UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Uma infra-estrutura para controle de versões e adaptação de páginas Web

por

RODRIGO GIACOMINI MORO

Dissertação submetida a avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser
Orientador

Porto Alegre, março de 2003.

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Rodrigo Giacomini Moro,

Uma infra-estrutura para controle de versões e adaptação de páginas Web / por Rodrigo Giacomini Moro. — Porto Alegre: PPGC da UFRGS, 2003.

78 f.: il.

Dissertação (mestrado) — Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2003. Orientador: Heuser, Carlos Alberto.

1. Versões. 2. Configurações. 3. Modelos de Hiperdocumentos. 4. Adaptação. 5. Web. I. Heuser, Carlos Alberto. II. Título.

Agradecimentos

Agradeço ao professor Carlos Alberto Heuser pela orientação durante o curso. Agradeço por todas as "dicas" e experiência que generosamente me transmitiu, o que contribuiu para a qualidade do trabalho e tornou seu desenvolvimento muito mais objetivo. Também agradeço pela minha primeira viagem aérea e pela primeira viagem internacional!

Agradeço aos alunos do grupo de pesquisa em banco de dados, pelo clima de coleguismo que imperou no nosso cotidiano durante os últimos dois anos. Um abraço a todos!

Agradeço ao Instituto de Informática da UFRGS por fornecer a infra-estrutura necessária para a realização do trabalho e aos funcionários que mantém essa infra-estrutura funcionando. Também agradeço aos professores do PPGC, cujos ensinamentos foram muito importantes no desenvolvimento do trabalho.

Agradeço aos meus pais, Tarcísio e Teresinha, pelos valores que me ensinaram, que me permitiram atingir este objetivo. Agradeço pelo amor, incentivo e preocupação que sempre me dedicaram. Agradeço à minha irmã, Renata, por seu companheirismo.

Agradeço ao Pai Celestial pela inspiração que me proporcionou, o que me fez enxergar uma "luz no fim do túnel" quando as coisas ficavam complicadas...

Por fim, agradeço ao financiamento recebido durante a realização do trabalho. Este trabalho teve financiamento parcial da CAPES, bem como de recursos da contra-partida à fruição dos benefícios fiscais de que trata o artigo 1º do Decreto 3.800 de 20/04/2001.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

CAD	Computer Aided Design
COV	Change Oriented Versioning
DTD	Document Type Definition
GDOC	Gestão de Documentos
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
GIF	Graphics Interchange Format
ICE	Incremental Configuration Environment
NCM	Nested Context Model
UBCC	Usefulness-Based Copy Control
UML	Unified Modeling Language
URI	Uniform Resource Identifier
SQL	Structured Query Language
SCM	Software Configuration Management
SGBD	Sistema Gerenciador de Bancos de Dados
SMIL	Synchronized Multimedia Integration Language
UDB	User Database
UVM	Uniform Version Model
WebDAV	Web Distributed Authoring and Versioning
WAP	Wireless Application Protocol
WBMP	Wireless Bitmap
WML	Wireless Markup Language
XML	Extensible Markup Language
XSLT	XML Stylesheet Language Transformation

Resumo

Conforme os sites Web crescem em número de páginas, sua manutenção torna-se mais complicada. Assim, os administradores dos sites necessitam de métodos e ferramentas que tornem sua manutenção mais organizada e automatizada. Entretanto, a criação de tais mecanismos é dificultada pelo formato das páginas Web (HTML), que mistura o conteúdo e a formatação da página em um mesmo arquivo. Uma solução usual para esse problema é separar estes componentes da página em documentos XML (conteúdo) e folhas de estilo XSLT (formatação).

Pode-se notar várias semelhanças entre páginas Web e programas de computador (software), pois ambos têm componentes de tipos diferentes que evoluem com o tempo. Assim, técnicas oriundas da área de Gerência de Configuração de Software, como o controle de versões, podem ser adaptadas para auxiliar a manutenção de sites. Além da melhoria na manutenção, outra necessidade cada vez mais comum aos sites é a adaptação automática das páginas. Por meio desta, páginas podem ser automaticamente adequadas (adaptadas) a determinado usuário, o que potencialmente atrai um maior número de visitantes ao site.

Se forem mantidas versões de cada componente de página, pode-se combiná-las para gerar muitas páginas alternativas. Através da escolha cuidadosa das versões dos componentes que compõem uma página, é possível obter páginas adaptadas automaticamente. Na área de Gerência de Configuração de Software, o chamado processo de configuração é responsável por selecionar automaticamente versões de módulos para compor um programa completo.

