>
    <bind component="prato4" role="stop"/>
    <bind component="resultadoPrato1" role="stop"/>
    <bind component="resultadoPrato2" role="stop"/>
    <bind component="resultadoPrato3" role="stop"/>
    <bind component="resultadoPrato4" role="stop"/>
    <bind component="instrucaoPratos" role="stop"/>
    <bind component="fundoPratos" role="stop"/>
  </link>

  </body>
</nclRaw.ecore:NCL>
```

APÊNDICE D – Código NCL - Vestiba-TV

Code 54 Codigo NCL VestibaTV

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nclRaw.ecore:NCL xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:nclRaw.ecore="/modelo/nclRaw.ecore" id="vtv">
<head>

<connectorBase>

<causalConnector id="onBegin1StartN">
  <simpleCondition role="onBegin"/>
  <simpleAction role="start" max="unbounded" qualifier="par"/>
</causalConnector>

<causalConnector id="onEnd1StopN">
  <simpleCondition role="onEnd"/>
  <simpleAction role="stop" max="unbounded" qualifier="par"/>
</causalConnector>

<causalConnector id="onEndStart">
<simpleCondition role="onEnd"/>
<simpleAction role="start"/>
</causalConnector>

<causalConnector id="onKeySelection1StartNStopN">
  <connectorParam name="keyCode" />
  <simpleCondition role="onSelection" key="$keyCode" />
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" qualifier="par"/>
    <simpleAction role="stop" max="unbounded" qualifier="par"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelection1StopN">
  <connectorParam name="keyCode" />
  <simpleCondition role="onSelection" key="$keyCode" />
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" qualifier="par"/>
  </compoundAction>
</causalConnector>

```

Code 55 Codigo NCL VestibaTV

```
1
2  <causalConnector id="onKeySelectionStart">
3    <connectorParam name="keyCode"/>
4    <simpleCondition key="$keyCode" role="onSelection"/>
5  <simpleAction role="start"/>
6  </causalConnector>
7
8  <causalConnector id="onKeySelectionSet">
9    <connectorParam name="keyCode"/>
10   <connectorParam name="var"/>
11   <simpleCondition key="$keyCode" role="onSelection"/>
12 <simpleAction role="set" value="$var"/>
13 </causalConnector>
14
15
16 <causalConnector id="onBeginSet">
17   <connectorParam name="var"/>
18   <simpleCondition role="onBegin"/>
19   <simpleAction role="set" value="$var"/>
20 </causalConnector>
21
22 </connectorBase>
23
24 </head>
25
26 <body>
27
28 <port id="entry" component="VideoAbertura"/>
29
30 <media  id="VideoAbertura" src="media/BACON_Abertura_vestiba_mpeg.mp4">
31 <property name="left" value="0%"/>
32 <property name="top" value="0%"/>
33 <property name="width" value="100%"/>
34 <property name="height" value="100%"/>
35 <property name="explicitDur" value="2s"/>
36 </media>
```

Code 56 Codigo NCL VestibaTV

```
<media  id="video1" src="media/trailer1.mp4">
<area id="DicaAos5s" begin="5s" end="10s"/>
<!-- âncora no vídeo que indica que existe a quarta dica -->
<area id="DicaAos30" begin="30s" end="40s"/>
<!-- âncora no vídeo que indica que existe a quinta dica -->
<area id="DicaAos60" begin="60s" end="70s"/>
<!-- âncora no vídeo que indica que existe a sexta dica -->
<area id="DicaAos90" begin="90s" end="100s"/>
<!-- âncora no vídeo que indica que existe a setima dica -->
<area id="DicaAos120" begin="120s" end="150s"/>
<!-- âncora no vídeo que indica que existe a decima dica -->
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
<property name="explicitDur" value="165s"/>
</media>
<media  id="DicaCoruja5s" src="media/BEACON_Coruja+I.png">
<property name="left" value="80%"/>
<property name="top" value="80%"/>
<property name="width" value="15%"/>
<property name="height" value="15%"/>
</media>
<media  id="DicaCoruja30s" src="media/BEACON_Coruja+I.png">
<property name="left" value="80%"/>
<property name="top" value="80%"/>
<property name="width" value="15%"/>
<property name="height" value="15%"/>
</media>
<media  id="DicaCoruja60s" src="media/BEACON_Coruja+I.png">
<property name="left" value="80%"/>
<property name="top" value="80%"/>
<property name="width" value="15%"/>
<property name="height" value="15%"/>
</media>
```

Code 57 Codigo NCL VestibaTV

```
<media  id="DicaCoruja90s" src="media/BEACON_Coruja+I.png">
<property name="left" value="80%"/>
<property name="top" value="80%"/>
<property name="width" value="15%"/>
<property name="height" value="15%"/>
</media>
<media  id="DicaCoruja120s" src="media/BEACON_Coruja+I.png">
<property name="left" value="80%"/>
<property name="top" value="80%"/>
<property name="width" value="15%"/>
<property name="height" value="15%"/>
</media>
<media  id="Dica05" src="media/Dica05.png">
<property name="left" value="8%"/>
<property name="top" value="72%"/>
<property name="width" value="85%"/>
<property name="height" value="23%"/>
</media>
<media  id="Questao001" src="media/Questao001.png">
<property name="explicitDur" value="10s"/>
        <property name="left" value="8%"/>
<property name="top" value="72%"/>
<property name="width" value="85%"/>
<property name="height" value="23%"/>
</media>

<media  id="Dica60" src="media/Dica60.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="correta01" src="media/correta01.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
```

Code 58 Codigo NCL VestibaTV

```
<media  id="questao02" src="media/questao02.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="correta02" src="media/correta02.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="questao03" src="media/questao03.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="correta03" src="media/correta03.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="errada"   src="media/errada.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="errada02" src="media/errada.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
```

Code 59 Codigo NCL VestibaTV

```
<media  id="errada03" src="media/errada.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="errada04" src="media/errada.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="errada05" src="media/errada.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="errada06" src="media/errada.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="errada07" src="media/errada.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
<media  id="errada08" src="media/errada.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>
```

Code 60 Codigo NCL VestibaTV

```
<media  id="errada09" src="media/errada.png">
<property name="left" value="0%"/>
<property name="top" value="0%"/>
<property name="width" value="100%"/>
<property name="height" value="100%"/>
</media>

<link id="FimVideoAbertura" xconnector="onEndStart">
<bind component="VideoAbertura" role="onEnd"/>
<bind component="video1" role="start"/>
</link>

<link id="TerceiraDica" xconnector="onBegin1StartN">
<bind component="video1" interface="DicaAos5s" role="onBegin" />
<bind component="DicaCoruja5s" role="start" />
</link>

<link id="PassaTerceiraDica" xconnector="onEnd1StopN">
<bind component="video1" interface="DicaAos5s" role="onEnd" />
<bind component="DicaCoruja5s" role="stop" />
</link>

<link id="AbreTerceiraDica" xconnector="onKeySelection1StartNStopN">
<bind component="DicaCoruja5s" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="Dica05" role="start" />
<bind component="DicaCoruja5s" role="stop" />
</link>
```

Code 61 Codigo NCL VestibaTV

```
<link id="QuintaDica" xconnector="onBegin1StartN">
<bind component="video1" interface="DicaAos30" role="onBegin" />
<bind component="DicaCoruja30s" role="start" />
</link>

<link id="PassaQuintaDica" xconnector="onEnd1StopN">
<bind component="video1" interface="DicaAos30" role="onEnd" />
<bind component="DicaCoruja30s" role="stop" />
</link>

<link id="AbreQuintaDica" xconnector="onKeySelection1StartNStopN">
<bind component="DicaCoruja30s" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="Questao001" role="start" />
<bind component="DicaCoruja30s" role="stop" />
</link>

<link id="PrimeiraPergunta" xconnector="onBegin1StartN">
<bind component="video1" interface="DicaAos60" role="onBegin" />
<bind component="DicaCoruja60s" role="start" />
</link>

<link id="PassaPrimeiraPergunta" xconnector="onEnd1StopN">
<bind component="video1" interface="DicaAos60" role="onEnd" />
<bind component="DicaCoruja60s" role="stop" />
</link>

<link id="AbrePrimeiraPergunta" xconnector="onKeySelection1StartNStopN">
<bind component="DicaCoruja60s" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="Dica60" role="start" />
<bind component="DicaCoruja60s" role="stop" />
</link>

