1
Introdução

O constante aumento na largura de banda e espaço de armazenamento disponível para usuários domésticos agiram como principais catalisadores da revolução no tráfego de conteúdo multimídia. Deixa-se uma rede de informações baseadas em textos e imagens estáticas para adentrar em uma nuvem de conteúdo baseada em mídias continuas e dinâmicas, como áudio e vídeo. Consolidou-se, assim, a necessidade de criação de aplicações interativas multimídias, denominadas de aplicações hipermídia e que, segundo (Hardman, 1998), são coleções de unidades de informações, e elos de referências, unificadas através de relacionamento de sincronização espacial e temporal, bem como momentos de interação com o usuário.

Geralmente, aplicações hipermídia podem ser divididas entre aplicações declarativas e aplicações imperativas. Entretanto, aplicações hipermídia são, geralmente, híbridas no sentido que elas podem ter entidades declarativas e imperativas. O paradigma a ser utilizado depende da preferência e habilidade do criador do conteúdo, assim como o propósito da aplicação hipermídia. Dentre as principais linguagens hipermídia declarativas destacam-se NCL (*Nested Context Language*) (ABNT, 2007), SMIL (*Synchronized Multimedia Markup Language*) (W3C, 2008) , BML (*Broadcast Markup Language*) (Lee et al., 2002) e XHTML (The Extensible *HyperText Markup Language*) (W3C, 2002). As linguagens imperativas mais usuais são ECMAScript (ECMA International, 2009) , Lua (Ierusalimschy, 2006) e Java.

De uma forma geral, as linguagens hipermídia declarativas permitem estruturar e relacionar informações. Tipicamente, elas definem um modelo específico para desenvolver aplicações em um domínio também específico. Essas especificações declarativas costumam estar mais perto de uma especificação em alto nível, quando comparadas com as especificações em linguagens de programação imperativas (Soares et al., 2009).

Não importa qual paradigma de programação é utilizado, escrever código-fonte para aplicações complexas é uma tarefa extremamente difícil. No início da Web, a autoria de aplicações hipermídia tinha um cenário composto pela figura de um *webmaster*, *webdesigner* ou *webdeveloper*. O conteúdo era produzido, geralmente, por pessoas que tinham expertise sobre a tecnologia e não por pessoas que detinham a informação do conteúdo. A Web se popularizou depois do surgimento de ferramentas de autoria simples e completas que possibilitaram transformar os usuários passivos em usuários criadores de aplicações.

É possível traçar um paralelo com o atual estágio de desenvolvimento de aplicações hipermídia para Televisão Digital (TVD), onde o conteúdo interativo ainda vem sendo desenvolvido, quase exclusivamente, por profissionais com profundo conhecimento das linguagens empregadas. O cenário de concepção e criação de aplicações Televisão Digital Interativa (TVDi), no entanto, é composto por diferentes atores que pertencem a diferentes grupos de usuários potencialmente interessados, e com perfis tão diversos quanto: usuários domésticos, designers gráficos, comunicadores e programadores experientes.

Cada um desses grupos de usuários tem expectativas diferentes acerca de uma ferramenta de autoria para aplicações interativas. Mais ainda, esses grupos estão inseridos em diferentes ambientes de produção de conteúdo, conforme pode-se acompanhar pela Figura 1. Designers gráficos, no estúdio de autoria (2), esperam trabalhar em um nível de abstração mais próximo da apresentação final do que programadores inseridos em um ambiente de desenvolvimento e testes (3). Os profissionais no ambiente de radiodifusão (1) necessitam de uma ferramenta que possibilite a transmissão das aplicações e em diferentes redes de distribuição de conteúdo. O próprio telespectador também anseia por uma ferramenta para enriquecer o conteúdo com suas próprias anotações, promovendo a chamada TV Social (Oehlberg et al., 2006).

Diversos trabalhos avaliam ferramentas de autoria baseadas nas necessidades do autor, isto é, nos aspectos funcionais que uma ferramenta deve prover. Dois tipos de ferramentas sempre são discutidas (Azevedo, Lima, Neto, & Texeira, 2009): ferramentas de autoria textuais, que ajudam os autores na codificação direta do código-fonte com funcionalidades como auto-completação, sugestão de palavras e depuração; e ferramentas de autorias baseadas em

abstrações visuais que tentam esconder do autor a complexidade da linguagem (Soares Neto & Soares, 2009).