O presente trabalho propõe uma infra-estrutura para um servidor Web que realiza controle de versões e suporta a adaptação de páginas Web de forma transparente ao visitante. Para tanto, é projetado um modelo de versões de páginas que separa conteúdo e formatação em componentes distintos. É proposto um processo de configuração que é responsável pela geração de páginas dinâmicas, no que é suportado por informações presentes no modelo de versões.

Os autores de páginas e o próprio servidor Web podem interferir nas escolhas do processo de configuração, fornecendo critérios de seleção de versões. Esses critérios guiam as escolhas do processo de configuração, pois representam características que as versões escolhidas devem (necessariamente ou preferencialmente) apresentar.

Palavras-chave: Versões, Configurações, Modelos de Hiperdocumentos, Adaptação, Web.

TITLE: "A INFRASTRUCTURE FOR VERSION CONTROL AND ADAPTATION OF WEB PAGES"

Abstract

As the number of pages of Web sites grows, maintenance becomes harder. Therefore, Web site managers need methods and tools to help them in organizing and automating site maintenance. However, creating such mechanisms is difficult due to the fact that HTML merges page content and formatting in the same file. This problem is commonly solved by separating page components in XML documents (content) and XSLT stylesheets (formatting).

Web pages and computer programs (software) are similar in that both have different types of components that evolve overtime. Thus, Software Configuration Management techniques, e.g. version control, can be adapted to help site maintenance.

In addition, Web sites need automatic page adaptation. Pages can be automatically adapted to a particular user, what potentially attracts more visitors to the site.

When versions of page components are maintained, it's possible to combine them to generate many alternative pages. By carefully choosing versions to compose a page, it's possible to get automatically adapted pages. In Software Configuration Management area, the configuration process is responsible for automatically selecting module versions to compose a complete program.

This work proposes a Web server infrastructure to accomplish version control and to support adaptation of Web pages in a way that is transparent to the site visitor. To achieve this goal, a page version model that separates page content and formatting in distinct components is described. A configuration process is also proposed and is responsible for generating dynamic pages.

Page authors and the Web server itself may intervene in configuration process choices, by providing criteria for version selection. These criteria guide choices of configuration process, because they represent preferred features for selected versions.

Keywords: Versions, Configurations, Hyperdocument Models, Adaptation, Web.

1 Introdução

Atualmente, existem muitos sites similares na Web e seu crescimento em número acirrou a competição entre eles por audiência; para ser atrativo, cada site investe em serviços novos e úteis, mantém o conteúdo constantemente atualizado e cria uma apresentação agradável. Fazendo isso, sua complexidade em número de páginas aumenta muito, dificultando sua manutenção manual.

Portanto, os profissionais responsáveis pela manutenção do site necessitam de ferramentas que facilitem e automatizem a manutenção do conteúdo. No entanto, os sites são normalmente construídos usando HTML (*"Hypertext Markup Language"*) [RAG 99], o que dificulta a criação de tais ferramentas, pois o conteúdo e a formatação das páginas são misturados em um mesmo arquivo.

A separação entre os componentes de uma página (conteúdo, formatação, imagens, etc.) é importante para a gerência de conteúdo Web [MOR 2001, MOR 2002], pois simplifica a manutenção de páginas tanto por pessoas quanto por ferramentas. Esta separação pode ser (e freqüentemente é [KAY 2000]) implementada através do armazenamento do conteúdo em documentos XML (*"Extensible Markup Language"*) [BRA 2000, BRA 2002] e da formatação em folhas de estilo XSLT (*"XML Stylesheet Language Transformation"*) [CLA 99, KAY 2000]. A tradução de documentos XML para o formato HTML, usando XSLT, é uma forma comum de apresentar documentos XML em um navegador Web [KAY 2000].

Hiperdocumentos (páginas Web) são similares a software em muitos aspectos [BIE 98]: ambos têm componentes, seus componentes podem ser de vários tipos, são mantidos por equipes de muitos desenvolvedores, evoluem com o passar do tempo, etc. Como forma de melhorar o desenvolvimento de software, sistemas de Gerência de Configuração de Software [TIC 85, BER 88, MUN 93, EST 95, MAG 96, ZEL 95, ZEL 97] armazenam todos os estados dos componentes dos programas (módulos, arquivos de cabeçalho, etc.), desde sua criação até atingirem seu estado final. Eles também mantém versões de programas inteiros (chamados configurações), de forma a guardar as configurações entregues aos clientes.

