7
Conclusão e Trabalhos Futuros

Esta dissertação apresenta a linguagem X-SMIL para autoria declarativa de documentos hipermídia. Nessa direção, o primeiro passo do trabalho foi definir NCL 2.1, refinando alguns dos módulos de NCL 2.0 e introduzindo dois novos módulos, um para a especificação de funções de custo e outro para regras de apresentação. Esses módulos aumentam tanto o reuso quanto a expressividade de NCL. Foi também refinada a linguagem XConnector, com a especificação de XConnector 2.1; e estendida a linguagem XTemplate (XTemplate 2.1), permitindo que *templates* definam relacionamentos não somente por meio de conectores, mas também de inclusão. Além disso, XTemplate 2.1 foi especificada de forma modular, permitindo que outras linguagens definam o perfil de XTemplate que utilizam. XTemplate 2.1 se torna, assim, uma das principais contribuições deste trabalho.

A partir de NCL 2.1, foi possível a especificação de X-SMIL e de seus perfis: XT-SMIL e XC-SMIL. O perfil XT-SMIL, apesar de não aumentar a expressividade de SMIL, adiciona novas funcionalidades para o reuso, flexibilizando os tipos de composição oferecidos pela linguagem. Assim, o autor não fica obrigado a definir uma hierarquia de composições dos tipos básicos (*par*, *seq* e *excl*) e um conjunto de eventos para representar relacionamentos complexos entre componentes. A definição de XT-SMIL utiliza um perfil de XTemplate que contempla relacionamento de inclusão, que somente pôde ser especificado devido à abordagem modular e às novas funcionalidades de XTemplate 2.1. O perfil XC-SMIL aumenta o reuso e expressividade de SMIL ao introduzir o conceito de conectores hipermídia nessa linguagem. Finalmente, X-SMIL combina as facilidades dos perfis XT-SMIL e XC-SMIL, permitindo, por exemplo, a definição de *templates* com relações de inclusão e por meio de conectores. Por aumentar tanto o reuso quanto a expressividade de SMIL, X-SMIL está também entre as principais contribuições desta dissertação.

Para facilitar o processamento de documentos especificados em SMIL, NCL e X-SMIL, foi definido um *framework* genérico (*meta-framework*) para o processamento de documentos XML. Baseados nesse *framework*, diversos *frameworks* de compiladores e suas instâncias (compiladores) foram implementados, permitindo, por exemplo, a conversão entre documentos NCL, SMIL e X-SMIL; e a edição e exibição de documentos especificados nessas linguagens no sistema HyperProp. A especificação do *framework* genérico de processamento e de suas instâncias são outra importante contribuição desta dissertação.

Cabe destacar, ainda, o processador de *templates*, um subproduto deste trabalho que realiza o processamento de todos os *templates* referenciados em um documento XML e, ao contrário do processador de XTemplate 2.0 (Muchaluat-Saade, 2003), oferece suporte a todas as facilidades de XSLT. O processador de *templates* implementado foi utilizado, por exemplo, na conversão de documentos XT-SMIL em documentos SMIL, demonstrando a compatibilidade entre essas linguagens.

7.1. Templates

Com o padrão MPEG-4 (Koenen, 2002), novas linguagens declarativas baseadas em XML para especificação de documentos hipermídia (ou cenas) foram definidas: XMT-A e XMT-O (ISO, 2001). Um trabalho concluído (Costa, 2005) é a definição de XT-XMT-O, uma extensão de XMT-O que introduz o conceito de *templates* nessa linguagem.

Como o conceito de *templates* de composição não é restrito à hipermídia, perfis de XTemplate podem ser definidos para outras linguagens de autoria, como linguagens de *workflow* (Thatte, 2003). Em (Silva, 2003), é apresentada uma comparação entre conceitos de hipermídia e de *workflow*, apontando como conectores hipermídia podem ser utilizados na especificação de padrões de *workflow* (Aalst, 2003) e como *templates* podem ser definidos e reutilizados no processo de autoria de documentos nessas linguagens.

Um trabalho em andamento constitui a definição de *templates* que utilizam outros *templates*. Ou seja, composições definidas como recursos em *templates*

poderão referenciar outros *templates*, reusando-os e facilitando o processo de autoria. O processador de *templates* está sendo modificado para que, após o processamento de cada *template*, seja verificado se algum componente contido na composição processada referencia outro *template*. Se houver, deve-se processar novamente esses componentes, e assim sucessivamente. Para garantir que esse algoritmo seja correto, *templates* não podem, direta ou indiretamente, se auto referenciar.

