UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uma Extensão da Visão Estrutural do NCL Composer para Integração de Código Imperativo

Thales Pordeus Ferreira

JOÃO PESSOA-PB
Maio-2013

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uma Extensão da Visão Estrutural do NCL Composer para Integração de Código Imperativo

Thales Pordeus Ferreira

JOÃO PESSOA-PB
Maio-2013

Thales Pordeus Ferreira

Uma Extensão da Visão Estrutural do NCL Composer para Integração de Código Imperativo

DISSERTAÇÃO APRESENTADA AO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DA PARAÍBA, COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE EM INFORMÁTICA (SISTEMAS DE COMPUTAÇÃO).

Orientador: Prof. Dr. Guido L. S. Filho
Coorientador: Prof. Dr. Raoni Kulesza

JOÃO PESSOA-PB
Maio-2013

F383u Ferreira, Thales Pordeus.
Uma extensão da visão estrutural do NCL Composer para integração de código imperativo / Thales Pordeus Ferreira.- João Pessoa, 2013.
126f. : il.
Orientador: Guido L. S. Filho
Coorientador: Raoni Kulesza
Dissertação (Mestrado) - UFPB/CCEN
1. Informática. 2. Sistemas de computação. 3. Aplicações NCL. 4. NCL Composer - visão estrutural. 5. Ginga-NCL.

UFPB/BC CDU: 004(043)

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de **Thales Pordeus Ferreira**, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 24 de Julho de 2013.

Ao vigésimo quarto dia do mês de Julho do ano dois mil e treze, às dez horas, no laboratório 2 da Escola Superior de Redes - Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Computação Distribuída*", o Sr. **Thales Pordeus Ferreira**. A comissão examinadora foi composta pelos professores doutores: GUIDO LEMOS DE SOUZA FILHO (PPGI-UFPB), Orientador e Presidente da Banca, RAONI KULESZA (UFPB), examinador externo ao programa e AQUILES MEDEIROS FILGUEIRA BURLAMAQUI (UFRN) como examinador externo. Dando início aos trabalhos, o professor GUIDO LEMOS DE SOUZA FILHO cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "*Uma Extensão da Visão Estrutural do NCL Composer para Integração de Código Imperativo*". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "*Aprovado*". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, Micherlon Xavier Bezerra, Assistente em Administração, servindo de secretário, lavrei a presente ata que vai assinada por mim e pelos membros da Banca Examinadora. João Pessoa, 24 de Julho de 2013.

Micherlon Xavier Bezerra

Prof. Dr. Guido Lemos de Souza Filho
Orientador (PPGI-UFPB)

Prof. Dr. Raoni Kulesza
Examinador Externo ao Programa (UFPB)

Prof. Dr. Aquiles Medeiros Filgueira Burlamaqui
Examinador Externo (UFRN)

AGRADECIMENTOS

Gostaria de agradecer àquelas pessoas que me apoiaram e me auxiliaram no decorrer do mestrado, assim como na produção nesta dissertação.

A Deus, aos meus pais e ao meu irmão, obrigado pelo apoio, incentivo, amizade, animação e tudo mais.

Agradeço a minha namorada, Cristiane Sousa, por ter ficado ao meu lado me transmitindo força e determinação para que eu conseguisse concluir esta etapa.

Agradeço aos meus orientadores Guido Lemos e Raoni Kulesza, pela calma, pelas conversas e soluções das dúvidas me incentivando e colaborando desde o início desse trabalho. Ao pessoal do laboratório LAVID, em especial ao Eduardo Santana pelas discussões produtivas.

Agradeço, aos funcionários e a todos os professores do curso, com os quais tive a oportunidade de conviver.

E a todos aqueles que direta ou indiretamente contribuíram para a reflexão e realização deste trabalho, os meus sinceros agradecimentos.

RESUMO

Os requisitos das aplicações NCL, como a necessidade de conteúdo dinâmico resultado de processamento, tornam o seu desenvolvimento um desafio. Essas aplicações são definidas com uma parte declarativa e, adicionalmente, uma parte imperativa que relaciona os objetos de mídia com algum processamento de operações complexas (por exemplo, uma lógica descrita num *script* programado na linguagem Lua). Dentro do desenvolvimento de aplicações NCL, uma opção adotada pelos autores de documento é utilizar a visão estrutural do NCL Composer para relacionar de forma visual os objetos de mídia com os objetos de mídia imperativos. Atualmente, apesar do NCL Composer suportar o relacionamento com os objetos de mídia imperativos, é possível identificar uma ineficiência no tocante à integração de forma rápida e simples do código imperativo em uma aplicação NCL. O objetivo deste trabalho é propor uma extensão para a visão estrutural que permita melhor integrar objetos de mídia e código imperativo, de forma a diminuir o tempo de uso do conteúdo presente nas mídias imperativas. O trabalho traz também uma avaliação do impacto na produtividade do desenvolvimento de aplicações compatíveis com a especificação Ginga-NCL.

ABSTRACT

NCL application requirements such as the need for dynamic content processing, makes their development challenge. These applications are defined with a declarative part and additionally a portion imperative that relates the media objects to a operation processing complex (for example, a programmed logic described in a script language Lua). Within the development of NCL applications, an option taken by the authors of the document is to use the structural view of the NCL Composer to relate visually the media objects with media objects imperatives. Currently, despite the NCL Composer support the relationship with the media objects imperatives, it is possible to identify an inefficiency associated to integrate fast and simple imperative code in an application NCL. The objective of this work is to propose an extension to the structural view that better integrate media objects and imperative code, in order to decrease the time to use the content in this media imperative. The paper presents an evaluation of the impact on productivity of application development compatible with the specification Ginga-NCL.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

As mudanças provocadas pela substituição do paradigma analógico pelo digital viabilizaram as plataformas de TV Digital (TVD), a TV Digital Interativa (TVDi) e as TVs Conectadas (ALFONSI, 2005), que possibilitam ao telespectador vivenciar experiências de interação próximas àquelas já difundidas em aplicações presentes na *Web*. Nesse cenário, as formas já presentes de interação com os programas de TV (como votação, envio de mensagens, entre outras) podem ser aprimoradas com o uso das aplicações sobrepostas aos programas de TV (BACHMAYER, LUGMAYR e KOTSIS, 2010). A interatividade realizada pelas aplicações interativas facilita e torna mais transparente a interação do telespectador com o programa de TV.

Uma aplicação multimídia é definida como um software que possui interface com o usuário (UI – *User Interface*) com pelo menos um objeto de mídia (áudio, vídeo, imagens, animações, gráficos 2D ou 3D e etc.). Algumas aplicações, principalmente aquelas que produzem muito conteúdo dinâmico (SOARES, RODRIGUES, *et al.*, 2010), demandam relação entre os objetos de mídia e um processamento de operações complexas (lógica da aplicação), o qual é especificado numa linguagem imperativa (software). Por exemplo, deve ser possível incluir, remover, alterar os objetos de mídia a partir de uma lógica da aplicação em tempo de execução e/ou gerar eventos dos objetos de mídia para a lógica imperativa (PLEUB, 2005). Dessa forma, tais aplicações demandam novos requisitos para seu desenvolvimento.

Um cenário atual e emergente que envolve esse tipo de integração (entre objetos de mídia e lógica imperativa) é a utilização de serviços Web em aplicações NCL. Tais elementos oferecem, por meio de algum código imperativo, funcionalidades que permitem o consumo e/ou produção de informação para qualquer tipo de aplicação. As aplicações que utilizam serviços Web, a exemplo das aplicações *web* "*Mashups*", estão se tornando cada vez mais

comuns e o reuso desses elementos em aplicações multimídia também se faz cada vez mais necessário.

Atualmente, para tratar a complexidade do desenvolvimento de aplicações multimídia é comum utilizar linguagens específicas de domínio, que definem uma sintaxe próxima ao domínio das aplicações multimídia para tratar aspectos particulares da autoria. Por exemplo, para aplicações no padrão brasileiro de TV Digital é empregada a linguagem declarativa NCL (*Nested Context Language*) (SOARES e RODRIGUES, 2006), que tem como principal foco descrever em um nível maior de abstração a estrutura espacial e relação temporal dos objetos de mídia das aplicações. A principal justificativa é utilizar uma especificação que tenha uma notação simplificada, sendo mais fácil de escrever e entender. Para o processamento de operações complexas, é comum utilizar linguagens imperativas (normalmente baseadas em linguagens de *scripting*) para complementar as funcionalidades da parte declarativa da aplicação através da edição do estado de algum objeto de mídia ou processamento lógico (por exemplo, envio e recebimento de informações para e de fontes externas). Nesse caso, a integração entre a parte declarativa e a parte imperativa se dá através do tratamento dos módulos de *scritps* também como objetos de mídia.

Complementar ao uso dessa abordagem mista, usando linguagem declarativa e imperativa, é comum utilizar ferramentas para automatizar algumas etapas de desenvolvimento com o objetivo de reduzir a quantidade de código escrita por programadores e, assim, aumentar a produtividade. O desenvolvimento sendo feito inteiramente por uma linguagem textual (isto é, sem a ajuda de ferramentas visuais) está relacionada basicamente a duas tarefas: (i) escrita da sintaxe do elemento a ser utilizado e (ii) definição da forma que esse elemento é usado (isto é, seus atributos preenchidos com valores). O trabalho (AZEVEDO, NETO, *et al.*, 2011) automatiza parte dessas tarefas, pois tais ferramentas, apesar de resolverem parte do problema (acelerarem a escrita da sintaxe), não abordam a segunda tarefa, que é a edição rápida dos valores dos atributos.

Uma alternativa é usar as ferramentas de autoria visuais, que oferecem técnicas gráficas para proporcionar dois objetivos: (1) aumentar a produtividade do desenvolvimento, automatizando a edição rápida de valores dos atributos e (2) tornar simples e intuitivo a construção da aplicação (KASKALIS, TZIDAMIS e MARGARITIS, 2007), pois o autor nem sempre tem experiência e geralmente é uma pessoa não técnica. O sistema HyperProp (SOARES, RODRIGUES e SAADE, 2000), que é baseado em múltiplas visões, foi a primeira

ferramenta para autoria visual de documentos hipermídia no modelo NCM[1]. No HyperProp quatro visões (estrutural, temporal, espacial e textual), trabalhando de forma integrada, foram propostas para modelar cada uma um aspecto diferente das aplicações.

Dentre as visões, a visão estrutural merece destaque por modelar o comportamento da aplicação, apresentando graficamente como os elementos de mídias estão relacionados por meio de estruturas do tipo elo[2]. Em uma notação visual semelhante a um grafo, os elementos de mídia e os elos são criados para especificar os aspectos de sincronismo temporal e a interatividade da aplicação (SOARES, RODRIGUES e SAADE, 2000).

A visão estrutural ganhou destaque na comunidade NCL por ser uma notação gráfica que abstrai a linguagem NCL facilitando o entendimento do documento, como visto nos exemplos em (SOARES e BARBOSA, 2009). Essa visão cumpre um papel importante em descrever quando e como os elementos de mídia são apresentados, isto é, iniciados, parados, pausados, etc. A proposta inicial presente no sistema HyperProp envolve apenas os elementos de mídia de conteúdo, por exemplo, áudio, vídeo, imagem, texto, etc. Como um avanço à proposta original e a fim de se adequar ao contexto da NCL e da TV Digital, essa ferramenta passou por diversas modificações, o que culminou na criação do NCL Composer (GUIMARÃES, COSTA e SOARES, 2007).

O NCL Composer (TELEMÍDIA, 2013) (LIMA, AZEVEDO, *et al.*, 2010) complementa a visão estrutural com suporte aos elementos de mídia imperativos. Logo, ele além de implementar a visão estrutural original, adiciona suporte para integrar elementos declarativos juntamente com elementos imperativos. Realizar o relacionamento com elementos de mídia imperativos possibilita a integração do código imperativo dentro da aplicação NCL.

Apesar do Composer efetivamente realizar o relacionamento entre os elementos de mídia declarativos e imperativos, foram identificados problemas que tornam a integração desses elementos custosa e complexa para ser feita pelo autor de documentos. Sendo assim, a visão estrutural, apesar de permitir edição visual da parte declarativa das aplicações, não contempla de maneira simples e rápida a parte imperativa e, nem sua integração com a parte declarativa.

[1] O NCM (*Nested Context Model*) é um modelo de contextos aninhados para especificação de aplicações hipermídias do qual foi originado a linguagem NCL.

[2] O elemento elo na NCL relaciona elementos de mídia por meio de uma relação causal, em que uma *condição* deve ser satisfeita para que *ações* possam ser executadas.

Nesse sentido, é possível identificar uma lacuna que precisa ser preenchida no desenvolvimento de aplicações multimídia: ausência de notações e recursos de autoria na visão estrutural que permita lidar com a complexidade dos requisitos de aplicações que necessitam de linguagens declarativas e imperativas.

Por apresentar diferentes atividades, o processo de desenvolvimento de aplicações NCL deve considerar o envolvimento do autor de documentos da área de interface gráfica, do produtor de mídias e do programador responsável pelo código imperativo. O surgimento dessas aplicações trouxe para os envolvidos na produção de conteúdo para TV um contato com atividades de desenvolvimento de software (TAVARES e VEIGA, 2007). A utilização de código imperativo é complexa demais para um autor de documentos, necessitando às vezes de um profissional que tenha domínio em programação. Logo, a utilização de mídias imperativas em uma aplicação NCL por autores sem conhecimento em programação passa a ser um entrave.

Segundo (SOARES e BARBOSA, 2009), um requisito importante na integração de linguagens declarativas e imperativas é que haja o mínimo de intercalação entre seus domínios de modo a simplificar a divisão de tarefas entre equipes de profissionais técnicos e não-técnicos em linguagens de programação. O profissional não-técnico ao reusar uma mídia imperativa deve fazer de forma simples e rápida, de forma a não requerer conhecimento sobre a implementação interna do código imperativo.

1.1 Objetivos

Este trabalho tem como principal objetivo propor uma extensão da visão estrutural para tratar a integração de código imperativo de forma simples e rápida dentro de uma aplicação NCL/Lua. O intuito é aumentar a produtividade diminuindo o esforço durante a integração de código imperativo em uma aplicação qualquer. Como objetivo específico é proposto uma abordagem para reuso de serviços web utilizando o conceito elaborado aqui para as mídias imperativas dentro da visão estrutural.

A ideia principal é auxiliar o autor de documentos a reusar um conteúdo imperativo na forma de métodos e variáveis em aplicações de TVD. A extensão facilita a identificação do conteúdo presente na mídia sem que seja necessário o autor conhecer a linguagem imperativa. Alguns conceitos envolvendo a definição para usar o conteúdo imperativo são gerados automaticamente, tornando mais rápido o uso do código. O suporte aos serviços web pré-

existentes também é tratado. Espera-se que a ferramenta diminua o esforço de desenvolvimento da aplicação ganhando tempo em relação à visão estrutural original.

1.2 Contribuições

A principal contribuição deste trabalho é a inclusão de novos mecanismos na visão estrutural para melhorar a integração de conteúdo imperativo no tocante ao aumento da produtividade do desenvolvimento.

1.3 Metodologia

A metodologia de pesquisa adotada nesse trabalho consiste em três fases: (i) análise do problema; (ii) proposta e projeto da solução e (iii) avaliação da proposta.

A primeira fase teve como intuito analisar o problema da integração visual entre objetos de mídia e código imperativo do ponto de vista do autor de documentos. Devido aos poucos trabalhos existentes na literatura, a analise sucedeu basicamente pelas próprias experiências do autor deste trabalho na autoria visual de aplicações NCL utilizando a última implementação da visão estrutural feita pela ferramenta NCL Composer.

Esta etapa serviu para investigar, na prática, os problemas atuais do NCL Composer que impactam a autoria visual do documento NCL que inclui código imperativo com relação à simplicidade e rapidez. Com base nesses problemas foram elaborados os requisitos da nossa proposta, os quais foram utilizados para elaborar a extensão da visão estrutural existente. Outros trabalhos semelhantes ao nosso, mas em outro domínio, foram também pesquisados para servir de embasamento nos requisitos. Para tanto, a pesquisa foi dividida em duas etapas: (1) estudo sobre: o mecanismo da NCL para integração entre objetos de mídia e código imperativo e as situações na visão estrutural que demandam integração entre objetos de mídia NCL e código (2) elaboração dos requisitos para a rápida e simples integração de código imperativo na aplicação.

Na segunda fase, os requisitos e problemas da etapa anterior foram usados para projetar uma solução. O principal objetivo foi entender quais as tarefas e necessidades do autor de documentos que incluem elementos imperativos usando a visão estrutural no NCL Composer. Baseado nisso, foi possível propor extensões que possam tornar a autoria mais simples e rápida.

Na terceira fase, foi feito um estudo para investigar em que medida as extensões podem melhorar tempo de desenvolvimento de aplicações multimídia envolvendo lógica complexa em comparação a visão estrutural existente no NCL Composer. O objetivo foi avaliar se a extensão implantada na visão estrutural aumenta a produtividade, diminuindo o esforço durante a integração de código imperativo em uma aplicação NCL.

1.4 Organização da Dissertação

Esta dissertação está organizada em mais 7 capítulos. No capítulo 2 são apresentados os principais conceitos envolvidos neste trabalho relacionados à TV digital, ao desenvolvimento de aplicações para esta nova plataforma como também à linguagem NCL. O capítulo 3 empreende a revisão de literatura realizada para a visão estrutural e as principais ferramentas de autoria relacionadas. O capítulo 4 apresenta de forma mais detalhada os problemas existentes na visão estrutural que impactam na velocidade e simplicidade da integração de código imperativo. O capítulo 5 descreve a solução proposta para tratar os problemas detalhados no capítulo anterior e no capítulo 6 é apresentada a implementação. No capítulo 7 são expostos os estudos de caso e os resultados da avaliação da extensão. Já, o capítulo 8 discute as considerações finais com conclusões, as contribuições alcançadas e os possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 TV Digital Interativa

A TV Digital Interativa (TVDi) altera efetivamente a forma como a TV tradicional é consumida e produzida, introduzindo uma era em que o telespectador deixa de ser um elemento passivo no processo de comunicação, passando a ter condições de interferir no conteúdo enviado pela emissora de TV. O processo de digitalização torna possível a integração de vários serviços em uma onda de radiodifusão digital, sendo cada dispositivo receptor de sinal de TV uma via de acesso as informações oferecidas por meio de variadas infra-estruturas de redes digitais.

A interação coloca como requisitos para os aparelhos receptores de sinais televisivos a capacidade de processar e executar códigos que definem os elementos de um programa interativo. Podemos ter interatividade em várias plataformas receptoras de TV: aparelhos convencionais de TV, aparelhos de alta definição (HDTV – *High Definition Digital Television*), telas de computadores e celulares.

Para o desenvolvimento de sistemas interativos, é fundamental conhecer os níveis de interação que a infra-estrutura de transmissão proverá aos seus usuários (MORRIS e SMITH-CHAIGNEAU, 2005). Em sistema de TVD costuma-se separar em perfis os diferentes tipos de serviços oferecidos. Estes perfis possibilitam ter aplicações com diferentes capacidades e custos. Usando estes perfis, produtos podem ser direcionados para segmentos de mercados específicos de usuários ou para produtoras de conteúdo. Alguns padrões definem duas classes baseado-se no cenário de uso das aplicações:

- **Difusão aprimorada (*Enhanced Broadcast*)**: Possui apenas interação local. O conteúdo é transmitido unilateralmente para o receptor de uma só vez. A interação do usuário é restrita aos dados que ficam armazenados no receptor, não comunicando diretamente com a emissora ou com conteúdos disponibilizados por outras infra-estruturas de redes.
- **Difusão com interatividade (*Interactive Broadcast*)**: Possui interação remota. Além da interação local, a interatividade é também estabelecida a partir da troca de informações por meio de uma rede à parte do sistema de televisão, acessível através de um canal de retorno. Por meio de linhas telefônicas ou redes de banda larga, por exemplo, fluxos de dados podem ser transportados a central emissora de TV, ou até mesmo, receptores de outros usuários. A principal diferença ao anterior, é que aqui inclui um suporte para o canal de retorno. Como explicado anteriormente podemos ter diversos tipos de interatividade. Com a presença do canal de retorno o telespectador passará a enviar informações para a central produtora do conteúdo.

A difusão com interatividade constitui o patamar de mais alto nível nas possibilidades de recursos para o telespectador. Estes níveis de possibilidades de interação com a TV, caracterizados mais adiante, possuem dependência quanto a sua execução com relação à plataforma. À medida que se avança na produção de novos recursos o desenvolvimento de plataformas mais robustas torna-se necessário. Um Set-Top Box (STB) possibilita as mais ricas experiências de interatividade através do conteúdo. Permitindo utilizar a TV como um terminal de acesso WWW ou de Email. Embalados com a convergência digital, outros tipos de terminais além da televisão, podem usufruir das transmissões digitais da TV. Dessa forma computadores, dispositivos portáteis como notebooks e móveis como celulares podem exibir o conteúdo da TVD.

É importante ressaltar que é possível ter uma TV Digital sem interatividade, da mesma forma que é possível ter interatividade na TV analógica tradicional. A digitalização não define a interatividade na TV. Por exemplo, caso um telefone celular apenas receba um sinal digital, ele será considerado simplesmente como um terminal de TV Móvel. Existindo interação entre o mesmo e a emissora, passamos a considerá-lo como um terminal de TVDI.

Considerando a diversidade de soluções tecnológicas que podem ser adotadas para programar um sistema de TVDI, diversos órgãos de padronização concentraram esforços na especificação de padrões (FERNANDES, LEMOS e SILVEIRA, 2004). No Brasil, o

resultado desses esforços culminou no padrão brasileiro de TV Digital (SBTVD – Sistema Brasileiro de Televisão Digital).

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

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

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

No Ginga, *middleware* do padrão brasileiro, as aplicações interativas podem ser classificadas em dois tipos: (i) as aplicações imperativas que utilizam a linguagem Java e (ii) as aplicações declarativas que utilizam a linguagem descritiva NCL. A seção seguinte descreve as características das aplicações interativas e as linguagens empregadas para o seu desenvolvimento.

2.2 Desenvolvimento de Aplicações para TVDi

Aplicativos da TV digital em muito se parecem com os desenvolvidos para as plataformas computacionais. Isto é, eles são criados a partir de uma linguagem de programação e em seguida convertidos em um conjunto de instruções (no formato digital) que se torna passível de manipulação pelo computador. No ambiente da TVD não é diferente, descritas por uma linguagem, as aplicações são executadas dentro de uma plataforma projetada especificamente para entender sua estrutura interna e mostrar um resultado visual ao telespectador.

No contexto da TV digital o formato das aplicações são especializações de documentos multimídia que são apresentáveis aos telespectadores através da composição de elementos multimídia, por exemplo, áudio, vídeo, imagem, texto, entre outros. Todos estes elementos são dispostos espacialmente e relacionados entre si para criar informação visual interpretável pelas pessoas. As aplicações além de possuírem a característica de ser multimídia devem possibilitar a capacidade de o usuário interagir com o conteúdo. Dessa forma as aplicações devem lidar com a sincronização espacial e temporal de objetos de diferentes tipos de mídia, além dos objetos de vídeo e áudio que compõem o fluxo principal.

Na TVDi brasileira as aplicações podem ser desenvolvidas usando a linguagem Java, aplicações imperativas, ou a linguagem NCL, aplicações declarativas. O uso delas depende das características do desenvolvedor e seu grau de conhecimento em programação. As aplicações desenvolvidas em Java exigem uma decomposição algorítmica do problema e assim exigem um conhecimento maior do programador. Já as aplicações declarativas utilizam uma linguagem de marcação mais próxima do domínio de aplicações que descreve o comportamento das aplicações utilizando um conjunto restrito e mais especifico de conceitos.

As aplicações são casos particulares de aplicações hipermídia e devem lidar com a sincronização espacial e temporal de objetos de diferentes tipos de mídia formados por imagens, vídeo/áudio, componentes de interface, texto, entre outros (SOARES e RODRIGUES, 2006). Através destes componentes, a composição gráfica das telas é constituída basicamente pelos componentes de interface gráfica (*widgets*) e dos relacionamentos entre estes.

2.3 Linguagem NCL

A linguagem NCL (Nested Context Language) (SOARES e RODRIGUES, 2006) constitui numa linguagem declarativa para autoria de documentos hipermídia. Baseada em perfis de linguagem, ela destina um perfil especializado para desenvolvimento de aplicações

para a TV Digital. A NCL especifica dois elementos principais: os nós representando as abstrações das mídias e os elos que tratam a sincronização espacial e temporal dos nós. Os nós diferenciam-se em nós de conteúdo (*media*) que trata uma mídia ou um nó de composição (*context*) que agrupa outros nós recursivamente. Os nós também podem definir interfaces como as âncoras (*area*) e as propriedades (*property*).

O elemento ***elo*** (também conhecido como ***link***) realiza o relacionamento entre as mídias com base em uma relação causal, na qual uma *condição* deve ser satisfeita para que *ações* possam ser disparadas. O elo é uma estrutura que realiza o que foi pré-definido no ***conector***, associando aos papéis de condição e de ação os seus respectivos componentes. A Figura 2.1 apresenta um conector que define uma condição e uma ação, e para isto agrega a sua estrutura dois tipos de papéis: um papel para representar **qual** a condição de ativação da ação (ou seja, o papel da condição) e outro o papel para representar a ação.

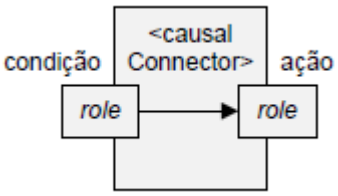


Figura 2.1 – Ilustração da estrutura do conector (SOARES e BARBOSA, 2009).

A Figura 2.2 apresenta como o elo realiza o que foi definido anteriormente pelo conector. Para o par de condição e ação do conector, o elo define uma associação (***bind***) com o seu respectivo componente de mídia.

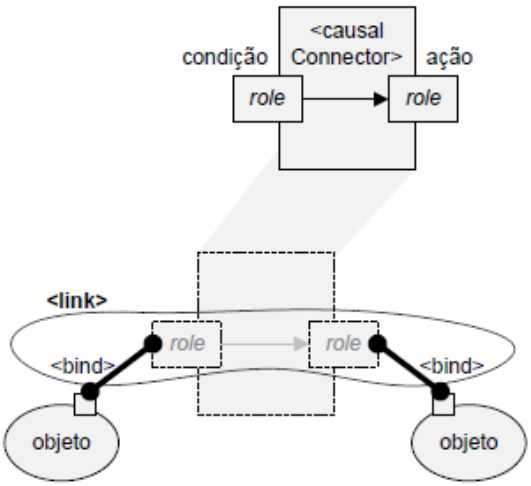


Figura 2.2 – Ilustração do elo realizando o conector (SOARES e BARBOSA, 2009).

A condição está associada à ocorrência de um dos três tipos de eventos da linguagem NCL: apresentação, seleção ou atribuição, os quais são disparados a partir de um componente de mídia (imagem, vídeo, código imperativo, etc.). As ações estão relacionadas às ações de apresentação das mídias e ações de atribuição que são para atribuir valores as propriedades da mídia.

Os nomes dos papéis fazem parte de um conjunto de papéis predefinidos, cujo comportamento é interpretado por um formatador NCL. Por padronização e para facilitar o trabalho do autor de documentos são predefinidos nomes de papéis para a condição e ação predefinidos e reconhecidos pelo formatador NCL. A Tabela 2.1 apresenta os papéis de condição predefinidos e a Tabela 2.2 apresenta os papéis predefinidos de ação.

Tabela 2.1 – Descrição dos papéis de condições agrupados.
Adaptado de (SOARES e BARBOSA, 2009).

Tipo de evento	Papel	Descrição (Quando o Elo Será Ativado)
Apresentação	**onBegin**	Quando a apresentação for iniciada
	onEnd	Quando a apresentação for terminada (naturalmente ou por uma ação stop)
	onAbort	Quando a apresentação for abortada
	onPause	Quando a apresentação for pausada
	onResume	Quando a apresentação for retomada após uma pausa
Seleção	**onSelection ou onBegin Selection**	Quando uma tecla (a ser especificada) for pressionada enquanto o objeto ligado a esse papel estiver sendo apresentado ou quando a tecla OK for pressionada enquanto o objeto ligado a esse papel estiver com o foco, ou quando um dispositivo apontador, por exemplo o mouse, selecionar o objeto em apresentação ligado a esse papel
	onEnd Selection	Quando uma tecla (a ser especificada) terminar de ser pressionada enquanto o objeto ligado a esse papel estiver sendo apresentado ou quando a tecla OK terminar de ser pressionada enquanto o objeto ligado a esse papel estiver com o foco, ou quando um dispositivo apontador, por exemplo, o mouse, terminar a seleção do objeto em apresentação ligado a esse papel
Atribuição	**onBegin Attribution**	Logo antes que um valor (a ser especificado) seja atribuído a propriedades ligadas a esse papel

	onEnd Attribution	Logo após um valor (a ser especificado) ter sido atribuído a propriedades ligadas a esse papel

Tabela 2.2 – Descrição dos papéis de ações. Adaptado de (SOARES e BARBOSA, 2009).

Tipo de ação	Papel	Descrição (Ação a Ser Realizada Quando o Elo for Ativado)
Apresentação	**start**	inicia a apresentação dos objetos associados a esse papel
	stop	termina a apresentação dos objetos associados a esse papel
	abort	aborta a apresentação dos objetos associados a esse papel
	pause	pausa a apresentação do objeto associados a esse papel
	resume	retoma a apresentação do objeto associados a esse papel (caso esteja em pausa)
Atribuição	**set**	estabelece um valor (a ser especificado) às propriedades associadas a esse papel

2.3.1 Integrando Objetos Imperativos à NCL

Um código imperativo (*script*) pode ser incluído numa aplicação NCL utilizando a ideia de que uma mídia (conceito já existente na NCL) está associada a um código na linguagem Lua. A mesma abstração presente nas mídias de conteúdo da NCL é utilizada para permitir, aos autores de documentos, definirem elos para iniciar, parar, pausar ou abortar a execução do código imperativo. Assim, do ponto de vista da linguagem NCL, um objeto imperativo é tratado da mesma forma que um objeto de mídia de texto, imagem, vídeo. A diferença é que nesse objeto devem-se incluir mecanismos para acessar o código imperativo (conteúdo imperativo) como métodos e variáveis.

Dentro de um objeto de mídia imperativa pode existir três tipos de trechos de código passiveis de serem executados, isto é, usados pelo autor de documentos: (1) execução de um método (com ou sem retorno de valor) do código Lua; (2) uso de variáveis para atribuição ou leitura de valores; e (3) execução de código presente na mídia que não está necessariamente relacionado a um método. Um método que retorna um valor possibilita o uso deste valor na parte declarativa através de um evento. Uma abordagem para execução destas estruturas é

definida pela NCL para que o autor de documentos possa utilizar cada um deles. Esta abordagem é descrita a seguir.

De modo a proporcionar a integração de mídias imperativas na NCL, o formatador NCL mantém controle da execução da mídia por meio da máquina de execução imperativa. Uma interface baseada em eventos é definida para que a comunicação entre as máquinas aconteça. Desta forma, tanto a parte declarativa NCL pode executar o código da mídia imperativa, como também a parte imperativa pode comandar ações nas ancoras de conteúdo da parte declarativa. A criação da interface envolve responsabilidades de desenvolvimento tanto para o autor do documento (programador NCL) quanto para o programador do código imperativo. Dessa forma, é estabelecida uma sincronização de duas vias entre código imperativo e o restante do documento NCL. Estas responsabilidades são descritas a seguir.

A abordagem para execução do código exige do autor de documentos a realização das seguintes tarefas: (1) definir uma interface no objeto de mídia imperativo que represente o código a ser acessado e (2) especificar a ação que realiza a execução do trecho de código em questão.

No caso da **execução de um método** ou **atribuição/leitura de valores às variáveis** pode-se definir as interfaces do tipo **property** (propriedade) ou **area** (âncora). Caso o método retorne um valor para a parte declarativa, uma propriedade adicional deve ser declarada para receber o retorno do método por ela. Estas interfaces devem necessariamente possuir o mesmo nome que os métodos ou variáveis que se queria executar no código da mídia imperativa.

Por meio desta associação (interface possuindo o mesmo nome que o método/variável) e quando as ações sobre estas interfaces são especificadas, tanto o formatador como o programador da mídia imperativa terão informações suficientes para saber qual o código que se quer acessar na mídia, quando ele for chamado. Estas informações são encapsuladas como eventos e enviadas pelo formatador para a máquina de execução imperativa, de modo que eles possam ser tratados (pelo programador imperativo) para consequentemente chamar o código imperativo relacionado ao evento.