São várias as razões para gerenciar múltiplas versões [WES 96]: reuso de versões, manutenção de versões já entregues a clientes, armazenamento de versões de *"backup"*, suporte a gerência de alterações, etc. Controlar versões de hiperdocumentos também é importante, conforme verificado por Halasz [HAL 88]. Para atender a essa necessidade, vários modelos de versões de (hiper)documentos foram propostos [DEL 87, HAA 92, ØST 92, SOA 95, HAA 96, HIC 98, BIE 98, NOR 98].

No entanto, uma deficiência geral destes modelos é que a separação entre conteúdo e formatação não é atendida satisfatoriamente. Poucos destes modelos realizam esta separação e, entre os modelos que o fazem, nenhum controla versões de conteúdo e formatação independentemente. Recentemente surgiram modelos de versões específicos para documentos XML [CHI 2001a, CHI 2001, CHI 2002]; entretanto, a ênfase destes modelos está no suporte ao armazenamento e consultas complexas.

Além da melhoria na manutenção dos sites, outro requisito cada vez mais comum é a adaptação das páginas dependendo de características particulares dos usuários. Desta forma o servidor Web pode prover conteúdo e apresentação específicas para uma classe de usuários. A geração de páginas adaptadas em servidores

Web comuns é normalmente obtida através da execução de algum tipo de programa ou *"script"* criado especificamente para este fim. Desta forma, o mecanismo de adaptação está embutido em (ou distribuído por) cada página.

Alguns servidores de conteúdo Web também usam esta abordagem (como o SIM [ARN 2000], que disponibiliza sua linguagem ACE para a criação de aplicações Web). Apesar de esta ser uma abordagem bastante flexível, ela é também propensa a erros porque é complexa, bastante manual e, principalmente, pela necessidade de implementar o mecanismo de adaptação em cada página.

Alguns sites desenvolvem sistemas dedicados e específicos para realizar a adaptação de páginas; por exemplo, o Yahoo! [MAN 2000] (MyYahoo!, em especial) desenvolveu inclusive um SGBD (Sistema Gerenciador de Bancos de Dados) para manter os dados dos perfis de usuário, chamado UDB (*"User Database"*). Como o mecanismo de adaptação está centralizado, alterações nele são implementadas em apenas um local, ao invés de em um conjunto de páginas dinâmicas (*"scripts"*).

Entretanto, como esse mecanismo de adaptação é específico, dificilmente ele poderá ser reusado. Além disso, como ele é projetado para adaptar determinadas características das páginas, alterações mais amplas podem requerer uma reestruturação do sistema. Portanto, é necessário um mecanismo geral de adaptação de páginas.

Se forem mantidas versões de cada componente de página, é possível combiná-las para gerar muitas páginas alternativas. Escolhendo-se estas versões cuidadosamente, pode-se gerar páginas adaptadas. Muitos sistemas de Gerência de Configuração de Software têm o chamado processo de configuração ou configurador, que escolhe automaticamente uma versão específica de cada componente de software e gera um programa completo (configuração). Se existir um gerenciamento adequado das versões dos componentes das páginas, um processo de configuração similar pode ser implementado para configurar páginas. Assim, este processo pode servir de base para a construção de um mecanismo específico de adaptação de páginas.

Como é mostrado neste trabalho, um processo de configuração pode ser usado como um mecanismo geral de adaptação de páginas Web, oferecendo ao site uma vantagem importante sobre seus rivais na atração de visitantes. Páginas que são adequadas a alguns períodos do ano (como promoções e feriados) podem ser publicadas automaticamente. Além da data, uma adaptação pode considerar características do usuário ou dispositivo que requisitou a página, usando esta informação para selecionar a melhor (mais adequada) versão de cada componente da página.

A adaptação pode ainda ajudar a melhorar o desempenho do servidor Web. Ela pode considerar a carga de trabalho atual do servidor Web e fornecer ao usuário uma página mais leve [ADB 99], caso uma condição de sobrecarga seja detectada. Para gerar páginas adaptadas, o processo de configuração pode ter ajuda do autor da página e/ou do servidor Web. Eles fornecem critérios para seleção de versões ao processo, que guiam-no na escolha da versões que compõem a página.

O objetivo deste trabalho é propor uma infra-estrutura para controle de versões e suporte a adaptação de páginas Web. Para facilitar a manutenção do site, a modelagem das páginas deve separar o conteúdo e a formatação das páginas e controlar as versões destes dois tipos de componentes independentemente. A adaptação de uma página não deve depender de uma programação específica, mas sim de um mecanismo geral de seleção de versões de seus componentes.