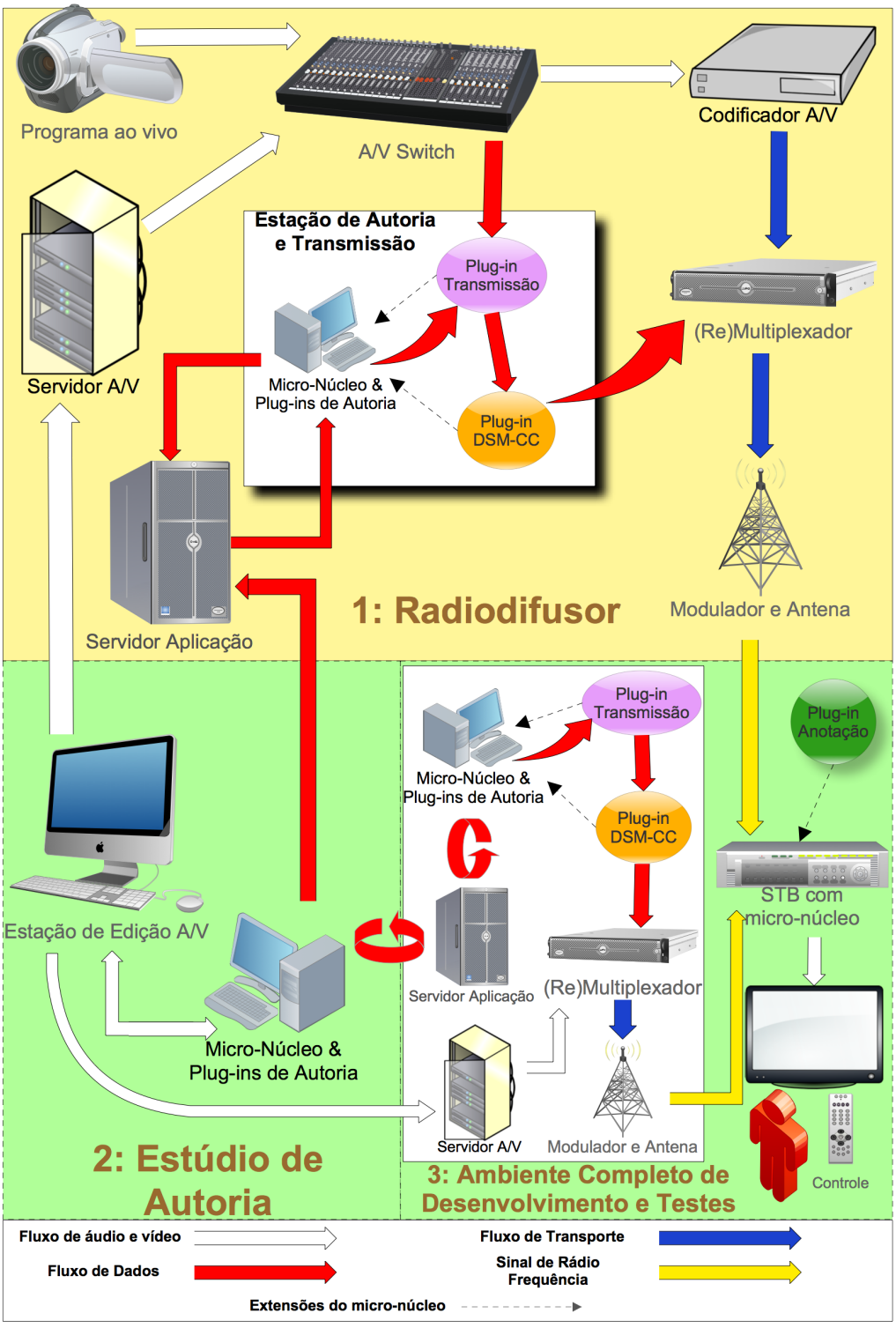

Figura 1 - Cadeia de radiodifusão de aplicativos hipermídia

Atualmente, as ferramentas de autoria para aplicações de TVDi buscam atender perfis de usuários específicos, comentados anteriormente. Nesse sentido, as ferramentas que utilizam abstrações visuais podem ser baseadas em grafos

compostos (Coelho, Rodrigues, & Soares, 2004), grafos de cena, paradigma de linha temporal e mesmo no conceito de "*What You See is What You Get*" (Araújo, 2009), mais voltadas para os usuários iniciantes. Por outro lado, as ferramentas textuais trazem facilidades para programadores, seu público-alvo.

Ainda com relação a aspectos funcionais das ferramentas, existem trabalhos (Azevedo, Lima, Neto, & Texeira, 2009) discutindo as vantagens e desvantagens de ferramentas de autoria textuais sobre ferramentas visuais. Dentre as principais desvantagens dessas últimas, está o fato dessas ferramentas inserirem uma camada de abstração que tem por consequência a quebra da comunicação entre o projetista da linguagem e o público-alvo.

Ferramentas de autoria visuais em geral oferecem múltiplas visões, para representar diferentes aspectos do modelo conceitual da linguagem alvo. Uma visão pode ser considerada como uma abstração que restringe o olhar do autor a uma determinada perspectiva da aplicação, como por exemplo, uma visão de linha do tempo (visão temporal). Não importa o tipo (textual ou visual), uma ferramenta de autoria tem um relacionamento intrínseco com o modelo semântico ao qual provê suporte. Apesar de possuir uma interface gráfica sofisticada ou diversos recursos para manipulação direta de entidades da aplicação, um autor não pode negligenciar os aspectos fundamentais da linguagem alvo.