Ações sobre as interfaces disparam a ocorrência da execução do código dentro da mídia imperativa. A interface do tipo *área* junto com a ação de inicio (*start*) é utilizada normalmente quando se deseja executar um método que não precisa de parâmetros. Já a interface *property* juntamente com a ação de atribuição (*set*) é necessária quando se precisar

executar a chamada de métodos que requer a passagem de parâmetros, como também para atribuir ou ler valores das variáveis presentes no código.

As listagens a seguir apresentam o uso do mecanismo para executar um método e atribuir um valor para uma determinada variável do código. A Listagem 1 (parte de um documento NCL) apresenta a mídia imperativa (linha 1) e as ações dos elos sobre as interfaces que são responsáveis pela execução de um método com parâmetros (linha 9) e pela execução de um método sem parâmetros (linha 16).

A mídia imperativa (linha 1) possui o atributo ***src*** referenciando um *script* na linguagem Lua (**código_imperativo.lua**) e define três interfaces para acesso as estruturas deste mesmo código Lua: uma propriedade (**variavel**) para atribuir um valor a uma variável de mesmo nome no código Lua, uma propriedade (**metodoComParametros**) para usar um método Lua de mesmo nome, que retorna um valor e possui dois parâmetros de entrada e por fim uma área (**metodoSemParametros**) com o intuito de usar um método Lua também de mesmo nome, que não possui parâmetros de entrada e nenhum valor de retorno, ou seja, um método do tipo procedimento.

O elo da linha 7 especifica no seu *bind* a ação ***set*** (linha 9)para acessar um método que possui dois parâmetros de entrada. Para chamar o método passando os valores para cada parâmetro, a ação de atribuição atribui à propriedade **metodoComParametros** o valor "1, 2", especificando o valor "1" para o primeiro parâmetro e "2" para o segundo. Os valores para cada parâmetro são especificados separados por vírgula utilizando o elemento **bindParam** da linha 10.

O elo da linha 26 especifica a forma como é recebido um valor de retorno resultante do processamento de um método. Ele possui o *bind* de condição de final de atribuição (**onEndAttribution**), o qual espera o evento de final de atribuição de valores para a propriedade **resultado**. Esta propriedade é utilizada pela mídia imperativa para enviar o valor do processamento como um evento de final de atribuição. Quando o método Lua gera o evento de final de atribuição encapsulado com o valor, a condição do elo é disparada resultando na leitura do valor pelo *bind* da linha 28 e, por conseguinte na atribuição deste valor para a mídia **globalVar** (linha 29).

O elo da linha 14 especifica no seu *bind* de ação ***start*** (linha 16) a ação para acessar um método Lua sem parâmetros. O *bind* usado apenas especifica uma ação de ***start*** para à área **metodoSemParametros** da mídia imperativa. Isto ocasiona a chamada do método de mesmo nome no *script* Lua.

```
...
1.  <media id="midiaImperativa" src="codigo_imperativo.lua"
    type="application/x-ginga-NCLua">
2.      <property name="variavel"/>
3.      <property name="metodoComParametros"/>
4.      <área id="metodoSemParametros" label="metodoSemParametros"/>
5.  </media>

6.  <media id="midiaImagem" src="imagem.png"/>

7.  <link xconnector="onSelectionSet">
8.      <bind role="onSelection" component="midiaImagem"/>
9.      <bind role="set" component="midiaImperativa"
          interface="metodoComParametros">
10.         <bindParam name="var" value="1, 2"/>
11.     </bind>
12. </link>
13.
14. <link xconnector="onSelectionStart">
15.     <bind role="onSelection" component="midiaImagem"/>
16.     <bind role="start" component="midiaImperativa"
         interface="metodoSemParametros"/>
17. </link>
18.
19. <link xconnector="onSelectionSet">
20.     <bind role="onSelection" component="midiaImagem"/>
21.     <bind role="set" component="midiaImperativa"
        interface="variavel">
22.     <bindParam name="var"value="1"/>
23.     </bind>
24. </link>
25.
26. <link xconnector="onEndAttributionSet">
27.     <bind role="onEndAttribution" component="midiaImperativa"
        interface="resultado"/>
28.     <bind role="getResultado" component="midiaImperativa"
        interface="resultado"/>
29.     <bind role="set" component="globalVar" interface="resultado">
30.         <bindParam name="var" value="$getResultado"/>
31.     </bind>
32. </link>
...
```

Listagem 1 – Documento NCL com elos para uso de métodos e variáveis de uma mídia imperativa.

O elo da linha 19 especifica no seu *bind* de ação ***set*** (linha 21) uma ação para atribuir um valor para a variável (**variavel)** presente no código Lua. Este *bind* é semelhante ao *bind* para chamada de método com parâmetros, a diferença é que o nome da propriedade utilizada refere-se a uma variável no código Lua. O elemento **bindParam** é utilizado para ser o valor para atribuir a variável no código Lua. A distinção se é uma atribuição de variável ou

chamada de método com parâmetros fica a cargo do programador Lua. A seguir é apresentado como a mídia imperativa é implementada.

```
1.  -- definicao da variavel variavel
2.  local variavel = 0
3.
4.  -- definicao do metodo metodoComParametros
5.  function metodoComParametros(parametro1, parametro2)
6.      local soma = parametro1 + parametro2
7.
8.      local eventoSaida = {
9.          class = 'ncl'
10.         type  ='attribution',
11.         name  ='resultado',
12.     }
13.
14.     --sinaliza NCL de que o resultado foi processado
15.     eventoSaida.value = soma
16.     eventoSaida.action = 'start'; event.post(eventoSaida)
17.     eventoSaida.action = 'stop';  event.post(eventoSaida)
18. end
19.
20. -- definicao do metodo metodoSemParametros
21. function metodoSemParametros()
22.
23. end
24.
25. -- funcao tratadora de eventos
26. function tratador(evt)
27.     -- bloco condicional para tratar chamadas de metodos com parametros
28.     -- atribuição de valores a variaveis
29.     if evt.class == 'ncl'and evt.type == 'attribution'and evt.action=='start'then
30.         if evt.name == 'metodoComParametros'then
31.             local parametros = extraiParametros(evt.value)
32.             metodoComParametros(parametros[1], parametros[2])
33.         elseif evt.name == 'variavel'then
34.             variavel = evt.value
35.         end
36.     -- bloco condicional para tratar chamadas de metodos sem parametros
37.     elseif evt.class == 'ncl'and evt.type == 'presentation'and evt.action=='start'
38. then
39.         if evt.label == 'metodoSemParametros'then
40.             metodoSemParametros()
41.         end
42.     end
43. end
44. -- registra a funcao tratadora de eventos
45. event.register(tratador)
```

Listagem 2 – Código da mídia imperativa Lua.

A Listagem 2 apresenta o código do *script* na linguagem Lua, definindo os métodos e variáveis imperativas (linhas 2, 5 e 10) e o método tratador de eventos (linha 15), que é responsável por realizar as chamadas dos métodos do *script* Lua, a partir dos eventos disparados da parte NCL (declarativa). Cada ação descrita anteriormente faz com que seja disparado um evento NCL (atribuição ou apresentação) pelo formatador NCL para o código imperativo. Este evento fica disponível para o programador imperativo poder tratá-lo e assim poder atuar na chamada do método ou atribuição do valor as variáveis da mídia imperativa.

Como explicado anteriormente, o elo da linha 26 permite receber um valor da parte imperativa a partir de uma propriedade da mídia imperativa. No método **metodoComParametros** envia-se o valor da soma dos dois parâmetros como um evento de final de atribuição para a propriedade da mídia imperativa **resultado**. Por exemplo, o método **metodoComParametros**, entre as linhas 8 e 17, é definido o evento de atribuição **eventoSaida** com o valor (**soma**) e a propriedade da mídia imperativa (**resultado**) que receberá este valor. Em seguida é gerado um evento de inicio e fim de atribuição (linhas 16 e 17) para notificar a parte declarativa sobre o valor. Dessa forma, quando o evento for lançado, o elo da parte declarativa poderá ler o valor enviado pela parte imperativa, como descrito anteriormente.

Para a mídia imperativa tratar todos os eventos disparados pelo documento NCL, o trecho de código da linha 33 serve para registrar um método Lua (método tratador de eventos) que recebe todo e qualquer evento disparado. Um exemplo típico de evento é apresentado na Listagem 3, descrevendo um evento que é gerado a partir de uma ação de atribuição (linha 9 da Listagem 1), o qual possui o seu tipo com valor *attribution*, o seu nome com valor *metodoComParametros*, a ação com valor *start* e por fim o valor “1, 2”.

O método tratador de eventos usa estruturas condicionais para tratar cada evento especifico e assim atuar de acordo, ou seja, chamando um método ou atribuindo valor a uma variável. Por exemplo, na linha 18 o bloco condicional atua para tratar todas as chamadas de métodos com parâmetros e atribuição de valores a variáveis, que é o evento de atribuição. Dentro desse bloco as estruturas condicionais atuam para tratar as chamadas em particular, por exemplo, na linha 19 se o nome da propriedade envolvida for **metodoComParametros** o método de mesmo nome é chamada, com seus respectivos valores de parâmetros, os quais são obtidos na linha 20. Na linha 23 é atribuído o valor a variável.

Continuando com os blocos, o bloco condicional da linha 26 trata os eventos de apresentação e atua para chamar as funções sem parâmetros. Na linha 27 se o *label* do evento for **metodoSemParametros** o método de mesmo nome é chamado.

```
evento = {
        class   =   'ncl',
        type    =   'attribution',
        name    =   'metodoComParametros',
        action  =   'start',
        value   =   '1, 2',
}
```

Listagem 3 – Descrição do evento para a ação de uso do método metodoComParametros.

O terceiro tipo de trecho de código é executado sem ser preciso especificar qualquer interface, pois por padrão ele está sempre associado à âncora de conteúdo principal do objeto imperativo. Este código é executado toda vez que um objeto imperativo é iniciado (ação de inicio no objeto) sem que seja especificada uma de suas âncoras de conteúdo ou de propriedade. Nestas condições, a "âncora de conteúdo principal" da mídia é assumida e, como consequência, o trecho de código a ela associado é executado. Este tipo de trecho de código é usado normalmente para registrar um ou mais tratadores de evento para comunicação com o formatador NCL.

2.4 Ferramentas de Autoria

No começo da popularização da computação pessoal, a produção de conteúdo multimídia era feita apenas por especialistas na área. Na cadeia de negócio, a produção de conteúdo se deu principalmente pelos que dominavam certas tecnologias e o usuário final ficava sempre dependente dos serviços oferecidos. Entretanto, atualmente as ferramentas permitem que uma pessoa não especialista na área criem seu próprio conteúdo e disponibilizem para outras pessoas.

O surgimento da Web 2.0, baseada em colaboração, foi um dos fatores que permitiu que essa produção de conteúdo multimídia também ganhasse espaço na Internet, ou seja, uma nova forma do usuário expressar e produzir seu próprio conteúdo, se libertando da

dependência de outrora. Neste cenário os usuários usam seus computadores para compor suas produções multimídia.

Autoria é a criação de conteúdo para qualquer tipo de documento (MYERS, 1998) que também podem incluir ou ser especificados na forma de aplicações. Sabemos que por ser o alicerce da computação, a programação, é o primeiro método a ser recorrido na produção de qualquer artefato computacional. Entretanto, a universalidade que linguagens de programação podem oferecer em autoria multimídia é contraposta pela busca da redução de complexidade (KASKALIS, TZIDAMIS e MARGARITIS, 2007). Assim, desenvolvimento de aplicações multimídia acontece tipicamente em um ambiente interativo de desenvolvimento que esconde (do desenvolvedor) detalhes de programação de baixo nível responsáveis por controlar os objetos multimídia envolvidos na apresentação.

Ambientes de desenvolvimento de aplicações multimídia facilitam e automatizam a autoria (criação) de documentos de multimídia. Há uma diversidade alta de tais ambientes (também chamados como vimos de sistemas de autoria) (MAKEDON, MATTHEWS, *et al.*, 2001). Eles dependem dos diferentes tipos de aplicações que os guiam. O tipo de aplicações atravessa uma variedade de tópicos e níveis, desde simples panfletos eletrônicos à sofisticadas aplicações envolvendo computação e apresentação de informações. A autoria de um documento de multimídia é um processo complexo e é mais que a combinação de elementos multimodais de diversas mídias. Uma característica desejável é a definição da interatividade e do sincronismo temporal do documento de multimídia.

3 REVISÃO DE LITERATURA

Este capítulo reúne a revisão de literatura empreendida para apresentar as ferramentas de autoria existentes. Além disso, ela serve para descobrir algumas limitações que precisam ser resolvidas na visão estrutural. Tais limitações constituem nos principais problemas endereçados nesta dissertação. Primeiro, foram estudados os métodos e ferramentas de autoria. No segundo estudo, foram levantadas as principais características da visão estrutural no desenvolvimento de aplicações multimídia com foco na integração entre UI e mídias imperativas.

As seções 3.1 a 3.4 descrevem as ferramentas de autoria empregadas para o desenvolvimento de aplicações na TVDi. A seção 3.5 apresenta o que consideramos ser, a principal visão gráfica para tratar a integração entre NCL e mídias imperativas no desenvolvimento de aplicações multimídia.

3.1 JAME Author

O JAME Author (IST, 2012) é uma ferramenta comercial desenvolvida pela empresa alemã Fraunhofer IAIS. Ela tem por objetivo facilitar autoria de aplicativos imperativos escritos na linguagem Java e voltados para o padrão europeu MHP (*Multimedia Home Platform*) de TVDI. Esta ferramenta trabalha com uma classe especial de aplicativos, chamados de serviços baseados em páginas (*page-based service*), cuja navegação se assemelha à navegação na Web. Durante a criação de seu projeto, o autor poderá criar suas próprias páginas ou se basear em páginas pré-definidas. O processo de edição dos aplicativos é feito através de dois modos: o modo de design e o modo de navegação. No modo de design,

os componentes podem ser selecionados e inseridos na página em edição. Um conjunto padrão de componentes já vem especificado na ferramenta como mostra a Figura 3.1.

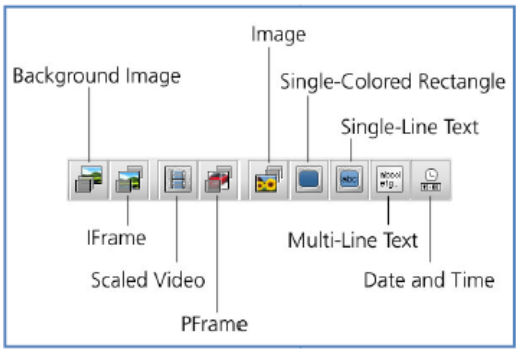


Figura 3.1 - Componentes de interface disponíveis no JAME Author.

Já no modo de navegação, é possível definir a estrutura navegacional intra e inter páginas. A navegação é definida através de regras de transferência de foco entre componentes ou páginas, acionadas quando as teclas do controle remoto forem pressionadas. A Figura 3.2 mostra a tela principal da ferramenta.

Figura 3.2– Modo de design da ferramenta JAME Author.

3.2 NCL Eclipse

O NCL Eclipse (AZEVEDO, NETO, *et al.*, 2011) é um editor textual de código para a linguagem NCL. Integrado ao ambiente do Eclipse como um *plug-in*, ele tem o foco em oferecer uma ferramenta de apoio à geração de código fonte. Dentre as principais funcionalidades implementadas pelo NCL Eclipse estão: coloração de elementos e atributos XML, suporte para que determinados elementos XML sejam escondidos ou exibidos pelo usuário, wizards para a criação de documentos NCL simples, autoformatação do código XML, validação do documento NCL, sugestão de código NCL de forma contextual (autocomplete), navegação no documento como uma árvore, execução do documento NCL, dentre outras.

O plug-in não oferece funcionalidades de apoio ao reaproveitamento de código ou mesmo a especificações de projeto mais abstratas. Além disto, este tipo de ferramenta apesar de acelerar a escrita da sintaxe, não aborda o problema de edição rápida dos valores dos atributos relacionados.

3.3 NCL Composer 3

O NCL Composer 3 (LIMA, AZEVEDO, *et al.*, 2010) (TELEMÍDIA, 2013) ganhou destaque na comunidade NCL como um editor visual para reduzir o tempo e o esforço necessário para desenvolver aplicações. O NCL Composer é um ambiente de autoria desenvolvido pelo Laboratório de Telemídia do Departamento de Informática da PUC-Rio.

Ao longo do desenvolvimento de um projeto, o código-fonte da aplicação é automaticamente criado, porém é recorrente ser necessário acessar o código para realizar procedimentos comuns. A Figura 3.3 apresenta várias visões do projeto, são elas: (1) estrutural, (2) de propriedades, (3) de leiaute, (4) de hierarquia de elementos e (5) textual.

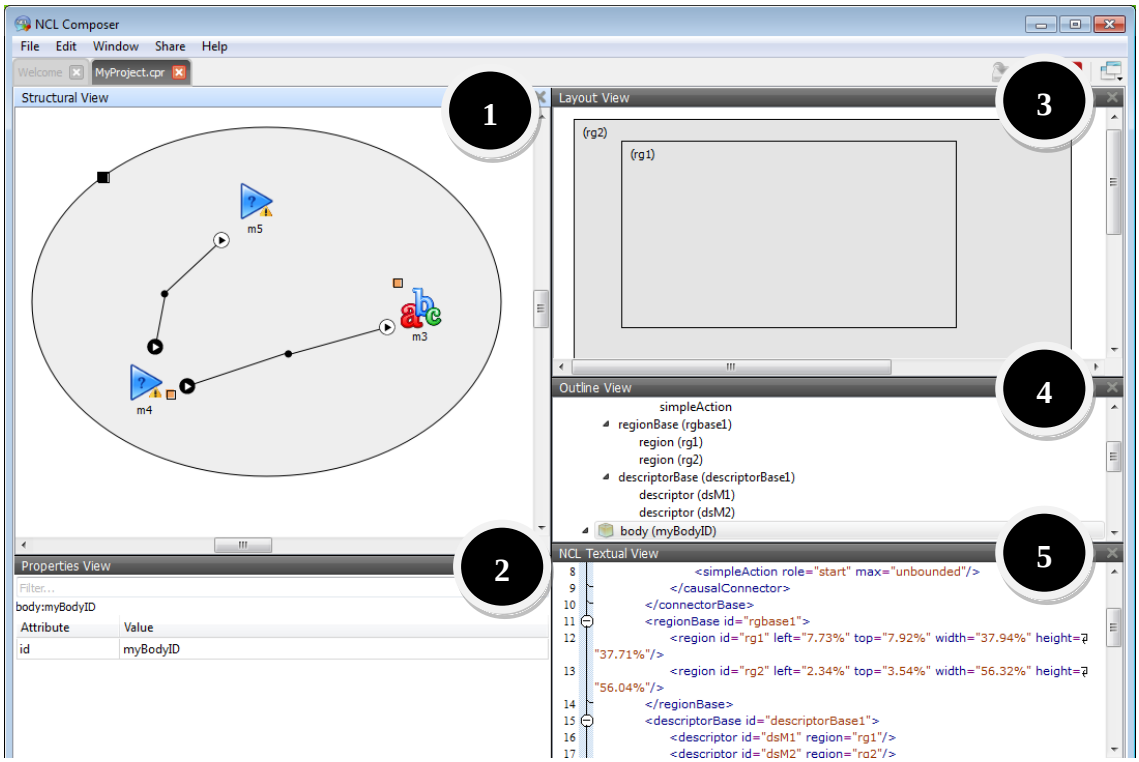


Figura 3.3 – Tela da ferramenta de autoria NCL Composer.

O NCL Composer pode ser entendido como uma interface para manipulação de objetos NCL. A forma como ele foi desenvolvido necessita de que o usuário conheça a linguagem NCL, ou seja, conheça de programação em NCL.

A grande vantagem deste editor é auxiliar na criação rápida de aplicações NCL. Entretanto, ele requer do usuário conhecimento especializado da linguagem NCL para desenvolver uma aplicação. A notação visual faz uso de muitos conceitos presentes na NCL, específicos ao domínio de aplicações multimídia interativas.

3.4 Visão Estrutural no NCL Composer

A visão estrutural aqui descrita é explicada tomando como base a versão do NCL Composer que é desenvolvida e disponibilizada em (TELEMÍDIA, 2013). Ela foi escolhida por ser a ferramenta de autoria mais atual e voltada para o desenvolvimento da linguagem NCL na TV digital.

A visão estrutural permite a criação das mídias e modela visualmente o relacionamento entre elas, facilitando a especificação da interatividade e do sincronismo da aplicação. O sincronismo descreve as ações realizadas (iniciar, pausar, parar, etc.) em

decorrência dos eventos ocorridos nos elementos de mídias, já a interatividade descreve as ações a partir da interação feita pelo usuário.

Um objeto de mídia dentro da visão estrutural representa uma abstração visual sobre os elementos de mídia do documento NCL e são utilizados pelo autor do documento para compor a aplicação visualmente. Um objeto de mídia pode representar três tipos de entidades possíveis: (i) as mídias já existentes no NCL (vídeo, áudio, imagem, etc.); (ii) as mídias imperativas NCLua.

O NCL Composer no momento da criação da mídia não faz distinção entre o tipo de mídia que pode ser criado (mídia de áudio, vídeo, texto, etc.). O autor de documentos cria primeiramente uma mídia genérica e somente após indicar o seu tipo de conteúdo (pela propriedade **src** da mídia), é que a visão estrutural determina o tipo da mídia. Com as mídias definidas na visão, o autor de documentos pode definir como a apresentação irá proceder a partir de um conjunto de ações sobre estes elementos de mídia.

O principal objetivo da visão estrutural é fazer os relacionamentos entre os elementos de mídia, de modo a compor a apresentação da aplicação. Sendo assim, o principal elemento gráfico na visão é o elo da linguagem NCL. Como dito anteriormente, um elo relaciona dois ou mais elementos de mídia, baseado numa relação causal, isto é, uma *condição* deve ser satisfeita para que *ações* possam ser disparadas. A Figura 3.4 apresenta uma visão estrutural com quatro elementos de mídia relacionados por dois elos, pontos 1 e 2 na figura.

As ações na visão estrutural são sempre ocasionadas a partir dos eventos que ocorrem nas mídias, sejam elas de conteúdo principal (áudio, vídeo, texto, imagem, etc.) ou imperativas. Por exemplo, a visão estrutural da Figura 3.4 apresenta o fluxo de ações realizadas quando o usuário pressiona o botão verde sobre o elemento de mídia **img** do tipo imagem. O ponto 1 especifica um elo, que quando o usuário interagir apertando o botão verde sobre **img**, um valor é atribuído para a propriedade do elemento de mídia **m1** e o elemento de mídia **audio** é iniciado. O elo do ponto 2 especifica que quando o elemento de mídia **audio** for iniciado o elemento de mídia **texto** é inicializado.

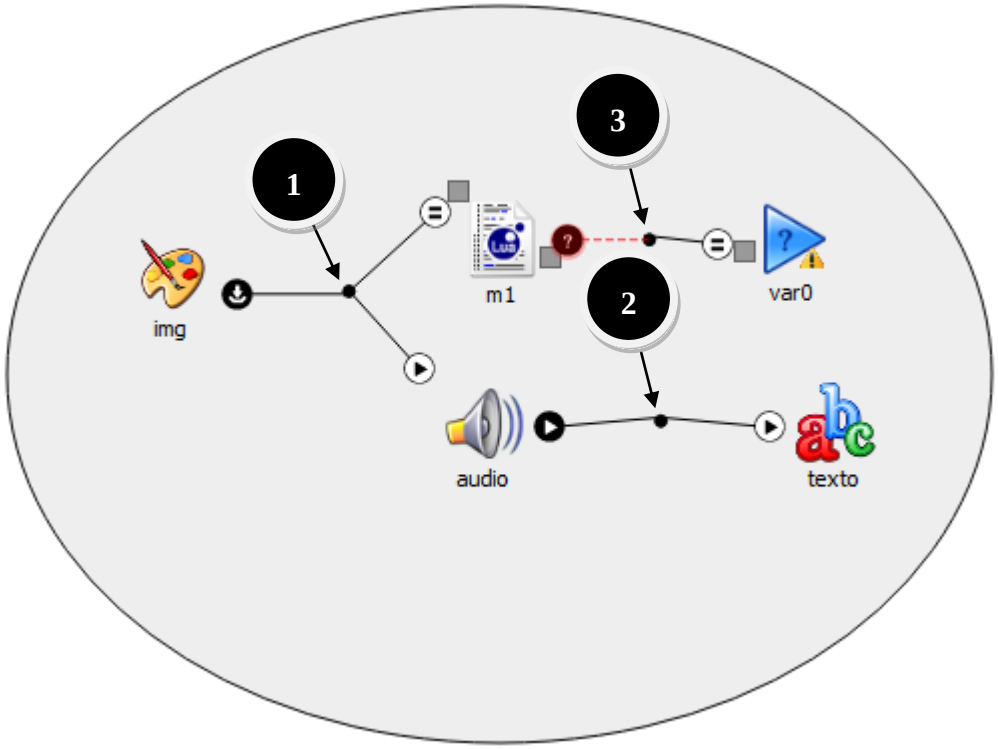


Figura 3.4 – Visão estrutural do NCL Composer descrevendo ações em decorrência da interatividade.

No NCL Composer, cada elo é representado visualmente por um ponto cheio (pontos 1 e 2 da figura), o qual serve para unir, por meio de uma reta, os *binds's* de condições e de ações. Como visto anteriormente, cada *bind* serve para associar um determinado elemento de mídia ou interface para um papel de condição ou papel de ação do *bind*. Normalmente, os elementos de mídia que fazem o papel da condição são postos à esquerda do ponto (elo) e os elementos de mídia que fazem o papel da ação ficam à direita do ponto.

Visualmente o *bind* é representado por dois elementos: um ícone (que tipifica o *bind* de condição ou de ação) e uma reta, a qual sai do elo (ponto cheio) e vai ate o elemento de mídia ou interface. Por exemplo, na figura o elo do ponto 1 possui três *binds*, sendo um de condição (pressionamento da tecla verde), ligado ao elemento de mídia **img**, e os outros dois de ações que estão ligados a propriedade do elemento **m1** (ação de atribuição de valor à propriedade) e ao elemento **audio** (ação de inicio).

As figuras Figura 3.5 e Figura 3.6 exemplificam as mídias **img** e **m1** com seus respectivos *binds*. A Figura 3.5, apresenta o *bind* que associa a mídia **img** ao papel de condição do tipo ***onSelection***. A Figura 3.6, apresenta o *bind* que associa a mídia **m1** ao papel de ação de atribuição (*set*).

Figura 3.5 – Mídia de imagem com *bind* referente ao evento de pressionamento da tecla verde.

Cada *bind* seja ele de condição ou de ação possui um circulo e um ícone específico. Os ícones dos papeis de condições são sempre representados por um circulo de fundo escuro e um imagem branca que caracterize o tipo de condição do papel. Já os ícones dos papeis de ações são representados por um circulo de fundo branco e no centro um ícone escuro. Por exemplo, o ícone do papel de condição de seleção é representado por uma seta para baixo, já o ícone do papel de condição de inicio (*onBegin*) é representado por uma seta para a direita.

Na Figura 3.6, a mídia **m1** possui o *bind* de ação de atribuição sobre sua propriedade. Esta ação representa uma forma de atribuir valores para a propriedade do elemento de mídia imperativo **m1**. Por ser uma ação de atribuição, o ícone do papel de ação é representado por um símbolo de igualdade, simbolizando assim uma atribuição de valores as propriedades.

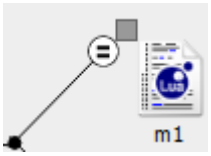


Figura 3.6 – Mídia com bind referente à ação de atribuição.

O *bind* da ação de atribuição ilustra um ponto importante na questão da integração de objetos de mídia imperativos na NCL. Como descrito anteriormente, esta ação é primordial para utilizar um método que se encontra em um *script* Lua. No exemplo, este *bind* associado à propriedade do elemento imperativo significa que um método com o mesmo nome da propriedade seria executado no *script* Lua.

Por prover suporte a esta ação, o NCL Composer oferece um passo além de (SOARES, RODRIGUES e SAADE, 2000) no uso de *scripts* NCLua, pelo fato de adicionar suporte a ações de atribuição e configuração dos valores do *bind*. Apesar do suporte a esta ação, o NCL Composer possui alguns problemas nessa integração, um deles é negligenciar o recebimento de valores do método, caso tenha-se um método que retorne um valor. Por exemplo, o *bind* de condição do elo do ponto 3 da Figura 3.4 que seria responsável pela leitura do valor retornado pelo método (utilizando a propriedade da mídia **m1**) é apresentado em vermelho indicando que o NCL Composer ainda não reconhece este tipo de evento. Logo,

a leitura do valor retornado por métodos, como descrito na seção 2.3.1, não é possível. Estes e outros problemas são descritos no próximo capítulo.

A criação do elo começa pela escolha dos elementos de mídia que irão participar da relação. Para criar um relacionamento entre duas mídias, basta que o autor de documentos, escolha a mídia que estará associada ao papel da condição e em seguida escolha a que estará associada ao papel da ação. Assim que é escolhido o elemento para fazer o papel da ação, uma janela de dialogo aparece para que os papéis do elo possam ser escolhidos.

A Figura 3.7 apresenta a janela que permite criar o elo a partir de um conector já existente ou então escolhendo a condição e a ação usando as escolhas pré-estabelecidas. Esta ação cria automaticamente um *bind* para a condição e outro para a ação. Para criar outros *binds* basta, na própria visão estrutural, usar o ponto do elo como origem e outros componentes como destino.

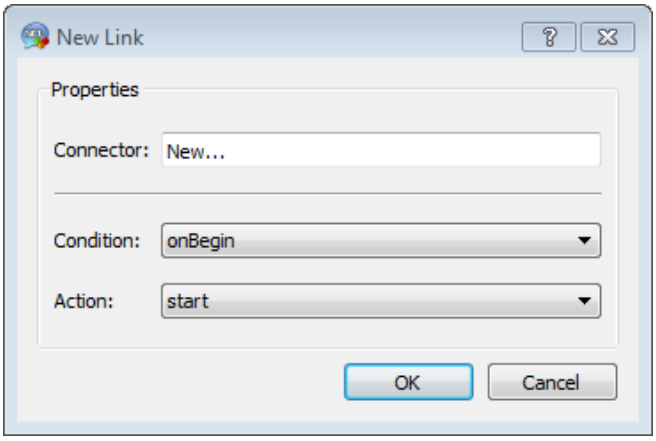


Figura 3.7 – Janela de dialogo para definir os papeis de condições e de ações do elo sendo criado.

4 DESCRIÇÃO DO PROBLEMA

A especificação feita por uma linguagem textual está relacionada basicamente a duas tarefas: (i) definição da sintaxe do elemento da linguagem a ser utilizado e (ii) de que forma este elemento é usado (isto é, seus atributos preenchidos com valores). O trabalho (AZEVEDO, NETO, *et al.*, 2011) contorna em parte esse problema, pois apesar de resolverem parte do problema (acelerarem a escrita da sintaxe), não abordam a segunda tarefa, que é a edição rápida dos valores dos atributos relacionados à especificação da aplicação.

O emprego de uma ferramenta visual dentro do processo de autoria trás facilidades na criação do documento para autores inexperientes, assim como rapidez na edição dos valores dos atributos para automatizar a geração da aplicação. Utilizando abstrações visuais o autor lida diretamente com estruturas gráficas que substituem o contato direto com detalhes da linguagem textual NCL.

O desenvolvimento de aplicações multimídia NCL envolve dois tipos de projeto: o projeto de mídia e o projeto do software (lógica imperativa). Pelo fato delas serem comumente desenvolvidas independentemente (por profissionais e equipes distintas) é preciso que os artefatos produzidos em cada projeto sejam integrados. O projeto de mídia demanda uma integração com o que foi produzido no projeto de software, pois compreende uma grande parte da aplicação e necessita, mesmo que às vezes, do projeto de software.

O projeto de mídia, realizado pelo designer, corresponde à produção criativa da interface com o usuário da aplicação por meio dos elementos de mídia e *widgets* e da sua especificação dos relacionamentos entre estes elementos de mídia. O projeto de software é realizado pelo programador e refere-se à lógica imperativa, que está normalmente presente em alguns tipos aplicações.

Um requisito atual que se encaixa no cenário desta integração é a utilização dos serviços web dentro da aplicação. A automatização do uso do código imperativo presente nos serviços *web* facilitaria a vida do autor de documentos, o qual necessita facilidade para reusar este tipo de código (conteúdo) na aplicação.

Entre os fatores que influenciam na dependência estão: (i) eventos de elementos de mídia propagados para realizar alguma lógica da aplicação e vice versa; e (ii) criação, exclusão, alteração de elementos de mídia a partir da lógica imperativa em tempo de execução. É necessário que a ferramenta de autoria defina uma visão onde as inter-relações entre o projeto de mídia e projeto de software sejam suportadas.