```

Code 62 Codigo NCL VestibaTV

```
<link id="ErraPrimeira01" xconnector="onKeySelection1StartNStopN">
<bind component="Dica60" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada02" role="start" />
<bind component="Dica60" role="stop" />

</link>

<link id="ErraPrimeiraEContinua01" xconnector="onKeySelection1StopN">
<bind component="errada02" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada02" role="stop" />

</link>

 <link id="ErraPrimeira02" xconnector="onKeySelection1StartNStopN">
<bind component="Dica60" role="onSelection">
<bindParam name="keyCode" value="GREEN" />
</bind>
<bind component="Dica60" role="stop" />
<bind component="errada" role="start" />
</link>

<link id="ErraPrimeiraEContinua02" xconnector="onKeySelection1StopN">
<bind component="errada" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada" role="stop" />

</link>

 <link id="ErraPrimeira03" xconnector="onKeySelection1StartNStopN">
<bind component="Dica60" role="onSelection">
<bindParam name="keyCode" value="BLUE" />
</bind>
<bind component="errada03" role="start" />
<bind component="Dica60" role="stop" />

</link>
```

Code 63 Codigo NCL VestibaTV

```
<link id="ErraPrimeiraEContinua03" xconnector="onKeySelection1StopN">
<bind component="errada03" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada03" role="stop" />

</link>

<link id="RespostaCorreta01" xconnector="onKeySelection1StartNStopN">
<bind component="Dica60" role="onSelection">
<bindParam name="keyCode" value="YELLOW" />
</bind>
<bind component="correta01" role="start" />
<bind component="Dica60" role="stop" />

</link>

<link id="AcertaPrimeiraEContinua" xconnector="onKeySelection1StopN">
<bind component="correta01" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="correta01" role="stop" />

</link>

<link id="SegundaPergunta" xconnector="onBegin1StartN">
<bind component="video1" interface="DicaAos90" role="onBegin" />
<bind component="DicaCoruja90s" role="start" />
</link>

```

Code 64 Codigo NCL VestibaTV

```
<link id="PassaSegundaPergunta" xconnector="onEnd1StopN">
<bind component="video1" interface="DicaAos90" role="onEnd" />
<bind component="DicaCoruja90s" role="stop" />
</link>

<link id="AbreSegundaPergunta" xconnector="onKeySelection1StartNStopN">
<bind component="DicaCoruja90s" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="questao02" role="start" />
<bind component="DicaCoruja90s" role="stop" />
</link>

<link id="ErraSegunda01" xconnector="onKeySelection1StartNStopN">
<bind component="questao02" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada04" role="start" />
<bind component="questao02" role="stop" />

</link>

<link id="ErraSegundaEContinua01" xconnector="onKeySelection1StopN">
<bind component="errada04" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada04" role="stop" />

</link>

<link id="ErraSegunda02" xconnector="onKeySelection1StartNStopN">
<bind component="questao02" role="onSelection">
<bindParam name="keyCode" value="GREEN" />
</bind>
<bind component="questao02" role="stop" />
<bind component="errada05" role="start" />
</link>
```

Code 65 Codigo NCL VestibaTV

```
<link id="ErraSegundaEContinua02" xconnector="onKeySelection1StopN">
<bind component="errada05" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada05" role="stop" />

</link>

<link id="ErraSegunda03" xconnector="onKeySelection1StartNStopN">
<bind component="questao02" role="onSelection">
<bindParam name="keyCode" value="YELLOW" />
</bind>
<bind component="errada06" role="start" />
<bind component="questao02" role="stop" />

</link>

<link id="ErraSegundaEContinua03" xconnector="onKeySelection1StopN">
<bind component="errada06" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada06" role="stop" />

</link>

<link id="RespostaCorreta02" xconnector="onKeySelection1StartNStopN">
<bind component="questao02" role="onSelection">
<bindParam name="keyCode" value="BLUE" />
</bind>
<bind component="correta02" role="start" />
<bind component="questao02" role="stop" />

</link>
```

Code 66 Codigo NCL VestibaTV

```
<link id="AcertaSegundaEContinua" xconnector="onKeySelection1StopN">
<bind component="correta02" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="correta02" role="stop" />

</link>

<link id="TerceiraPergunta" xconnector="onBegin1StartN">
<bind component="video1" interface="DicaAos120" role="onBegin" />
<bind component="DicaCoruja120s" role="start" />
</link>

<link id="PassaTerceiraPergunta" xconnector="onEnd1StopN">
<bind component="video1" interface="DicaAos120" role="onEnd" />
<bind component="DicaCoruja120s" role="stop" />
</link>

<link id="AbreTerceiraPergunta" xconnector="onKeySelection1StartNStopN">
<bind component="DicaCoruja120s" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="questao03" role="start" />
<bind component="DicaCoruja120s" role="stop" />
</link>

<link id="ErraTerceira01" xconnector="onKeySelection1StartNStopN">
<bind component="questao03" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada07" role="start" />
<bind component="questao03" role="stop" />

</link>
```

Code 67 Codigo NCL VestibaTV

```
<link id="ErraTerceiraEContinua01" xconnector="onKeySelection1StopN">
<bind component="errada07" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada07" role="stop" />

</link>

<link id="ErraTerceira02" xconnector="onKeySelection1StartNStopN">
<bind component="questao03" role="onSelection">
<bindParam name="keyCode" value="GREEN" />
</bind>
<bind component="questao03" role="stop" />
<bind component="errada08" role="start" />
</link>

<link id="ErraTerceiraEContinua02" xconnector="onKeySelection1StopN">
<bind component="errada08" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada08" role="stop" />

</link>

<link id="ErraTerceira03" xconnector="onKeySelection1StartNStopN">
<bind component="questao03" role="onSelection">
<bindParam name="keyCode" value="BLUE" />
</bind>
<bind component="errada09" role="start" />
<bind component="questao03" role="stop" />

</link>
```

Code 68 Codigo NCL VestibaTV

```
<link id="ErraTerceiraEContinua03" xconnector="onKeySelection1StopN">
<bind component="errada09" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="errada09" role="stop" />

</link>

<link id="RespostaCorreta03" xconnector="onKeySelection1StartNStopN">
<bind component="questao03" role="onSelection">
<bindParam name="keyCode" value="YELLOW" />
</bind>
<bind component="correta03" role="start" />
<bind component="questao03" role="stop" />

</link>

<link id="AcertaTerceiraEContinua" xconnector="onKeySelection1StopN">
<bind component="correta03" role="onSelection">
<bindParam name="keyCode" value="RED" />
</bind>
<bind component="correta03" role="stop" />
</link>

</body>
</nclRaw.ecore:NCL>

```

APÊNDICE E – Código GI - Primeiro João

Code 69 Codigo GI - Primeiro João

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<grafo.ecore:GrafoNCL xmi:version="2.0" xmlns:xmi="http://www.omg.org
/XMI" xmlns:grafo.ecore="/modelo/grafo.ecore">

 <ElementosMidias quantia="4">
 <Midia idMidia="animation" contexto="main" play="1" nend="70s">
    <LigacoesEntrada quantiaTotal="4">
    <EntradaTipos quantia="3" TipoEntrada="set">
    <Entrada classificacao="2" vizinho="icon" idLink="lBeginShoes1"
    idConector="onKeySelectionStopSetStart" acao="set"
    vizinhoAcao="onSelection"></Entrada>
    <Entrada classificacao="2" vizinho="icon" idLink="lBeginShoes2"
    idConector="onKeySelectionStopSetStart" acao="set"
    vizinhoAcao="onSelection"></Entrada>
    <Entrada classificacao="2" vizinho="shoes" idLink="lEndShoes"
    idConector="onEndSetStop" acao="set" vizinhoAcao="onEnd"></Entrada>
    </EntradaTipos>
    <EntradaTipos quantia="1" TipoEntrada="play">
    <Entrada classificacao="3" vizinho="controle" idLink="Lcontroleclick"
    idConector="Ccontroleclick" acao="play" vizinhoAcao="click"></Entrada>
    </EntradaTipos>
    </LigacoesEntrada>
    <LigacoesAcoes quantiaTotal="4">
    <AcaoTipos quantia="2" TipoAcao="onBegin">
    <Acao classificacao="1" idLink="lIcon" idConector="onBeginStart"
    acao="onBegin" tecla="" ancora="segIcon" ordem="15">
    <CondicaoSimples classificacao="1" role="onBegin"></CondicaoSimples>
    <LigacoesSaida quantiaTotal="1">
       <Saida classificacao="1" vizinho="icon" idLink="lIcon"
       idConector="onBeginStart" acao="start"></Saida>
    </LigacoesSaida>
    </Acao>
```

Code 70 Codigo GI - Primeiro João