Os autores de páginas devem poder intervir na seleção de versões, fornecendo

critérios de seleção de versões que guiarão as escolhas do processo de configuração. Deve ser possível que o servidor Web também intervenha na seleção de versões, caso tenha essa capacidade. Isso facilita o uso do processo de configuração como base para a adaptação de páginas. Deve ser possível ainda implementar esta abordagem usando padrões da Web, como XML e relacionados.

O presente trabalho obedece à seguinte organização. O capítulo 2 introduz conceitos gerais sobre o controle de versões e gerência de configurações, que serão usados no decorrer do trabalho; alguns trabalhos relacionados ao controle de versões e à gerência de configurações também são analisados neste capítulo. O modelo de versões das páginas proposto e seu respectivo processo de configuração são descritos no capítulo 3. O capítulo 4 conclui o trabalho e apresenta sugestões de trabalhos futuros. Um protótipo de servidor Web foi desenvolvido para testar a infra-estrutura proposta. O anexo A apresenta a arquitetura deste protótipo.

2 Controle de versões e gerência de configurações

O presente trabalho utiliza controle de versões e gerência de configurações para suportar a adaptação de páginas Web. Este capítulo analisa trabalhos relacionados a estes assuntos. Primeiramente, conceitos gerais relacionados a estes temas são introduzidos (seção 2.1). Em seguida, modelos de versões de hiperdocumentos e documentos estruturados são analisados com relação a suas características gerais (seção 2.2).

Uma vez que existem versões dos componentes de documentos, são necessárias maneiras de selecionar versões adequadas de cada um deles, a fim de formar documentos completamente definidos. A seleção de versões foi bastante estudada na área de Gerência de Configuração de Software (*"Software Configuration Management"*, de agora em diante chamada SCM), por isso alguns desses trabalhos também são analisados (seção 2.3).

Em seguida, é explicado um protocolo de rede que padroniza o controle de versões na Web (seção 2.4). O foco desta explicação é o modelo de dados deste protocolo, que serviu de base para o modelo de versões de páginas proposto. Ao final do capítulo, são destacadas algumas deficiências dos trabalhos analisados (seção 2.5).

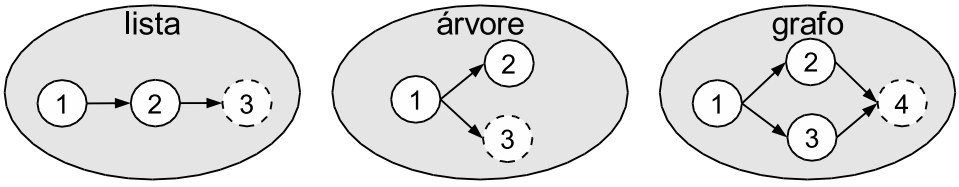
2.1 Conceitos gerais sobre versões e configurações

Esta seção introduz os conceitos gerais relacionados ao controle de versões e à gerência de configurações. Os conceitos são apresentados de forma a tentar aproximar conceitos semelhantes provenientes das áreas de CAD (*"Computer Aided Design"*) [KAT 90], hipertexto [ØST 92, HAA 92, HAA 94] e SCM [EST 95, CON 96, CON 98].

2.1.1 Objeto versionado e versões

Sistemas para a manutenção de objetos (arquivos, artefatos de software, etc.) que evoluem com o tempo costumam armazenar todos os estados (*"snapshots"*) importantes destes objetos. As razões para registrar os estados dos objetos podem ser: reuso de estados, registro de estados entregues a clientes, *"backup"*, análise da evolução do produto (objeto), auditoria, entre outras. O objeto para o qual o sistema mantém um histórico de estados é chamado de *objeto versionado* (ou algo semelhante, dependendo do domínio de aplicação).

Cada estado do objeto versionado que é registrado pelo sistema é chamado de versão. *Versão* é a descrição de um objeto em determinado instante de tempo, cujo registro é importante para a aplicação [GOL 95]. Uma versão tem propriedades que expressam características de seu conteúdo ou informações sobre o controle de versões. Entre estas propriedades está ao menos um nome ou número que diferencia uma versão das outras do mesmo objeto versionado. Uma das versões representa o estado atual do objeto versionado e é chamada de *versão corrente*. Freqüentemente (mas não obrigatoriamente) a versão corrente é a versão criada mais recentemente.