Um elemento de mídia imperativo é desenvolvido pelo programador e refere-se a alguma lógica imperativa que pode ser agregada na parte declarativa da aplicação. A visão estrutural (COELHO, RODRIGUES e SOARES, 2004) (SOARES, RODRIGUES e SAADE, 2000) aborda o relacionamento visual dos elementos, entretanto no seu projeto original foi tratado apenas o relacionamento entre os elementos de mídia de conteúdo (áudio, vídeo, imagem, texto, etc.) e aspectos de implementação da navegação pela visão como o tratamento gráfico em olho de peixe.

Trazendo um avanço a essa proposta inicial, o NCL Composer complementa a visão por meio do suporte aos elementos de mídia imperativos, que são comumente chamados de objetos de mídia NCLua (SANT'ANNA, CERQUEIRA e SOARES, 2008). Logo, o NCL Composer além de implementar a visão estrutural original, adiciona suporte para integrar elementos declarativos juntamente com elementos imperativos. Apesar do NCL Composer mostrar-se efetivo para realizar o relacionamento entre os elementos de mídias declarativos e imperativos, foram identificados problemas com a integração destes elementos na aplicação.

De maneira geral, a visão estrutural atualmente não prove um suporte eficiente ao relacionamento com elementos de mídia imperativos, de modo que o uso do conteúdo imperativo (método e variáveis) seja feito de maneira rápida e simples. Os problemas identificados tornam a integração destes elementos custosa e complexa para ser feita por um autor de documentos.

A integração de uma mídia imperativa envolve o relacionamento entre dois domínios: o declarativo e o imperativo. A parte declarativa usa o conteúdo imperativo e este por sua vez pode retornar valores para serem usados na parte declarativa. O uso do conteúdo imperativo deve ser feito da maneira mais simples possível, isto é, evitando ao máximo o contato do

autor de documentos com os conceitos envolvendo o uso do conteúdo sem que seja necessário entender da linguagem imperativa e dos detalhes de implementação desta mídia.

Além disto, deve ser a mais rápida possível, sempre procurando ao máximo o uso automático deste conteúdo. A seguir são pontuadas situações nas quais o NCL Composer não deixa intuitivo nem rápida a utilização de mídias imperativas por autores de documentos.

1 – Falta de exposição do conteúdo imperativo. Uma mídia imperativa, quando criada na visão estrutural, não tem seu conteúdo como métodos e variáveis expostos automaticamente. A exposição automática evita que a criação destas estruturas seja feita pelo autor de documentos.

A Figura 4.1 apresenta um exemplo de mídia imperativa recém adicionada na visão estrutural. Após ela ser criada, a visão não disponibiliza de maneira simples quais os métodos e variáveis disponíveis nesta mídia. Para acessar algum método é preciso que o usuário crie os elementos de propriedades que representam os métodos que ele queira acessar.

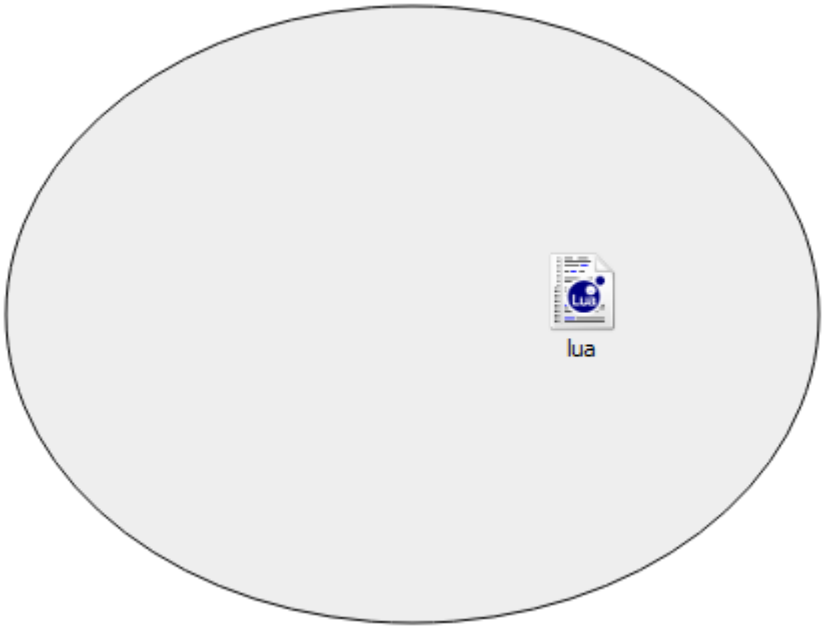


Figura 4.1 – Exemplo de mídia imperativa recém adicionada.

No NCL Composer, atualmente, para criar o conteúdo imperativo o autor de documentos pode proceder de duas formas: (1) o próprio autor atua na identificação das estruturas alvos (métodos, variáveis) olhando a implementação do código imperativo ou (2) os métodos e variáveis são criados a partir de um documento de especificação do código imperativo criado pelo programador.

A primeira forma implica que o autor tenha conhecimento da linguagem imperativa para poder encontrar o código (conteúdo) necessário para a aplicação. Isto implica além de perda tempo para achar tais estruturas, experiência da linguagem imperativa, o que nem sempre o autor possui.

A segunda forma evita que o autor de documentos lide com a implementação imperativa, disponibilizando o conteúdo através de um documento de especificação. Ele é criado pelo programador e contém os métodos e variáveis existentes na mídia, a assinatura (nome do método e parâmetros) bem como o propósito de cada um deles. Partindo desta especificação, o autor de documentos cria as estruturas na mídia imperativa para poder acessar o conteúdo imperativo.

Apesar do documento de especificação auxiliar o autor, ele ainda necessita criar as propriedades que representam os métodos. Isto implica que o autor tenha o conhecimento sobre a linguagem NCL.

2 – Falta de identidade visual das propriedades e ação de atribuição como forma de uso do conteúdo imperativo. A notação visual das propriedades empregadas para representar os métodos e variáveis dificulta a identificação das propriedades no caso de métodos que retornam um valor ou a leitura de valores das variáveis. Além disso, o emprego da ação de atribuição não se adéqua a semântica por trás da execução de um método do código imperativo.

Na linguagem NCL uma propriedade serve para representar uma característica do objeto de mídia, a qual pode armazenar valores. O uso das propriedades foi criado para se ter um melhor controle dinâmico da apresentação da mídia (SOARES, RODRIGUES, *et al.*, 2010). Por exemplo, utilizando elos é possível dinamicamente modificar as propriedades dos objetos de mídia para alterar as suas características de exibição como, por exemplo, posicionamento espacial (*left*, *top*, *width* e etc.) e características adicionais de apresentação como as propriedades de duração (*explicitDur*), visibilidade (*visible*), transparência (*transparency*), entre outras.

Para o autor de documentos, é difícil enxergar uma propriedade, como um elemento que permite executar um método (isto é, realizar um processamento) e retornar um valor ou efetuar a leitura de um determinado valor da mídia imperativa.

O problema surge quando é preciso mapear este mesmo conceito para executar métodos que retornam um valor do processamento ou ler uma variável da mídia imperativa.

Quando uma mídia imperativa possui um método que retorna um valor, para usá-lo é preciso que se tenham duas propriedades: uma propriedade associada ao método e outro associada ao valor de retorno do método. Como as propriedades visualmente possuem a mesma aparência (identidade visual), nesta situação o autor não sabe (de forma direta) se a propriedade refere-se a um método ou a uma propriedade que recebe o valor de retorno do método. Neste caso seria preciso verificar propriedade por propriedade para poder identificá-las.

A identificação fica ainda mais prejudicada quando na mídia existe mais de um método. Por exemplo, imaginemos que uma mídia imperativa (**lua**) possua dois métodos, cada um retornando um valor para o NCL (Figura 4.2). Nesta situação, não é possível saber quais das propriedades são os métodos e quais são os valores de retorno.

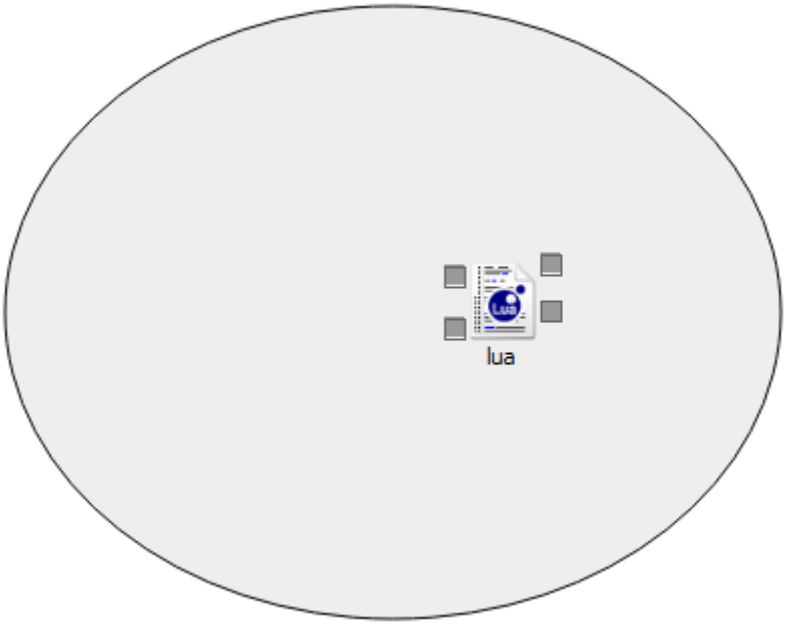


Figura 4.2 – Falta de critérios para identificar métodos e valores de retorno.

3 – Necessidade do autor informar a ação quando usando um método ou variável. Como já descrito na seção 2.3.1, o uso do conteúdo imperativo necessita que o autor de documentos tenha noções de como usar um determinado método, envolvendo, por exemplo, a especificação de qual é a interface do método e a ação a ser realizada. No NCL Composer, quando usando um método, sempre é necessário que o autor especifique qual a ação a ser utilizada nestes casos.

Este cenário é demonstrado na Figura 4.3. Quando no momento da ligação com um método da mídia imperativa (uso de método), a ferramenta pede ao autor que especifique qual

será a ação do novo elo sendo criado, utilizando o campo ***action*** presente na janela de dialogo para definição do elo.

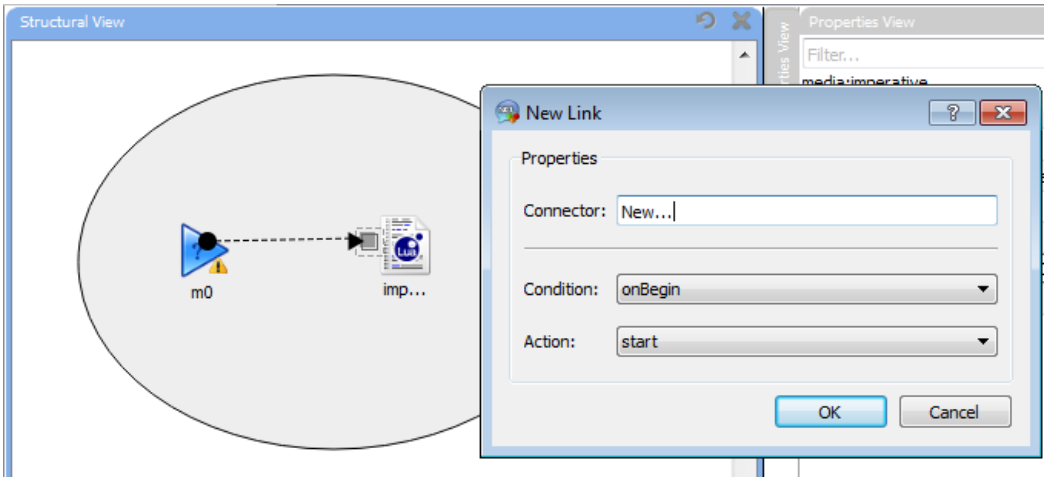


Figura 4.3 – Definição do elo com ação que usa método da mídia imperativa.

Sendo assim, quando criando um elo cuja ação é o uso de um método, é necessário que o usuário informe qual é a ação que será realizada, o que pode tornar a especificação do elo mais demorada. Já que a ferramenta mantém controle das entidades que se referem a propriedades comuns ou a métodos, a informação da ação poderia ficar implícita na ferramenta, deixando livre deste trabalho o autor de documentos.

4 – Ausência de meios para especificar os valores de cada parâmetro do método. Não existe nenhum método visual que auxilie o autor a especificar quais os valores para serem usados em cada parâmetro durante o uso do método. Um método pode possuir mais de um parâmetro e a especificação do valor para cada um em separado é importante.

No NCL Composer atualmente, a Figura 4.4 apresenta a abordagem empregada para especificar os valores dos parâmetros do método. A figura apresenta um elo cuja ação é o uso de um método. A modificação do valor a ser atribuído (valores dos parâmetros do método) é feita clicando duas vezes sobre o *bind*. A edição é feita pela janela de dialogo, que apresenta apenas o parâmetro do *bind* (**params**), referente ao valor a ser atribuído pela ação de atribuição.

A janela depois de preenchida e confirmada, irá gerar um **bindParam** para o *bind* de atribuição com o valor entrado pelo autor. Observa-se que não é possível especificar os valores caso o método possua mais de um parâmetro, nem mesmo saber quais são os parâmetros do método.

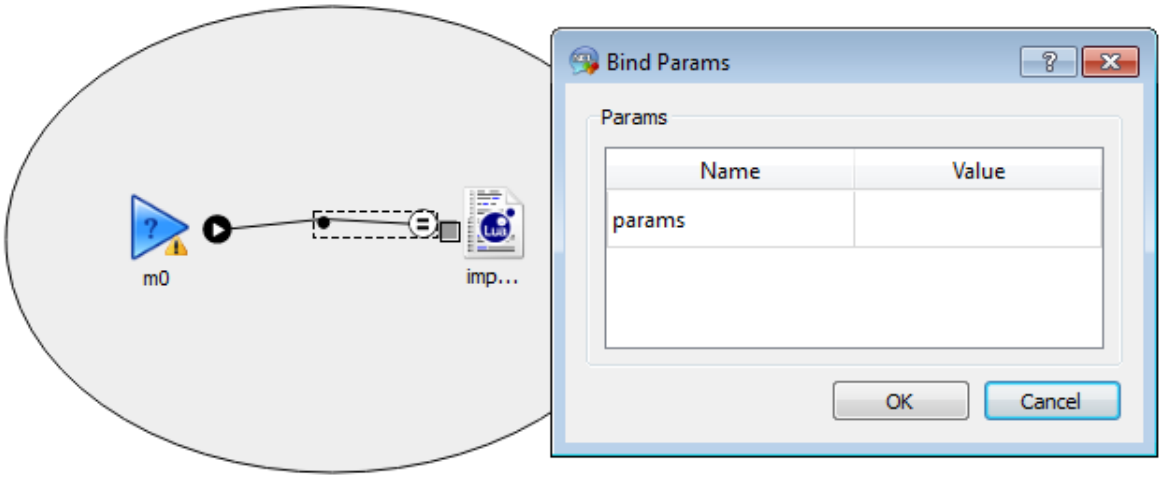


Figura 4.4 – Especificação dos valores dos parâmetros do método.

5 – Impossibilidade do uso de propriedades de outras mídias como valores de parâmetros. Ainda no problema anterior, não é possível especificar um valor para um determinado parâmetro que esteja presente em outras mídias. Por exemplo, às vezes é de interesse que o valor de um parâmetro advenha de uma propriedade presente em outra mídia ou variável global do documento.

6 – Omissão no uso dos valores de retorno dos métodos ou de variáveis na parte declarativa. Este problema tem relação com a ausência do suporte a ações do tipo **get** e **set** (passagem de valores) (SOARES e BARBOSA, 2009). Com este tipo de ações, é possível direcionar um valor de retorno de um método para outra propriedade em outra mídia.

Nas mídias imperativas, este cenário é constantemente utilizado quando queremos armazenar o resultado de algum processamento e enviá-lo para outra mídia, realizando assim a integração do código imperativo com o documento NCL. Outra função empregada é quando queremos realizar ações decorrentes de eventos específicos disparados dentro da mídia imperativa.

Atualmente, não é possível receber um valor de uma mídia imperativa e repassá-lo para outro elemento de mídia. A Figura 4.5 exemplifica esta situação ao não suportar o direcionamento do valor retornado pela mídia para ser reusado em outra mídia. As ações neste elo são disparadas quando o evento de final de atribuição ocorre na propriedade da mídia **m1**. Entretanto na janela de dialogo de criação do elo, não está presente este tipo de evento. O elo

criado entre a propriedade que retorna o valor da mídia **m1** e a propriedade da mídia **m2** indica que o autor de documentos quer repassar os valores entre estas propriedades.

Identifica-se aqui uma semelhança com o terceiro problema, necessitando que o autor saiba como repassar o valor para outra mídia. Essas informações estão implícitas na visão e podem ser geradas automaticamente sem a necessidade de interação do usuário.

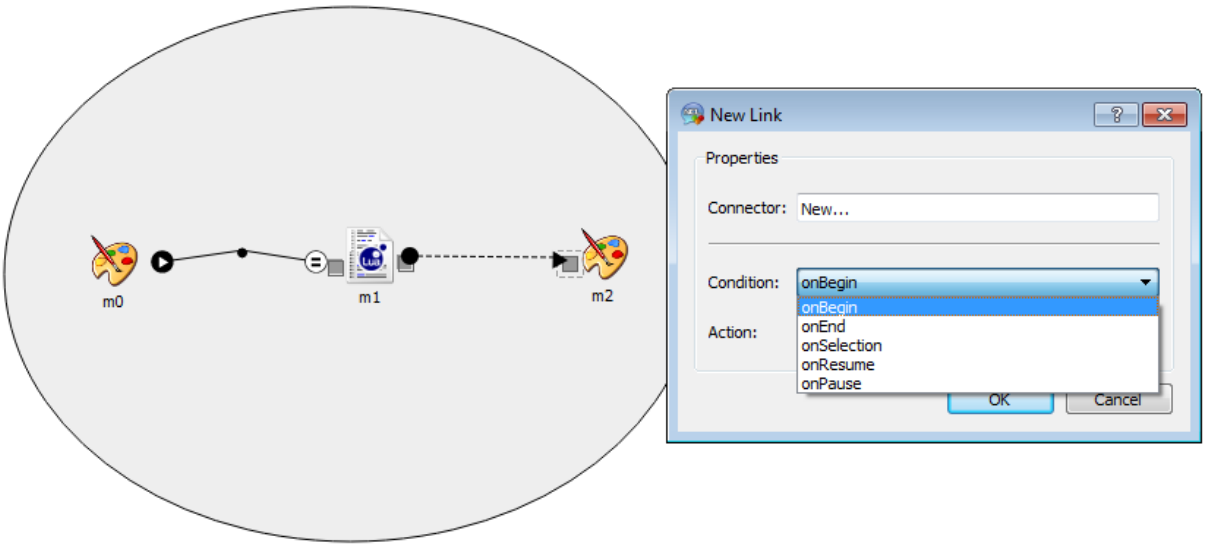


Figura 4.5 – Omissão do tratamento do evento de final de atribuição.

Como observado, alguns problemas ainda persistem quando integrando objetos de mídia imperativos na visão estrutural do NCL Composer. Estes problemas impactam no desenvolvimento rápido e simples do documento NCL utilizando objetos de mídia imperativa. Nesta dissertação, é proposta uma extensão da visão estrutural para adequá-la as situações envolvendo o uso rápido e simples do conteúdo imperativo na linguagem Lua em aplicações NCL. A Tabela 4.1 apresenta a relação entre os problemas a serem endereçados e a solução proposta.

Tabela 4.1 – Relação entre os problemas e a solução proposta.

Problema	Descrição	Solução
1	Falta de exposição do conteúdo imperativo	Emprego de anotações do código Lua para rastreamento de métodos e variáveis

2	Falta de identidade visual ao empregar propriedades e ação de atribuição como forma de uso do conteúdo imperativo	Criação de novos elementos visuais: • Para representar o conteúdo imperativo; • Ação para usar o conteúdo; • Condição para receber valor de retorno;
3	Necessidade de o autor informar a ação quando usando um método ou variável	Criação automática a partir do conhecimento sobre o conteúdo imperativo
4	Ausência de meios para especificar os valores de cada parâmetro do método	Método visual para especificar o valor para cada parâmetro existente no método

5 EXTENSÃO DA VISÃO ESTRUTURAL

Este capítulo apresenta a proposta destinada a resolver os problemas propostos anteriormente para assim tornar mais rápida e simples a integração de código imperativo em aplicações NCL. A primeira seção explicita como se dá o processo de autoria das aplicações NCL, quando, devido à natureza da aplicação em questão, demanda a necessidade do autor de documentos integrar um código imperativo desenvolvido por um programador. A seção 5.2 descreve os principais requisitos envolvidos para que a integração seja feita de forma rápida e simples e estes servem como base para propor as extensões que são apresentadas na seção 5.3. As seções seguintes apresentam como seria o uso das mídias imperativas (seção 5.4) e de serviços web (seção 5.5) utilizando as propostas das extensões da visão estrutural.

5.1 Visão Geral do Processo de Autoria

Os universos declarativo e imperativo envolvem normalmente a presença de profissionais com capacidades e interesses distintos. A NCL, ao contrário das linguagens imperativas, é facilmente compreendida por autores que possuem domínio sobre uma linguagem de marcação (por exemplo, HTML) para produção de conteúdo audiovisual. Em contra partida, a utilização de código imperativo é complexo demais para um autor de documentos, exigindo a presença de um profissional que tenha domínio em programação. Logo, a utilização de mídias imperativas em uma aplicação NCL por autores sem conhecimento em programação passa a ser um entrave.

Segundo (SOARES e BARBOSA, 2009), um fator importante na integração de linguagens declarativas e imperativas é que haja o mínimo de intercalação entre seus

domínios de modo a simplificar a divisão de tarefas entre equipes de profissionais técnicos e não-técnicos em linguagens de programação. O profissional não-técnico ao reusar uma mídia imperativa deve fazê-lo de forma simples, sem que seja necessário conhecimento sobre a implementação interna do código imperativo ou conhecimento na linguagem imperativa.

O desenvolvimento de uma aplicação NCL exige a interação de três profissionais distintos: o designer, o programador de código imperativo e o autor de documentos NCL. Os dois primeiros profissionais atuam como produtores de conteúdo (cada qual com seu tipo) de modo a cumprir os requisitos da aplicação.

Os conteúdos de mídia (imagens, trilha sonora, textos, etc.) são artefatos criados de maneira criativa pelo profissional de mídia *designer*, o qual utiliza normalmente suas próprias ferramentas de interesse. Além disso, em algumas aplicações com características mais dinâmicas, é preciso a produção do código imperativo que é desenvolvido pelo programador imperativo por meio da linguagem NCLua (SANT'ANNA, CERQUEIRA e SOARES, 2008). As mídias imperativas NCLua cumprem um papel na aplicação de realizar tarefas que necessitam da especificação de algoritmos e estruturas de dados que não estão disponíveis na linguagem NCL (SOARES, RODRIGUES, *et al.*, 2010).

Estes dois tipos de conteúdo são apenas artefatos que até o momento da sua criação ainda não fazem parte da aplicação. Cabe ao autor de documentos integrá-los no documento NCL.

A Figura 5.1 apresenta como ocorre a relação entre o autor de documentos e o programador Lua durante a autoria da aplicação NCL. Ao surgir uma demanda por processamento imperativo na aplicação, o autor de documentos cria uma mídia imperativa para satisfazer essa necessidade (passo 1). Para isso, ele importa o código imperativo e a sua especificação que foram previamente criados pelo programado Lua.

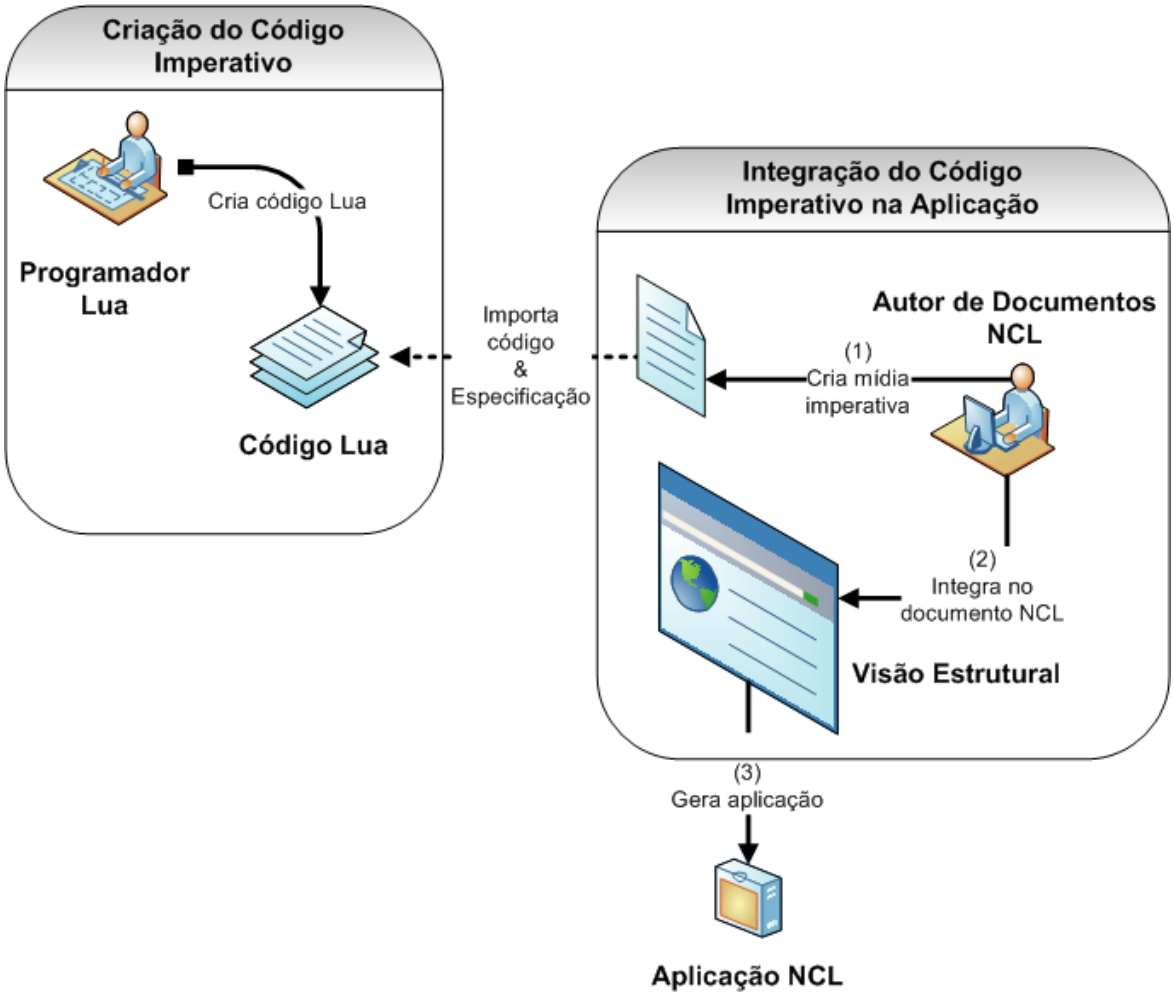


Figura 5.1 – Processo de autoria para integração de código imperativo.

Em posse da mídia, no passo 2, o uso da parte imperativa presente no código é feito na aplicação. Utilizando normalmente a visão estrutural como abordagem de autoria, a relação do documento NCL com o código imperativo ocorre em dois sentidos: do documento NCL para a mídia, através da execução de métodos e leitura de variáveis (conteúdo da mídia imperativo) e da mídia para o documento NCL, quando o resultado de um processamento é utilizado dentro do documento NCL.

A execução de um método Lua deve ser feito sem que o autor de documentos conheça a linguagem Lua. Quando o programador Lua cria o código Lua o ideal é que ele crie a especificação do código de modo a ser utilizada pelo autor para descobrir quais os métodos e variáveis que estão disponíveis no código. Ela descreve apenas os nomes e os propósitos dos métodos e variáveis presentes no código.

A especificação ajuda o autor na tarefa de mapear o conteúdo presente no código Lua (métodos e variáveis) para a mídia imperativa. Assim, durante o uso da mídia no documento,

o autor foca apenas no que o método faz, não sendo necessário conhecer detalhes de implementação.

Cada artefato criado (seja ele uma elemento de mídia ou o código imperativo) anteriormente é representado na visão estrutural como objetos de mídias NCL. De posse destes objetos, a tarefa do autor de documentos é definir a aplicação reutilizando cada artefato e compondo com cada objeto de mídia. A visão estrutural (seção 3.4) auxilia nessa etapa por ajudar a compor rapidamente os objetos.

5.2 Requisitos

A rapidez e simplicidade da autoria do documento NCL constitui-se requisitos importantes para o reuso de mídias imperativas na ferramenta de autoria. Essa busca envolve principalmente abstrair do autor de documentos a complexidade envolvida no uso deste tipo de mídia. A visão estrutural oferece um passo para a rapidez na composição da aplicação, entretanto, como descrito no capítulo 4, ela não se mostrou adequada e intuitiva o suficiente para a inclusão de mídia imperativa de forma rápida pelo autor de documentos.

Um código imperativo pode ser incluído na visão estrutural definindo um elemento de mídia com conteúdo composto por códigos na linguagem imperativa Lua (*script*). Um elemento de mídia imperativa é criado pelo programador como uma entidade independente do documento NCL, mas que pode ser referenciado pelo domínio declarativo. Como descrito na seção 2.3.1, para acessar o código imperativo é definido um mecanismo que consiste na representação em NCL das estruturas e definições que deverão ser usadas do código imperativo, como por exemplo, métodos e variáveis.

Essa tarefa gera uma dependência para o autor de documentos que precisa ter conhecimento detalhado do código da mídia imperativa que ele pretende utilizar. Por exemplo, é necessário definir quais trechos de código serão úteis e relacioná-los com os elementos do documento NCL. Normalmente, o autor de documentos é auxiliado por quem programou a mídia imperativa, mas isto nem sempre é possível.

Devido às características do autor de documentos, é importante abstrair o conteúdo da mídia imperativa para que o mesmo possa acessá-la de maneira simples e rápida. A filosofia adotada neste trabalho faz um paralelo com o conceito de componentes, onde o reuso de um componente requer apenas o conhecimento da sua interface e abstrai a implementação interna do componente (SZYPERSKI, 1999).

Além dos elementos de mídia imperativos já conhecidos, é comum na aplicação a presença de mídias imperativas que acessam serviços web da internet. Isso permite suportar cenários do uso de serviços web em aplicações NCL. A integração destes serviços demanda a criação de mídias imperativas exclusivas para comunicação com os serviços web.

Com o objetivo de suportar esse cenário, os serviços web são suportados como mídias imperativas, permitindo reusar as funcionalidades de um serviço web qualquer.

A partir dos problemas identificados no capítulo 4, foi identificado um requisito importante para o reuso de mídias imperativas na visão estrutural:

- Permitir a **rápida ligação** com o código presente na mídia imperativa, facilitando a **identificação** e **uso** do conteúdo da mídia imperativa como **métodos** e **variáveis**, que podem ter ou não relação com o documento NCL;

5.3 Extensões Propostas

O objetivo das extensões é simplificar e tornar rápido o uso da mídia imperativa e assim solucionar os entraves descritos nos problemas do capítulo anterior. Assim, foram definidos quatro pontos principais de extensões para a visão estrutural presente no NCL Composer 3:

- (1) criar um novo elemento visual agregado à mídia imperativa que represente o conteúdo (métodos, retorno de valores e variáveis) a ser acessado;
- (2) definir uma nova ação e um novo tipo de evento para melhor caracterizar o acesso ao conteúdo imperativo e sua relação com os outros elementos de mídia presentes da visão estrutural;
- (3) uso automático do conteúdo imperativo:
  - (3.1) exposição automática do conteúdo imperativo por meio do rastreamento dos métodos e variáveis presentes na mídia imperativa;
  - (3.2) criação automática das ações e eventos para acessar o conteúdo;
- (4) integração de serviços *web* a partir de uma mídia imperativa;

A primeira extensão é responsável por auxiliar o autor de documentos a identificar facilmente que conteúdo está presente na mídia. O principal objetivo desta extensão é proporcionar uma distinção quanto ao tipo de conteúdo, facilitando o acesso rápido e simples ao código-fonte imperativo.

Realizar uma distinção quanto ao tipo de conteúdo facilita o autor de documentos a descobrir rapidamente com qual conteúdo ele esta lidando, como também ajuda a distinguir as propriedades da mídia. Como visto no capítulo 4 (problema 2), na visão estrutural, representar o conteúdo por meio das propriedades ocasiona diversos problemas, sendo um deles, a questão da mesma aparência visual gerar uma dificuldade em distinguir se uma propriedade está representando um método, um retorno de um valor ou uma variável.