```
    <Acao classificacao="1" idLink="lBackground" idConector="onBeginStart"
    acao="onBegin" tecla="" ancora="">
    <CondicaoSimples classificacao="1" role="onBegin"></CondicaoSimples>
    <LigacoesSaida quantiaTotal="1">
       <Saida classificacao="1" vizinho="background" idLink="lBackground"
       idConector="onBeginStart" acao="start"></Saida>
    </LigacoesSaida>
    </Acao>
    </AcaoTipos>
    <AcaoTipos quantia="2" TipoAcao="onEnd">
    <Acao classificacao="1" idLink="lEndIcon" idConector="onEndStop"
    acao="onEnd" tecla="" ancora="segIcon" ordem="21">
    <CondicaoSimples classificacao="1" role="onEnd"></CondicaoSimples>
    <LigacoesSaida quantiaTotal="1">
       <Saida classificacao="1" vizinho="icon" idLink="lEndIcon"
       idConector="onEndStop" acao="stop"></Saida>
    </LigacoesSaida>
    </Acao>
    <Acao classificacao="1" idLink="lEndBackground" idConector="onEndStop"
    acao="onEnd" tecla="" ancora="">
    <CondicaoSimples classificacao="1" role="onEnd"></CondicaoSimples>
    <LigacoesSaida quantiaTotal="1">
       <Saida classificacao="1" vizinho="background" idLink="lEndBackground"
       idConector="onEndStop" acao="stop"></Saida>
    </LigacoesSaida>
    </Acao>
    </AcaoTipos>
    </LigacoesAcoes>

```

Code 71 Codigo GI - Primeiro João

```
    <PontosAncoras quantiaTotal="2">
    <Ancoras nome="segPhoto" inicio="6s" fim="40s"></Ancoras>
    <Ancoras nome="segIcon" inicio="15s" fim="21s"></Ancoras>
    </PontosAncoras>
    <LisPropriedades quantiaTotal="5">
    <Propriedades nome="width" valor="100%"></Propriedades>
    <Propriedades nome="height" valor="100%"></Propriedades>
    <Propriedades nome="zIndex" valor="2"></Propriedades>
    <Propriedades nome="bounds" valor=""></Propriedades>
    <Propriedades nome="explicitDur" valor="70s"></Propriedades>
    </LisPropriedades>
</Midia>
<Midia idMidia="icon" contexto="main" start="1" stop="3" selStop="2"
select="2" sequencia="1">
    <LigacoesEntrada quantiaTotal="4">
    <EntradaTipos quantia="3" TipoEntrada="stop">
    <Entrada classificacao="1" vizinho="animation" idLink="lEndIcon"
    idConector="onEndStop" acao="stop" vizinhoAcao="onEnd"></Entrada>
    <Entrada classificacao="1" vizinho="icon" idLink="lBeginShoes1"
    idConector="onKeySelectionStopSetStart" acao="stop"
    vizinhoAcao="onSelection"></Entrada>
    <Entrada classificacao="1" vizinho="icon" idLink="lBeginShoes2"
    idConector="onKeySelectionStopSetStart" acao="stop"
    vizinhoAcao="onSelection"></Entrada>
    </EntradaTipos>
    <EntradaTipos quantia="1" TipoEntrada="start">
    <Entrada classificacao="1" vizinho="animation" idLink="lIcon"
    idConector="onBeginStart" acao="start" vizinhoAcao="onBegin"></Entrada>
    </EntradaTipos>
    </LigacoesEntrada>
```

Code 72 Codigo GI - Primeiro João

```
1     <LigacoesAcoes quantiaTotal="2" quantiaSelecoes="2">
2     <AcaoTipos quantia="2" TipoAcao="onSelection">
3     <Acao classificacao="2" idLink="lBeginShoes1"
4     idConector="onKeySelectionStopSetStart" acao="onSelection"
5     tecla="" ancora="">
6     <CondicaoComposta classificacao="1" juncao="and">
7        <CondicaoSimples classificacao="2" key="$keyCode"
8        role="onSelection"></CondicaoSimples>
9        <CondicaoSimples classificacao="3" variavel="systemLanguage"
10       valor="$vIdioma"></CondicaoSimples>
11    </CondicaoComposta>
12    <LigacoesSaida quantiaTotal="4">
13       <Saida classificacao="1" vizinho="shoes" idLink="lBeginShoes1"
14       idConector="onKeySelectionStopSetStart" acao="start"></Saida>
15       <Saida classificacao="2" vizinho="animation" idLink="lBeginShoes1"
16       idConector="onKeySelectionStopSetStart" acao="set"></Saida>
17       <Saida classificacao="1" vizinho="icon" idLink="lBeginShoes1"
18       idConector="onKeySelectionStopSetStart" acao="stop"></Saida>
19    </LigacoesSaida>
20    <BindParametros quantia="2">
21       <Parametro nome="keyCode" valor="RED"></Parametro>
22       <Parametro nome="vIdioma" valor="pt"></Parametro>
23    </BindParametros>
24    </Acao>
25    <Acao classificacao="2" idLink="lBeginShoes2"
26    idConector="onKeySelectionStopSetStart" acao="onSelection"
27    tecla="" ancora="">
28    <CondicaoComposta classificacao="1" juncao="and">
29       <CondicaoSimples classificacao="2" key="$keyCode"
30       role="onSelection"></CondicaoSimples>
31       <CondicaoSimples classificacao="3" variavel="systemLanguage"
32       valor="$vIdioma"></CondicaoSimples>
33    </CondicaoComposta>
34
```

Code 73 Codigo GI - Primeiro João

```
    <LigacoesSaida quantiaTotal="4">
       <Saida classificacao="1" vizinho="shoes" idLink="lBeginShoes2"
       idConector="onKeySelectionStopSetStart" acao="start"></Saida>
       <Saida classificacao="2" vizinho="animation" idLink="lBeginShoes2"
       idConector="onKeySelectionStopSetStart" acao="set"></Saida>
       <Saida classificacao="1" vizinho="icon" idLink="lBeginShoes2"
       idConector="onKeySelectionStopSetStart" acao="stop"></Saida>
    </LigacoesSaida>
    <BindParametros quantia="2">
       <Parametro nome="keyCode" valor="RED"></Parametro>
       <Parametro nome="vIdioma" valor="en"></Parametro>
    </BindParametros>
    </Acao>
    </AcaoTipos>
    </LigacoesAcoes>
    <LisPropriedades quantiaTotal="5">
    <Propriedades nome="left" valor="67.5%"></Propriedades>
    <Propriedades nome="top" valor="11.7%"></Propriedades>
    <Propriedades nome="width" valor="8.45%"></Propriedades>
    <Propriedades nome="height" valor="6.7%"></Propriedades>
    <Propriedades nome="zIndex" valor="4"></Propriedades>
    </LisPropriedades>
    <LisVarAmbiente quantiaTotal="2">
    <varAmbiente nome="systemLanguage" valor=""
    conector="onKeySelectionStopSetStart"></varAmbiente>
    <varAmbiente nome="keyCode" valor=""
    conector="onKeySelectionStopSetStart"></varAmbiente>
    </LisVarAmbiente>
</Midia>
```

Code 74 Codigo GI - Primeiro João

```
<Midia idMidia="shoes" contexto="main" start="2" nend="14s">
   <LigacoesEntrada quantiaTotal="2">
   <EntradaTipos quantia="2" TipoEntrada="start">
   <Entrada classificacao="1" vizinho="icon" idLink="lBeginShoes1"
   idConector="onKeySelectionStopSetStart" acao="start"
   vizinhoAcao="onSelection"></Entrada>
   <Entrada classificacao="1" vizinho="icon" idLink="lBeginShoes2"
   idConector="onKeySelectionStopSetStart" acao="start"
   vizinhoAcao="onSelection"></Entrada>
   </EntradaTipos>
   </LigacoesEntrada>
   <LigacoesAcoes quantiaTotal="1">
   <AcaoTipos quantia="1" TipoAcao="onEnd">
   <Acao classificacao="1" idLink="lEndShoes" idConector="onEndSetStop"
   acao="onEnd" tecla="" ancora="">
   <CondicaoSimples classificacao="1" role="onEnd"></CondicaoSimples>
   <LigacoesSaida quantiaTotal="2">
      <Saida classificacao="2" vizinho="animation" idLink="lEndShoes"
      idConector="onEndSetStop" acao="set"></Saida>
   </LigacoesSaida>
   </Acao>
   </AcaoTipos>
   </LigacoesAcoes>
   <LisPropriedades quantiaTotal="6">
   <Propriedades nome="left" valor="15%"></Propriedades>
   <Propriedades nome="top" valor="60%"></Propriedades>
   <Propriedades nome="width" valor="25%"></Propriedades>
   <Propriedades nome="height" valor="25%"></Propriedades>
   <Propriedades nome="zIndex" valor="3"></Propriedades>
   <Propriedades nome="explicitDur" valor="14s"></Propriedades>
   </LisPropriedades>
</Midia>
```

Code 75 Codigo GI - Primeiro João