A linguagem XTemplate também está sendo refinada para permitir a definição de parâmetros. Elementos do tipo *resourceParam*, que contém os atributos *name* e *value*, poderão ser declarados como filhos de elementos *resource*, determinando que um atributo com o nome *name* e o valor *value* seja adicionado a um recurso. Os atributos *name* e *value*, quando iniciados com o caractere "$", referenciam parâmetros do *template*. Por exemplo, para adicionar, a um recurso, um atributo com o nome *region* e com valor definido pela variável "*r1*" do *template*, deve-se especificar: *<resourceParam name="region" value="$r1">*. Parâmetros também poderão ser referenciados por elementos *xsl:templateParam*, utilizados de modo semelhante ao elemento *xsl:value-of* em transformadas XSLT. Durante o processamento de *templates*, elementos *xsl:templateParam* serão substituídos pelo valor do parâmetro do *template* indicado pelo atributo *value*. Por exemplo, para definição do atributo *systemLanguage*[53] de forma parametrizada (pela variável *langX*), pode-se especificar: *<xsl:attribute name = "systemLanguage"> <xsl:templateParam value="langX" /> </xsl:attribute>*[54]. Como *templates* são referenciados, em documentos XML, pelo valor do atributo *xtemplate*, será utilizado o padrão de *query strings* (W3C, 2001c) para passagem de parâmetros. Por exemplo, para referenciar o *template* "*áudioComLegendasXY*" (similar ao *audioComLegendasEnPt* do Capítulo 4, mas com a escolha entre as alternativas de legendas sendo baseada nas línguas parametrizadas pelas variáveis *langX* e *langY*), deve-se declarar: *xtemplate = "audioComLegendasXY.xml ? langX=en &*

[53] Ver exemplo da Figura 4:3 do Capítulo 4.

[54] Um valor padrão para os atributos *name* (de *resourceParam*) e *value* (de *resourceParam* e de *xsl:templateParam*) podem ser especificados, respectivamente, pelos atributos *defaultName* e *defaultValue*.

langY=pt" (que define o valor *en* para a variável *langX* e *pt* para a variável *langY* do *template*).

Em adição a esses refinamentos a XTemplate, um trabalho futuro é a definição de uma linguagem de domínio especifico para a autoria de *templates*. O uso de XSLT em XTemplate traz grande expressividade à linguagem e permite utilizar processadores XSLT padronizados para o processamento de *templates*. Entretanto, o uso de XSLT adiciona uma grande complexidade na especificação de *templates*, sendo importante estudar funcionalidades que simplifiquem sua autoria. Essas novas funcionalidades podem seguir a abordagem adotada neste trabalho, que estende XSLT, ou pode-se definir uma outra linguagem. Em ambos os casos, é desejável um mapeamento para folhas de estilo XSL (W3C, 1999d).

Um outro trabalho futuro consiste em explorar o conceito de *templates* e adotá-lo tanto no ambiente de autoria quanto no formatador do sistema HyperProp. Uma abordagem para introduzir *templates* no ambiente de autoria gráfica é a definição de uma *interface* Java (por exemplo, "*Template.java*") declarando um método "*process*", que recebe como parâmetro uma composição hipermídia. Classes que implementem essa interface (ou seja, que definam *templates*) devem implementar esse método para processar o *objeto composição* recebido como parâmetro para atribuir a semântica do *template* (incluindo elos na composição, por exemplo). Na interface gráfica, deve-se permitir ao usuário escolher um *template* (uma classe que especifique um *template*) para processar composições sendo editadas. Adicionalmente, a interface "*Template.java*" pode definir os métodos "*checkConstraints*" e "*drawTimeView*", responsáveis, respectivamente, por verificar restrições em uma composição e definir como ela deve ser representada na visão temporal do ambiente de autoria (Coelho, 2004). Esse último método pode ser útil para adicionar o conceito de composições na visão temporal do sistema.

O mecanismo de *plug-ins* do formatador HyperProp (Rodrigues, 2003) pode ser explorado para introduzir o conceito de *templates* no formatador. Dessa forma, cada *template* deve ser implementado como um *plug-in*, sendo responsável pelo controle da exibição de uma composição que referencia esse *template*. Por exemplo, pode-se definir *plug-ins* para composições com semântica temporal paralela e seqüencial. O uso desse conceito pode facilitar a implementação de um formatador modular e distribuído.

Se o processo de incorporação de *templates* nos componentes do sistema HyperProp for bem sucedido, deve-se estabelecer mecanismos para o mapeamento, se possível automático, entre *templates* especificados em NCL, no ambiente de autoria, e no formatador. Paralelamente a esses estudos, a edição gráfica de *templates*, como apresentada na Seção 6.1, deve ser analisada para *templates* NCL.