Este problema é resolvido com a representação do código a ser acessado (método ou variável) por meio de duas estruturas visuais: (1) pino de entrada e (2) pino de saída. Ambos os elementos visuais são agregados ao próprio objeto de mídia. O **pino de entrada** é definido para representar cada possível trecho de código imperativo que pode ser referenciado pelo documento NCL, por exemplo, métodos e variáveis. O pino de entrada possui uma seta com o sentido para dentro da mídia, indicando o acesso ao conteúdo interno.

Complementar ao pino de entrada, outro elemento visual denominado **pino de saída** é definido para representar o retorno de valores dos métodos e variáveis. Ele é usado para poder relacioná-los com outros objetos de mídia da visão estrutural. O pino de saída possui uma seta com sentido externo a mídia, indicando que por ele é possível capturar o retorno de um método. A cor diferente e os sentidos das setas nos pinos ajudam o autor a distinguir os tipos de pinos.

Os pinos de entrada e de saída reforçam a ideia do fluxo de dados envolvendo o relacionamento entre a mídia imperativa e o documento NCL. O sentido nos pinos indica que os objetos de mídia retornam valores (usando os pinos de saída) e realizam procedimentos (usando os pinos de entrada) por meio de ações requisitadas a partir de objetos de mídia presentes na visão.

A outra ideia por trás dos pinos é que eles possibilitam a exposição dos métodos (conteúdo) existentes no código-fonte da mídia imperativa. Assim como é feito com as propriedades, eles facilitam a identificação do conteúdo. Mais ainda, os pinos resolvem o segundo problema (capítulo 4), ao resolver a questão da falta de identidade visual das propriedades.

A Figura 5.2 apresenta uma mídia imperativa com seus três pinos de entrada e um pino de saída, que servem para mapear o conteúdo da mídia. Internamente à mídia, o conteúdo Lua é composto de dois métodos (**método 1** e **método 2**) e uma variável (**variável 1**) prontos para serem usados pela parte declarativa. O **método 1**, quando executado, retorna um valor para a parte declarativa, já o **método 2** não possui nenhum valor de retorno.

O conteúdo é acessado por meio dos pinos. Dessa forma, cada pino de entrada ou saída são mapeados para a sua respectiva estrutura (método ou variável) na mídia imperativa. Por padrão, um pino de saída está sempre associado ao retorno de um método ou de uma variável. Logo, se um método retorna mais de um valor (seção 2.3.1), um ou mais pinos de saídas devem ser criados, com o objetivo de acessar cada valor de retorno do método. Na Figura 5.2 o pino de entrada (**pino de entrada 1**) esta mapeado para um método que retorna um valor, logo se faz necessário um pino de saída.

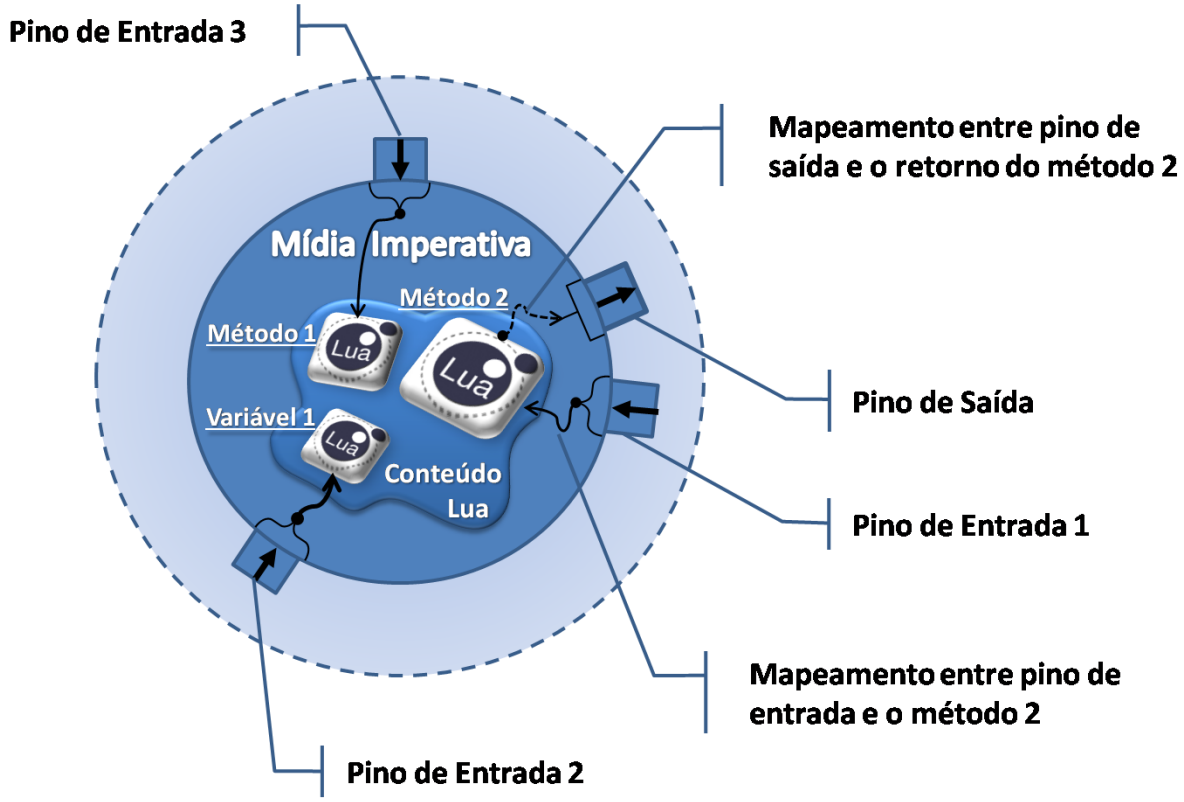


Figura 5.2 – Mapeamento entre os pinos e o conteúdo da mídia imperativa.

Um par de pinos (um pino de entrada e outro de saída) pode estar associado a um determinado método ou variável. Se associado com um método que não retorna valores o pino de saída não se faz necessário. Os pinos de entrada e saída em conjunto formam uma espécie de interface de comunicação com os objetos de mídias externos.

Os pinos servem para estabelecer uma interface de comunicação com os objetos de mídia, tudo isto para estabelecer uma ligação com o código interno do objeto imperativo. Como a visão estrutural modela visualmente o relacionamento entre os objetos de mídia declarativo e imperativo da aplicação, expor visualmente o acesso ao conteúdo disponível no objeto imperativo na visão estrutural pode facilitar a tarefa do autor.

A forma como os pinos são criados a partir o código-fonte, ou seja, o mapeamento entre os pinos e o código-fonte, é descrito na terceira extensão utilizando o conceito de anotações do código NCLua.

Além da representação visual do conteúdo da mídia (pinos), a segunda extensão criada para a simplificação é relacionada à forma como é feito o acesso ao conteúdo. Para isso, ela define um novo tipo de ação e de evento exclusivamente para tratar o acesso aos métodos dos objetos de mídia imperativos. O objetivo é melhor caracterizar o uso do conteúdo imperativo e sua relação com outros elementos de mídia. No tocante a atribuição de valores a variáveis presentes na mídia a ação de atribuição continua valendo.

Na visão estrutural os objetos de mídia imperativos, assim como os objetos de mídia da NCL, são representados como uma entidade visual que possui os eventos padrões da NCL (Tabela 2.1) como também as ações padrões de mídia (Tabela 2.2). Por ser uma mídia com um conteúdo diferente, referente ao código imperativo, um novo tipo de evento e ação foi definido para tratar a integração com este tipo de mídia.

A nova ação denominada **chamada de método** (*callMethod*) permite ao autor de documentos executar um método pertencente a um objeto de mídia imperativa. A execução pode envolver a presença de parâmetros, dessa forma a visão deve facilitar a especificação dos valores para cada parâmetro. Esta ação abstrai todo o código NCL necessário para realizar a execução de um método, o qual envolve o conhecimento do método a ser executado, a especificação dos valores dos parâmetros e o tratamento do valor de retorno do método. A Figura 5.3 representa o ícone escolhido para representar a ação de chamada de método. Na ação a parte exterior do círculo é clara e o símbolo interno escuro, seguindo assim o mesmo padrão de aparência dos ícones de eventos utilizados na ferramenta NCL Composer.

Figura 5.3 – Ícone para representar a ação de chamada de método.

É comum no uso de um método a passagem de parâmetros (valores) para serem utilizados durante a execução do método. Estes valores são passados pela parte declarativa e é utilizado para parametrizar o processamento imperativo. Dependendo da quantidade de

parâmetros do método, a visão deve oferecer a possibilidade de especificar valores diferentes para cada parâmetro em separado.

No caso do uso de variáveis, o autor sempre está restrito a especificar apenas um valor que será atribuído a variável em questão. Por exemplo, variáveis no código armazenam apenas um único valor, assim o autor especifica o valor que será atribuído.

A especificação do valor para um determinado parâmetro pode ser feita de duas formas distintas: (1) na primeira o valor pode ser uma *string* formada por letras e/ou números; (2) na segunda o valor pode ser o conteúdo presente na propriedade de uma mídia. O segundo caso abre possibilidades maiores para parametrização do comportamento da aplicação, usando valores presentes em outras mídias, sejam elas de conteúdo, imperativas ou de serviços web.

Primeiramente, a interface visual utilizada para especificar os parâmetros da chamada do método, orienta o autor mostrando a definição do método com seu nome e parâmetros. Em seguida, para cada parâmetro do método uma tabela possibilita ao autor especificar qual a forma utilizada na especificação do valor, se por um valor (*string*) ou por um valor presente em outra mídia (mídia). Se o autor escolher por valor, um campo de entrada de texto é disponibilizado, caso o autor escolha por mídia é disponibilizado para ele as propriedades de uma determinada mídia presente na visão estrutural.

O novo evento denominado ***onValueReturn*** (quando valor for retornado) permite disponibilizar para a parte declarativa o valor presente no pino de saída.O valor é obtido a partir do processamento realizado pelo método e disponibilizado para outras mídias utilizarem.

A Figura 5.4 apresenta o ícone utilizado para este novo tipo de evento. No evento a parte exterior do circulo é escura e a parte interna clara, seguindo o padrão de aparência dos ícones de eventos utilizados na ferramenta NCL Composer.

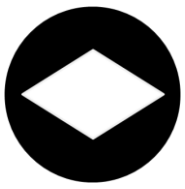


Figura 5.4 – Ícone para evento do tipo *onValueReturn*.

Unificando as duas extensões anteriores, tem-se uma nova abordagem para uso de mídias imperativas. A Figura 5.5 apresenta o uso de um método que retorna um valor para a parte declarativa. O *bind* do **elo 1** especifica uma ação de chamada de método, descrita pelo

novo ícone, sobre o pino de entrada da mídia imperativa. Utilizando o evento ***onValueReturn*** (*bind* de condição do **elo 2**) sobre o pino de saída é possível receber o valor do processamento do método e reusá-lo na parte declarativa. Como visto mais adiante, tanto a ação como o evento para estes *binds* são determinados automaticamente quando no momento da criação dos elos.

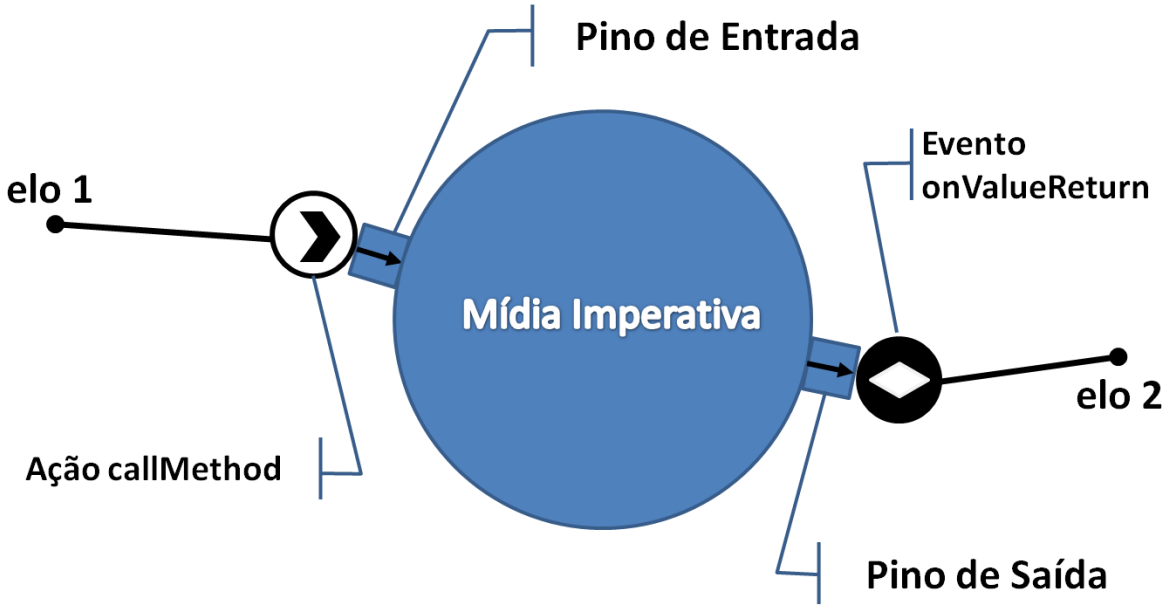


Figura 5.5 – Abordagem para uso de uma mídia imperativa.

É preciso dizer que a criação destes novos elementos (pinos, ações e eventos) não impõe mudanças na linguagem NCL. Eles são uma notação visual para tratar de outro aspecto o conceito existente na NCL de usar um conteúdo imperativo por meio da ação de atribuição sobre a propriedade que se refere ao método. Em outras palavras a nova notação visual é mapeada diretamente para os elementos existentes na linguagem NCL.

Continuando com as extensões propostas, a terceira extensão possibilita automatizar muitos dos passos que são necessários para acessar o conteúdo imperativo. O objetivo é realizar a criação automática tanto dos pinos de entrada e de saída como das ações e eventos que usam o conteúdo da mídia.

A criação automática dos pinos é feita para expor ao autor de documentos quais os métodos e variáveis estão acessíveis para serem usados, sem que o mesmo precise se preocupar com a criação dos mesmos. Dessa forma, o autor de documentos não se ocupa com a questão de criar as estruturas que representam os métodos e nem as ações que usam os métodos, resolvendo assim o primeiro e o terceiro problema (capítulo 4).

Cada tipo de conteúdo da mídia (método ou variável) deve ser mapeado para um pino. Para facilitar este processo, propõe-se um esquema de anotações para serem usadas junto ao código Lua. As anotações são inseridas explicitamente no código imperativo e relacionadas diretamente com as estruturas do código como métodos e variáveis. Dessa forma, a descoberta do conteúdo disponível em uma mídia imperativa é feita de forma automática.

Um método é anotado usando a anotação *@Method* e uma variável com *@Variable*. Ambas são inseridas como um comentário Lua antes da declaração de ambos. A Figura 5.6 e a Figura 5.7 apresentam as anotações do método e da variável respectivamente. A anotação do método e da variável deve acompanhar uma definição de outros atributos como o nome do método ou da variável em questão (**name**), zero ou mais parâmetros do método (**params**) separados por vírgulas, zero ou mais valores de saída (**outputValue**) separados por vírgulas (caso o método necessite retornar um ou mais valores) e por fim uma breve descrição da utilidade do método (**description**).

```
@Method(name=”methodName” params=”param1, param2, ...,
        paramN” outputValue=”outputValue1, outputValue2, …,
        outputValueN” description=”brief description”)
```

Figura 5.6 – Anotação do tipo @Method para ser utilizada em métodos.

```
@Variable(name=”variableName” outputValue=”outputValue”
        description=”brief description”  )
```

Figura 5.7 – Anotação do tipo @Variable para ser utilizada em variáveis.

Utilizando os atributos da anotação *@Method* um pino de entrada e vários pinos de saída podem ser criados. Como descrito anteriormente, um método Lua pode retornar um ou mais valores para a parte declarativa. Os valores ao invés de serem retornados em um único evento, eles são retornados em eventos separados, sendo um evento diferente para cada valor. Sendo assim, para cada ***outputValue*** existente na anotação, um pino de saída é criado e por ele o valor pode ser lido. Na anotação o atributo ***name*** é utilizado para nomear o pino de entrada.

No caso da anotação *@Variable* é mapeada para um pino de entrada e um pino de saída. O pino de entrada é criado com o nome do atributo da anotação e o pino de saída com o atributo ***outputValue***. O pino de saída é utilizado para receber o valor contido na variável.

Com as anotações postas no código-fonte, tarefa esta feita pelo programador imperativo, a mídia imperativa torna-se passível de ser integrada rapidamente pelo autor de documentos utilizando a visão estrutural.

A Figura 5.8 apresenta um exemplo de código-fonte Lua como conteúdo imperativo e descreve como é feito o mapeamento entre as anotações e os pinos do objeto de mídia. A mídia na figura possui dois pinos (um de entrada e outro de saída) e ilustra o caso de execução de um método e retorno de um valor do mesmo para ser usado na visão.

O código-fonte Lua define o método **getValue** (linha 2) que possui o parâmetro **tag** e retorna um valor para a parte declarativa. Este valor esta presente no arranjo **tabela** (linha 3) e é acessado utilizando o parâmetro de entrada como índice. Por ser de interesse para outros objetos de mídia, este método é anotado com a anotação de método @Method (linha 1). A anotação criada possui o nome do método **getValue**, o único parâmetro **tag** e o único evento de saída **tagValue**.

Como visto na figura, a anotação como um todo é mapeada para o pino de entrada e o atributo **outputValue** é mapeado para o pino de saída. O valor de saída **tagValue** presente na anotação, indica o pino de saída que irá retorna o valor presente na tabela. Nesta circunstancia um pino de entrada é criado com nome **getValue** e um pino de saída com nome **tagValue**.

Como já dito, o valor de retorno do método é enviado para a parte declarativa como um evento. O envio deste evento sempre segue um mesmo padrão de código (seção 2.3.1). Ao invés de colocarmos este código dentro do método, é proposta uma diretiva para organizar o código e tornar mais entendível o envio do evento.

A diretiva criada tem como objetivo encapsular o código responsável pelo envio do evento em um método Lua. O método segue um padrão de nomeação começando com ***return*** e seguindo pelo nome do evento de saída (**returnNomeDoEventoDeSaida(valor)**). No seu corpo a implementação envia o evento especifico para o evento de saída. Este evento é enviado usando uma chamada para este método em qualquer lugar. Por exemplo, o evento de saída **tagValue** é enviado na linha 4 através da chamada ao método **returnTagValue** (implementado na linha 6).

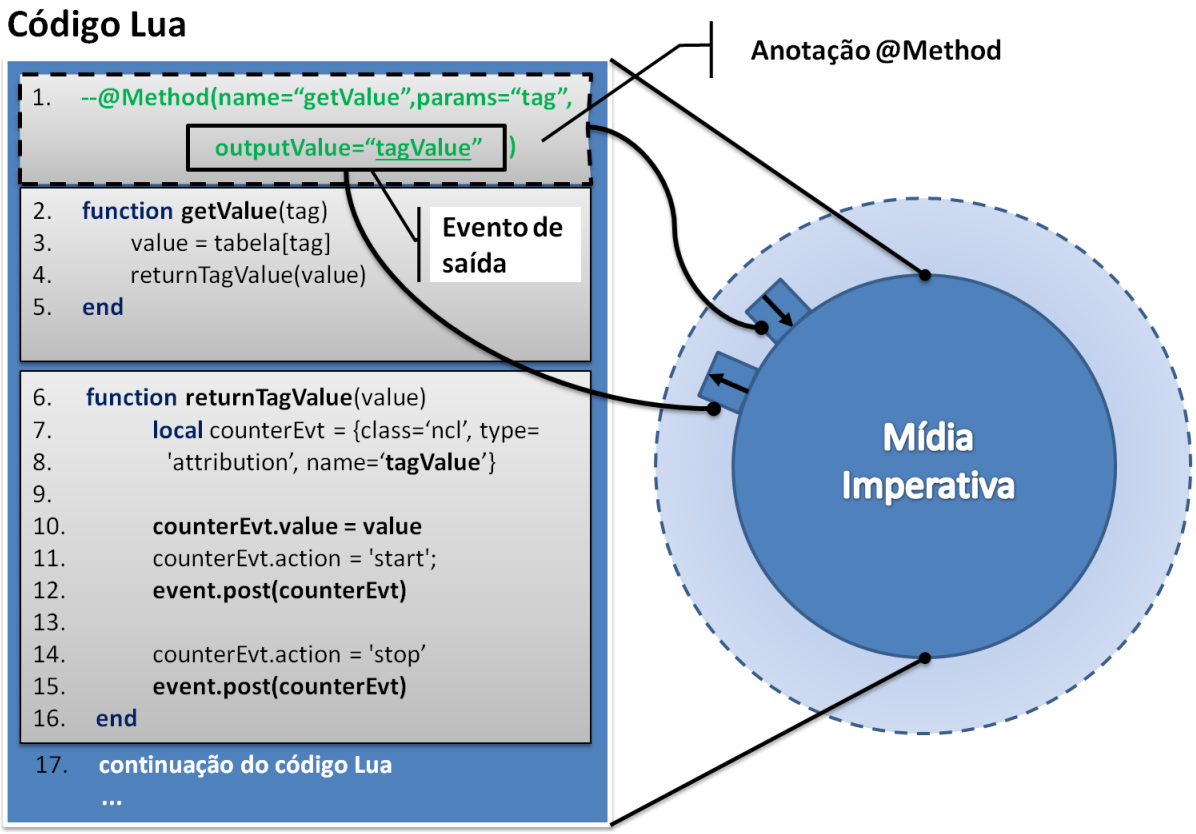


Figura 5.8 – Mapeamento entre as anotações do código Lua e os pinos da mídia imperativa.

O mecanismo de anotações do código resolve o terceiro problema (capítulo 4), pois oferece um meio para expor automaticamente os métodos e variáveis presentes na mídia. Através de cada anotação um pino de entrada e os pinos de saída são descobertos sem necessidade do autor documentos.

Além da criação das estruturas que representam os métodos, as ações e eventos para usar o conteúdo são determinados automaticamente quando no momento da criação dos elos, evitando do autor de documentos especificá-los. A diminuição do tempo na autoria é conseguida pela geração automática da forma de uso do conteúdo existente na mídia imperativa. Quando se ligando com um pino de entrada (método) ou recebendo o valor do pino de saída do objeto de mídia imperativo, a escolha tanto da ação quanto do evento é feita de maneira automática pela visão estrutural. Dessa forma, evita-se a interação do usuário em escolher qual o tipo de ação ou evento a ser realizado.

Na seção seguinte, é apresentado o uso (na perspectiva do autor de documentos) de um código imperativo como um objeto de mídia imperativo na visão estrutural.

5.3.1 Usando Objeto de Mídia Imperativo

O processo de criação da mídia imperativa ainda é idêntico ao existente no NCL Composer, a única diferença é que no momento da escolha do arquivo do código-fonte, é feito a descoberta automática dos métodos e variáveis utilizando as anotações existentes no código. Para cada anotação é criado os pinos de entrada e saída da mídia, assim o conteúdo da mídia imperativa pode ser identificado e usado facilmente pelo autor de documento.

A Figura 5.9 apresenta o uso de um método do objeto imperativo semelhante ao que foi utilizado no exemplo da visão estrutural na seção 3.4. A visão demonstra como é o uso de um código Lua que possui um método com um parâmetro e que retorna um valor para a parte declarativa.

Tomando como base a definição do código-fonte Lua presente na Listagem 4, a anotação (linha 1) é criada com os atributos nome como **method**, parâmetro como **param** e o valor de retorno do método como **value**. O método **method** chama o método **returnValue** para enviar o evento para o NCL com o valor que foi processado no método. Com isso a mídia fica pronta para ser usada pelo autor de documentos.

Após o código-fonte Lua ser indicado para a mídia imperativa **m5**, é criado automaticamente o pino de entrada e de saída, representando respectivamente o método do código e o valor de retorno do mesmo (Figura 5.9). A representação visual dos pinos possui a cor vermelho e as setas indicando sentido interno à mídia no caso do pino de entrada e externo no caso do pino de saída. O autor do documento diferencia o pino de entrada e pino de saída a ser usado para executar o método a partir da sua representação visual. Cada pino também possui o seu nome para ajudar na decisão.

O autor do documento estabelece a ligação com o método Lua escolhendo o pino de entrada a ser usado para executar o método. A ligação procede criando um elo que tem como condição o inicio da mídia **m2** e ação de chamada de método (*callMethod*) sobre o pino de entrada. Para criar este elo o autor de documentos informa apenas qual a condição que irá disparar a ação do elo. A ação de chamada de método é determinada automaticamente pela visão. Na Figura 5.9 o uso do método Lua **method** ocorre pelo *bind* de ação de chamada de método (*callMethod*) pertencente ao elo que relaciona a mídia **m2** e o pino de entrada da mídia imperativa **m5**.

Da mesma forma, um valor de retorno de um pino de saída pode ser obtido para ser usando pela parte declarativa. O valor de retorno do método geralmente é utilizado pelo autor na visão para armazenar em uma propriedade de uma mídia para utilizá-lo futuramente. A Figura 5.9 apresenta este caso em que um valor é obtido da mídia através do evento

onValueReturn sobre o pino de saída da mídia e em seguida repassado para ser armazenado na propriedade da mídia **m3**. A criação tanto da condição como da ação deste elo são feitos automaticamente pela visão.

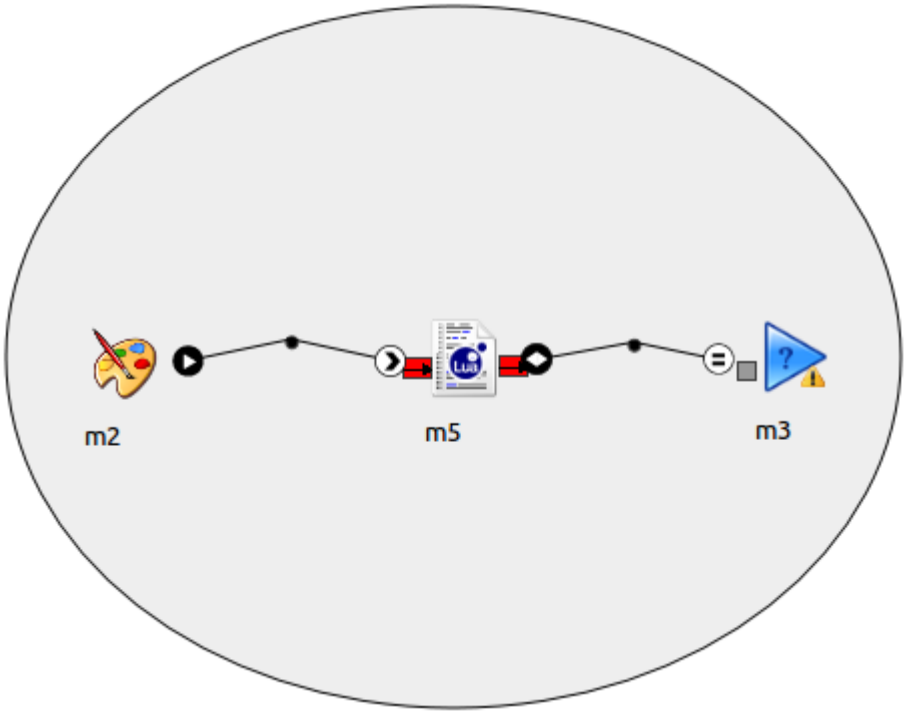


Figura 5.9 – Visão estrutural utilizando uma mídia imperativa com um pino de entrada e outro de saída.

```
1.      --@Method(name="method" params="param"
2.      --      outputValue="value")
3.      functionmethod(param)
4.              ...
5.              returnValue(valueProcessed)
6.      end
7.
8.      function returnValue(aValue)
9.              local outputEvent = {
10.             class= 'ncl',
11.             type = 'attribution',
12.             name  ='value',
13.             }
14.             --sinaliza documento NCL do resultado processado
15.             outputEvent.value= aValue
16.             outputEvent.action ='start'; event.post(outputEvent)
17.             outputEvent.action= 'stop';  event.post(outputEvent)
18.     end
        ...
```

Listagem 4 – Código Lua com método que retorna um valor para o NCL.

As funcionalidades dos serviços web também são reusadas em termos dos objetos de mídia imperativa, abaixo segue uma descrição de como eles são utilizados pela visão.

5.4 Objetos de Mídia para Integração com Serviços Web

Além dos objetos de mídia representando as mídias imperativas, a visão estrutural é estendida para disponibilizar objetos de mídia com suporte aos serviços web. Eles são responsáveis por adicionar rapidamente suporte aos serviços da internet bem como de algum canal de retorno (servidor web) para a aplicação.

Um objeto de mídia de serviço web assemelha-se a um objeto de mídia imperativo. Ele possui as mesmas características de uma mídia imperativa ao disponibilizar execução de código imperativo através dos pinos. A diferença é que uma mídia de serviço realiza a execução de procedimentos a partir de serviços web. Por ser semelhante a uma mídia imperativa, a mídia de serviço web disponibiliza acesso a serviços na web utilizando a mesma interface encontrada na mídia imperativa. Com isso, o autor de documentos não nota muita diferença ao reusar um serviço na visão estrutural.

Para proporcionar essa característica, a mídia de serviço é composta por duas entidades: a primeira é uma mídia imperativa necessária para representar em termos de pinos de entrada e saída os métodos existentes no serviço web e uma segunda responsável pelo acesso ao serviço, que é realizado pelo módulo cliente do serviço web.

A mídia imperativa tem o papel de relacionar pinos de entrada e ou saída para cada método presente no serviço web, e assim poder exportar o serviço para entidades externas. Além disto, a mídia de serviço possui internamente uma biblioteca cliente, que é responsável pela comunicação com o serviço.

A Figura 5.10 apresenta o esquema de funcionamento da mídia de serviço. Semelhante a uma mídia imperativa, ela possui os pinos de entrada e saída, entretanto neste caso eles são mapeados para chamadas de métodos do serviço. As chamadas de serviço web são feitas pelo módulo cliente do serviço, que compreende uma biblioteca cliente Lua encapsulada como um módulo. O módulo é responsável em efetuar as requisições como também receber as respostas do servidor. Na figura o módulo cliente do serviço web possui três métodos que são responsáveis para comunicação com os métodos do serviço web. O **Método 2** representa um método de mesmo nome no serviço web e possui uma resposta associado à chamada, já o **Método 1** e o **Método 3** não possuem resposta associada às chamadas.

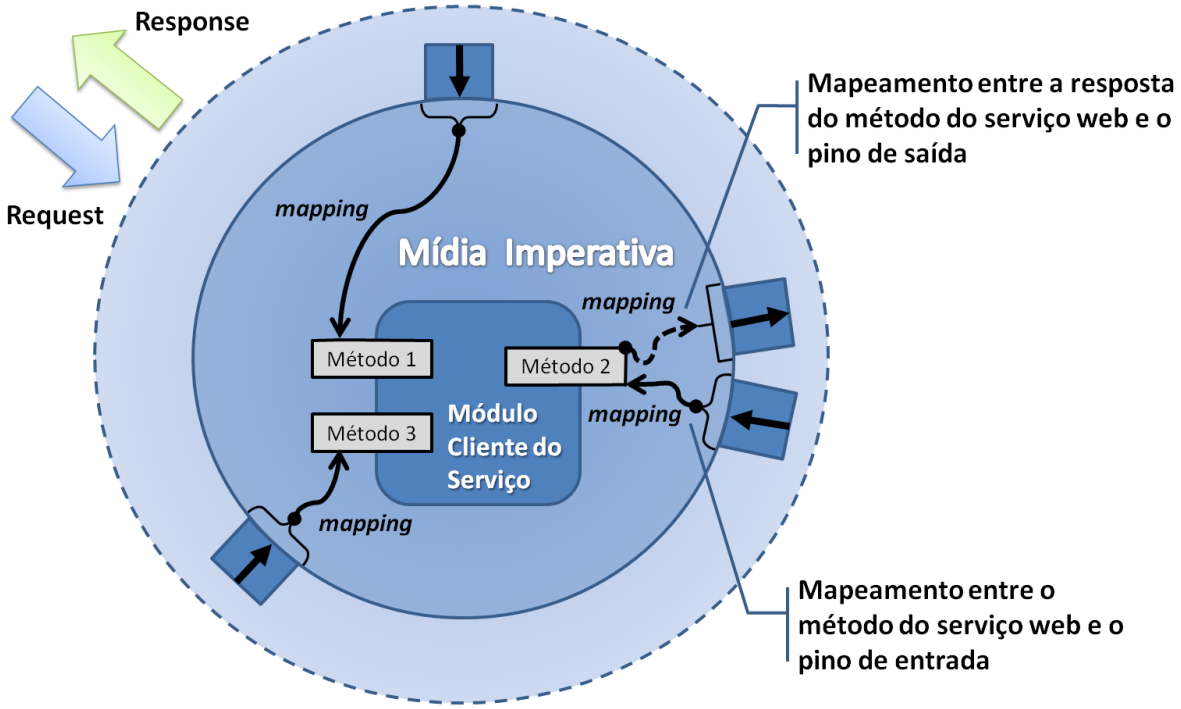


Figura 5.10 – Representação esquemática da mídia de serviço web.