(a) Estrutura em lista (b) Estrutura em árvore (c) Estrutura em grafo

FIGURA 2.1 – Tipos de organização lógica das versões

2.1.2 Estrutura do histórico de versões

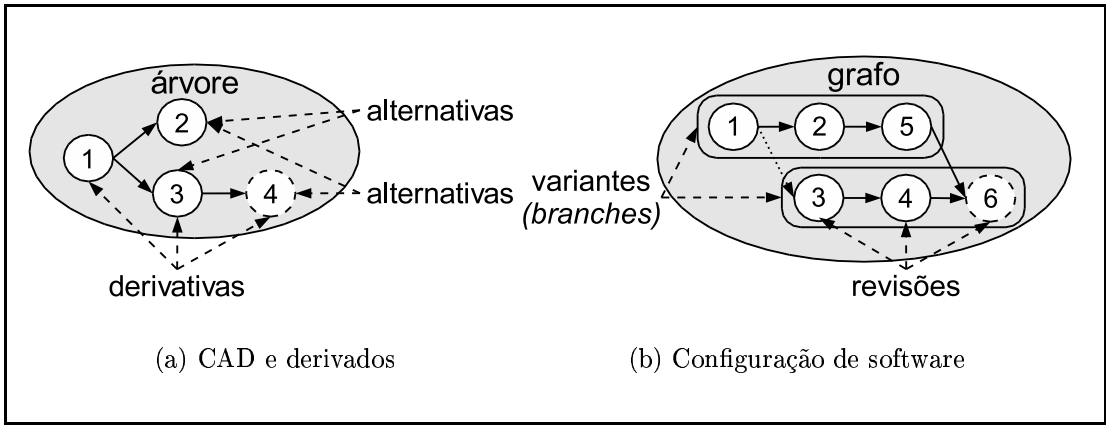
Conforme o objeto versionado é alterado, novas versões são criadas para armazenar as alterações, formando um *histórico de versões*. A estrutura mais simples do histórico de versões é aquela em forma de lista, como mostrado na figura 2.1(a). Nesta figura (e nas seguintes), o objeto versionado é mostrado como uma elipse preenchida com seu nome ao topo; as versões são mostradas como círculos brancos identificadas com seus respectivos números; a versão com contorno tracejado é a versão corrente; as setas entre duas versões indicam que a versão apontada foi criada a partir da outra. Por exemplo, a versão 3 foi criada em resposta a uma modificação realizada no objeto versionado quando a versão corrente era a 2.

Quando uma versão é criada a partir de uma versão base (V_B), diz-se que ela é uma *versão derivada* de V_B. Assim, as setas indicam uma relação de derivação entre as versões e apontam para as versões derivadas. A versão derivada (V_D) é dita *sucessora* de V_B, enquanto que V_B é dita *predecessora* de V_D. Por exemplo, tomando-se a versão 2 como referência, sua predecessora é a versão 1 e sua sucessora é a versão 3.

A organização do histórico das versões nem sempre é restrita a uma lista de versões. Quando uma versão tiver mais de uma sucessora, bifurcações são geradas no histórico, como mostra a figura 2.1(b). Esta figura mostra uma estrutura em forma de árvore que pode ser obtida da seguinte forma. Um objeto versionado, representando um módulo de um programa, é criado com sua versão 1. Uma atualização no módulo é feita para otimizar o desempenho, criando a versão 2. Neste ponto, um problema introduzido pela otimização é descoberto na versão 2 e a versão corrente volta a ser a 1. Uma nova modificação é feita no módulo para melhorar a segurança, criando a versão 3 a partir da 1.

Uma nova versão poderia ser criada de forma a implementar tanto a otimização quanto a melhoria de segurança. Para isso ela deveria ser derivada tanto da versão 2 quanto da 3. Assim, ela ficaria com duas predecessoras, e causando a fusão (*"merge"*) de duas versões em uma única. Isto gera um histórico de versões com estrutura de derivação em forma de um grafo acíclico dirigido, como mostra a figura 2.1(c).

Em qualquer das três estruturas de derivação mostradas, a versão que não tem predecessoras é normalmente referida como a *raiz* (no caso, a versão 1). Similarmente, as versões que não têm sucessoras são chamadas de *folhas*. Por exemplo, na figura 2.1(a), a versão 3 é folha; na 2.1(b), as versões 2 e 3 são folhas e na 2.1(c) a versão 4 é folha.