```
<Midia idMidia="background" contexto="main" start="1" stop="1">
   <LigacoesEntrada quantiaTotal="2">
   <EntradaTipos quantia="1" TipoEntrada="stop">
   <Entrada classificacao="1" vizinho="animation" idLink="lEndBackground"
   idConector="onEndStop" acao="stop" vizinhoAcao="onEnd"></Entrada>
   </EntradaTipos>
   <EntradaTipos quantia="1" TipoEntrada="start">
   <Entrada classificacao="1" vizinho="animation" idLink="lBackground"
   idConector="onBeginStart" acao="start" vizinhoAcao="onBegin"></Entrada>
   </EntradaTipos>
   </LigacoesEntrada>
   <LisPropriedades quantiaTotal="3">
   <Propriedades nome="width" valor="100%"></Propriedades>
   <Propriedades nome="height" valor="100%"></Propriedades>
   <Propriedades nome="zIndex" valor="1"></Propriedades>
   </LisPropriedades>
</Midia>
</ElementosMidias>
<Contexto nome="main" totMidias="4" totCont="0">
<midiasFilhas nome="animation" totE="1" totS="4">
   <ParamChamadaProcS nome="lIcononBeginStart" origem="animation"
   tipo="onBegin" ancora="segIcon"></ParamChamadaProcS>
   <ParamChamadaProcS nome="lEndIcononEndStop" origem="animation"
   tipo="onEnd" ancora="segIcon"></ParamChamadaProcS>
   <ParamChamadaProcS nome="lBackgroundonBeginStart" origem="animation"
   tipo="onBegin"></ParamChamadaProcS>
   <ParamChamadaProcS nome="lEndBackgroundonEndStop" origem="animation"
   tipo="onEnd"></ParamChamadaProcS>
   <ParamChamadaProcE nome="LcontroleclickCcontroleclick" origem="controle"
   tipo="play"></ParamChamadaProcE>
</midiasFilhas>
```

Code 76 Codigo GI - Primeiro João

```
<midiasFilhas nome="icon" totE="4" totS="2">
   <ParamChamadaProcS nome="lBeginShoes1onKeySelectionStopSetStart"
   origem="icon" tipo="onSelection"></ParamChamadaProcS>
   <ParamChamadaProcS nome="lBeginShoes2onKeySelectionStopSetStart"
   origem="icon" tipo="onSelection"></ParamChamadaProcS>
   <ParamChamadaProcE nome="lIcononBeginStart" origem="animation"
   tipo="start"></ParamChamadaProcE>
   <ParamChamadaProcE nome="lEndIcononEndStop" origem="animation"
   tipo="stop"></ParamChamadaProcE>
   <ParamChamadaProcE nome="lBeginShoes1onKeySelectionStopSetStart"
   origem="icon" tipo="stop"></ParamChamadaProcE>
   <ParamChamadaProcE nome="lBeginShoes2onKeySelectionStopSetStart"
   origem="icon" tipo="stop"></ParamChamadaProcE>
   <LisVarAmbiente quantiaTotal="2">
   <varAmbiente nome="systemLanguage" valor=""
   conector="onKeySelectionStopSetStart"></varAmbiente>
   <varAmbiente nome="keyCode" valor=""
   conector="onKeySelectionStopSetStart"></varAmbiente>
   </LisVarAmbiente>
</midiasFilhas>
<midiasFilhas nome="shoes" totE="2">
   <ParamChamadaProcE nome="lBeginShoes1onKeySelectionStopSetStart"
   origem="icon" tipo="start"></ParamChamadaProcE>
   <ParamChamadaProcE nome="lBeginShoes2onKeySelectionStopSetStart"
   origem="icon" tipo="start"></ParamChamadaProcE>
</midiasFilhas>
<midiasFilhas nome="background" totE="2">
   <ParamChamadaProcE nome="lBackgroundonBeginStart" origem="animation"
   tipo="start"></ParamChamadaProcE>
   <ParamChamadaProcE nome="lEndBackgroundonEndStop" origem="animation"
   tipo="stop"></ParamChamadaProcE>
</midiasFilhas>
```

Code 77 Codigo GI - Primeiro João

```
<LisLigaLocais total="15">
   <Parametro nome="lEndIcononEndStop" origem="animation"
   tipo="onEnd"></Parametro>
   <Parametro nome="lBeginShoes1onKeySelectionStopSetStart"
   origem="icon" tipo="onSelection"></Parametro>
   <Parametro nome="lEndBackgroundonEndStop" origem="animation"
   tipo="onEnd"></Parametro>
   <Parametro nome="LcontroleclickCcontroleclick" origem="controle"
   tipo="play"></Parametro>
   <Parametro nome="lBackgroundonBeginStart" origem="animation"
   tipo="start"></Parametro>
   <Parametro nome="lEndIcononEndStop" origem="animation"
   tipo="stop"></Parametro>
   <Parametro nome="lBeginShoes1onKeySelectionStopSetStart"
   origem="icon" tipo="stop"></Parametro>
   <Parametro nome="lBeginShoes2onKeySelectionStopSetStart"
   origem="icon" tipo="start"></Parametro>
   <Parametro nome="lEndBackgroundonEndStop" origem="animation"
   tipo="stop"></Parametro>
   <Parametro nome="lBackgroundonBeginStart" origem="animation"
   tipo="onBegin"></Parametro>
   <Parametro nome="lBeginShoes2onKeySelectionStopSetStart"
   origem="icon" tipo="stop"></Parametro>
   <Parametro nome="lIcononBeginStart" origem="animation"
   tipo="start"></Parametro>
   <Parametro nome="lIcononBeginStart" origem="animation"
   tipo="onBegin"></Parametro>
   <Parametro nome="lBeginShoes2onKeySelectionStopSetStart"
   origem="icon" tipo="onSelection"></Parametro>
   <Parametro nome="lBeginShoes1onKeySelectionStopSetStart"
   origem="icon" tipo="start"></Parametro>
</LisLigaLocais>
```

Code 78 Codigo GI - Primeiro João

```
1
2  <LisVarAmbienteContexto quantiaTotal="2">
3     <varAmbiente nome="systemLanguage" valor=""
4     conector="onKeySelectionStopSetStart"></varAmbiente>
5     <varAmbiente nome="keyCode" valor=""
6 conector="onKeySelectionStopSetStart"></varAmbiente>
7  </LisVarAmbienteContexto>
8  </Contexto>
9  <MidiasInterativas quantia="1">
10 <nomeMidia>icon</nomeMidia>
11 </MidiasInterativas>
12 </grafo.ecore:GrafoNCL>
13
14
```

APÊNDICE F – Código Fiacre - Primeiro João

Code 79 Codigo Fiacre - Primeiro João

```
type vet is int

type flag is array 2 of bool


 /*********************************************************/
 process OB_040_shoes_background_5  [mstart1:bool,mstop1:bool,
 mstart2:bool,mstop2:bool] is

     states idle, running1, running2, running3, erro, eem, eemt_5
     var aviso:bool:=false

     init to idle

     from idle   select
     mstart1? aviso;      to running1
     [] mstart2? aviso;   to running2
     end
     from running1    select
     mstop1? aviso;      to erro
     [] mstart2? aviso;   to running3
     end
     from running2    select
     mstop2? aviso;      to erro
     [] mstart1? aviso;   to running3
     end
     from erro   select
     mstop1? aviso;      to erro
     [] mstop2? aviso;   to erro
     [] mstart1? aviso;   to erro
     [] mstart2? aviso;   to erro
     end
     from running3   select
     mstop1? aviso;      to eem
     [] mstop2? aviso;   to eem
     [] wait[5,5];   to eemt_5
     end

```

Code 80 Codigo Fiacre - Primeiro João