Um pino de entrada é mapeado para um método do serviço, já um pino de saída é mapeado para a resposta do método. O pino de saída é especificado apenas no caso do método possuir alguma informação útil associada a sua resposta. Uma resposta pode possuir uma ou mais informações de interesse. No caso de se querer utilizar, na parte declarativa, mais de um tipo de informação presente na resposta, ao invés de criar um único pino de saída é criado um pino de saída para cada informação de interesse presente na resposta.

Por exemplo, um método do serviço web para conversão de taxa de moedas entre países retorna apenas um valor que representa o valor da moeda, logo neste caso um único pino de saída é preciso. Já em um método para identificar a localização geográfica baseada em IP, retorna diversos atributos de interesse como o país, a cidade, indicações se o IP está ativo, etc. Neste caso, um pino de saída é criado para cada informação de interesse.

A especificação dos parâmetros do método de uma mídia de serviço web procede da mesma forma que quando na especificação dos parâmetros do método de uma mídia imperativa.

A responsabilidade por realizar a requisição para o método do serviço é tratado em separado pelo módulo cliente de acesso ao serviço. Desta forma, o acesso ao servidor é isolado em um código separado da mídia imperativa. Como na mídia imperativa, os pinos

também cumprem o mesmo papel de representar uma interface para acesso aos métodos, entretanto a implementação do método agora se refere a uma requisição ao serviço web, a qual é feita em separada pelo módulo cliente do serviço. Como visto na figura, a seta azul de requisição (***request***) e a seta verde de resposta (***response***), de uma forma geral, concebem que a mídia do serviço web está realizando uma requisição e tratando uma resposta com um servidor web dentro a partir de um documento NCL.

5.4.1 Reusando Serviços Web

Para reusar as mídias de serviços web e permitir o uso dos métodos pelo autor de documentos foi definida uma nova visão no NCL Composer para disponibilizar um conjunto de serviços web. A nova visão, denominada **Aba de Serviços Web**, é responsável por disponibilizar de maneira intuitiva os serviços e os seus métodos. O autor reusa o serviço web a partir de um método encontrado nesta visão.

O objetivo da visão é: (1) auxiliar o autor na localização do método do serviço web que ele deseja utilizar e (2) criar automaticamente a mídia de serviço web para rapidamente reusar o método do serviço web no documento NCL. Depois de escolhido o método do serviço, o autor de documentos utiliza um mecanismo de arrastar-e-soltar para a visão estrutural de modo a efetivamente usar o método do serviço.

Quando o método é arrastado e solto na visão estrutural, é criado um novo objeto de mídia imperativo na visão estrutural para representar a mídia do serviço web. Além da mídia, também é criado o pino de entrada para representar o método escolhido na visão e dependendo se o método possui resposta, o pino de saída. Por meio destes passos, a mídia de serviço web passa a disponibilizar os pinos para realizar a ligação de outros objetos de mídia com o serviço desejado. Como visto anteriormente, isto é feito pela ação de chamada de método sobre os pinos de entrada da mídia, que se traduz no final como uma chamada de método ao servidor web.

Os serviços web por também serem tratados como mídias imperativas, permitem a rápida integração de serviços dentro da aplicação possibilitando chamadas que são feitas diretamente para os métodos do serviço web.

O autor de documentos pode querer reusar mais de um método do mesmo serviço na visão estrutural. Como dito anteriormente, se arrastar-e-soltar o método na visão estrutural uma mídia de serviço web é criada juntos com os pinos. Entretanto, quando reusando outro método do mesmo serviço, duas possibilidades existem: (1) soltar o método numa mídia de

serviço web já existente ou (2) soltar o método para criar uma nova mídia de serviço web com os pinos de entrada e saída.

No primeiro caso, para representar o acesso ao método escolhido um novo pino de entrada e de saída é criado junto à mídia de serviço escolhido. Já no segundo caso, é criada uma mídia de referencia para a mídia de serviço web e os pinos de entrada e saída adequados. Neste sentido, poderá existir mais de um objeto de mídia na visão estrutural representando o serviço web, entretanto no final todos os objetos de mídia (incluindo as referencias) irão referir-se a apenas uma mídia de serviço web.

A Figura 5.11 apresenta a visão da aba de serviços web referente a três provedores de serviços web bastante comuns na web Yahoo (YAHOO, 2013a), AppInventor (APPINVETOR, 2013) e WebServiceX (WEBSERVICEX.NET, 2013). Normalmente cada provedor de serviço web disponibiliza mais de um serviço web, como é o caso do Yahoo, que possui serviços web para busca, encontrar lugares, armazenamento de imagens, etc. e do Google que possui vários serviços web disponíveis para uso (GOOGLE, 2013).

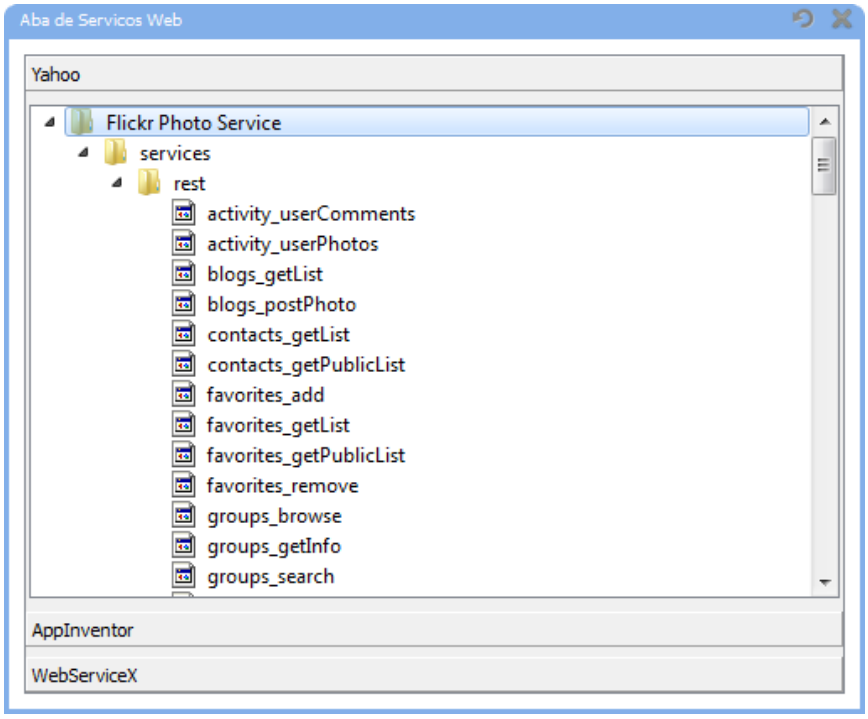


Figura 5.11 – Aba de serviços web exibindo os métodos presentes no serviço *Flickr*.

Por este fato, a aba disponibiliza os provedores de serviço como um painel deslizante abrigando numa árvore em particular cada serviço existente no provedor. Por exemplo, a Figura 5.11 apresenta no painel do Yahoo a árvore do serviço web do Flickr (YAHOO,

2013b). A árvore apresenta o recurso **services/rest** e os seus métodos como **ativity_userComment**, **activity_userPhotos**, etc.

Quando um painel do provedor é escolhido o conjunto de serviços web existentes são visualizados cada um em sua árvore exclusiva. Foi empregada uma visualização em forma de árvore por melhor adequar-se a representação hierárquica dos métodos e recursos (quando serviços *web Rest*) presentes nos serviços web (*Rest* ou *Soap*). A árvore do serviço representa tanto o serviço como os recursos do serviço, como um ícone de diretório (uma pasta fechada) e os métodos pertencentes ao recurso, como um ícone de uma folha.

A aba de serviços web é construída a partir de descrições criadas para informar os provedores de serviço web e a descrição dos serviços web. Para a descrição do serviço foi utilizado padrões já existentes na indústria, como é o caso do WADL e do WSDL, que são bastante utilizadas para a descrição das funcionalidades dos serviços, ou seja, quais são os seus métodos e respostas disponíveis para reuso. As descrições desenvolvidas na linguagem XML são apresentadas nas listagens a seguir.

Para facilitar a apresentação das descrições, será utilizado o nome **provider** para identificar um provedor genérico e **web service** como um nome de um serviço web qualquer presente na web. A descrição do provedor de serviços (Listagem 5) informa todos os provedores de serviços a serem incluídos na aba, além de cada serviço web disponível. O elemento **provider** descreve cada provedor em separado, possuindo como atributos o nome do provedor e o seu endereço. Cada provedor pode agrupar um ou mais serviços, o que é feito pela inclusão do elemento **service**, o qual possui o atributo *url* para indicar onde se encontra a descrição do serviço web. Por exemplo, na listagem a localização do primeiro serviço referente ao primeiro provedor é a url "/provider/webservice/Service1.xml". Um padrão de diretórios foi criado para organizar esta estrutura de armazenamento das descrições.

O diretório **service** armazena todos os provedores e cada provedor possui seu diretório exclusivo. Dentro do diretório do provedor, cada serviço web é também organizado em um diretório particular, o qual armazena a descrição do serviço web.

```
<saas-serviceprovider>
<provider name="name"url="url">
<service url="service/provider/webservice/Service1.xml"/>
<serviceurl="service/provider/anotherwebservice/Service2.xml"/>
<anotherService>
…
</provider>
<anotherProvider/>
  …
</saas-serviceprovider>
```

Listagem 5 – Exemplo da descrição utilizada para os provedores de serviços web.

A descrição do serviço web (Listagem 6) possui dois elementos principais: (1) o elemento **service-description** indica qual o tipo do serviço em questão (se REST ou SOAP) e a funcionalidade do serviço web em si, que é responsável por informar quais são os métodos e respostas que estão disponíveis e (2) o elemento **service-impl**, que descreve qual a localização da implementação do serviço web, sendo formada pela biblioteca cliente do serviço web (**serviceClientLib**) e pela a mídia imperativa (**imperativeMedia**). Por exemplo, neste caso a biblioteca encontra-se no diretório **WebServiceClient** e a mídia imperativa no *script* Lua **webService.lua**, ambos encontrados no próprio diretório do serviço web.

```
<saas-services id="id" api-doc="doc ">
<description>description</description>
<display-name>name</display-name>

<service-description type="REST">
service/provider/webservice/WebServiceWadl.xml
</service-description>

<service-impl base="resources/services-impl">
<serviceClientLibtype="Lua" url="webservice/WebServiceClient"/>
<imperativeMedia type="Lua" url="webservice/webService.lua"/>
</service-impl>
</saas-services>
```

Listagem 6 – Exemplo de XML empregado para descrever um serviço web.

A descrição do serviço web **service-description** é feita em um XML separado, utilizando padrões existentes que já são bastante empregados na indústria. Serviços web do tipo REST usam a WADL (*Web Application Description Language*) (JAVA.NET, 2013) como uma descrição em XML para aplicações baseadas no protocolo HTTP (tipicamente serviços web REST), que é voltada basicamente para descrever os recursos e métodos presentes no serviço. Esta linguagem foi criada pela Sun Microsystems com o objetivo de ser mais simples e mais voltada para serviços REST do que a linguagem existente WSDL.

Os serviços web do tipo SOAP usam como base a linguagem WSDL (*Web Services Description Language*) (W3C, 2013) como padrão. Serviços REST também podem usar tal linguagem. Ela foi criada pela W3C e é usada para descrever a funcionalidade oferecida por um serviço web. Uma descrição WSDL de um serviço web (também referida como um arquivo WSDL) fornece uma descrição de como um método pode ser chamado, quais os parâmetros que ele espera, e que estruturas de dados ele retorna. A versão atual do WSDL é a WSDL 2.0.

A Listagem 7 apresenta um exemplo da descrição em WADL de um serviço web REST. O serviço é descrito usando um conjunto de elementos do tipo recurso (**resources**). Cada recurso contém elementos para indicar um conjunto de métodos que descrevem a solicitação (**request**) e a resposta (**response**) de um recurso. O elemento de solicitação especifica a forma de representar os parâmetros (as entradas), quais os seus tipos são necessários e quaisquer cabeçalhos HTTP específicos. A resposta descreve a representação da resposta do serviço, bem como qualquer informação de falha, para no caso de lidar com erros.

```
<application xmlns="http://wadl.dev.java.net/2009/02">
<doc xml:lang="en" title="WebService"/>
<resources base="url">
<resource path="resourceName" id="id">
<doc xml:lang="en" title="doc"/>
<method name="POST" id="id">
<request>
<param name="param1" type="xs:string" required="false" default="" style="query"/>
<param name="param2" type="xs:string" required="false" default="" style="query"/>
<representation mediaType="application/xml"/>
</request>
<response status="200">
<representation mediaType="application/jsonrequest"/>
</response>
</method>
<anotherMethod/>
    ...
</resource>
<anotherResource/>
  ...
```

Listagem 7 –WADL descrevendo um serviço web REST a partir de seus recursos e métodos.

6 IMPLEMENTAÇÃO DA EXTENSÃO

Este capítulo descreve o projeto da proposta de extensão da visão estrutural apresentando como foi feito o seu desenvolvimento. A proposta consiste num projeto que estende a visão estrutural do NCL Composer 3 para adicionar um suporte imperativo nesta visão, além de um *plugin* que implementa a aba de serviços *web*.

O projeto é descrito em termos da metodologia utilizada para o seu desenvolvimento e como é feito o funcionamento interno para cumprir os casos de uso do autor de documentos. Também é explicado o contexto em que o projeto está inserido dentro da plataforma do Composer 3.

O objetivo é ter condições necessárias para avaliar na prática a solução proposta, a qual é abordada no capítulo seguinte.

6.1 Metodologia de Desenvolvimento

Como já descrito anteriormente, o desenvolvimento da extensão teve como base o reuso da visão estrutural da ferramenta NCL Composer 3. A escolha de reuso do Composer 3 deve-se principalmente em relação a solução proposta ser uma funcionalidade particular em relação a visão estrutural como um todo (e a uma própria ferramenta de autoria) e para poupar tempo no desenvolvimento reutilizando uma ferramenta existente.

Além da economia de tempo, outro fator que influenciou na escolha foi dele ter sido pensado desde o início para ser extensível. Como descrito adiante, sua arquitetura baseada em *plugins* possibilita a qualquer desenvolvedor inserir novas funcionalidades de autoria por meio de um *plugin*. Outro motivo é integrar uma nova funcionalidade dentro de uma

ferramenta de autoria bastante funcional. Com isso, abre-se espaço para que as pessoas que já usam o NCL Composer 3 tornem usuários em potencial para utilizar a extensão proposta, devido ao amplo grau de uso e aceitação que esta ferramenta possui no mercado de desenvolvimento de aplicações NCL.

O NCL Composer 3 surgiu da ideia de criar um ambiente integrado para desenvolvimento (IDE – *Integrated Development Enviroment*) de aplicações NCL que fosse capaz de ser facilmente estendido para adicionar facilmente novas funcionalidades de autoria. Esta característica vai de encontro ao fato de que o ambiente de desenvolvimento de aplicações NCL possui perfis de usuários bastante diversificados.

Uma ferramenta de autoria baseada em múltiplas visões atende este requisito, entretanto é fundamental abrir a possibilidade para adicionar novos mecanismos de autoria (LIMA, AZEVEDO, *et al.*, 2010). Ou seja, ao criar novas visões que sejam específicas para determinado autor, é importante que esta seja integrada com as visões existentes. Como também adaptar as visões existentes as características do usuário. Além disso, a ferramenta também deve se adaptar ao ambiente de produção que exige contato com diferentes tipos de equipamentos, sejam estes relacionados a transmissão, exibição ou anotação do conteúdo (LIMA, AZEVEDO, *et al.*, 2010).

6.2 Projeto do Suporte Imperativo

A extensão da visão estrutural é tratada dentro da ferramenta Composer 3 como um *plugin* de suporte imperativo. Apesar do Composer 3 possuir uma arquitetura voltada para a extensibilidade, a mesma não trata extensões internas aos *plugins* existentes. Sendo assim, o desenvolvimento do suporte imperativo concentrou-se principalmente em dois esforços de projeto: (i) modificações no código do *plugin* que é responsável pela visão estrutural e (ii) criação de *plugin* responsável por aspectos exclusivos do suporte imperativo, e que não tem relação com a visão estrutural.

Para facilitar o entendimento das responsabilidades atribuídas para os dois *plugins*, a Tabela 6.1 apresenta qual *plugin* é responsável por qual extensão e por qual caso de uso (descritos na seção 5.3.1 e 5.4.1).

Tabela 6.1 – Mapeamento entre as extensões propostas e o seu respectivo *plugin*.

Extensão	Caso de Uso	*Plugin*
(1) Notação visual dos pinos de entrada e de saída (3.1) Exposição automática do conteúdo imperativo	Uso de código imperativo NCLua	Plugin da Visão Estrutural
(2)Notação visual da ação de chamada de método e evento de valor de retorno (3.2) Criação automática das ações e eventos para acessar o conteúdo	Fazer uma chamada de método ou ler um valor de retorno	
(5) Integração com serviços web	Reuso de serviços a partir da aba de serviços web	*Plugin* de Suporte Imperativo

O *plugin* da visão estrutural é responsável pela implementação da nova notação visual dos pinos, da ação e eventos que foram definidos, exposição automática do conteúdo imperativo, além da criação automática das ações e eventos, quando no momento da criação do elo. O *plugin* de suporte imperativo fica responsável pela implementação de descobrir quais os métodos e variáveis presentes no código imperativo e do reuso de serviços web através da aba de serviços web.

Como visto na Tabela 4 cada extensão anterior tem uma relação direta com seu caso de uso do Composer 3. Como visto nas seções 5.4 e 5.5.1, basicamente têm-se três casos de uso: (i) o usuário deseja reusar determinado código imperativo, (ii) o usuário deseja fazer uma chamada de método ou ler um valor de retorno da mídia imperativa e por último, (iii) o usuário reusa um serviço *web* a partir da aba de serviços web.

No primeiro caso de uso, o usuário necessita usar os métodos ou variáveis existentes no código imperativo. Esse processo inicia-se modificando o atributo *src* da mídia com o código NCLua, o qual dispara a realização da extensão de exposição automática do conteúdo imperativo e em seguida da aplicação da notação visual aos pinos de entrada e de saída. A exposição automática cria os pinos de saída ou entrada e mapeia-os para cada conteúdo (método ou variável) do código imperativo.

O mapeamento é feito com o *plugin* da visão estrutural consultando o *plugin* de suporte imperativo para obter o conteúdo que esta disponível na mídia. Em posse do conteúdo, é feito o mapeamento entre cada método e variável com os seus respectivos pinos. Com os pinos criados é aplicanda a notação visual dos pinos de entrada ou saída.

A Figura 6.1 apresenta o diagrama de componentes que descreve o relacionamento do *plugin* estrutural com o *plugin* de suporte imperativo para fazer o mapeamento. Na figura cada componente representa um *plugin* do Composer. O *plugin* da visão estrutural **StructuralPlugin** possui os métodos: **mapContentToPins**, que realiza o mapeamento entre o conteúdo imperativo e os pinos, além do método **handleServiceDrop** que realiza o reuso de serviços web na visão estrutural.

O método **mapContentToPins** recebe como entrada o conteúdo da mídia imperativa, isto é, os métodos e variáveis que são descobertos com a ajuda do *plugin* de suporte imperativo **ImperativeSupportPlugin** através do método **retrieveMediaContent** da interface **ImperativeSupportPlugin**.

O *plugin* de suporte imperativo implementa o método **retrieveMediaContent** para recuperar o conteúdo imperativo disponível na mídia. A descoberta é feita por meio da classe **ImperativeMedia**, que é responsável por fazer o *parser* da mídia e descobrir os métodos e variáveis. A descoberta é feita com base em cada anotação presente no código.

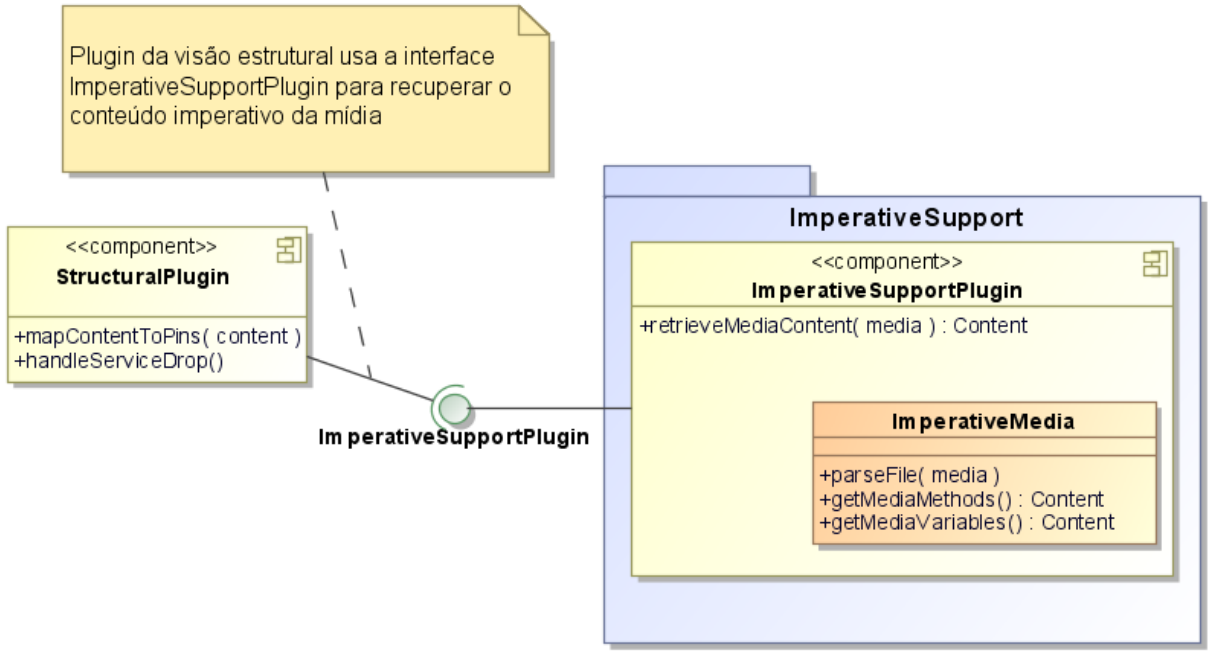


Figura 6.1 – Relacionamento entre o *plugin* estrutural e o *plugin* de suporte imperativo para recuperar o conteúdo da mídia imperativa.

O segundo caso de uso, que faz a criação automática da chamada de método ou a leitura do valor de retorno, é implementado no próprio *plugin* da visão estrutural. A criação automática é feita com base se um pino de saída ou de entrada está participando na criação elo. Eles são detectados automaticamente devido ao mapeamento feito na etapa anterior.

Caso uma ação de chamada de método seja feito, o *plugin* foi modificado para apresentar apenas a janela de diálogo para informar a condição. O mesmo caso para o evento de leitura do valor de retorno que apresenta apenas a janela para informa a ação.

O terceiro caso de uso é implementado também em conjunto com ambos os *plugins*. Esse caso de uso é feito quando o usuário arrasta um serviço da aba de serviços web e solta sobre a visão estrutural. A aba de serviços web é implementada pelo *plugin* de suporte imperativo. Como visto anteriormente, a aba constrói os serviços web a partir das descrições dos serviços web. Para cada método do serviço, o usuário pode arrastar e soltar sobre a visão estrutural.

A ação de arrastar é implementada no *plugin* de suporte imperativo e a ação de soltar é implementada no *plugin* da visão estrutural. A ação de arrastar cria um evento de arrastar contendo o método e serviço selecionado pelo usuário. Quando o usuário solta sobre a visão estrutural, o *plugin* é notificado com este evento, e então manipula a ação de soltar através do método **handleServiceDrop.** Esse método é responsável por criar a mídia referente ao serviço web e o pino de entrada ou de saída referente ao método selecionado.

De fato como percebido, o *plugin* da visão estrutural não possui mecanismos que facilitem sua extensão e dessa forma foi preciso uma contínua modificação do mesmo para implementar o suporte imperativo. Apesar disso, de certo modo ajudou a criar a extensão da visão estrutural rapidamente e assim ser usada para aplicá-la em estudos de caso e avaliá-la na prática por usuário autores de documento NCL, como é visto no capítulo a seguir.

Como toda arquitetura baseada em *plugins* ou componentes, os plugins seguem um padrão de desenvolvimento bem definido. Seguindo um protocolo bem definido, eles são identificados para facilitar o seu gerenciamento pela plataforma como também padronizar a comunicação entre os *plugins*. A próxima seção explica o padrão de desenvolvimento do *plugin* e na seção 6.4 a arquitetura do Composer 3, na qual ele está inserido.

6.3 *Plugin* de Suporte Imperativo

O *plugin* é desenvolvido implementando duas interfaces principais definidas pelo Composer 3: *IPluginFactory* e *IPlugin* (TELEMÍDIA, 2013). O *IPluginFactory* é responsável

por criar novas instâncias do *plugin* definido pelo usuário, e manter informações globais sobre o *plugin*, como seu fornecedor, versão, etc. O método **createPluginInstance** da interface *IPluginFactory* é chamado pelo Composer 3 para criar uma nova instância do plugin.

A Listagem 8 apresenta a implementação desta interface para o plugin de suporte imperativo. A implementação define o nome ImperativeSupportFactory para a classe e implementa os métodos com as informações relacionadas ao plugin como o id do *plugin* (br.lavid.ufpb.ImperativeSupportPlugin), o *vendor* (Lavid - UFPB / Thales Ferreira), descrição, entre outros. O método para criar a instância do *plugin* de suporte imperativo é implementado retornando uma instância da classe ImperativeSupportPlugin (Listagem 9), a qual implementa a interface *IPlugin*.

```
class ImperativeSupportFactory : public QObject,
public IPluginFactory {
    Q_OBJECT
    Q_INTERFACES(IPluginFactory)
public:
     ImperativeSupportFactory();
~ImperativeSupportFactory();
     IPlugin* createPluginInstance();
void releasePluginInstance(IPlugin *);

     QString id() const {return"br.lavid.ufpb.ImperativeSupportPlugin";};
     QString name() const {return"Aba de Servicos Web";};
     QString version() {return"0.0.1"; }
     QString compatVersion() {return"0.0.1";}
     QString vendor() {return"Lavid - UFPB / Thales Ferreira";}
     QString copyright() {return"Lavid/UFPB";}
     QString description() {return"Imperative Support Plugin improves support to use imperative
media leaving faster and simpler to use methods and variables. Also is available a set of web
services for use in structural view";}
     QString url() {return"";}
     QString category() {return"";}
};
```

Listagem 8 – Implementação da interface *IPluginFactory* do *plugin* de suporte imperativo.

A interface *IPlugin* define o plugin do Composer NCL em si. Quando um documento é aberto, uma nova instância do plugin será criada e se ele possuir interface gráfica ela será exibida. A Listagem 9 apresenta a implementação do *plugin* **ImperativeSupportPlugin**. Os métodos implementados da interface são **getWidget**, que retorna um objeto contendo a interface do *plugin* e **init** que inicializa o *plugin*.

Percebe-se também a definição do método **retrieveMediaContent**, o qual foi explicado anteriormente. Os outros métodos (**onEntityAdded**, **onEntityRemoved**, etc.) são executados quando na ocorrência da inserção ou remoção de entidades nodocumento por outros *plugins*. Estes métodos são sempre chamados pelo micro-núcleo, por exemplo, quando outro plugin modifica a estrutura do documento.

```
class ImperativeSupportPlugin : public IPlugin {
   Q_OBJECT
private:
   QWidget*window;
public:
explicit ImperativeSupportPlugin();
~ImperativeSupportPlugin();
   QWidget* getWidget();
void init();
Content*retrieveMediaContent(QString media);
public slots:
void onEntityAdded(QString ID, Entity *);
void onEntityChanged(QString ID, Entity *);
void onEntityRemoved(QString ID, QString entityID);
void errorMessage(QString error);
};
```

Listagem 9 – *Plugin* de suporte imperativo implementando a interface *IPlugin*.

6.4 Arquitetura do NCL Composer 3

Pensando em atender os requisitos de estensibilidade, o NCL Composer foi desenvolvido com uma arquitetura baseada em micro-núcleo e extensões. O micro-núcleo serve como um hospedeiro para as extensões e coordena a forma como elas comunicam-se (LIMA, AZEVEDO, *et al.*, 2010).

No NCL Composer as extensões de funcionalidades do micro-núcleo são feitas por meio de *plugins*. *Plugins* interagem com a aplicação hospedeira, com o objetivo de estendê-la ao adicionar funcionalidades e/ou recursos de cunho especifico. O micronúcleo é responsável por controlar a comunicação entre os diferentes *plugins*, gerenciar o acesso dos *plugins* a um modelo único que representa o documento hipermídia e notificar as modificações nesse modelo para os *plugins* interessados.

A Figura 6.2, retirada de (LIMA, AZEVEDO, *et al.*, 2010), detalha o funcionamento do micro-núcleo por passagem de mensagem. O núcleo faz o gerenciamento do modelo central, controlando as modificações que são iniciadas a partir da interação feita pelos *plugins*. Qualquer alteração no modelo, os outros *plugins* são notificados com uma passagem de mensagem.

Por exemplo, no momento em que o autor da aplicação interage com o *plug-in* textual modificando a estrutura do documento nesta visão (1), ele emite um sinal para o núcleo notificando-o sobre as mudanças efetuadas (2). O núcleo então realiza a validação (3) e, caso a validação não retorne erros, realiza as devidas mudanças no modelo central (4). Em seguida chega a hora de notificar os outros *plug-ins*, então ele emite um sinal com a respectiva mudança (5). Dessa forma, os *plug-ins* podem incrementalmente realizar as mudanças necessárias no modelo central ficando sempre único para todos os *plug-ins*.

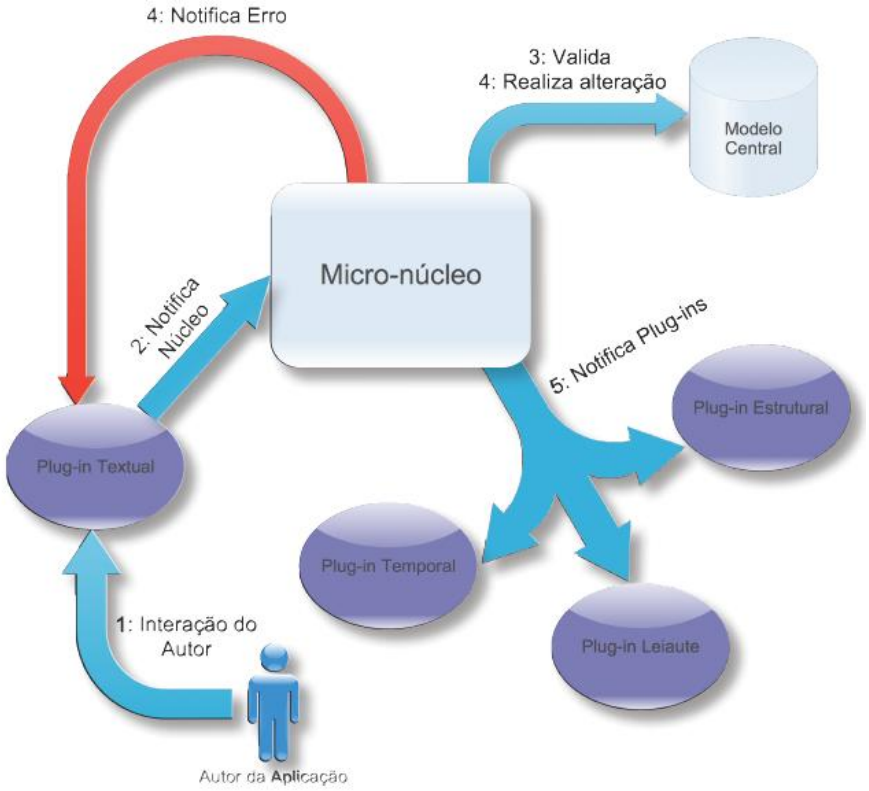


Figura 6.2 – Padrão de comunicação entre micro-núcleo e *plugins*. (LIMA, AZEVEDO, *et al.*, 2010).

O esquema de organização do Composer 3 em camadas é apresentado na Figura 6.3. No topo da arquitetura têm-se os *plugins* dos usuários, os quais usam diretamente a três camadas abaixo: (i) *CoreModel*, (ii) *CoreControl* e (iii) *ComposerGUI*. Na parte inferior (de infraestrutura) encontra-se o framework QT, que provê um conjunto de API's para construção de interface gráfica, manipulação XML e multimídia (áudio, vídeo, imagens) e acesso a interface de rede.