(a) CAD e derivados (b) Configuração de software

FIGURA 2.2 – Organização de versões paralelas (alternativas)

Alguns sistemas suportam a derivação de versões diretamente a partir de qualquer versão, não só da corrente (através do objeto versionado). Os modelos de versões mais recentes utilizam uma estrutura de derivação de versões em forma de grafo ou árvore, e alguns deles serão comentados na seção 2.2. Um exemplo de sistema que utiliza derivação em lista é o sistema hipertexto Neptune [DEL 87].

2.1.3 Versões alternativas

Em sistemas de CAD, duas versões podem ser classificadas como derivativas ou alternativas [KAT 90]. *Versões derivativas* são aquelas que estão no mesmo caminho de derivação até a raiz. *Versões alternativas* são as que não estão no mesmo caminho. Isso é mostrado na figura 2.2(a).

Alguns sistemas de SCM usam estruturas de derivação similares às da figura 2.1, chamadas de organização de um nível (*"one-level"*) [CON 98]. Apesar disso, existem outros tipos de organizações [CON 98] como a de dois níveis (*"two-level"*) e de matriz.

A figura 2.2(b) mostra uma organização de dois níveis que é bastante comum. Neste tipo de organização existem dois tipos de derivação de versões: *"offspring"* (indicada por uma seta pontilhada), que gera variantes ou galhos (*"branches"*) e *"successor"* que gera revisões. *Variantes* são seqüências de versões que devem coexistir com outras variantes; por exemplo, duas seqüências de versões de um programa, onde cada uma implementa-o para um sistema operacional diferente. Variantes evoluem através de uma seqüência de *revisões*, que são versões que devem substituir sua predecessora devido a alguma correção implementada.

Note-se, na figura 2.2, a semelhança entre revisões de variantes diferentes e versões alternativas, bem como entre revisões da mesma variante e versões derivativas. A diferença é que versões alternativas e derivativas são classificações relativas, que consideram duas ou mais versões, enquanto que variantes e revisões são classificações estruturais (por isso o nome de organização de dois níveis).

2.1.4 Objetos compostos

Uma necessidade na modelagem de sistemas complexos é a possibilidade de expressar composição de objetos. Por exemplo, hiperdocumentos são compostos de fragmentos de texto, figuras, etc., enquanto que sistemas de software (programas) são

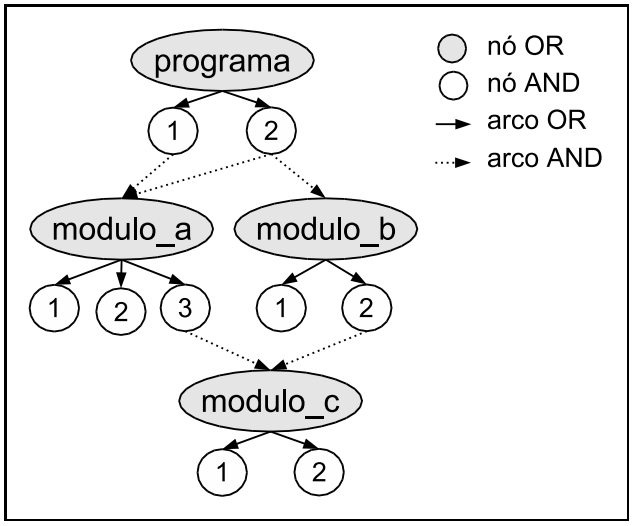


FIGURA 2.3 – Modelagem de objetos compostos

compostos de módulos e bibliotecas de funções. Portanto, um *objeto composto* (ou complexo) é aquele formado pela agregação de outros objetos.

Em sistemas de SCM, um modelo geral para os objetos versionados compostos é o de grafo AND/OR, que inclusive já foi usado para modelar hipertexto [BIE 98]. Esses grafos têm os dois tipos de nó que os nomeiam, onde os *nós OR* correspondem a objetos versionados e os *nós AND* correspondem a versões. Existem também arcos direcionados AND e OR que originam-se em nós de mesmo nome. *Arcos AND* representam relações de dependência (ou composição), enquanto que os *arcos OR* ligam um objeto versionado a suas versões.

Um exemplo de um programa modelado como um grafo AND/OR é mostrado na figura 2.3. O programa (módulo principal) tem duas versões; a versão 1 é composta pelo módulo "a" e a versão 2 é composta dos módulos "a" e "b". Por sua vez, tanto a versão 3 do módulo "a" quanto a versão 2 do módulo "b" são compostas pelo módulo "c".