```
1  /*****************************************************************
2  ******** PROCESSO, GLUE E COMPONENTE DA MIDIA animation
3  *****************************************************************/
4  /*parametros: entradas, saidas*/
5      process p_animation  [mstart:bool,mstop:bool,lIcononBeginStart:bool,
6      lEndIcononEndStop:bool,lBackgroundonBeginStart:bool,
7      lEndBackgroundonEndStop:bool]   is
8
9
10     states dormindo, avisandoFim, est1 ,est2 ,est3 ,est4 ,est5 ,est6 ,est7
11
12     var aviso:bool:=false
13
14     init to dormindo
15
16     from dormindo   mstart? aviso;   to est1
17
18    from est1    lBackgroundonBeginStart!aviso;  to est2
19    from est2    wait[15,15];  to est3
20    from est3    lIcononBeginStart!aviso;  to est4
21    from est4    wait[6,6];  to est5
22    from est5    lEndIcononEndStop!aviso;  to est6
23    from est6    wait[49,49];  to est7
24    from est7    lEndBackgroundonEndStop!aviso;  to avisandoFim
25
26     from avisandoFim   mstop! aviso;   to dormindo
27
```

Code 81 Codigo Fiacre - Primeiro João

```
/*parametros: saidas, entradas*/
    process glue_animation  [mstart:bool,mstop:bool,
    LcontroleclickCcontroleclick:bool] is

    states dormindo, est1, start
    var aviso:bool:=false

    init to dormindo

    from dormindo    select
    LcontroleclickCcontroleclick? aviso;   to start
    []      mstop? aviso;   to dormindo
    end

    from est1    select
    LcontroleclickCcontroleclick? aviso;   to est1
    []      mstop? aviso;   to dormindo
    end

    from start   mstart! aviso;   to est1

/*parametros: entradas, saidas*/
    component C_animation  [LcontroleclickCcontroleclick:bool,
    lIcononBeginStart:bool,
    lEndIcononEndStop:bool,lBackgroundonBeginStart:bool,
    lEndBackgroundonEndStop:bool]   is

    port
    mstart:bool in [0,0],mstop:bool in [0,0]

    par * in
p_animation[mstart,mstop,lIcononBeginStart,lEndIcononEndStop,
lBackgroundonBeginStart,lEndBackgroundonEndStop]
|| glue_animation[mstart,mstop,LcontroleclickCcontroleclick]

    end
```

Code 82 Codigo Fiacre - Primeiro João

```
1  /*****************************************************************
2  ******** PROCESSO, GLUE E COMPONENTE DA MIDIA icon
3  *****************************************************************/
4  /*parametros: entradas, saidas*/
5      process p_icon  [mstart:bool,mstop:bool] (&systemLanguage:vet,
6      &keyCode:vet, &estatus:flag)  is
7
8
9      /*tipoG  chega start chega stop select*/
10
11     states dormindo, est1, est2
12
13     var aviso:bool:=false
14
15     init to dormindo
16
17
18     from dormindo   mstart? aviso;   to est1
19
20     from est1    wait[0,0];estatus[1]:=true;    to est2
21
22     from est2    mstop? aviso;estatus[1]:=false;   to dormindo
23
```

Code 83 Codigo Fiacre - Primeiro João

```
1  /*parametros: saidas, entradas*/
2       process glue_icon  [mstart:bool,mstop:bool,lIcononBeginStart:bool,
3       lEndIcononEndStop:bool, lBeginShoes1onKeySelectionStopSetStart:bool,
4       lBeginShoes2onKeySelectionStopSetStart:bool] is
5
6       states dormindo, executando, start, stop, eselect
7
8    var aviso:bool:=false
9
10   init to dormindo
11
12   from dormindo    select
13         lIcononBeginStart? aviso;   to start
14         [] lEndIcononEndStop? aviso;   to dormindo
15         [] lBeginShoes1onKeySelectionStopSetStart? aviso;   to dormindo
16         [] lBeginShoes2onKeySelectionStopSetStart? aviso;   to dormindo
17         end
18
19   from executando    select
20         lIcononBeginStart? aviso;   to executando
21         [] lEndIcononEndStop? aviso;   to stop
22         [] lBeginShoes1onKeySelectionStopSetStart? aviso;   to eselect
23         [] lBeginShoes2onKeySelectionStopSetStart? aviso;   to eselect
24         end
25
26   from start   mstart! aviso;   to executando
27
28   from stop   mstop! aviso;   to dormindo
29
30   from eselect    mstop! aviso;   to dormindo
31
```

Code 84 Codigo Fiacre - Primeiro João

```
/*parametros: entradas, saidas*/
     component C_icon  [lIcononBeginStart:bool,lEndIcononEndStop:bool,
     lBeginShoes1onKeySelectionStopSetStart:bool,
     lBeginShoes2onKeySelectionStopSetStart:bool]
     (&systemLanguage:vet,&keyCode:vet, &estatus:flag)  is

     port
     mstart:bool in [0,0],mstop:bool in [0,0]

     par * in
 p_icon[mstart,mstop] (&systemLanguage,&keyCode, &estatus)
 || glue_icon[mstart,mstop,lIcononBeginStart,lEndIcononEndStop,
 lBeginShoes1onKeySelectionStopSetStart, lBeginShoes2onKeySelectionStopSetStart]

     end

 /*****************************************************************
 ******** PROCESSO, GLUE E COMPONENTE DA MIDIA shoes
 *****************************************************************/
 /*parametros: entradas, saidas*/
     process p_shoes  [mstart:bool,mstop:bool]   is

     states dormindo, avisandoFim, est1

     var aviso:bool:=false

     init to dormindo

     from dormindo   mstart? aviso;   to est1

     from est1  wait[14,14];  to avisandoFim

     from avisandoFim   mstop! aviso;   to dormindo

```

Code 85 Codigo Fiacre - Primeiro João

```
/*parametros: saidas, entradas*/
    process glue_shoes  [mstart:bool,mstop:bool,
    lBeginShoes1onKeySelectionStopSetStart:bool,
    lBeginShoes2onKeySelectionStopSetStart:bool] is

    states dormindo, est1, start

    var aviso:bool:=false

    init to dormindo

    from dormindo    select
    lBeginShoes1onKeySelectionStopSetStart? aviso;   to start
    []      lBeginShoes2onKeySelectionStopSetStart? aviso;   to start
    []      mstop? aviso;   to dormindo
    end

    from est1    select
    lBeginShoes1onKeySelectionStopSetStart? aviso;   to est1
    []      lBeginShoes2onKeySelectionStopSetStart? aviso;   to est1
    []      mstop? aviso;   to dormindo
    end

    from start   mstart! aviso;   to est1

/*parametros: entradas, saidas*/
    component C_shoes  [lBeginShoes1onKeySelectionStopSetStart:bool,
    lBeginShoes2onKeySelectionStopSetStart:bool]   is

    port
    mstart:bool in [0,0],mstop:bool in [0,0]

    par * in
p_shoes[mstart,mstop]
|| glue_shoes[mstart,mstop,lBeginShoes1onKeySelectionStopSetStart,
lBeginShoes2onKeySelectionStopSetStart]

    end
```

Code 86 Codigo Fiacre - Primeiro João

```
/*****************************************************************
******** PROCESSO, GLUE E COMPONENTE DA MIDIA background
*****************************************************************/
    process p_background  [mstart:bool,mstop:bool]   is
    states dormindo, est1

    var aviso:bool:=false

    init to dormindo

    from dormindo   mstart? aviso;   to est1
    from est1   mstop? aviso;   to dormindo

    process glue_background  [mstart:bool,mstop:bool,lBackgroundonBeginStart:bool,
    lEndBackgroundonEndStop:bool] is

    states dormindo, est1, start, stop
    var aviso:bool:=false

    init to dormindo

    from dormindo    select
    lBackgroundonBeginStart? aviso;   to start
    []      lEndBackgroundonEndStop? aviso;   to dormindo
    end

    from est1    select
    lBackgroundonBeginStart? aviso;   to est1
    []      lEndBackgroundonEndStop? aviso;   to stop
    end

    from start   mstart! aviso;   to est1

    from stop   select
    mstop! aviso;   to dormindo
    []      lEndBackgroundonEndStop? aviso;   to stop
    end

```

Code 87 Codigo Fiacre - Primeiro João