O modelo central (*CoreModel*) é a representação interna da aplicação em desenvolvimento pelo usuário da ferramenta. Ele foi desenvolvido considerando os elementos da linguagem NCL. Para possibilitar a comunicação entre *CoreControl* e *CoreModel* foi desenvolvida uma fachada que provê a *CoreControl* uma API simples para manipulação sobre *CoreModel*. *ModelFacade* é responsável por receber requisições de modificações sobre o modelo interno e executar os procedimentos necessários para que essas modificações sejam realizadas corretamente.

O *CoreControl* é responsável pelo micro-núcleo em si. É composto de quatro módulos principais: *Message*, *Transaction*, *Plugin* e *Project*. O módulo *Message* é encarregado pela gerência das trocas de mensagens entre os *plugins* e o micro-núcleo. Esse módulo recebe os

sinais emitidos pelos plug-ins, interpreta-os e delega suas respectivas ações para o *ModelFacade*.

O módulo *Transaction* é responsável por gerenciar as transações que visam manipular o modelo central. O *Plugin* é o módulo responsável pelo carregamento dos *plugins* externos. Esse módulo executa a função de filtro para cada um dos plug-ins e faz a conexão entre os sinais específicos do micronúcleo, sobre os elementos filtrados, com as funções que irão tratar esses sinais. O módulo *Project* gerencia operações sobre projetos (abrir, deletar, salvar) e manipula documentos que serão inseridos ou criados dentro dos projetos.

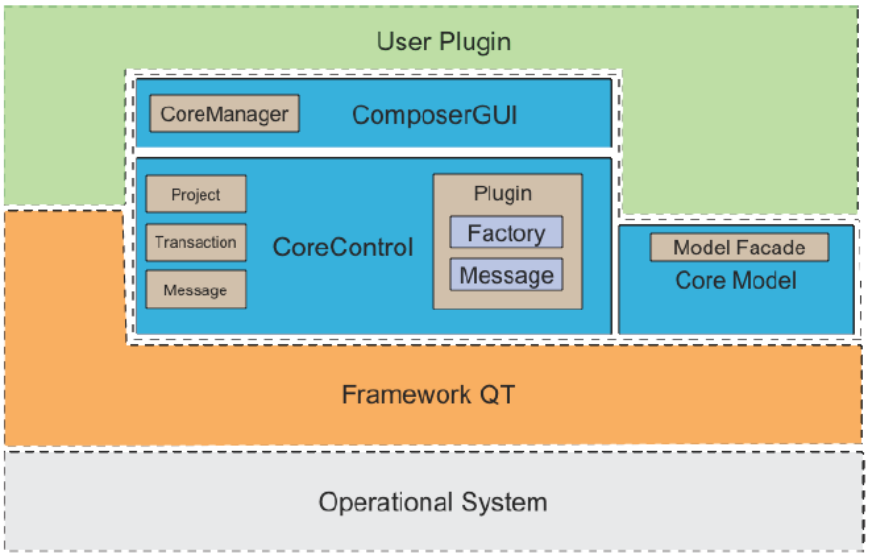


Figura 6.3 – Esquema das camadas do Composer 3.
Retirado de (LIMA, AZEVEDO, *et al.*, 2010).

O *ComposerGUI* representa a interface gráfica do Composer 3 como um todo e uma parte da GUI que é utilizada como base para embutir a interface existente nos *plug-ins*. Cada *plugin* possui um método específico onde são definidos seus próprios elementos gráficos. A partir deste método a interface do plug-in é incorporada ao Composer 3. Esse método é explicado mais adiante na interface de programação dos *plug-ins*.

Para os *plugins* interagirem com o micro-núcleo a interface discutida na seção 6.3 apresenta um exemplo de como integrar um *plug-in* dentro do Composer 3 utilizando uma interface de programação única e ortogonal.

7 AVALIAÇÃO, RESULTADOS E DISCUSSÕES

Este capítulo apresenta a avaliação da extensão por meio de um conjunto de estudos de casos (seção 7.1) que demonstram o uso da visão estrutural estendida na tarefa de integração de código imperativo. Além disso, a seção 7.2 apresenta a avaliação empírica empreendida para avaliar na prática os efeitos da extensão com relação ao tempo e facilidade de integração.

7.1 Estudos de Caso

Esta seção apresenta uma série de estudos de caso que demonstram como a visão estrutural estendida integra código imperativo em uma aplicação NCL. São quatro aplicações modeladas na visão, sendo a primeira (aplicação de Quiz) que exemplifica a integração com uma mídia imperativa e as outras aplicações que exemplificam mídias imperativas que acessam serviços web.

7.1.1 Aplicação de Quiz

A aplicação de Quiz realiza um jogo de perguntas e respostas. Neste exemplo, o Quiz é composto por três perguntas onde somente uma das suas respostas (três ao todo) está correta. A resposta de cada pergunta é feita pelos botões ‘1’, ‘2’ ou ‘3’. Quando o usuário pressionar o botão ‘1’, ele responde a pergunta escolhendo a primeira resposta e assim por diante. O valor da resposta de cada pergunta deverá ser enviada para um *script* poder computar quantas respostas foram respondidas corretamente. Saber a quantidade de acertos é o lado imperativo da aplicação de Quiz, e este processamento é feito pelo código Lua.

Após a resposta da primeira pergunta, a segunda pergunta deverá aparecer no lugar da primeira e assim sucessivamente. Depois de respondida a terceira pergunta, uma mensagem no formato “X acertos”, onde X é o número de acertos deverá aparecer na tela informando quantas questões foram respondidas corretamente. A quantidade de acertos é recuperada do código imperativo.

O código imperativo é responsável por computar a quantidade de respostas que foram corretamente respondidas. Ele armazena as questões e qual a sua resposta correta. Baseado no que o usuário responde, o cálculo de respostas corretas é feito.

O código possui o método **answerQuestion(question, choose)** que é utilizado para responder cada questão com a resposta escolhida. Ele recebe como parâmetro o identificador da questão e o número da resposta (1, 2 ou 3). Esta resposta é escolhida pelo usuário do quiz. Este método não possui valor de retorno. O outro método **getHits**() retorna a quantidade de perguntas corretas (acertos) através do valor de retorno hits. O valor de retorno **hits** armazena a quantidade de perguntas corretas.

A Figura 7.1 apresenta a visão estrutural da aplicação de Quiz. A aplicação começa com o inicio da mídia vídeo e da primeira pergunta **pergunta 1**. Para cada pergunta três elos são utilizados para quando o usuário escolher a resposta seja realizado a escolha da resposta escolhida (dada pelo botão pressionado) e em seguida a exibição da pergunta seja parada. Por exemplo, o primeiro elo da primeira pergunta, tem como evento a seleção do botão ‘1’ e a ação de chamada de método sobre o pino de entrada **answerQuestion** (mídia imperativa **quiz)** passando os valores ‘1’ e ‘1’, ou seja, primeira resposta da primeira pergunta. Após a chamada do método, a pergunta é parada.

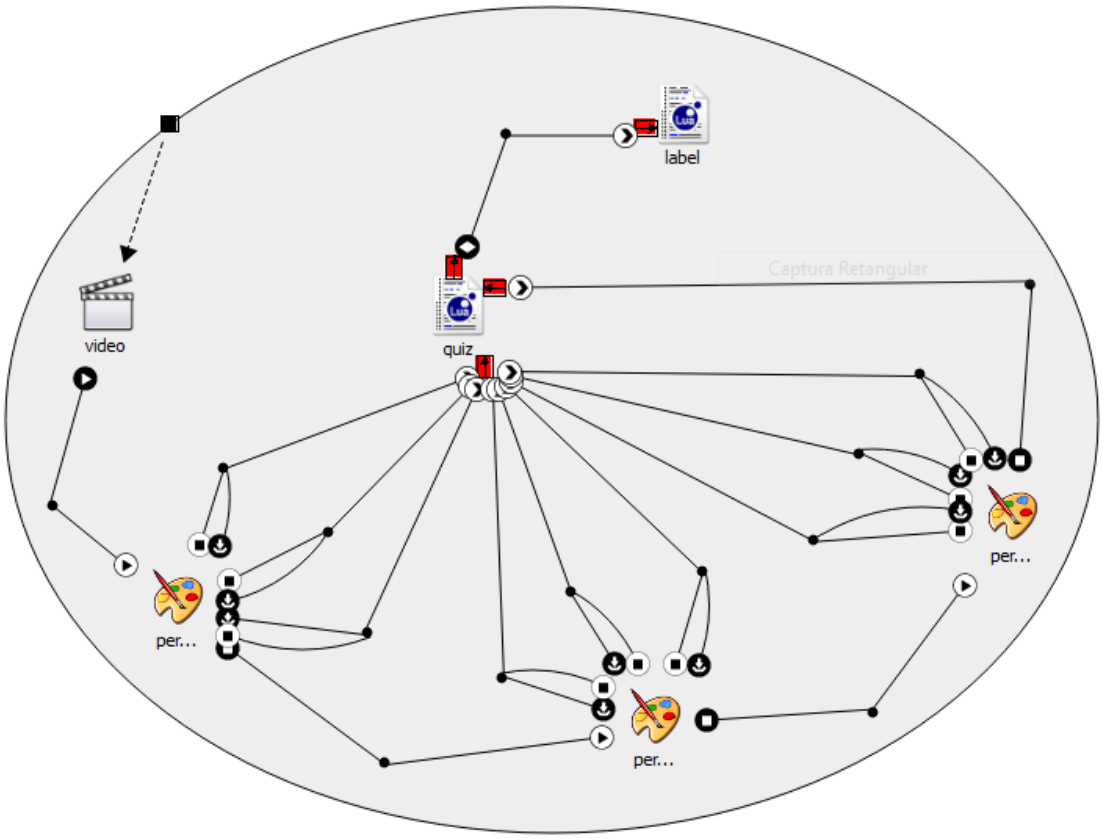


Figura 7.1 – Visão estrutural da aplicação de Quiz integrando uma mídia imperativa.

O outro elo da pergunta apresenta a próxima pergunta quando a anterior terminar. Este elo possui como condição a parada da mídia e a ação de inicio da próxima pergunta. A segunda e a terceira pergunta possuem o mesmo mecanismo de sincronismo temporal que a primeira pergunta. A única diferença é que na última pergunta, quando ela termina de ser exibida a quantidade de acertos do Quiz é exibida na mídia **label**.

Para exibi-lo, o método **getHits** é chamado para obter a quantidade de acertos. Quando chamado o valor de acertos é colocado dentro do pino de saída **hits** da mídia **quiz** e ele fica pronto para ser lido pelo documento NCL. Este valor é lido pelo elo que interliga o pino de saída de **quiz** com o pino de entrada da mídia **label**. Com esta ligação o valor da quantidade de acertos presente no pino de saída do **quiz** é usado como entrada no pino de entrada da mídia **label** e assim o valor é exibido na tela.

A Figura 7.2 apresenta a configuração dos parâmetros **question** e **choose** para a chamada do método answerQuestion. Os parâmetros são configurados individualmente com o valor ‘1’ para a questão e também ‘1’ para o segundo parâmetro.

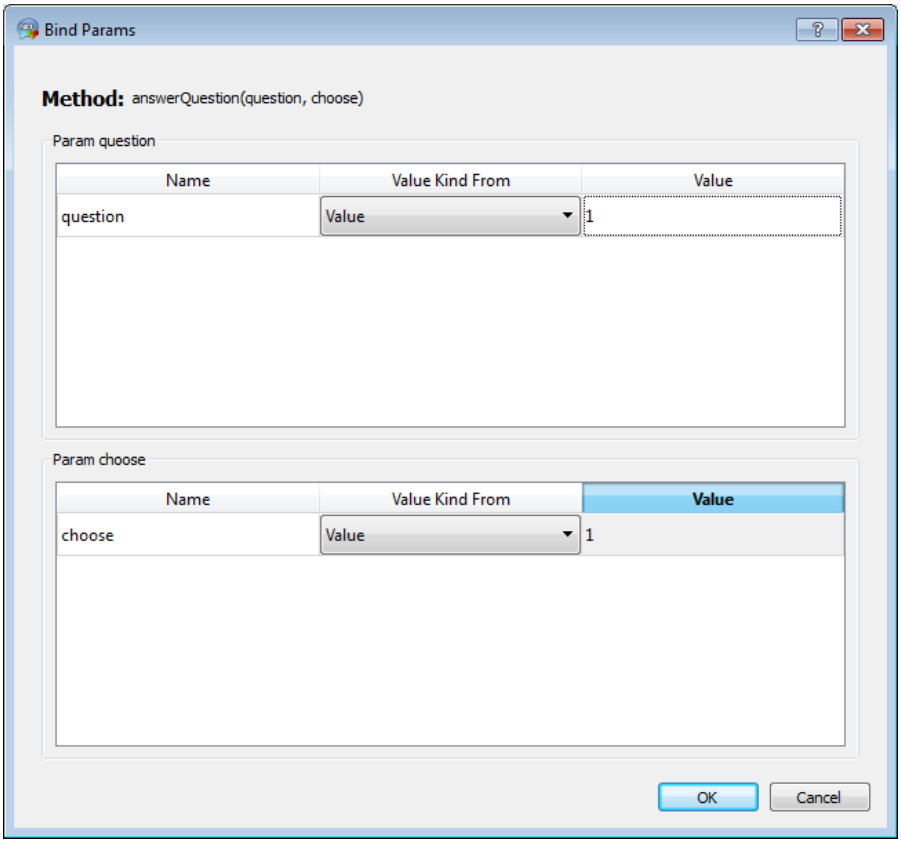


Figura 7.2 – Janela de diálogo que especifica os dois parâmetros do método answerQuestion.

7.1.2 Aplicação TinyWebDB com Serviço Web

A aplicação TinyWebDB representa um caso de uso para utilização de uma mídia de serviço web em uma aplicação NCL. Essa mídia de serviço é uma forma fácil de armazenar e recuperar dados na forma chave e valor a partir de um serviço web (TINYWEBDB, 2013). Esse exemplo contempla o relacionamento com a mídia para enviar solicitações e receber respostas de serviços web a partir da visão estrutural.

O serviço web possui um método (*storeavalue*) para armazenar um valor correspondente a uma chave e outro método (*getvalue*) para recuperar valores a partir de uma determinada chave. Para cumprir os requisitos de interação com este serviço, os pinos que fazem parte da mídia são: dois pinos de entrada (1) **storeValue**, que faz armazenar no servidor um valor, dado uma chave e (2) **getValue**, que realiza a recuperação do valor armazenado em uma determinada chave. E um pino de saída (1) **tagValue,** que é utilizado para ter acesso ao valor retornado pelo pino de entrada getValue.

Na aplicação a interação com o serviço web é feita da seguinte forma: um valor entrado pelo usuário em um campo de texto é enviado para o servidor quando ele pressionar o

botão de envio. Para recuperar o valor armazenado no servidor o usuário pressiona um botão, o valor é recuperado e em seguida o valor aparece na tela.

O reuso do serviço web TinyWebDB é feito a partir da aba de serviços web. A aba disponibiliza todos os métodos existentes no serviço web para serem utilizados na visão (Figura 7.3). Quando o autor de documentos arrasta-e-solta o método **GetValue** sobre a visão, a mídia para acessar o serviço web é criada automaticamente. Além disso, o pino de entrada **GetValue** e de saída **tagValue** também são criados sem a necessidade de interação do usuário.

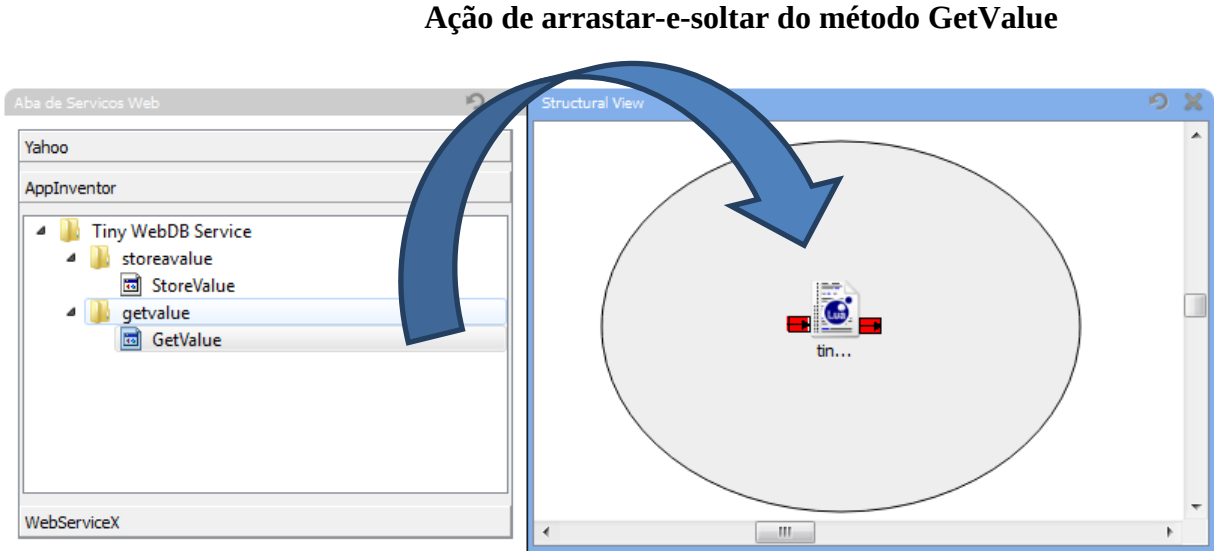


Figura 7.3 – Criação da mídia e pinos após o método GetValue do serviço ser solto na visão estrutural.

Com a mídia do serviço reusada, o restante da aplicação é modelada. A Figura 7.4 apresenta a visão estrutural da aplicação final. Duas mídias de imagens **buttonStore** e **buttonGet** são usadas para fazerem o papel do botão que envia o valor e do botão para recuperar o valor do servidor. Além dos botões, a mídia **inputText** é usada para o usuário entrar com o texto a ser enviado e a mídia **label** para exibir o valor recuperado do servidor.

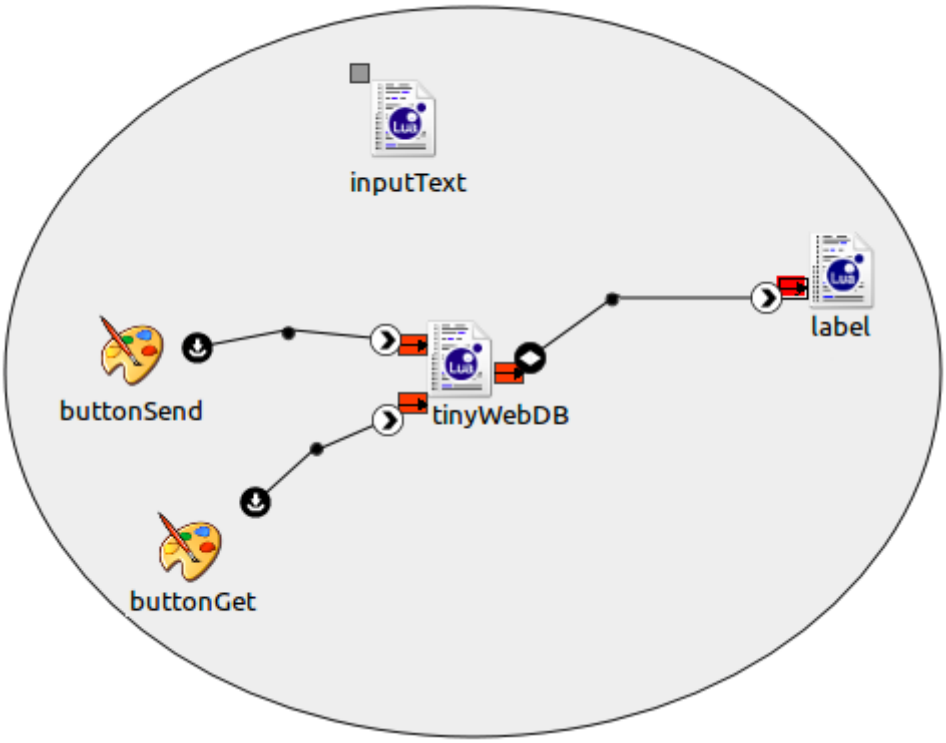


Figura 7.4 – Visão estrutural da aplicação TinyWebDB.

O elo entre o botão **buttonStore** e **tinyWebDB** modela o envio do valor presente na mídia **inputText** para o servidor. Este elo é ativado quando o usuário pressiona o botão, fazendo com que uma ação de chamada de método seja feita sobre o pino de entrada **storeValue** da mídia **tinyWebDB**. Esta ação faz com que o método correspondente no serviço web seja chamado e assim a aplicação apresenta uma comunicação com o servidor.

A Listagem 10 apresenta o código-fonte Lua da mídia imperativa **tinyWebDB**. No *script* são implementados os métodos, com seus parâmetros e na implementação é encapsulado o código do módulo do cliente de serviço web (**TinyWebDBClient**).

```
require 'lib/TinyWebDBClient/tinyWebDBClient'

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

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

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

function eventHandler(evt)
    ...
end

event.register(eventHandler)
```

Listagem 10 – Código-fonte Lua da mídia imperativa tinyWebDB.

A chamada para o método **storeValue** possui dois parâmetros para serem especificados e os mesmos são apresentados na Figura 7.5. Neste exemplo, a especificação dos parâmetros enquadra-se no segundo caso, onde os valores dos parâmetros são advindos de fontes distintas. O primeiro parâmetro **tag** é configurado para possuir o valor fixo “1”. Já o segundo parâmetro **value** é configurado para ter o seu valor partir de uma mídia presente na visão estrutural. Este valor é especificado para ser relativo à propriedade **text** da mídia **inputText**, a qual é responsável por conter o valor entrado pelo usuário quando o mesmo foi digitado.

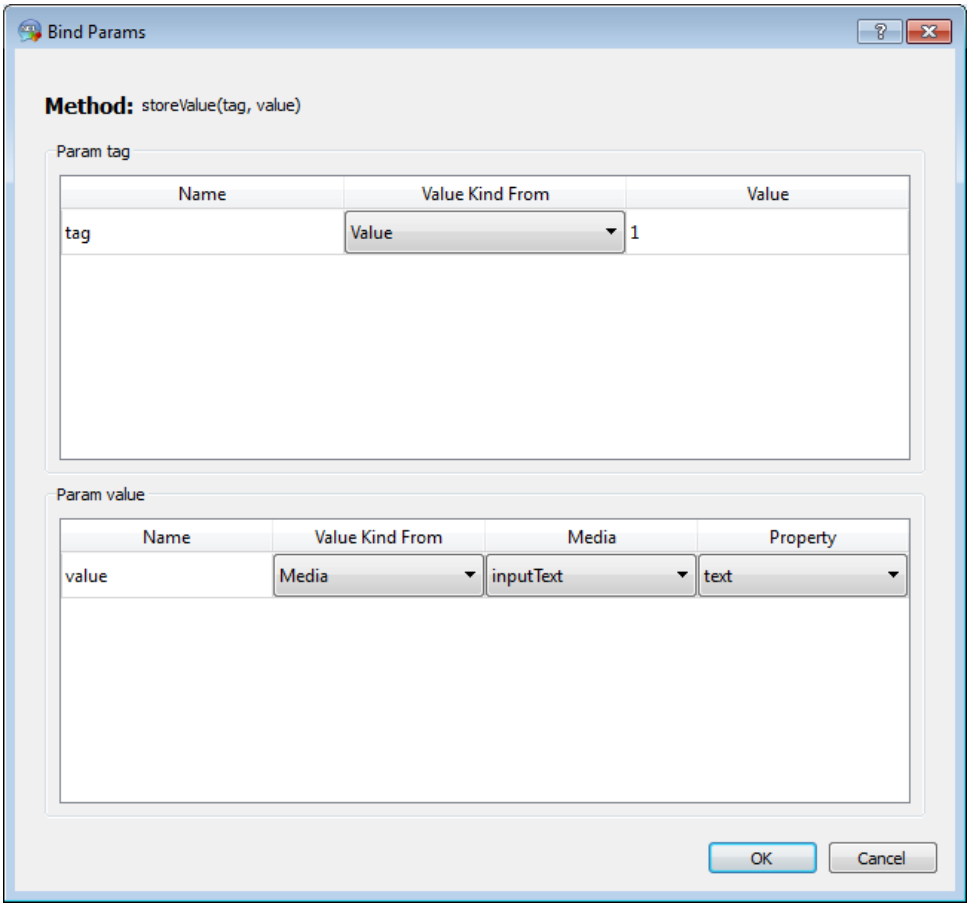


Figura 7.5 – Janela de dialogo para especificação dos parâmetros do método storeValue.

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

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

7.1.3 Aplicação de Enquete com Serviço Web

Este exemplo demonstra a integração de um serviço de enquete dentro da aplicação NCL. A aplicação escolhida é conhecida como *TV Voting*, que permite ao telespectador na

TVDi participar de uma enquete pela TV. Neste caso, foi definido que a aplicação teria interação baseada em canal de retorno, na qual a enquete envia resposta do telespectador e recebe o resultado “ao vivo” valendo-se de canal direto com um serviço Web implementado com Rest.

Para entender melhor a aplicação, a mesma é composta por quatro cenas: (1) cena com ícone de interatividade (inicia interação); (2) cena de votação (com exibição da pergunta e opções de respostas) (3) cena com informação sobre a resposta e confirmação do envio da resposta; e (4) se existir comunicação, uma cena com o resultado parcial ou final da enquete, caso contrário uma mensagem que houve erro de comunicação.

O tipo de aplicação de enquete aqui é uma enquete que é produzida pelo autor de documentos a partir de uma enquete já existente no servidor. Logo, é preciso que ele crie o conteúdo da enquete com as perguntas e respostas de acordo com a enquete do servidor e também informe qual a enquete que ele está se referindo na aplicação. Por motivos de simplificação apenas a terceira cena, a cena de votação, será descrita.

Quando o usuário escolhe a sua opção na segunda cena, a aplicação exibe a terceira cena com informações sobre a opção escolhida e um botão para confirmar o voto na escolha. A confirmação da escolha significa o envio da opção para a enquete no servidor.

A Figura 7.6 apresenta a visão estrutural referente à cena de votação. O envio da opção para o serviço é feita quando a tecla verde do controle remoto for pressionada para a mídia de imagem **aButton**. Após a tecla verde ser pressionada, o processo de envio da opção é realizado pela mídia de serviço **pollVoting**. Esta mídia possui um processo interno responsável pelo envio da opção ao serviço.

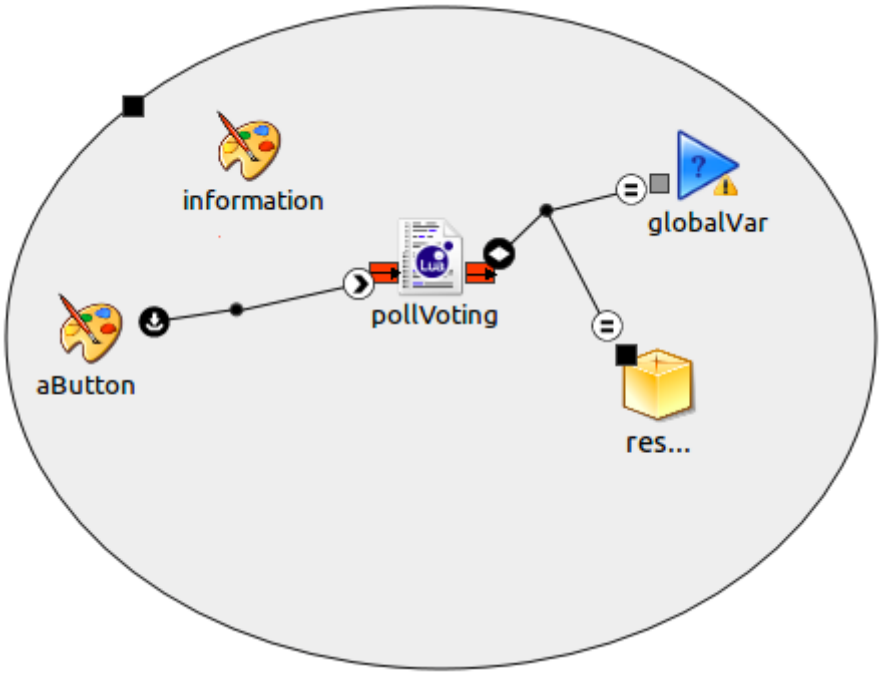


Figura 7.6 – Visão estrutural da aplicação de enquete.

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

A Listagem 11 apresenta o código Lua da mídia de serviço web bem como a anotação @Method. O método **sendPollOption** (linha 5) é anotado com o nome do método (**sendPollOption**), os parâmetros (**idPoll** e **option**) e o nome do evento de saída (**resultReceived**). A implementação do método chama o método do serviço responsável pelo envio do voto utilizando o módulo cliente denominado **PollServiceComunication** e o seu método **sendOption** (linha 7). O serviço web testado encontra-se no endereço a3tv.lavid.ufpb.br.

Quando a resposta do método do serviço web chega, o método **getResponse** (linha 11) é executado e neste momento o método **returnResultReceived**, o qual encapsula o envio do resultado recebido, é chamado para enviar o resultado como um evento. Com o envio do evento o resultado é deixado disponível no pino de saída **resultReceived** da mídia

pollVoting. Assim o resultado, vindo do serviço, pode ser acessado por outras mídias através do evento **onValueReturn**.

```
require 'lib/PollServiceComunication/pollServiceComunication'

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

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

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

function eventHandler(evt)
    ...
end

event.register(eventHandler)
```

Listagem 11 – Código-fonte lua da mídia imperativa pollVoting.

Os valores para cada parâmetro do método do serviço são escolhidos através da janela de diálogo de parâmetros. Como no exemplo anterior, a especificação dos parâmetros também se enquadra no segundo caso de passagem de parâmetros. A configuração é feita com o primeiro parâmetro **idPoll** (a identificação da enquete) configurado para ter o valor “1” e o segundo parâmetro **option** (a opção escolhida pelo usuário) configurado para ser o valor presente na propriedade **chooseOption** da variável global **information**.

O método **sendPollOption** possui uma resposta relacionada ao resultado parcial da enquete. Esta resposta é recebida com a quantidade de votos para cada opção separados por vírgulas. Para o autor de documentos recuperar o resultado parcial da enquete, a mídia do serviço **pollVoting** cria automaticamente o pino de saída relacionado a esta resposta do método.

O resultado da enquete é armazenado na mídia global **globalVar** apenas quando a resposta do serviço web é recebida. Este processo é feito pelo elo que liga o pino de saída **resultReceived** com a propriedade **result** da mídia **globalVar**. Este elo é criado automaticamente sem a interação do usuário, pois ele apenas redireciona o valor do pino de

saída (representando o resultado parcial da enquete) para a propriedade da mídia **globalVar**que armazena este resultado. O valor do resultado é posteriormente usado para exibir o resultado na cena de resultado.

7.1.4 Aplicação de Canal de Tempo com Serviço Web

Esta aplicação apresenta um exemplo em que um serviço web de clima tempo é integrado em uma aplicação NCL. A aplicação apresenta a situação climática da cidade com informações visuais relacionadas à condição do tempo (imagem representando clima chuvoso, ensolarado ou nublado, etc.) e textuais referentes à temperatura. Estas informações são apresentadas a partir de uma consulta ao serviço web de clima tempo, que mantém os dados de tempo atualizados para as grandes das cidades do mundo.

O serviço web possui apenas o método **GetWeather**, o qual retorna, entre outras informações, a temperatura e as condições do tempo (ensolarado, nublado, parcialmente nublado, etc.) para uma determinada cidade. Como forma de testar a mídia do serviço web, foi utilizado um serviço de clima tempo existente no site webservicex.net (WEBSERVICEX.NET, 2013). Para simplificar a entrada da cidade, a aplicação deve contar com um método para descobrir a cidade do usuário automaticamente, entretanto ele será mostrado aqui.

A Figura 7.7 apresenta a visão estrutural da aplicação de clima tempo. Quando a imagem de fundo **background** é iniciada, é disparada uma ação de chamada de método sobre o pino de entrada **getCityWeather** da mídia de serviço web **weatherClient**. Em outros termos, esta ação realiza uma requisição ao serviço web para consultar o clima e obter a resposta contento as informações do clima da cidade.