Existem diferentes tipos de grafos AND/OR [CON 98], inclusive alguns onde arcos AND apontam para nós AND. No entanto, o tipo mostrado na figura 2.3 é mais flexível, pois versões específicas de componentes não são referenciadas diretamente. Assim, pode-se usar sempre a versão corrente de um componente, mesmo que ela seja criada após a versão corrente do objeto composto. Os modelos de versões para hipertexto seguem a estrutura geral deste tipo de grafo, apesar de serem mais ricos em número de conceitos envolvidos.

2.1.5 Referências entre objetos

Em sistemas de CAD, as ligações (*"binding"*, também chamadas referências [GOL 95]) entre uma versão do objeto composto e seus componentes podem ser estáticas ou dinâmicas [KAT 90]. As *referências estáticas* são aquelas que apontam para uma versão específica do componente. Por outro lado, *referências dinâmicas* apontam para o próprio objeto versionado componente.

Quando uma versão do objeto composto é usada e contém referências dinâmicas, estas devem ser resolvidas a fim de obter-se um objeto completamente definido (mais detalhes são fornecidos na subseção 2.1.6). A *resolução de referência*

dinâmica consiste em selecionar uma versão específica do objeto versionado referenciado. Para isso, pode-se solicitar uma escolha manual do usuário, usar algum critério *"default"* (como selecionar a versão corrente ou a mais recente), ou usar critérios definidos pelo usuário.

A vantagem da referência estática é que as versões do objeto composto e do componente são garantidamente consistentes (pois o próprio usuário escolheu as versões dos componentes). Além disso, não é necessário um processamento adicional para resolvê-la posteriormente. Por sua vez, o principal benefício da referência dinâmica é que o objeto composto pode usar sempre a versão mais adequada do componente, como por exemplo a mais recente. Isso é possível porque a referência não está amarrada a uma versão específica, e será resolvida sob demanda (no momento em que o objeto for utilizado).

O processo de resolução de referências dinâmicas também existe em sistemas de SCM, mas é chamado de processo de configuração, como explicado a seguir.

2.1.6 Configurações

Nos casos em que os conceitos de histórico de versões e hierarquias de componentes (objetos compostos) são combinados, o resultado é uma *configuração* [KAT 90]: uma versão do objeto composto que é composta de versões específicas de seus objetos componentes. Uma configuração também pode ser definida como [CON 96] uma versão consistente e completa de um objeto composto, ou seja um conjunto de versões de componentes e suas relações.

Uma configuração é uma versão de um objeto composto que não contém, direta ou mesmo indiretamente, referências dinâmicas para seus componentes; ela pode ser de três tipos [CON 96] (apesar de os dois últimos violarem as definições):

- resolvidas (*"bound"*) ou *"baselines"*: contém exclusivamente versões como componentes;
- não resolvidas (*"unbound"*): contém exclusivamente objetos versionados como componentes;
- parcialmente resolvidas (*"partly bound"*): compostas tanto de versões como de objetos versionados.

O objetivo dos sistemas que gerenciam configurações é gerar configurações resolvidas, pois elas representam produtos (programas, hiperdocumentos, etc.) completos e consistentes, que podem ser entregues aos clientes. Para gerar configurações resolvidas, é preciso escolher versões específicas de cada componente referenciado, ou seja resolver referências dinâmicas.

Dependendo de quando as referências dinâmicas são ligadas (resolvidas) a versões específicas, os sistemas são classificados como de ligação estática ou dinâmica [CON 96, CON 98]. Sistemas de *ligação estática* resolvem todas as referências dinâmicas antes que qualquer componente seja acessado. Por outro lado, os de *ligação dinâmica* resolvem as referências dinâmicas sob demanda.

A seleção de versões pode ser *manual*, quando o usuário seleciona as versões dos componentes usando seus próprios critérios (como no sistema COOP/Orm [MAG 96]). Entretanto, o número de combinações possíveis de versões de componentes pode ser muito grande, principalmente quando vários componentes (cada um com muitas versões) estão envolvidos. Para um objeto composto

contendo c componentes com v versões, existem v^c combinações possíveis, ou seja o número de combinações potenciais cresce polinomialmente em v [CON 96].

Entre esse grande número de configurações resolvidas possíveis, muitas podem ser inconsistentes. Portanto, é necessário restringir a escolha de versões, gerando apenas combinações consistentes. Estabelecendo estas restrições, torna-se possível realizar a *seleção automática de versões*, na qual o sistema escolhe as versões sem a intervenção de um usuário. A seleção de versões pode ainda ser *semi-automática*, quando as seleções automática e manual são combinadas por não ser suficiente ou adequado utilizar apenas uma delas.