```
/*parametros: entradas, saidas*/
    component C_background  [lBackgroundonBeginStart:bool,
    lEndBackgroundonEndStop:bool]   is

    port
    mstart:bool in [0,0],mstop:bool in [0,0]

    par * in
p_background[mstart,mstop]
|| glue_background[mstart,mstop,lBackgroundonBeginStart,
lEndBackgroundonEndStop]

    end

/*************************************************************************/
process Oglue_040_shoes_backgroundA [mstartA:bool,mstopA:bool,
lBeginShoes1onKeySelectionStopSetStart:bool,
lBeginShoes2onKeySelectionStopSetStart:bool] is
 states idle, idle2, idle3, strA, endA
     var aviso:bool:=false

    init to idle

    from idle   select
lBeginShoes1onKeySelectionStopSetStart? aviso;      to strA
   []  lBeginShoes2onKeySelectionStopSetStart? aviso;      to strA
 end

    from idle2   select
lBeginShoes1onKeySelectionStopSetStart? aviso;      to idle2
   []  lBeginShoes2onKeySelectionStopSetStart? aviso;      to idle2
   end

    from idle3   select
lBeginShoes1onKeySelectionStopSetStart? aviso;      to idle3
   []  lBeginShoes2onKeySelectionStopSetStart? aviso;      to idle3
   end
     from strA   mstartA!aviso;  to idle2
    from endA   mstopA!aviso;  to idle3

```

Code 88 Codigo Fiacre - Primeiro João

```
1
2  process Oglue_040_shoes_backgroundB [mstartB:bool,mstopB:bool,
3  lBackgroundonBeginStart:bool,lEndBackgroundonEndStop:bool] is
4
5  states idle, idle2, idle3, strB, endB
6
7      var aviso:bool:=false
8
9      init to idle
10
11     from idle   select
12 lBackgroundonBeginStart? aviso;      to strB
13
14     []  lEndBackgroundonEndStop? aviso;      to idle
15
16  end
17
18     from idle2   select
19 lBackgroundonBeginStart? aviso;      to idle2
20
21
22     []  lEndBackgroundonEndStop? aviso;      to endB
23
24  end
25
26     from idle3   select
27 lBackgroundonBeginStart? aviso;      to idle3
28
29     []  lEndBackgroundonEndStop? aviso;      to idle3
30
31  end
32
33
34     from strB   mstartB!aviso;  to idle2
35     from endB   mstopB!aviso;  to idle3
36
```

Code 89 Codigo Fiacre - Primeiro João

```
component C_OBS_INTER_O40_shoes_background_5
[lBeginShoes1onKeySelectionStopSetStart:bool,
lBeginShoes2onKeySelectionStopSetStart:bool,
lBackgroundonBeginStart:bool,lEndBackgroundonEndStop:bool] is
     port
     mstartA:bool in [0,0],mstopA:bool in [0,0],
     mstartB:bool in [0,0],mstopB:bool in [0,0]

 par * in
     Oglue_O40_shoes_backgroundA [mstartA,mstopA,
     lBeginShoes1onKeySelectionStopSetStart,
     lBeginShoes2onKeySelectionStopSetStart]
     || Oglue_O40_shoes_backgroundB [mstartB,mstopB,
     lBackgroundonBeginStart,lEndBackgroundonEndStop]
     || OB_O40_shoes_background_5 [mstartA,mstopA,mstartB,mstopB]
 end

 /*****************************************************************
 ******** PROCESSO RELOGIO
 *****************************************************************/

     process p_relogio[LcontroleclickCcontroleclick:bool]   is
     /*sai start*/

     states dormindo, tic, tac

     var aviso:bool:=false

     init to dormindo

     from dormindo   LcontroleclickCcontroleclick? aviso;   to tic

     from tic   wait[1,1];    to tac
     from tac   wait[1,1];    to tic

```

Code 90 Codigo Fiacre - Primeiro João

```
1
2  /*****************************************************************
3  ******** PROCESSO CONTROLE
4  *****************************************************************/
5  /*parametros: saida*/
6      process p_controleStart [LcontroleclickCcontroleclick:bool]
7      (&estatus:flag) is
8
9  states final, est1
10     var aviso:bool:=false
11
12     init to est1
13
14     from est1 LcontroleclickCcontroleclick! aviso;    to final
15
16     process p_controleStartSelect [lBeginShoes1onKeySelectionStopSetStart:bool,
17     lBeginShoes2onKeySelectionStopSetStart:bool]  (&estatus:flag) is
18
19     states botao, volta
20
21     var aviso:bool:=false
22
23     init to botao
24
25     from botao select
26 if (estatus[1]=true) then  lBeginShoes1onKeySelectionStopSetStart! aviso;
27 to volta
28 else wait[0,0]; to volta end
29 [] if (estatus[1]=true) then  lBeginShoes2onKeySelectionStopSetStart! aviso;
30 to volta
31 else wait[0,0]; to volta end
32
33 [] wait[1,1]; to volta
34 [] wait[0,0]; to volta
35     end
36     from volta  wait[1,1]; to botao
37
38
```

Code 91 Codigo Fiacre - Primeiro João