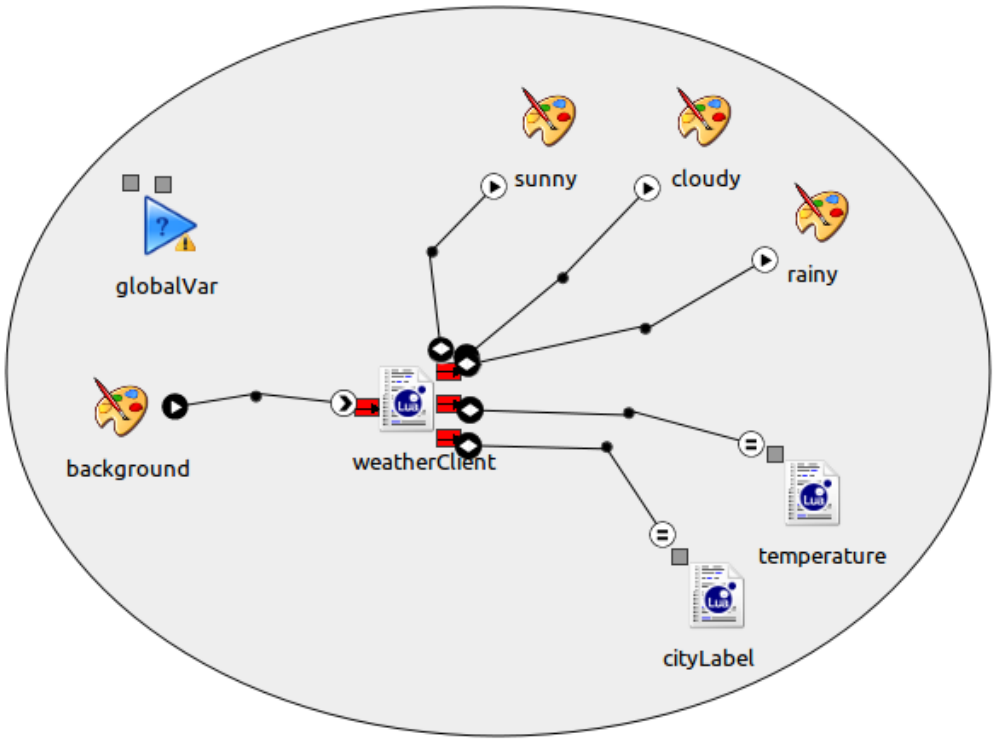


Figura 7.7 – Visão estrutural da aplicação de clima tempo.

Além do pino de entrada (mapeado para o único método do serviço **GetWeather**), na figura a mídia **weatherClient** possui três pinos de saída: **weatherCondition** representando o valor para a condição do tempo, **temperature** representando o valor da temperatura da cidade e **city** representando o valor da cidade. Como estamos interessados em mais de uma informação presente na resposta (condição do tempo, temperatura e cidade), este pino de saída enquadra-se no segundo tipo de pino de saída (ver seção 5.3). Dessa forma, o pino de saída da resposta é desmembrado em outros três pinos de saída, sendo um para tratar o recebimento da condição do tempo, da temperatura e outro da cidade.

A Listagem 12 apresenta o código Lua da mídia de serviço web bem como a anotação @Method. O método **getCityWeather** (linha 5) é anotado com o nome do método (**getCityWeather**), dois parâmetros (**cityName** e **countryName**) e três eventos de saída (**weatherCondition**, **temperature** e **city**). A implementação do método chama o método do serviço responsável pela consulta do clima da cidade utilizando o módulo cliente denominado **WeatherServiceClient** e o seu método **getCityWeather** (linha 7). O serviço web testado encontra-se no endereço www.webservicex.net.

Quando a resposta do método do serviço web chega, o método **getResponse** (linha 10) e o mesmo extrai as informações de interesse (condição do tempo, temperatura e cidade) da resposta e envia o evento de saída com o valor. No código foi exemplificado apenas o

tratamento da condição do tempo através do método **returnWeatherCondition** (linha 19). Este método, que encapsula o valor da condição do tempo, é chamado para enviar a condição do tempo como um evento. Com o envio do evento o valor vindo do serviço é deixado disponível no pino de saída **weatherCondition** da mídia **weatherClient**. Assim o valor da condição do tempo, pode ser acessado por outras mídias através do evento **onValueReturn**.

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

Listagem 12 – Código-fonte da mídia imperativa weatherClient.

A ação de chamada de método possui um parâmetro para a cidade e o outro para informar o país que será feita a consulta do tempo. Estes valores são determinados, respectivamente, a partir das propriedades **cityName** e **countryName** da mídia **globalVar**. O usuário realiza esta configuração pela janela de configuração de parâmetros associando cada parâmetro a sua respectiva propriedade.

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

automaticamente sem a interação do usuário, pois apenas transfere o valor do pino de saída para ação de chamada de método no pino de entrada.

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

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

7.2 Avaliação Empírica

Esta seção descreve o estudo experimental para avaliar na prática os benefícios da extensão. A proposta é aplicada no contexto de reais autores de documentos NCL, sejam eles experientes ou não. O objetivo da avaliação empírica é conduzir um estudo experimental para mostrar indícios que a extensão proposta consegue ser mais produtiva e menos complexa ao integrar objetos de mídia imperativos em documentos NCL.

Experimentação envolvendo software consiste em confrontar com fatos as suposições, especulações, intuições e crenças que abundam no contexto de desenvolvimento de software (JURISTO e MORENO, 2010). Ela serve para embasar por meio de investigações científicas se as afirmações feitas pela comunidade de desenvolvedores de software são válidas ou não. As investigações científicas são estudos objetivos, baseados em observações ou experimentação com o mundo real e suas mudanças mensuráveis (JURISTO e MORENO, 2010).

Na ciência existem quatro tipos de métodos para empreender experimentação: científico, de engenharia, experimental e o analítico. O método experimental também chamado de empírico propõe um método estatístico com o objetivo de validar uma dada hipótese (ZELKOWITZ e WALLACE, 1998). Nesse estudo utilizamos unicamente o método experimental e o mesmo será descrito a seguir.

No método experimental são conduzidos experimentos controlados ou estudos de casos para colher os dados e assim verificar ou refutar a hipótese. De forma geral, o método experimental sugere o modelo, desenvolve o método qualitativo e/ou quantitativo, aplica um

experimento, mede e analisa, avalia o modelo e repete o processo (TRAVASSOS, GUROV e DO AMARAL, 2002).

Para guiar a realização da avaliação empírica nesse trabalho foi utilizado o processo definido por (WOHLIN, RUNESON, *et al.*, 2000), o qual vem sendo seguido por diversos pesquisadores empreendendo estudos experimentais. O processo é dividido em 5 (cinco) fases: (i) definição: (ii) planejamento; (iii) operação do experimento; (iv) análise e interpretação dos resultados; e (v) apresentação dos resultados. O objetivo dessas etapas é estruturar os elementos importantes do experimento para permitir o foco no controle e nas variáveis significativas para a obtenção dos resultados. Os experimentos também terão sua validade avaliada através de 4 (quatro) tipos de ameaças: (conclusão, construção, interna e externa).

As próximas seções descrevem as etapas de definição do estudo, do planejamento do experimento, do projeto experimental, da execução e da análise dos resultados relacionadas ao estudo experimental da extensão da visão estrutural.

7.2.1 Definição do Experimento

O objetivo global do estudo é verificar se a extensão destinada à integração de mídia imperativa consegue tornar mais produtivo e simples o uso de um código imperativo na visão estrutural. Utilizando (WOHLIN, RUNESON, *et al.*, 2000), a definição do experimento delineia os objetivos do estudo e são descritos na seguinte estrutura:

- **Analisar a** integração de código imperativo em aplicações declarativas NCL utilizando a visão estrutural (abordagem visual);
- **Com o objetivo de** caracterizar a adoção da extensão da visão estrutural;
- **Com respeito à** produtividade e simplicidade conseguidas durante a autoria;
- **Do ponto de vista do** autor de documentos NCL;
- **No contexto de** estudantes de graduação dos cursos da área de computação;

O intuito de caracterizar a adoção da extensão é verificar o quanto de tempo de autoria da aplicação é economizado e se existe alguma melhora com relação à facilidade de integrar um código imperativo nas aplicações NCL. Assim, pretende-se responder, por meio de coleta e análise de dados quantitativos e qualitativos durante a operação do experimento, as seguintes questões de pesquisas:

- **Questão 1:** O uso da extensão proposta na visão estrutural consegue diminuir o tempo para integrar um código imperativo na aplicação NCL quando comparado à integração sem uso da extensão?
- **Questão 2:** O uso da extensão proposta na visão estrutural aumenta o grau de facilidade para integrar um código imperativo quando comprado à integração sem uso da extensão?

No contexto do desenvolvimento de aplicações NCL, o escopo para comparar com a visão estrutural sem a extensão fica limitado às seguintes medições:

- **Tempo de integração**: tempo para usar métodos (com um ou mais parâmetros) da mídia imperativa e para utilizar o resultado do método, ou seja, seu valor de retorno com outra mídia da aplicação;
- **Simplicidade**: facilidade em executar métodos de uma mídia imperativa e usar seus valores de retorno de forma correta sem a necessidade de escrever código na linguagem textual;

As métricas são relacionadas aos dados do tempo para integrar um código imperativo em casos particulares de processamento imperativo numa aplicação. A unidade de medida desses dados é segundos.

7.2.2 Planejamento do Experimento

Seleção do Contexto

O estudo foi realizado em ambiente interno (laboratório) para melhor controle, e dessa forma, tem modo off-line. Mesmo com a ciência que profissionais representam melhor o contexto de estudo, foram utilizados estudantes como sujeitos dos experimentos devido a dificuldade de acesso a profissionais da área. Porém, já que o estudo procura avaliar a integração entre duas visões de programação da plataforma Ginga-NCL, a escolha dos sujeitos considerou indivíduos com pouca experiência em linguagens de programação imperativa e realizou um treinamento para tentar nivelar o conhecimento da linguagem NCL.

Sobre os problemas abordados, foram escolhidas aplicações que são reais e que possuem como um dos requisitos a integração com lógica imperativa complexa, porém, de maneiras distintas. Como consequência, é importante também deixar claro que os estudos

trataram de um contexto específico do domínio de aplicações de TV Digital, ou seja, os resultados obtidos só podem ser considerados em situações que se exige o uso de código imperativo nas aplicações de TVDi.

Seleção de Variáveis

O tempo de integração define a métrica de produtividade de desenvolvimento das aplicações e por isso é a variável resposta ou **variável dependente** do experimento. Como **variáveis independentes**, que definem como empregar os dois tipos de visão (com e sem extensão), foram definidas: (1) desenvolvimento de aplicações NCL que utilizam o código-fonte Lua apenas para implementar lógica imperativa complexa; (2) especificação dos requisitos da aplicação que será implementada utilizando uma linguagem precisa e comum na área fazendo uso de *storyboards* e *checklist*;; e (3) utilização de ambientes de desenvolvimento de aplicações NCL e Lua que são compatíveis com a norma Ginga-NCL definindo na ABNT.

Fatores e Níveis

Este experimento apresenta um único fator, que se trata da ferramenta a ser utilizada. Assim, são definidos dois níveis desse fator: o uso do NCL Composer na versão original, isto é, sem a extensão para integração de mídia imperativa e uso do NCL Composer com a extensão. A descrição das duas ferramentas foi apresentada no capítulo anterior. Os níveis e os respectivos identificadores são:

- C1: utilização do NCL Composer na versão 0.1.5 disponibilizada em (TELEMÍDIA, 2013), representando o uso do NCL Composer original, sem a extensão para integração de mídias imperativas.
- C2: utilização do NCL Composer na versão 0.1.5 com a extensão para integração de mídias imperativas.

Ambas as metodologias dependem de conhecimento prévio da linguagem NCL e do entendimento da autoria utilizando o NCL Composer. Para tanto, foram realizados treinamento para uso da tecnologia (NCL) e das funcionalidades das ferramentas. Nesse último caso, foi realizado um treinamento com uma carga horária igual de treinamento para ambas os fatores.

Hipóteses

As hipóteses aqui levantadas foram obtidas a partir da primeira questão de pesquisa proposta durante a etapa de definição do experimento. Tais hipóteses, após a execução do experimento, serão submetidas a testes estatísticos, que fornecem informações definitivas para rejeitar (ou não) as hipóteses nulas em favor das hipóteses alternativas. Estes testes são realizados durante a análise dos resultados do experimento (seção 7.2.4), e constituem a principal fonte de informação para saber quais são as conclusões deste estudo experimental.

Considerando Cx , x= 1 e 2 as versões do NCL Composer utilizadas nos experimentos e a métrica TI (Fx) sendo o tempo de implementação da aplicação-problema utilizando C1 ou C2, as seguintes hipóteses nulas e as respectivas hipóteses alternativas foram definidas:

- **Hipótese Nula** ($H_0 1$): O NCL Composer sem e com a extensão são semelhantes com relação às propriedades de velocidade (TI) durante a integração de código imperativo nas aplicações NCL.

$$H_0 1: TI\ (C1) \cong TI(C2)$$

- **Hipótese Alternativa** ($H_1 1$): O NCL Composer sem e com a extensão são diferentes com relação às propriedades de velocidade (TI) durante a integração de código imperativo.

$$H_1 1: TI\ (C1)\ \neq\ TI(C2)$$

Objetos do Experimento

Os objetos do experimento representam os problemas a serem resolvidos pelos participantes tomando como entrada as ferramentas a serem avaliadas. Foram selecionadas duas aplicações comuns ao contexto da TV digital: *Quiz* (apêndice B.1) e *Slide Show* (apêndice B.2). A escolha foi feita com base no fato dos requisitos delas demandarem a necessidade, em algum momento do seu sincronismo temporal, de integração com procedimento feito por um código imperativo.

Sujeitos

Todos os sujeitos para o estudo são estudantes de graduação na área de computação e conhecem alguma linguagem de programação. Eles não possuem experiência no uso de código imperativo no NCL, entretanto eles já tinham tido contato com a linguagem NCL e outras linguagens de marcação como é o caso do HTML. Para nivelar o conhecimento, um treinamento prévio foi feito para uniformizar o conhecimento da linguagem NCL e do uso de código imperativo. Eles recebem um treinamento prévio na ferramenta NCL Composer e também na extensão proposta nesta dissertação.

Projeto Experimental

O objetivo do estudo é analisar o uso de dois tratamentos (NCL Composer com e sem extensão) com relação a velocidade para usar um código imperativo. O experimento é caracterizado como um estudo de dois fatores. Como a experiência do autor pode influenciar no resultado do experimento, o primeiro fator refere-se ao nível de experiência dos autores de documentos com relação à linguagem NCL, a ferramenta Composer e ao uso de objetos de mídia imperativos. O segundo fator está relacionado à complexidade envolvida no problema a ser resolvido pelo tratamento. Um problema mais complexo é resolvido em mais tempo e problemas simples em menos tempo. Tratar diferentes tipos de problemas aumenta a validade do estudo para diferentes tipos de aplicações-problema como também visa diminuir o efeito de aprendizado entre os dois tratamentos.

Considerando esses fatores o experimento deste estudo utiliza um plano experimental de quadrado latino (BOX, HUNTER e HUNTER, 2005). A Tabela 7.1 exemplifica a organização do experimento, onde cada quadrado é representado por um par de sujeitos, que são selecionados para desenvolver a mesma aplicação (**Aplicação 1**), cada um utilizando uma ferramenta. Num segundo momento, cada sujeito desenvolve outra aplicação (**Aplicação 2**) alterando a ferramenta utilizada. Qual ferramenta irá ser utilizada e qual será a aplicação-problema são feitas aleatoriamente.

Tabela 7.1 – Exemplos de possíveis *layouts* do experimento (quadrado latino).

	Aplicação 1	**Aplicação 2**		**Aplicação 1**	**Aplicação 2**	
Sujeito 1	F1	F2	**Sujeito 3**	F1	F2	
Sujeito 2	F2	F1	**Sujeito 4**	F2	F1	**...**
	Quadrado 1			**Quadrado 2**		

Os resultados foram analisados utilizando os testes de análise variância ANOVA (BOX, HUNTER e HUNTER, 2005). Ferramentas visuais também foram empregados como suporte à análise dos resultados. A seguir são explicados os detalhes da execução e resultados de cada experimento.

7.2.3 Operação

Ao todo participaram do estudo 8 estudantes do curso da área de Computação. Tais estudantes alegaram ter em média de 2 a 3 anos de experiência em programação e possuírem experiência em linguagens declarativas como é o caso de HTML. A metade deles já possuíam conhecimento na linguagem NCL e alguns deles já haviam tido contato alguma vez com o uso de objetos imperativos NCLua na NCL. Nunca os participantes haviam utilizado o NCL Composer para desenvolvimento.

Para efeito de aprendizagem na ferramenta, os participantes receberam o mesmo tipo de treinamento. Inicialmente, as pessoas receberam um treinamento para aprender sobre os conceitos da linguagem NCL e do uso de mídias imperativas. Essa fase serviu para suprir e equiparar entre os participantes o conhecimento básico que é necessário quando eles fossem utilizar o NCL Composer. Em seguida, foi feito um treinamento em como realizar o uso de mídias imperativas no NCL Composer e posteriormente como isso seria feito na extensão desenvolvida. Nesse sentido, ambos os participantes tiveram um conhecimento equiparável a respeito dos requisitos básicos para autoria, bem como com relação ao uso das duas ferramentas.

Como descrito no projeto experimental, as duas ferramentas são testadas em sequencia pelos participantes do estudo, são elas o NCL Composer com e sem a extensão da visão estrutural. De modo a diminuir o efeito de aprendizagem das duas ferramentas, a escolha da ferramenta entre os participantes é feita aleatoriamente e a aplicação a ser desenvolvida por eles são diferentes.

Cada ferramenta é utilizada para complementar a parte imperativa de duas aplicações pré-desenvolvidas: aplicação de *Quiz* e aplicação de *Slide Show*. Considerando o projeto do quadrado latino (explicado na seção anterior), um sujeito que implementa a aplicação 1 utilizando o NCL Composer sem extensão, vai obrigatoriamente, num segundo momento, implementar a aplicação 2 utilizando a extensão da ferramenta.

O experimento foi realizado no laboratório LAVID (Laboratório de Aplicações em Vídeo Digital) da UFPB. A definição das duplas participantes das 4 réplicas dos quadrados

latinos foi realizada aleatoriamente por meio de um sorteio. O primeiro estudante retirava uma ficha aleatoriamente para definir qual seria a primeira aplicação a ser desenvolvida, e por fim, fazia-se o sorteio de qual ferramenta seria utilizada para desenvolvê-la. Com esses dois sorteios o quadrado estava definido (Tabela 7.1), pois necessariamente o primeiro sujeito do quadrado deveria usar a outra versão do Composer para desenvolver a outra aplicação. E o segundo sujeito usaria a versão do Composer oposta do primeiro para desenvolver as aplicações.

Após definidas a ferramenta e a primeira aplicação, os participantes recebiam a descrição da aplicação e começavam o desenvolvimento marcando o tempo total para a integração do código imperativo. Os próprios participantes receberam instruções para operar o relógio que marcava o tempo total para integração. Em caso de dúvidas, pausas ou problemas no ambiente, o usuário parava o relógio e poderia continuar posteriormente de onde parou.

A descrição da aplicação continha os requisitos básicos da aplicação e uma lista de tarefas que necessitavam ser feitas para integrar o código imperativo e assim completar o desenvolvimento da aplicação. Já que esse estudo foca-se apenas em medir o esforço da integração de um código imperativo, foi tomado cuidado dos participantes não desenvolverem a aplicação por completo. A tarefa se resumiu apenas em complementar a parte imperativa da aplicação que estava faltando no esqueleto da aplicação. O apêndice B contém a descrição, o esqueleto e a lista de tarefas imperativas que os participantes usaram para as duas aplicações.

Os esqueletos das aplicações (apêndice B.1 e apêndice B.2) consistiam em um projeto pré-pronto do NCL Composer que já continha na visão estrutural as mídias de conteúdo visual (imagens, vídeo e etc.) e os elos que fossem necessários para implementar o sincronismo temporal da aplicação. Na resolução da questão o usuário cria as mídias imperativas e as ações e/ou os eventos que estão envolvidos ou com a chamada de um método ou com a leitura do valor de retorno de um método da mídia imperativa. Nesse sentido, o participante apenas usava como base os elos existentes no esqueleto para complementá-lo com as ações e eventos relacionados especificamente com o processamento imperativo da aplicação.

Ao final do desenvolvimento os participantes recebiam um questionário, que mais tarde foi usado na avaliação qualitativa para descobrir a opinião quanto à simplicidade conseguida por meio da extensão.

7.2.4 Análise e Interpretação dos Dados

Análise Quantitativa

Os dados obtidos com a execução do experimento foram analisados com o *R Statistical Software*[3]. O formato do arquivo de entrada é o mesmo utilizado por (KULESZA, 2013) e (ACCIOLY, 2012), visto que foi empregado um projeto de experimento semelhante.

A primeira análise obtida está relacionado à distribuição dos dados utilizando um gráfico "*box-plot*" (JAIN, 1991). Este apresenta média, mediana, medidas de quartis e pontos-limite. Na Tabela 7.2 é possível ver que na média a abordagem *Com-Extensão* possui valores menores para o tempo de integração e também que o valor médio do tempo de implementação da abordagem *Com-Extensão* representa um ganho de produtividade na ordem de 1,84, (dividindo o tempo médio obtido com o uso das duas ferramentas). Acredita-se que o desvio padrão encontra-se elevado por conta da heterogeneidade do perfil e experiência dos participantes em relação à plataforma Ginga-NCL.

Tabela 7.2 – Tempos médio e desvio padrão do experimento.

	Com-Extensão	**Sem-Extensão**
Tempo médio (em segundos)	965	1758
Tempo médio (em minutos e segundos)	16m:12s	29m:40s
Desvio padrão	445	470

Além disso, ainda foram observados os resultados de cada sujeito individualmente. A Figura 7.10 mostra que em todos os casos a abordagem *Com-Extensão* obteve um tempo de implementação menor do que a abordagem *Sem-Extensão*.

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

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

Figura 7.8 – Modelo linear utilizado no experimento.

[3] R: Software para análise estatística. Disponível em: http://www.r-project.org/

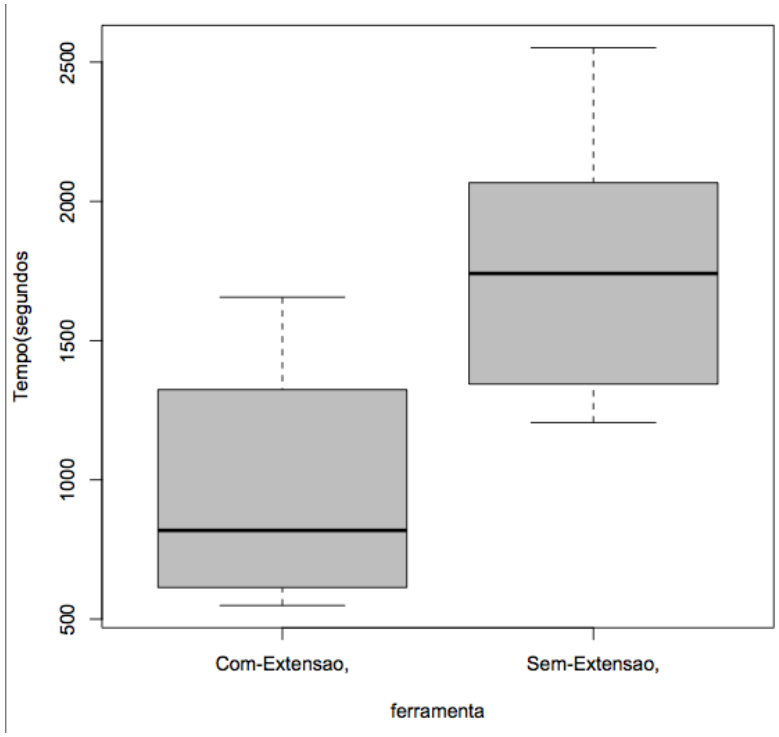


Figura 7.9 – Gráfico *box-plot* do segundo experimento.

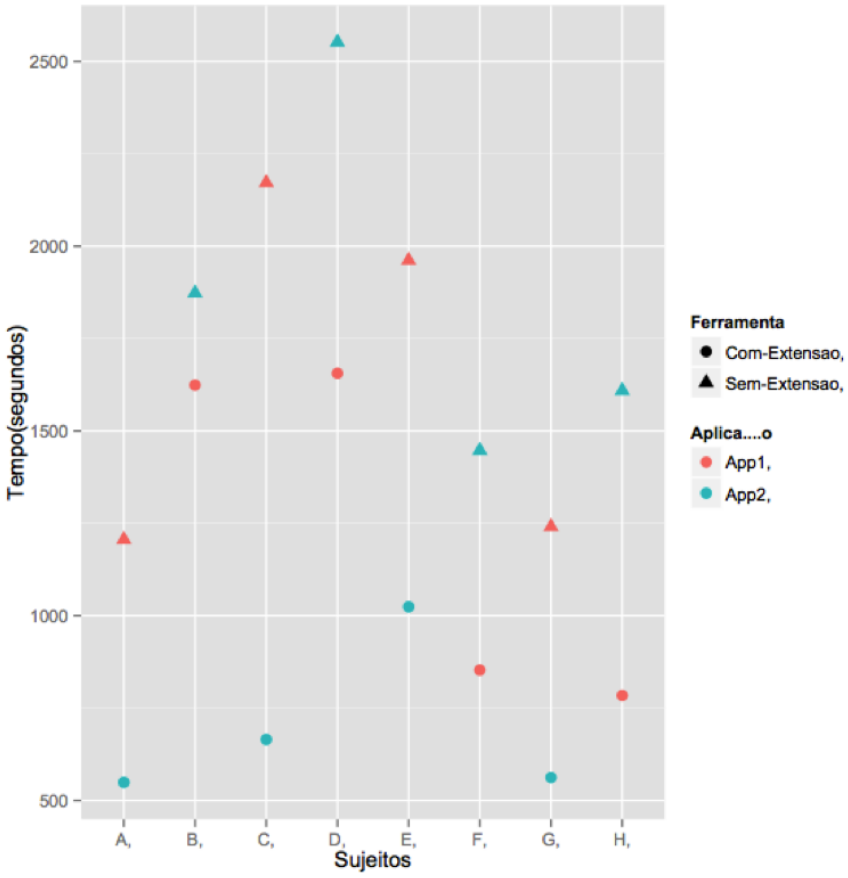


Figura 7.10 – Resultados individuais para cada abordagem.

A análise estatística dos experimentos foi realizada utilizando um modelo linear apresentado por (KEMPTHORNE e HINKELMANN, 1994) e ilustrado na Figura 7.8. Tal modelo atende o modelo de replicação de quadrado latino de “linhas cruzadas com colunas embutidas dentro das réplicas”. Todavia, antes de realizar o teste estatístico, mais dois processos foram executados sobre os dados coletados.

O primeiro, proposto por (SAKIA, 1992), aplica uma técnica de transformação do tipo *Box-Cox* para verificar se os dados possuem anomalias de não-aditividade e não-normalidade. Para tanto, é gerado um gráfico e é avaliado se a curva tem seu valor máximo entre 0 e 1. Caso sim, não é necessário fazer nenhuma transformação nos dados. De acordo com a Figura 7.11, é possível observar que não é necessário modificar os dados neste experimento.

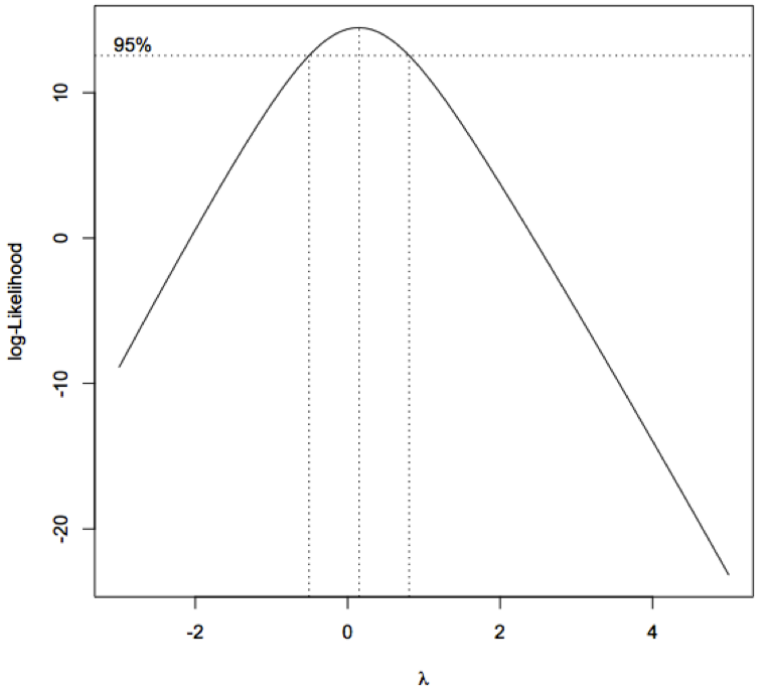


Figura 7.11 – Método Box-Cox para o primeiro experimento.

A segunda verificação executa o Teste de Aditividade de Tukey (BOX, HUNTER e HUNTER, 2005) para testar se o modelo de efeitos é aditivo, em outras palavras, analisa se os fatores das colunas e linhas do quadrado latino não afetam de modo significativo as respostas. No caso deste experimento, por exemplo, se a variação do tempo de integração das duas aplicação escolhidas podem ser comparadas diretamente. Tal teste possui as seguintes hipóteses:

H_0: o modelo é aditivo

H_1: H_0 é falso

O resultado do Teste de Aditividade de *Tukey* obteve um valor de p de 0.302, não dando evidência significativa (menor que 0.5) para rejeitar a hipótese H_0, ou seja, o modelo é aditivo.

Com a aditividade avaliada, a análise ANOVA foi aplicada para comparar o efeito das duas ferramentas (tratamento) no tempo de integração nas aplicações (variável resposta). A Tabela 7.3 exibe os resultados do teste estatístico. Assim (pelo valor em destaque vermelho), é aceitável afirmar, com um intervalo de confiança de 95% (valor de p < 0,05) , que podemos rejeitar a hipótese $H_0 1$ da seção 7.2.2, ou seja, a abordagem C2 teve efeito significativo sobre o tempo de integração nas aplicações em relação a outra abordagem C2.

Tabela 7.3 – Resultado da ANOVA do primeiro experimento

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
replica	3	1045633	348544	5.774	0.033444	*
aplicacao	1	92568	92568	1.534	0.261833	
ferramenta	1	2516189	2516189	41.685	0.000654	***
replica:sujeito	4	1433225	358306	5.936	0.027867	*
Residuals	6	362173	60362			

Análise Qualitativa

Um questionário foi aplicado com o objetivo de obter uma resposta dos participantes com relação aos ganhos de simplicidade conseguidos no uso de mídias imperativas e as opiniões gerais quanto à extensão proposta. O apêndice A apresenta o questionário usado.

De um total de 8 participantes, 5 (62,5%) disseram que a extensão tornou fácil a integração de mídia imperativa e 3 (37,5%) afirmaram que tornou muito fácil (Figura 7.12). De um modo geral os participantes disseram que a extensão ficou mais fácil e tornam mais intuitivo8 o desenvolvimento, pelo fato da criação automática fazer com que eles não se preocupassem com as propriedades, as ações e eventos durante o uso de um método. Outros ainda disseram que a extensão evita diversos tipos de erros como, por exemplo, a passagem errada de parâmetros na chamada do método e escrever o nome errado dos métodos ou dos valores de retorno. Um participante disse que simplificou, pois não precisou mais usar a visão textual durante a integração, segundo ele, “foi muito útil no sentido de agilizar a realização da integração, que no outro caso só foi possível editando o código diretamente”.

Em outra parte do questionário, foi perguntado aos participantes como eles avaliam a utilidade dos pinos de entrada e saída em ajudar a identificar o que é método e o que é valor

de retorno. Dos 8 participantes, 3 deles (37,5%) disseram que foi útil e o restante 5 (62,5%) falaram que foi extremamente útil (Figura 7.12). Segundo eles, os pinos ajudaram a visualizar melhor as entradas e saídas da mídia e são criados automaticamente, segundo um deles "remove-se uma possível confusão do que entra ou saí de informação trocada entre os componentes".

Por fim, foi perguntado para saber se a ação "chamada de método" e o evento "valor de retorno" melhor caracteriza o uso de métodos nas mídias imperativas do que a ação de atribuição e o evento "final de atribuição". Dos 8 participantes, 7 deles (87,5) responderam que sim e 1 (12,5) participante respondeu que não (Figura 7.12), por achar o conjunto ação e evento sem semelhança com o paradigma assíncrono (baseado em eventos), onde não se espera o valor de retorno logo em seguida da chamada de método.

Já outro participante disse que a ação de chamada de método resolveu confusões que ele tinha com relação a usar uma ação de *set* (ação de atribuição) para chamar um método que retorna um valor (métodos do tipo *get*). Segundo ele, ações desse tipo significaria uma chamada a um método que está recebendo um valor (ação *set*) e usando ela em um método que retorna um valor não fazia muito sentido.