Usualmente, a distância semântica entre o modelo de dados central e o modelo de dados interno de uma visão dificulta o processo de tradução entre eles. Por conta desse problema, algumas ferramentas não são extensíveis, trabalhando com um número limitado de visões; outras executam a tradução em apenas uma via, sem permitir a sincronização entre os modelos internos das visões (a mudança em uma visão não é refletida automaticamente nas demais); outras consomem grandes recursos computacionais, requerendo plataformas de ponta ou trabalhando com grandes atrasos, comprometendo seu uso nas edições em tempo real.

Esses e outros problemas relacionados a requisitos funcionais são levantados e discutidos em diversos trabalhos (Song, Kim, Ramalingam, Miller, & Yi, 1996) (Vazirgiannis, Kostalas, & Sellis, 1999) (Bulterman & Hardman, 2005) (Bulterman & Hardman, 2005) (Bulterman & Hardman, 2005) na literatura. Tais requisitos são reforçados e estendidos em (Guimarães, 2007). Embora muitos trabalhos reportem os aspectos funcionais das ferramentas, quase nenhum lida com os seus aspectos não-funcionais, tão ou mais importantes, principalmente

quando se quer dar suporte ao desenvolvimento de aplicações mais complexas, abraçar uma gama maior de usuários alvo e prover um ambiente de desenvolvimento robusto. Requisitos não-funcionais, doravante RNFs, podem ser definidos como requisitos de sistema que estão relacionados a restrições e medidas de qualidade (Malan & Bredemeyer, 2001). A literatura lista uma variedade de RNFs (Chung & Leite, 2009), mas este trabalho foca nos seguintes: desempenho (eficiência no uso de memória e CPU), costumabilidade, extensibilidade, escalabilidade, portabilidade e confiabilidade.

1.1.Objetivos e Contribuições

Um dos objetivos desta dissertação é levantar os requisitos não-funcionais RNF) que devem ser atendidos por uma ferramenta de autoria.

Esta dissertação também tem como objetivo a proposição de uma arquitetura para ferramentas de autoria hipermídia que visa atender as necessidades dos requisitos não-funcionais.

No entanto, o levantamento de requisitos não-funcionais muitas vezes traz uma relação de compromisso. Por exemplo, o desempenho é melhor atendido em uma ferramenta voltada para uma plataforma específica, mas que, portanto, peca em outro requisito não-funcional que é sua portabilidade a múltiplas plataformas.

Assim, baseado no levantamento feito nesta dissertação, e estabelecendo certas decisões de compromisso, esta dissertação também apresenta uma prova de conceito na implementação da segunda edição do *Composer*: uma ferramenta de autoria para aplicações NCL. NCL é a linguagem declarativa para o sistema terrestre de TVD Nipo-Brasileiro - ISDB-T_B (ABNT, 2007) e também é a recomendação ITU-T para serviços IPTV (ITU, 2009).

1.2. Organização da Dissertação

Esta dissertação está dividida como segue:

O Capítulo 2 discorre sobre algumas das principais ferramentas de autoria existentes atualmente, fazendo uma análise crítica das vantagens e desvantagens apresentadas por cada uma delas, levando em conta a linguagem à qual dá suporte e o seu público-alvo.

Tendo como base os problemas levantados no capítulo anterior, o Capítulo 3 traz um levantamento dos principais requisitos não-funcionais que uma ferramenta de autoria deve prover.

O Capítulo 4 apresenta a arquitetura que está sendo proposta pelo trabalho, detalhando cada módulo que compõe a arquitetura e discorrendo sobre as vantagens de sua adoção em ferramentas de autoria para aplicações hipermídia. O capítulo destaca como a arquitetura proposta endereça e resolve os problemas encontrados na análise dos requisitos não-funcionais.

O Capítulo 5 inicia estabelecendo algumas decisões de compromisso na implementação da nova versão da ferramenta Composer baseada na arquitetura proposta no capítulo anterior. Em seguida, o capítulo discute os detalhes relacionados a arquitetura baseada em micro-núcleo, expondo e discorrendo em profundidade a interface de programação para extensão do micro-núcleo. Além disso, traz aspectos específicos da implementação, apresentando as tecnologias utilizadas e justificando as suas escolhas. O Capítulo 5 ainda demonstra a forma de comunicação entre os plug-ins de extensão e o micro-núcleo.

No Capítulo 6 é apresentada a nova versão da ferramenta de autoria *Composer*, essa versão se sustenta na arquitetura proposta e tem como propósito servir de prova de conceito dessa arquitetura. Ainda no Capítulo 6 são apresentados os plug-ins desenvolvidos de maneira a validar a parte de extensibilidade da arquitetura. Os plug-ins foram incorporados na implementação da ferramenta *Composer*. Os plug-ins desenvolvidos foram: textual, leiaute, outline.

A forma de distribuição dos plug-ins através de um repositório, assim como as licenças que a implementação da ferramenta *Composer* e os plug-ins desenvolvidos irão utilizar também são discutidos no Capítulo 7. Ao final do capítulo, são apresentadas as conclusões e contribuições presentes nesse trabalho e as projeções de trabalhos futuros sobre a arquitetura e ferramenta *Composer*.