```
1
2   component  main is
3   /*************************************
4     Estrutura do contexto main
5   *************************************/
6
7   var
8       keyCode:vet :=1
9   ,    systemLanguage:vet :=1
10    ,estatus:flag := [false,false]
11
12   port
13  lEndIcononEndStop:bool in [0,0]
14  ,lBeginShoes1onKeySelectionStopSetStart:bool in [0,0]
15  ,lEndBackgroundonEndStop:bool in [0,0]
16  ,LcontroleclickCcontroleclick:bool in [0,0]
17  ,lBackgroundonBeginStart:bool in [0,0]
18  ,lBeginShoes2onKeySelectionStopSetStart:bool in [0,0]
19  ,lIcononBeginStart:bool in [0,0]
20
21
22  par * in
23  C_animation[LcontroleclickCcontroleclick,lIcononBeginStart,
24  lEndIcononEndStop, lBackgroundonBeginStart,lEndBackgroundonEndStop]
25  ||   C_icon[lIcononBeginStart,lEndIcononEndStop,
26  lBeginShoes1onKeySelectionStopSetStart,
27  lBeginShoes2onKeySelectionStopSetStart]
28  (&systemLanguage,&keyCode, &estatus)
29  ||   C_shoes[lBeginShoes1onKeySelectionStopSetStart,
30  lBeginShoes2onKeySelectionStopSetStart]
31  ||   C_background[lBackgroundonBeginStart,lEndBackgroundonEndStop]
32  ||   p_controleStartSelect[lBeginShoes1onKeySelectionStopSetStart,
33  lBeginShoes2onKeySelectionStopSetStart] (&estatus)
34  ||   p_controleStart[LcontroleclickCcontroleclick] (&estatus)
35  ||   p_relogio[LcontroleclickCcontroleclick]
36  || C_OBS_INTER_040_shoes_background_5
37  [lBeginShoes1onKeySelectionStopSetStart,
38  lBeginShoes2onKeySelectionStopSetStart,
39  lBackgroundonBeginStart,lEndBackgroundonEndStop]
40
41  end
42
43    main
44  /*************************************
45  *************************************/
46
```

APÊNDICE G – Lógica Temporal

A lógica temporal é um formalismo lógico feito para ser aplicado em sistemas que envolvem a noção de ordem no tempo (BERARD et al., 2010).

Este anexo apresenta uma visão geral sobre Lógica Temporal e as linguagens LTL e CTL.

G.1 VISÃO GLOBAL

Em um processo de validação de um sistema, pode-se usar lógica temporal para especificar as propriedades temporais que se deseja garantir durante sua execução. Tal execução gera uma sequência de estados que podem ser representados por um autômato, como aquele da Figura G.1.1.

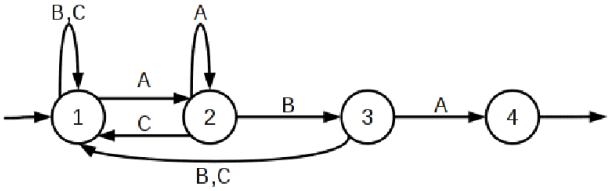


Figura G.1.1 – Autômato substring ABA

A lógica temporal usa proposições atômicas para fazer afirmações sobre os estados do autômato que representa a execução do sistema. Tais proposições são relações elementares, que possuem um valor definido em um dado estado.

Na lógica temporal, um autômato é uma quíntupla $A = (Q; E; T; Q_0; l)$ onde:

- Q é um conjunto de estados do autômato;
- E é um conjunto de etiquetas das transições existentes no autômato;
- $T \subseteq Q \times E \times Q$ é o conjunto das transições;
- Q_0 é o estado inicial;
- l é uma aplicação que associa a cada estado q de Q o conjunto finito de propriedades a ser verificado.

O autômato da Figura G.1.1 verifica se a *substring ABC* ocorre; algumas propriedades para esse autômato estão na Tabela G.1.1.

Tabela G.1.1 – Propriedades

Propriedades	
$P1$	a última letra digitada foi A
$P2$	a última letra digitada foi B
$P3$	o estado em execução aterior era o 2
$P4$	o estado em execução aterior era o 3

O autômato da Figura G.1.1 que verifica as propriedades apresentadas na Tabela G.1.1, pode ser representado por $A = (Q, E, T, q_0, l)$, onde:
$Q = \{1, 2, 3, 4\}$
$E = \{A, B, C\}$
$T = \{(1, A, 2), (1, B, 1), (1, C, 1), (2, A, 2), (2, B, 3), (2, C, 1), (3, A, 4), (3, B, 1), (3, C, 1)\}$

$$l = \begin{cases} 1 \longmapsto \oslash \\ 2 \longmapsto \{P1\} \\ 3 \longmapsto \{P2, P3\} \\ 4 \longmapsto \{P1, P4\} \end{cases}$$

G.1.1 A Linguagem da Lógica Temporal

A lógica temporal é composta por três tipos de estruturas que podem ser usadas para descrever suas propriedades: operadores lógicos, operadores temporais e quantificadores de caminhos.

Os operadores lógicos são compostos pelas constantes *true* e *false* e pelos operadores negação ($\neg$), conjunção ($\vee$), disjunção ($\wedge$), implicação ($\Rightarrow$) e dupla-implicação ($\Leftrightarrow$). Por exemplo $P \Rightarrow \neg(Q \wedge S)$.

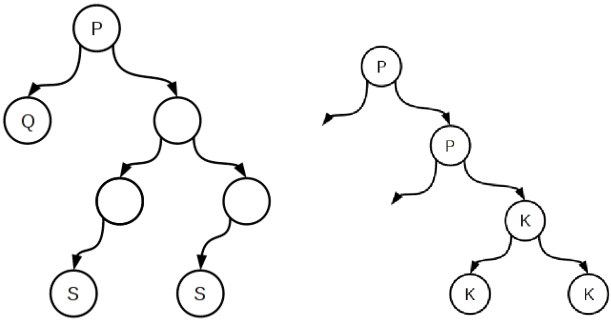
Os operadores temporais permitem que as fórmulas lógicas referentes à descrição das propriedades considerem também a evolução (sequência) dos estados. Na Tabela G.1.2 são descritos os principais operadores temporais.

Tabela G.1.2 – Operadores Temporais

Operadores Temporais		
Next (**X ou ()**)	**X** P	P é satisfeita no próximo estado
Future (**F** ou $\Diamond$)	**F** P	P é satisfeita em algum estado futuro
Globally (**G** ou $\Box$)	**G** P	P é satisfeita em todos os estados futuros
Until (**U**)	$P1$ **U** $P2$	$P1$ é satisfeita até que a $P2$ seja satisfeita

Um exemplo do uso de operadores lógicos e temporais é $P \Rightarrow (\mathbf{X}Q \vee \mathbf{F}S)$, onde a validação da propriedade P implica que no próximo passo a propriedade Q será verdadeira ou no futuro a propriedade S será verdadeira, como pode ser visto na Figura G.1.2.a.

Outro exemplo de uso de operadores lógico temporais é a propriedade P **U** **G**K, que significa que P será verdadeiro, até que K seja verdadeiro para todos os futuros, conforme apresentado na Figura G.1.2.b.



(a) Propriedade $P \Rightarrow (\mathbf{X}Q \vee \mathbf{F}S)$ (b) Propriedade P **U** **G**K

Figura G.1.2 – Autômatos - Operadores Temporais

Os quantificadores de caminho permitem verificar se uma proposição é válida em alguma das sequências possíveis (**E**) ou em todas as sequências (**A**). Na Tabela G.1.3 são mostradas as descrições dos quantificadores de caminhos.

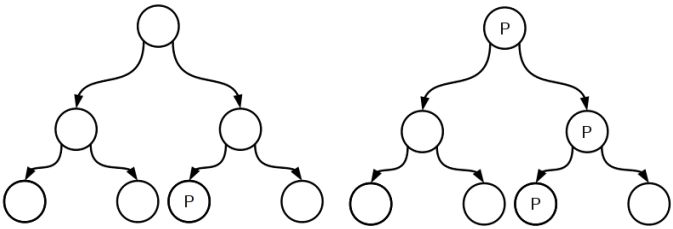
Tabela G.1.3 – Quantificadores de Caminhos

Quantificadores de Caminhos		
Existe (**E**)	**E** ϕ	Em alguma das execuções a partir do estado atual, a propriedade ϕ é satisfeita
All (**A**)	**A** ϕ	Todas as execuções a partir do estado atual satisfazem a propriedade ϕ

O exemplo da Figura G.1.3.a possui a propriedade **EF**P que verifica se existe um caminho que, no futuro, resultará P como verdadeiro (apenas um caminho é suficiente para tal propriedade ser satisfeita).

Já o exemplo da Figura G.1.3.b possui a propriedade **EG**P que verifica se existe um caminho no qual todos os tempos terão P como verdadeira (a despeito de um caminho ser suficiente, é necessário que ele tenha a propri-

edade *P* satisfeita em todos os nós do autômato partindo de sua raiz).

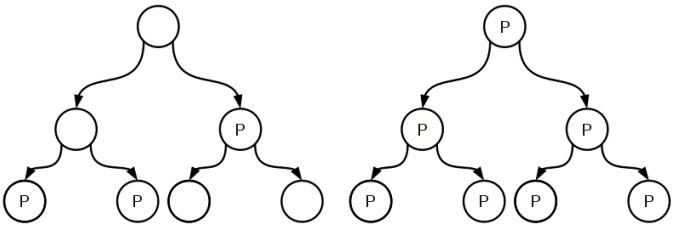


(a) Propriedade **EF***P* (b) Propriedade **EG***P*

Figura G.1.3 – Autômatos - Quantificadores de Caminhos “E”

O exemplo da Figura G.1.4.a mostra a propriedade **AF***P* que verifica se em todos os caminhos, no futuro, *P* será verdadeira. Essa propriedade exige que, em todos os caminhos partindo da raiz, a propriedade *P* deve ser satisfeita em algum momento. Quando, em um caminho cuja propriedade *P* ainda não tiver sido satisfeita, houver uma bifurcação, em ambas as direções da bifurcação a propriedade *P* deverá ser satisfeita em algum momento.

O exemplo apresentado na Figura G.1.4.b apresenta a propriedade **AG***P* que verifica se em todos os caminhos e em todos os momentos *P* é verdadeira.