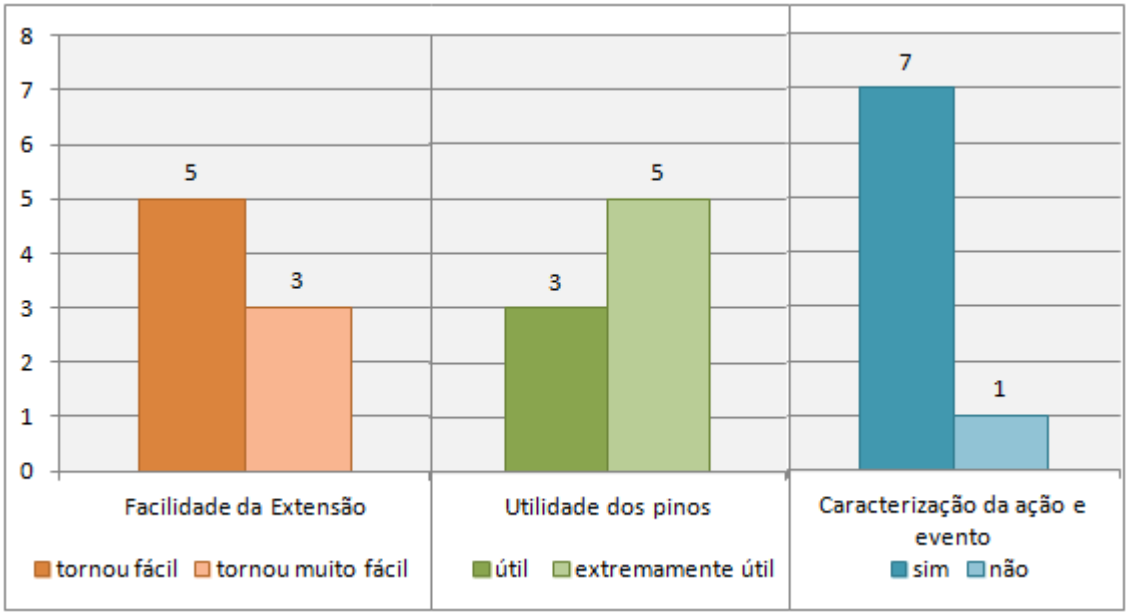


Figura 7.12 – Gráficos do estudo qualitativo.

7.3 Considerações Finais

Quanto às extensões propostas, com as anotações do código Lua foi possível obter a automatização do acesso ao conteúdo imperativo (criação automática dos pinos e criação da ação), deixando o processo mais produtivo e simples. Nesse sentido, facilitou a criação do

conteúdo presente na mídia sem que seja necessário o autor conhecer a linguagem imperativa ou depender da documentação do código. Além disso, as tarefas envolvendo a definição para usar o conteúdo imperativo são geradas automaticamente, tornando mais rápido o uso do código.

Mais ainda, o uso automático abre novas possibilidades de uso de código imperativo na NCL para autores com menos experiência, pois o conceito NCL envolvido por trás do uso de um método fica escondido do autor de documentos. Com essa simplificação as pessoas talvez não precise necessariamente entender os conceitos de ações para usar determinado conteúdo imperativo.

Com as alterações visuais foi possível tratar o uso de código imperativo de uma maneira que facilitou a distinção do conteúdo quanto a ele ser um método, um valor de retorno ou uma variável. De acordo com os participantes, os pinos auxiliaram-nos a distinguir os pontos de entrada e saída da mídia. A interface de preenchimento individualmente dos parâmetros permitiu ao autor visualizar melhor as necessidades de passar valores para a mídia.

Usar o mesmo elemento *property* para representar os trechos de conteúdo independentemente do tipo de objeto de mídia (por exemplo, posicionamento visual da mídia como também a execução de conteúdo imperativo), deixa mais simples o desenvolvimento. Em contra partida, visualmente o autor pode confundir quando propriedades de apresentação se misturam com propriedades que se referem a métodos. Nesse contexto, os pinos ajudam o autor a distinguir as propriedades de método das propriedades relacionadas à apresentação do objeto de mídia, criando uma identidade de qual tipo de conteúdo esta sendo acessado. Mais ainda, a diferenciação dos pinos situações resolve o caso de distinguir o caso de mídia com vários métodos e valores de retorno.

Com relação à criação de uma nova ação de chamada de método e evento de valor de retorno, eles são conceitos visuais criados com o intuito de facilitar a autoria. Sintaticamente eles são representados em termos dos conceitos presentes na linguagem NCL. Do ponto de vista semântico, o emprego da ação de atribuição, para usar um método, não traduz de maneira ideal a execução de um método. Ela traduz a atribuição de valores a nós de propriedades, o que pode trazer um desentendimento com relação ao uso de conteúdo imperativo. Por exemplo, um participante do experimento disse que a ação de chamada de método resolveu confusões com relação a usar uma ação de set (ação de atribuição set) para chamar um método que retorna um valor (métodos do tipo get).

Este trabalho suporta o uso de qualquer tipo de mídia imperativa, integrando desde os códigos para processar dados e adicionar uma característica mais dinâmica para a aplicação, como também códigos para reutilizar serviços web em aplicações NCL.

8 CONSIDERAÇÕES FINAIS

Este trabalho propôs uma extensão para a visão estrutural que visa facilitar e tornar mais produtivo a tarefa de autoria de aplicações NCL envolvendo a integração de código imperativo. A extensão auxilia o autor de documentos durante o uso de métodos e variáveis presentes numa mídia NCLua que foi desenvolvida por um programador Lua. A pesquisa aqui empreendida sucedeu-se devido à visão estrutural atualmente possuir diversos casos que impactam negativamente no tempo durante o uso de uma mídia imperativa. O problema foi resolvido como demonstrado no Capítulo 5, em que foi desenvolvido um conjunto de extensões, para tratar cada caso que impactava negativamente no tempo de uso de um código imperativo.

Por meio da extensão, com as anotações do código Lua, foi possível analisar a automatização do acesso ao conteúdo imperativo (criação de propriedades e criação da ação), deixando o processo mais produtivo e simples. Este trabalho suporta o uso de qualquer tipo de mídia imperativa, integrando desde os códigos para processar dados e adicionar uma característica mais dinâmica para a aplicação, como também códigos para reutilizar serviços web em aplicações NCL.

Com as alterações visuais foi possível tratar o uso de código imperativo de uma maneira que facilitou a distinção do conteúdo quanto a ele ser um método, um valor de retorno ou uma variável. A interface de preenchimento individualmente dos parâmetros permitiu ao autor visualizar melhor as necessidades de passar valores para a mídia.

Uma contribuição deste estudo é mostrar experimentalmente o quão mais rápido fica o uso do conteúdo de uma mídia imperativa em diversos casos envolvendo a integração de

código imperativo com os outros elementos de mídia na visão estrutural. Como mostrado na análise do experimento quantitativo o uso da extensão proposta reduziu o tempo de integração em 1,86, ou seja, quase pela metade.

Outra contribuição desta abordagem é na simplicidade, onde agora o conteúdo imperativo (muito complexo de ser entendido pelo autor de documentos) passa a ser disponibilizado automaticamente sem ser necessário contato com a estrutura interna do código Lua ou até mesmo com o programador.

8.1 Trabalhos Futuros

No prosseguimento desse trabalho, um trabalho futuro que pode ser atacado é com relação à realização de novos estudos empíricos. Um primeiro experimento a ser feito é realizar os mesmos estudos feitos, mas com um maior número de amostras para avaliar com mais veracidade a economia de tempo na integração de código imperativo.

No prosseguimento desse trabalho, um trabalho futuro que pode ser atacado é com relação à realização de novos estudos empíricos. Um primeiro experimento a ser feito é realizar os mesmos estudos feitos, mas com um maior número de amostras para avaliar com mais veracidade a economia de tempo na integração de código imperativo.

Outros tipos de estudos qualitativos podem ser feito para responder com mais eficiência se a abordagem trouxe mais simplicidade. Foi feito apenas um estudo qualitativo com aplicação de um questionário semi-estruturado e poucas pessoas, e assim não é possível afirmar realmente que houve uma melhora com relação a simplicidade.

Mais especificamente com relação à extensão um ponto que pode ser estudado é a inclusão semi-automática de qualquer serviço que se encontra na web dentro da *aba dos serviços web*. Um ponto negativo existente é que para incluir o serviço na aba, ele deve ser implementado por um programador Lua. A partir da descrição do serviço tanto o código da mídia como o código do cliente do serviço web poderiam ser gerados automaticamente sem esforço de programação.

Referências

ACCIOLY, P. R. G. **Comparing Different Testing Strategies for Software Product Lines**. Dissertação (Mestrado). Universidade Federal de Pernambuco, Recife, Brasil. [S.l.]. 2012.

ALFONSI, B. I Want My IPTV: Internet Protocol Television Predicted a Winner. **IEEE Distributed Systems Online**, Piscataway, NJ, USA, v. 6, n. 2, 2005. ISSN ISSN: 1541-4922 DOI: 10.1109/MDSO.2005.10. Disponivel em: <http://dx.doi.org/10.1109/MDSO.2005.10>.

APPINVETOR. **App Inventor**. Disponível em: http://appinventor.mit.edu. 2013. Último acesso janeiro de 2013.

AZEVEDO, R. G. A. et al. **Textual authoring of interactive digital TV applications**. Proceddings of the 9th international interactive conference on Interactive television (EuroITV '11). New York, NY, USA: ACM. 2011. p. 235-244.

BACHMAYER, S.; LUGMAYR, A.; KOTSIS, G. Convergence of collaborative web approaches and interactive TV program formats. **International Journal of Web Information Systems**, v. 6, p. 74-94, 2010. ISSN DOI: 10.1108/17440081011034493.

BOX, G. E. P.; HUNTER, J. S.; HUNTER, W. G. **Statistics for Experimenters:** Design, Innovation, and Discovery. 2. ed. [S.l.]: Wiley-Interscience, 2005. ISBN ISBN: 0471718130.

COELHO, R. M.; RODRIGUES, R. F.; SOARES, L. F. G. Integração de Ferramentas Gráficas e Declarativas na Autoria de Arquiteturas Modeladas através de Grafos Compostos. **X Simpósio Brasileiro de Sistemas Multimídia e Web – WebMedia**, 2004.

FERNANDES, J.; LEMOS, G.; SILVEIRA, G. Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas. **Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação, JAI-SBC**, 2004.

GOOGLE. **Google APIs Explorer**. Disponível em: https://developers.google.com/apis-explorer. 2013. Último acesso janeiro de 2013.

GUIMARÃES, R. L.; COSTA, R. M.; SOARES, L. F. G. Composer: Ambiente de autoria de aplicações declarativas para TV Digital Interativa. **Webmedia: Brazilian Symposion on Multimedia and the Web**, 2007.

IST. **Jame Production Technology**. IST. Disponível em http://www.iais.fraunhofer.de/jame.html?\&L=1. 2012. Último acesso em abril de 2012.

JAIN, R. **The Art of Computer Systems Performance Analysis:** Techniques for Experimental Design, Measurement, Simulation, and Modeling. [S.l.]: Wiley-Interscience, New York, NY, 1991.

JAVA.NET. **Web Application Description Language (WADL) - Specification and Tools Java.net**. Disponível em: http://wadl.java.net/. 2013. Último acesso janeiro de 2013.

JURISTO, N.; MORENO, A. M. **Basics of Software Engineering Experimentation**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN ISBN: 1441950117, 9781441950116.

KASKALIS, T.; TZIDAMIS, T. D.; MARGARITIS, K. G. Multimedia Authoring Tools: The Quest for an Educational Package. **Educational Technology \& Society**, v. 10, n. 3, p. 135-162, 2007. Disponivel em: <http://dblp.uni-trier.de/db/journals/ets/ets10.html#KaskalisTM07>.

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

KULESZA, R. **Uma abordagem dirigida por modelos para o desenvolvimento de aplicações multimídia**. Tese (Doutorado). Universidade Fedaral de Pernambuco, Recife, Brasil. [S.l.]. 2013.

LIMA, B. et al. Composer 3: Ambiente de autoria extensível, adaptável e multiplataforma. **WebMedia – Workshop de TV Digital Interativa (WTVDI)**, 2010.

MAKEDON, F. et al. **Multimedia authoring, development environments, and digital video editing.** Hanover, NH, USA. 2001.

MORRIS, S.; SMITH-CHAIGNEAU, A. **Interactive TV Standards:** A Guide to MHP, OCAP, and JavaTV. [S.l.]: Focal Press, 2005. ISBN ISBN: 0240806662.

MYERS, B. A. Authoring Interactive Behaviors. **The Ninth Annual NEC Research Symposium: Human Centric Multimedia Community**, 1998.

PLEUB, A. **Modeling the user interface of multimedia applications**. Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems. Berlin, Heidelberg: Springer-Verlag. 2005. p. 676-690.

SAKIA, R. M. The Box-Cox Transformation Technique: A Review. **Journal of the Royal Statistical Society. Series D (The Statistician)**, v. 41, n. 2, p. 169-178, #jan# 1992. ISSN ISBN: 00390526. Disponivel em: <http://www.jstor.org/stable/2348250>.

SANT'ANNA, F.; CERQUEIRA, R.; SOARES, L. F. G. **NCLua:** objetos imperativos lua na linguagem declarativa NCL. Proceedings of the 14th Brazilian Symposium on Multimedia and the Web. New York, NY, USA: ACM. 2008. p. 83-90.

SOARES, L. F. et al. Variable and state handling in NCL. **Multimedia Tools and Applications**, Hingham, MA, USA, v. 50, n. 3, p. 465-489, 2010. ISSN ISSN: 1380-7501 DOI: 10.1007/s11042-010-0478-2.

SOARES, L. F. G.; BARBOSA, S. D. J. **Programando em NCL 3.0:** Desenvolvimento de Aplicações para o Middleware Ginga, TV digital e Web. 1. ed. [S.l.]: Rio de Janeiro: Elsevier, 2009.

SOARES, L. F. G.; RODRIGUES, R. **Nested Context Language Part 8 - NCL Digital TV Profiles**. Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Rio de Janeiro. [S.l.]. 2006.

SOARES, L. F. G.; RODRIGUES, R. F.; SAADE, D. C. M. Modeling, authoring and formatting hypermedia documents in the HyperProp system. **Multimedia Systems**, Secaucus, NJ, USA, v. 8, n. 2, p. 118-134, 2000. ISSN ISSN: 0942-4962 DOI: 10.1007/s005300050155.

SZYPERSKI, C. **Components vs. Objects vs. Component Objects**. Object-Oriented Programming (OOP'99). [S.l.]: [s.n.]. 1999. p. 25-28.

TAVARES, T. A.; VEIGA, E. G. Um Modelo de processo para o desenvolvimento de programas para TV digital e interativa baseado em metodologias ágeis. **Workshop de Desenvolvimento Rápido de Aplicações**, 2007.

TELEMÍDIA. **Ferramenta NCL Composer**. Laboratório Telemídia. PUCRio. Disponível em: http://composer.telemidia.puc-rio.br/en/start. 2013. Último acesso janeiro de 2013.

TINYWEBDB. **TinyWebDB for Android**. Disponível em: http://appinvtinywebdb.appspot.com. 2013. Último acesso janeiro de 2013.

TRAVASSOS, G. H.; GUROV, D.; DO AMARAL, E. A. G. **Introdução à Engenharia de Software Experimental**. Programa de Engenharia de Sistemas e Computação COPPE / UFRJ. [S.l.]. 2002.

W3C. **Web Services Description Working Group**. Disponível em: http://www.w3.org/2002/ws/desc/. 2013. Último acesso janeiro de 2013.

WEBSERVICEX.NET. **Global Weather Service Description**. Disponível em: http://www.webservicex.net/globalweather.asmx. 2013. Último acesso janeiro de 2013.

WEBSERVICEX.NET. **WebserviceX.NET - XML Web Services solution provider**. Disponível em: http://www.webservicex.net/ws/default.aspx. 2013. Último acesso janeiro de 2013.

WOHLIN, C. et al. **Experimentation in software engineering:** an introduction. Norwell, MA, USA: Kluwer Academic Publishers, 2000. ISBN ISBN: 0-7923-8682-5.

YAHOO. **Yahoo! Developer Network - APIs \& Tools**. Disponível em: http://developer.yahoo.com/everything.html. 2013a. Último acesso janeiro de 2013.

YAHOO. **Métodos da API do Serviço Flickr**. Disponível em: http://www.flickr.com/services/api/. 2013b. Último acesso janeiro de 2013.

ZELKOWITZ, M. V.; WALLACE, D. R. Experimental models for validating technology. **Computer**, v. 31, n. 5, p. 23-31, 1998. ISSN ISSN: 0018-9162 DOI: 10.1109/2.675630.

APÊNDICE A – QUESTIONÁRIO DO EXPERIMENTO

Questionário Sobre a Ferramenta Composer 2

Por favor, responda com sinceridade. Qualquer dúvida consulte o instrutor.

Parte 1 - Integração de mídia NCLua

1) Numa escala de 1 a 5, onde 1 indica que não ajudou e 5 ajudou muito. Como você avalia a utilidade da ferramenta Composer 2 para usar as mídias imperativas numa aplicação NCL em relação ao Composer 1?
    a. Foi desnecessário (não ajudou nada)
    b. Pouco útil
    c. De alguma utilidade
    d. Muito útil
    e. Foi extremamente útil (ajudou muito)

    Por favor, justifique a nota atribuída na resposta acima.

2) Numa escala de 1 a 5, onde 1 indica que tornou muito fácil e 5 tornou muito difícil. Como você avalia a facilidade da ferramenta Composer 2 para usar mídias imperativas numa aplicação NCL em relação ao Composer 1?
    a. Tornou muito fácil (facilitou muito)
    b. Tornou fácil
    c. Ficou na mesma
    d. Tornou difícil
    e. Tornou muito difícil (complicou muito)

Por favor, justifique a nota atribuída na resposta acima.

3) Numa escala de 1 a 5, onde 1 indica que não ajudou e 5 ajudou muito. Como você avalia a utilidade dos pinos de entrada e de saída em ajudar a identificar o que é método e o que é valor de retorno?
    a. Foi desnecessário (não ajudou nada)
    b. Pouco útil
    c. De alguma utilidade
    d. Muito útil
    e. Foi extremamente útil (ajudou muito)
4) Do seu ponto de vista, a ação de chamada de método e o evento valor de retorno melhor caracteriza o uso de métodos nas mídias imperativas do que a ação de atribuição e o evento de final de atribuição?
    a. Sim
    b. Não

    Por favor, justifique a resposta.

Parte 2 – Informações Sobre o Participante

5) Qual o nome do seu curso e qual o seu período?

6) Se você já programa, quanto tempo você programa?
    a. Menos de 6 meses
    b. Entre 6 meses e 1 ano
    c. Entre 1 ano e 2 anos
    d. Mais que 2 anos
    e. Mais que 3 anos

7) Cite as linguagens de programação que você já conhece?

8) Você tem experiência com alguma linguagem de programação declarativa abaixo (marque mais de uma opção, se tiver)?
    a. HTML

b. XML
c. HTML 5
d. Flash
e. NCL

9) Você tem experiência com alguma tecnologia de linguagens de programação abaixo (marque mais de uma opção, se tiver)?
    a. Banco de dados
    b. Sockets
    c. HTTP
    d. Web Services (SOAP ou REST)
    e. Aplicações Web
    f. Outra:

APÊNDICE B – DESCRIÇÕES DAS APLICAÇÕES UTILIZADAS NO EXPERIMENTO

B.1 – Descrição da Aplicação 1

Aplicação de perguntas e respostas (Quiz)

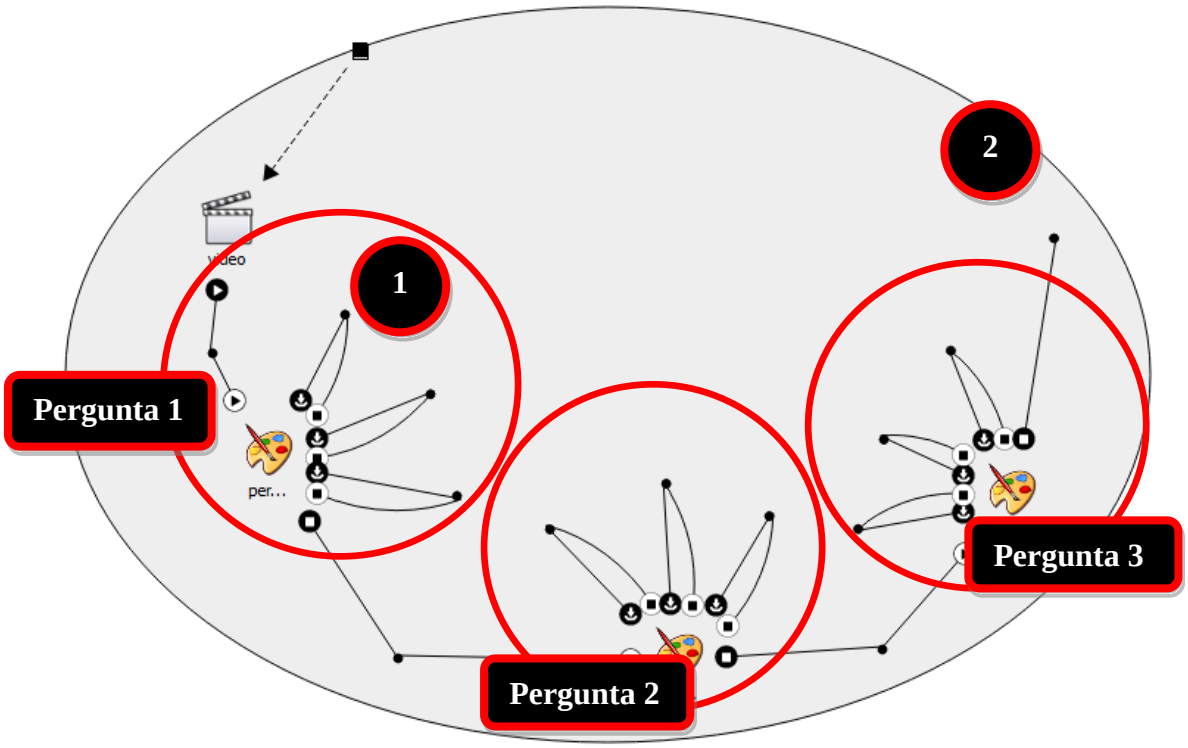
Complete o esqueleto da aplicação utilizando o NCL Composer criando as mídias Luas e preenchendo as ações restantes necessárias para executar os métodos nas mídias Lua se ler os seus valores de retorno.

Descrição da aplicação (Requisitos)

A aplicação de Quiz realiza um jogo de perguntas e respostas. O Quiz é composto por três perguntas onde somente uma das suas respostas (três ao todo) está correta. A resposta de cada pergunta é feita pelos botões ‘1’, ‘2’ ou ‘3’. Quando o usuário pressionar o botão ‘1’ escolhe a primeira resposta e assim por diante. Para cada resposta o método **answerQuestion(idQuestion, userChoose)** deverá ser chamado com dois parâmetros: (1) **idQuestion** ao identificador da questão atual que esta sendo respondida 1, 2 ou 3 (sendo 1 para a primeira questão e assim por diante) e (2) **userChoose** o valor da resposta escolhida pelo usuário sendo ‘1’ para a escolha da primeira resposta, ‘2’ para a segunda e ‘3’ para a terceira resposta.

Ao final de responder a terceira pergunta, uma mensagem no formato “X acertos”, onde X é o número de acertos, deverá aparecer na tela informando quantas questões foram respondidas corretamente.

Esqueleto



Elementos presentes

O esqueleto já possui criado as mídias do vídeo e das imagens das perguntas além de todos os links necessários para implementar a lógica de apresentação da aplicação.Para cada pergunta existem três links que implementam o requisito de escolher a resposta da questão pelos botões ‘1’, ‘2’ ou ‘3’. Por exemplo, os elos na pergunta 1 (ponto 1 na imagem) já possuem definido a condição de disparo da ação (evento de seleção da pergunta pelos botões ‘1’, ‘2’ ou ‘3’) e a ação de parada para quando a escolha do voto for feita.

Elementos ausentes

(1) As mídias Luas.

Código Lua **scriptQuiz.lua**

- registra as escolhas para cada questão.
- armazena a quantidade de perguntas respondidas corretamente.

Código Lua **label.lua**

- exibi a quantidade de acertos das questões respondidas.

Observação: utilize a descrição do código Lua (dada abaixo) para saber quais os métodos existentes bem como os seus parâmetros e os seus valores de retorno.

(2) Criar as ações para executar o método Lua (**answerQuestion(idQuestion, userChoose)**) que registra a resposta da questão a partir do pressionamento dos botões '1', '2' ou '3'. **Observação.** Utilize o link de escolha da resposta (ponto 1 da imagem) como base. Observe que para cada link uma escolha diferente de resposta é feita.

(3) Definir os parâmetros de cada método chamado anteriormente. **Observação.** No Composer 1 a entrada dos dois parâmetros do método são separados por virgula. Exemplos 1,2 1,3 2,3... No Composer 2 utilize a interface.

(4) Criar a ação que executa o método Lua (**getHits**()) para recuperar a quantidade de acertos do quiz (código Lua **scriptQuiz.lua**). **Observação.** Utilize o link presente no esqueleto no ponto 2 da imagem com base para criar a ação.

(5) Criar o link para ler o valor de retorno da execução do método anterior (método **getHits**()) e em seguida exibi-lo na tela.

Descrição do código Lua

1) **Código Lua do script quiz (scriptQuiz.lua)**

Este código lua é responsável por registrar as respostas de cada questão e também computar a quantidade de respostas que foram corretamente respondidas. Ele é formado por dois métodos:

Métodos

answerQuestion(idQuestion, userChoose)

answerQuestion responde cada questão com a resposta escolhida. O parâmetro **idQuestion** refere-se ao identificador da questão (1, 2 ou 3).O parâmetro **choose** refere-se ao número da resposta (1, 2 ou 3). Esta resposta é escolhida pelo usuário do quiz.

Valor de retorno: não possui nenhum

getHits()

getHits retorna a quantidade de perguntas corretas (acertos) através do valor de retorno hits.

Valor de retorno: hits. Armazena a quantidade de perguntas corretas após o método ser chamado.

2) Código Lua (label.lua)

Exibe um texto a partir de um determinado valor

Métodos

showText(text)

Exibe na tela um texto passado como parâmetro.

Valor de retorno: não possui valor de retorno.

B.2 – Descrição da Aplicação 2

Aplicação de *Slide Show*

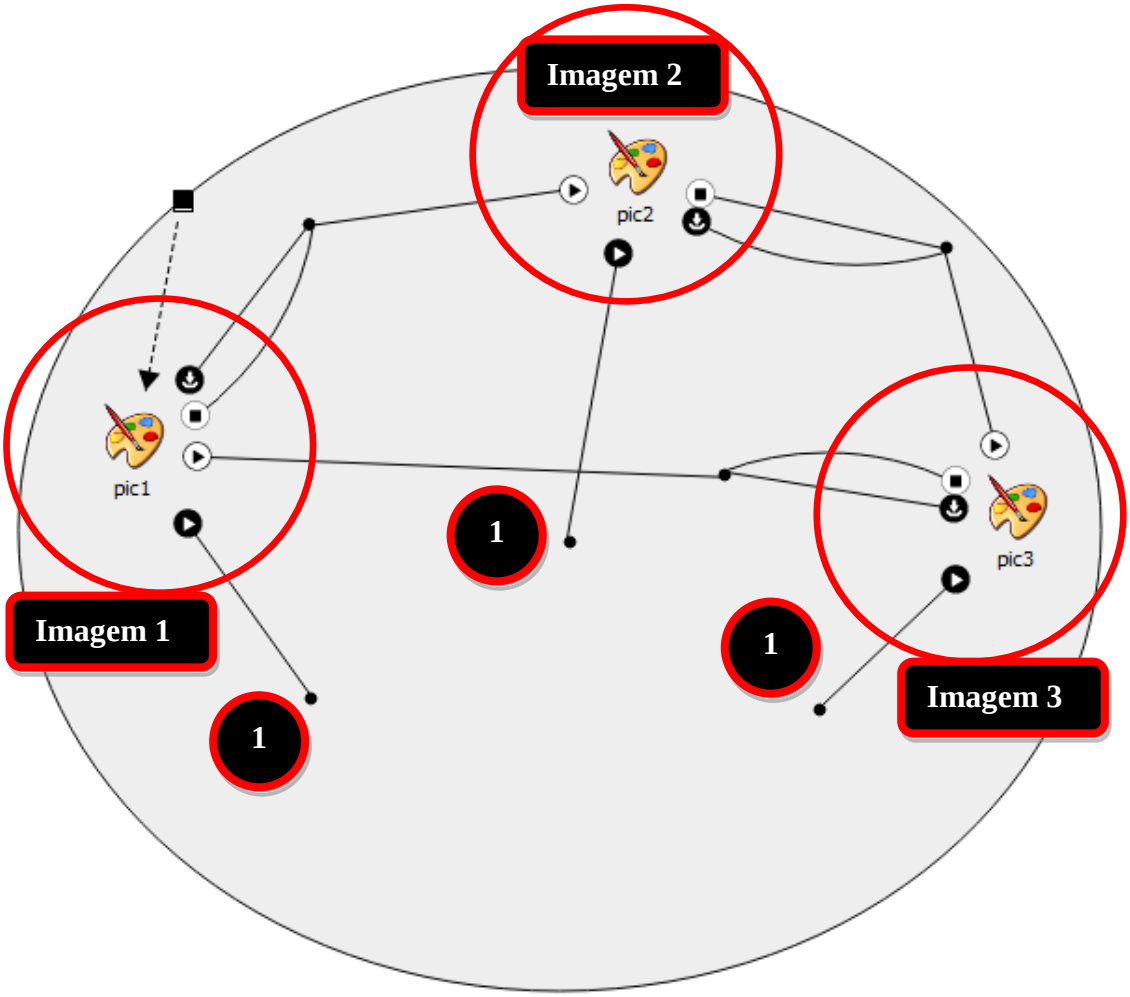
Completar o esqueleto da aplicação utilizando o NCL Composer criando as mídias Luas e as ações e os links restantes que são necessários para executar os métodos nas mídias Luas e ler os seus valores de retorno.

Descrição da aplicação (Requisitos)

A aplicação de *slide show* exibe três imagens e uma descrição para cada imagem. As imagens são exibidas em sequência à medida que é apertado o botão (*CURSOR_RIGHT*). Quando a imagem é exibida, a sua descrição deve também ser exibida na tela. Quando chegar na última imagem a primeira imagem e sua descrição é exibida novamente.

As descrições das imagens são obtidas a partir de uma mídia Lua que consulta as descrições das imagens a partir de um servidor da internet (URL da imagem).

Esqueleto



Elementos presentes

O esqueleto já possui criado as mídias das imagens do *slide show* além de todos os links necessários para implementar a lógica de apresentação da aplicação.As imagens são apresentadas pelo botão *CURSOR_RIGHT*, quando isso acontece a imagem seguinte é apresentada e a atual parada.

Elementos Ausentes

1) As mídias Luas

Código Lua **scriptDescription.lua**

- Recuperar a descrição da imagem a partir da URL

Código Lua **label.lua**

- exibi o texto da descrição a partir de um link da web

Observação: utilize a descrição do código Lua (dada abaixo) para saber quais os métodos existentes bem como os seus parâmetros e os seus valores de retorno.

2) Criar as ações que faltam para recuperar a descrição da imagem a partir do método **readTextFromHtml**(URL) do código Lua (**scriptDescription.lua**). **Observação.** Utilize como base cada link dos pontos 1 da imagem do esqueleto.

3) Definir os parâmetros de cada método chamado anteriormente. **Observação.** Utilize os valores das URL abaixo.

4) Criar os links para ler o valor de retorno da descrição (valor de retorno referente ao método anterior) e em seguida exibi-lo usando o método **showText(descricao)** do código Lua (**label.lua**).

Use os links abaixo para consultar as descrições das três imagens:

URL http://lavid.ufpb.br/~thales/pic1Description.txt para a descrição da primeira imagem.

URL http://lavid.ufpb.br/~thales/pic2Description.txt para acessar a descrição da segunda imagem

URL http://lavid.ufpb.br/~thales/pic3Description.txt para acessar a descrição da terceira imagem

Descrição do código Lua

1) **Código Lua (scriptDescription.lua)**

Script para recuperar a descrição relativa a cada imagem a partir do servidor web.

Métodos

readTextFromHtml(URL)

Recupera a descrição da imagem a partir da URL passada como parâmetro.

Valor de retorno: description. Armazena a descrição da imagem após o método ser chamado.

2) **Código Lua (label.lua)**

Exibe o texto da descrição da imagem a partir de um link da web que referencia o arquivo texto (txt) da descrição.

Métodos

showText(descricao)

Exibe na tela a descrição da imagem passando como parâmetro o texto da descrição.

Valor de retorno: Não possui valor de retorno.