7.2. Conversões entre Formatos

Em (Joung & Kim, 2002) é definida uma API para a conversão entre XMT-O e XMT-A, além da compilação de XMT-A em BIFS. Existem também ferramentas proprietárias para a compilação de XMT em BIFS ou em arquivos MPEG-4 (.mp4) (IBM, 2004) (GPAC, 2004). Porém, essas implementações não foram desenvolvidas visando a reutilização de código por parte de outros compiladores. Ao contrário, o trabalho descrito em (Costa, 2005) consiste em utilizar o *meta-framework* para definir *frameworks* de compiladores XMT-A e XMT-O; implementar instâncias desses *frameworks* (compiladores) para conversão de especificações XMT-O / XMT-A em BIFS e em NCL; e a construção de compiladores NCL-(XMT-A) e NCL-(XMT-O) - usando o *framework* de compiladores NCL. Como XMT-O é baseada no padrão SMIL, e como existe um mapeamento entre XMT-O e XMT-A, esses compiladores procuram aproveitar os esforços dedicados à construção dos compiladores NCL-SMIL e SMIL-NCL.

Outro trabalho em andamento utiliza a estrutura de dados (voltada para apresentação) definida pelo formatador HyperProp para geração de um documento SMIL com um menor número de composições *par*. Como visto na Seção 6.6, essa abordagem simplifica a exibição de documentos pelos *players* SMIL. No estágio atual, foi implementado (como parte dos trabalhos desenvolvidos nesta dissertação) um compilador NCL-SMIL modificado, que utiliza o compilador NCL-Formatador para determinar objetos com tempo de apresentação disjunto, visando estruturar o documento SMIL. Deve-se estudar outros algoritmos para essa estruturação, testando diferentes abordagens.

A geração de código pelo gerador automático de *framework* de compiladores ainda pode ser melhor explorada. Por exemplo, ao gerar esqueletos de códigos para compiladores específicos, a ferramenta gera novas classes independente se houve alteração do código previamente gerado. Isso dificulta a atualização de compiladores quando existe mudança na linguagem de origem. Um mecanismo de análise de código, para atualizar somente o que foi previamente gerado, deve ser estudado.

Um trabalho futuro interessante é utilizar o *meta-framework* para linguagens de outros domínios, que não o de hipermídia. Por exemplo, uma aplicação para teste foi implementada como parte dos trabalhos desenvolvidos nesta dissertação: um compilador para documentos em XML Schema foi criado utilizando o *gerador automático de frameworks* (usando um arquivo em XML Schema que define a própria linguagem XML Schema). A partir desse compilador, foi implementada uma outra versão para o próprio *gerador automático*. A nova versão do *gerador automático*, gerada a partir do *gerador automático* anterior, foi, então, testada satisfatoriamente para geração de *frameworks* de compiladores. Essa aplicação de teste demonstra como linguagens baseadas em XML, de diferentes domínios, podem aproveitar as facilidades apresentadas neste trabalho.

É importante destacar, também, que o uso do *gerador automático* tem uma outra vantagem: o auxilio na validação tanto de compiladores quanto de *schemas.* Por exemplo, foi possível realizar uma depuração nos arquivos XML Schema de NCL e X-SMIL a partir de *erros* (como ausência de métodos) encontrados nos compiladores gerados automaticamente. Além disso, durante a implementação dos compiladores gerados pelo *gerador automático*, erros de programação puderam ser corrigidos nos compiladores baseados em implementações anteriores.

Ainda um outro trabalho futuro é analisar o uso da abordagem SAX (Seção 6.2) no *framework* de compiladores. Alguns compiladores podem ser implementados sem o uso de árvores DOM, como, aparentemente, é o caso dos compiladores que geram cópias dos arquivos XML de entrada. A utilização da abordagem SAX pode aumentar a eficiência desses compiladores ao eliminar o passo intermediário de construção da árvore DOM. Essa abordagem também pode permitir que um documento seja compilado, sob demanda, enquanto é recebido em um dispositivo de exibição (via *streaming*, por exemplo).

Outra opção para melhorar o desempenho dos compiladores desenvolvidos nesta dissertação é utilizar implementações alternativas para o padrão DOM, como JDOM (JDOM, 2004) e DOM4J (DOM4J, 2004). Essas implementações, apesar de apresentarem algumas diferenças em relação ao padrão, são mais eficientes que as implementações padronizadas, segundo seus desenvolvedores, e permitem uma fácil integração com outras APIs baseadas em DOM. Implementações mais eficientes são possíveis ao se utilizar características particulares de Java, já que o padrão DOM foi definido de forma independente de linguagem de programação (JDOM, 2004).