(a) Propriedade **AF***P* (b) Propriedade **AG***P*

Figura G.1.4 – Autômatos - Quantificadores de Caminhos “A”

LTL e CTL: Duas Lógicas Temporais

A lógica temporal não incorpora explicitamente a ideia de tempo, existindo dois modos básicos para fazê-lo.

O primeiro modo de expressar o tempo é como um conjunto de sequências infinitas que começam no estado inicial do sistema (LTL - *Linear Temporal Logic*). Nesse modelo não são adotados os quantificadores de caminho.

O segundo modo é através de uma "árvore de execução" de profundidade ilimitada, onde cada ramificação da árvore corresponde a uma sequência alternativa de estados e a raiz representa o estado inicial (CTL - *Computation Tree Logic*). Nesse modelo, o operador temporal tem que ser imediatamente precedido por um quantificador de caminho.

As propriedades na lógica temporal podem ser classificadas nas seguintes categorias, conforme o objetivo a ser verificado:

- Alcançabilidade *(Reachability)*: certa situação (desejada) pode ser atingida (é alcançável). Em CTL, a propriedade de alcançabilidade estabelece que alguma situação particular pode ser atingida. Suas fórmulas do tipo **EF***P*, indicam que existe uma execução tal que no futuro a propriedade P será alcançável. A lógica CTL permite expressar alcançabilidade enquanto a lógica LTL permite expressar apenas a não alcançabilidade;
- Segurança *(Safety)*: sob certas condições, certo evento (indesejável) nunca ocorrerá. Em lógica CTL, a propriedade de segurança tem a forma **AG**$\neg$, que indica que em todos os caminhos e em todos os momentos algo não ocorrerá. Já para lógica LTL, por não permitir representar diferentes caminhos, a fórmula fica na forma **G**$\neg$ (em todos os momentos algo não ocorre);
- Vivacidade *(Liveness)*: sob certas condições, certa situação acabará por acontecer. Em CTL, pode-se expressar tal situação por meio da fórmula $\mathbf{AG}(solicito \Rightarrow \mathbf{AF}satisfeita)$. Já em LTL, isso é feito por meio de $\mathbf{G}(solicito \Rightarrow \mathbf{F}satisfeita)$;
- Ausência de Bloqueio *(Deadlock-Freeness)*: o sistema nunca entra em uma situação em que o progresso não é possível. Em CTL, tal propriedade é genericamente representada pela fórmula **AGEX** *true*, a qual indica que em todos os caminhos e em todos os momentos existirá um próximo estado. Quando o *model-checker* não suporta esse tipo de fórmula, deve ser criada uma fórmula específica para o modelo que verifique tal situação; e
- Justiça *(Fairness)*: sob certas condições, algo acontecerá infinitamente. Os operadores $\overset{\infty}{\mathbf{F}}$ (representação de **GF**) e $\overset{\infty}{\mathbf{G}}$ (representação de **FG**) podem ser utilizados para expressar infinitamente muitas vezes, mas eles não existem na lógica CTL.

APÊNDICE H – Observadores

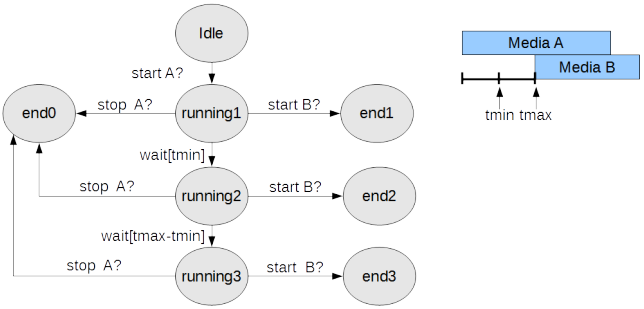


Figura H.0.5 – Observador

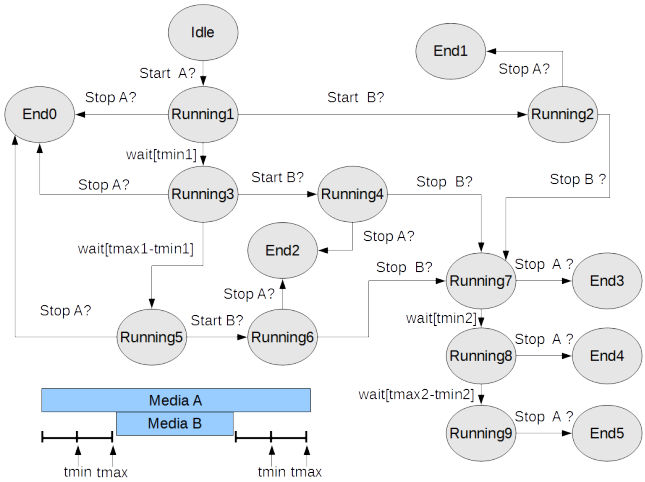


Figura H.0.6 – Observador

APÊNDICE I – O Ambiente: Manual de Instalação

Este manual de uso está dividido em três partes. Inicialmente apresenta-se uma introdução, onde é brevemente descrita a estrutura do ambiente. À seguir, apresenta-se os passos necessários para a instalação do ambiente, bem como das ferramentas necessárias para seu uso. Por fim, a terceira parte visa guiar a utilização do ambiente desenvolvido.

I.1 INTRODUÇÃO

O ambiente de verificação desenvolvido tem por objetivo permitir o uso de verificação formal na construção de aplicações hipermídia. Para realizar tal objetivo, usa-se diferentes tipos de ferramentas como: compiladores, ferramenta de especificação de propriedades, ferramentas de transformação e ferramentas de verificação formal. A Figura I.1.1 apresenta a estrutura do ambiente indicando o posicionamento de cada ferramenta no processo de verificação.

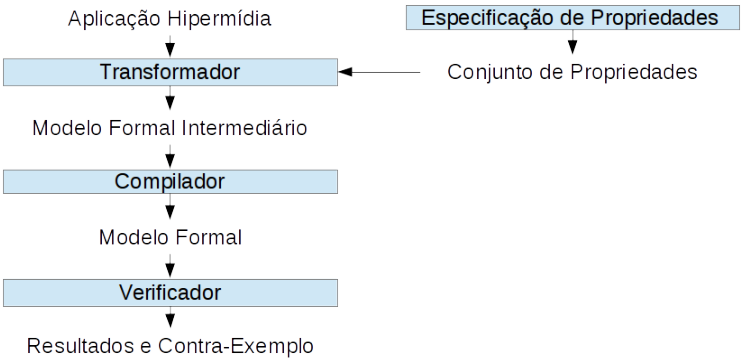


Figura I.1.1 – Estrutura do Ambiente

A próxima seção descreve os passos necessários para a instalação desse ambiente e das ferramentas necessárias.

I.2 INSTALAÇÃO

Para o uso do ambiente proposto, faz-se necessária a instalação das ferramentas adotadas. Após descompactar o arquivo contendo a aplicação, basta executar o *script* "./install.sh" para realizar a instalação completa. Algumas partes dessa instalação solicitarão a senha de administrador do sistema

operacional.

Caso o usuário queira saber mais detalhes sobre a instalação, na sequencia dessa seção serão descritos os passos da instalação de maneira mais detalhada possível, sempre considerando seu uso no sistema operacional Linux distribuição Debian ou baseada em Debian. É valido destacar que o uso do ambiente não se restringe apenas ao Linux Debian, podendo ser adotado também em outras distribuições Linux e Windows, mas nesses casos a instalação deverá ser manual.

I.2.1 Java

O ambiente foi desenvolvido utilizando o Java na versão 7. Experimentos realizado com a versão 8 do Java apresentaram problemas de compatibilidade, logo aconselha-se o uso da versão 7. Para instalar o Java na versão 7, em um terminal Linux Debian siga os seguintes passos:

(1) Remova outras versões do java:
*sudo apt-get purge openjdk**
*sudo apt-get remove oracle-java**

(2) Adicione o repositório do Oracle Java:
sudo add-apt-repository ppa:webupd8team/java

(3) Atualize os repositórios:
sudo apt-get update

(4) Instale o Java 7:
sudo apt-get install oracle-java7-installer

I.2.2 Compilador C

As ferramentas de verificação formal fazem uso de um compilador C, logo você precisa instala-lo. Em um terminal Linux Debian, efetue o seguinte comando:

sudo apt-get install build-essential

Embora o compilador *gcc* já esteja instalado por padrão em distribuições Linux, as principais bibliotecas não estão. Logo, faz-se necessário a execução do comando acima, que instalará tais bibliotecas.

I.2.3 Ferramentas de Verificação Formal

No ambiente desenvolvido usa-se o compilador Frac e o verificador TINA.

I.2.3.1 Compilador Frac

O compilador Frac pode ser obtido gratuitamente no seguinte endereço:

http://projects.laas.fr/fiacre/download.php

Após baixar o arquivo, e descomprimi-lo, deve-se editar o arquivo ”Makefile” atualizando o endereço contido na variável de ambiente FRACLIB. Essa variável deve conter o caminho absoluto para o diretório Lib do Frac. Para obter esse caminho entre no diretório Lib do Frac e execute o comando *pwd*.

I.2.3.2 Verificador TINA

O verificador TINA pode ser obtido gratuitamente no seguinte endereço:

http://projects.laas.fr/tina/download.php

Após baixar o arquivo, basta descomprimi-lo, pois o verificador TINA não requer configuração de variáveis de ambiente.

I.2.4 O Ambiente

Para instalar o ambiente de verificação desenvolvido neste trabalho, inicialmente descomprima o arquivo de instalação. Edite o arquivo “conf.txt” atualizando o endereço contido nas variável de ambiente ”fiacre” e ”tina”. Essas variáveis devem conter, respectivamente, o caminho absoluto para os arquivos executáveis do Frac e do Tina. Para obter o caminho dos executáveis do Frac entre no diretório do ”Frac” e execute o comando *pwd*. Para obter o caminho dos executáveis do TINA entre no diretório do ”Tina/bin” e execute o comando *pwd*.