Para realizar a seleção automática de versões, é necessário fornecer ao sistema uma *descrição de configuração* [CON 96] (também chamada configuração genérica [EST 95]). A descrição de configuração consiste de um objeto composto (ou uma de suas versões) e de critérios para selecionar as versões de seus componentes. Baseado nestas informações, o sistema é capaz de criar automaticamente configurações resolvidas (também chamadas instâncias de configuração [EST 95]).

Além dos critérios fornecidos com a descrição de configuração, alguns sistemas têm ainda um conjunto de critérios criado previamente pelos usuários; esse conjunto de critérios é chamado *base de critérios*. Durante a criação de uma configuração resolvida, o sistema utiliza critérios tanto da descrição de configuração quanto da base de critérios.

Os *critérios de seleção de versões* são expressões relacionando as propriedades das versões. São vários os formalismos usados para expressar os critérios, como opções em linha de comando (RCS [TIC 85]), expressões booleanas (Adele [EST 95]), *"feature logic"* (ICE [ZEL 95, ZEL 97]), funções heurísticas [NÁV 96], consultas semelhantes a SQL (SIO [BER 88]), etc.

Os critérios de seleção de versões costumam ser chamados de regras [CON 96]. Entretanto, algumas destas regras não são necessariamente atendidas durante a criação de uma instância de configuração. Por esse motivo, neste trabalho, o termo critério de seleção de versões [ØST 92] é preferido. Os critérios podem ser de três tipos:

- restrições (*"constraints"*) - critérios que devem ser obrigatoriamente satisfeitos. Normalmente, representam características necessárias para garantir a consistência da configuração resolvida;

- preferências - são critérios opcionais, que só se aplicam quando puderem ser satisfeitos por pelo menos uma versão;

- *"default"* - são critérios opcionais mais fracos, que somente se aplicam se a seleção de uma única versão não for possível de outra forma. Um critério *"default"* freqüentemente utilizado é a seleção da versão mais recente.

A figura 2.4 mostra a seleção de versões para configurar (gerar uma configuração resolvida) o programa modelado na figura 2.3. A seleção automática de versões é feita pelo chamado *processo de configuração*. Neste exemplo, a descrição de configuração fornecida consiste do objeto versionado "programa" e de um conjunto vazio de critérios de seleção. Primeiramente o processo escolhe uma versão (nó AND) do programa, por exemplo a versão 2 (passo "A"). O processo navega no grafo AND/OR e resolve os arcos AND escolhendo nós AND; neste exemplo o grafo é percorrido em profundidade.

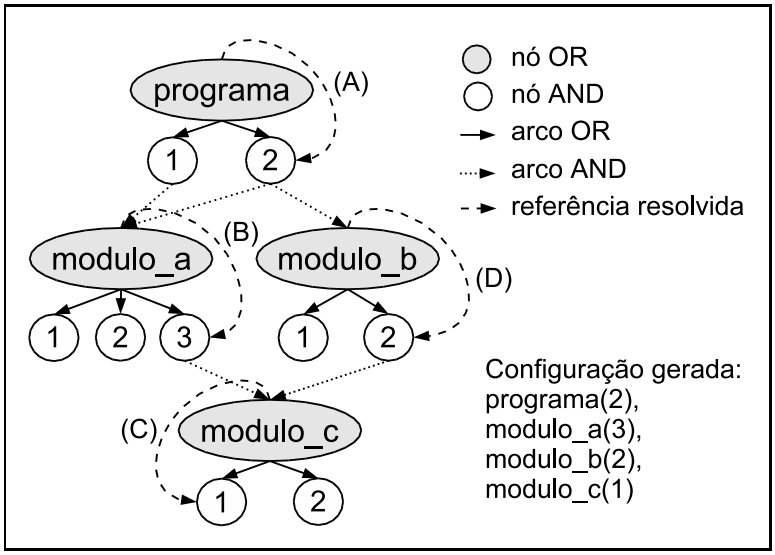


FIGURA 2.4 – Seleção de versões para criação de configuração

Continuando a partir da versão 2 de `produto` (`produto(2)`), o processo resolve o arco AND para `modulo_a` escolhendo a versão 3 (passo "B"). Em seguida, o processo resolve a referência de `modulo_a(3)` escolhendo `modulo_c(1)` (passo "C"). Por fim, o processo retorna à resolução de referências de `produto(2)` e resolve a referência para `modulo_b` escolhendo a versão 2 (passo "D"). A referência de `modulo_b(2)`, assim como a de `modulo_a(3)`, também deve ser resolvida para `modulo_c(1)`.

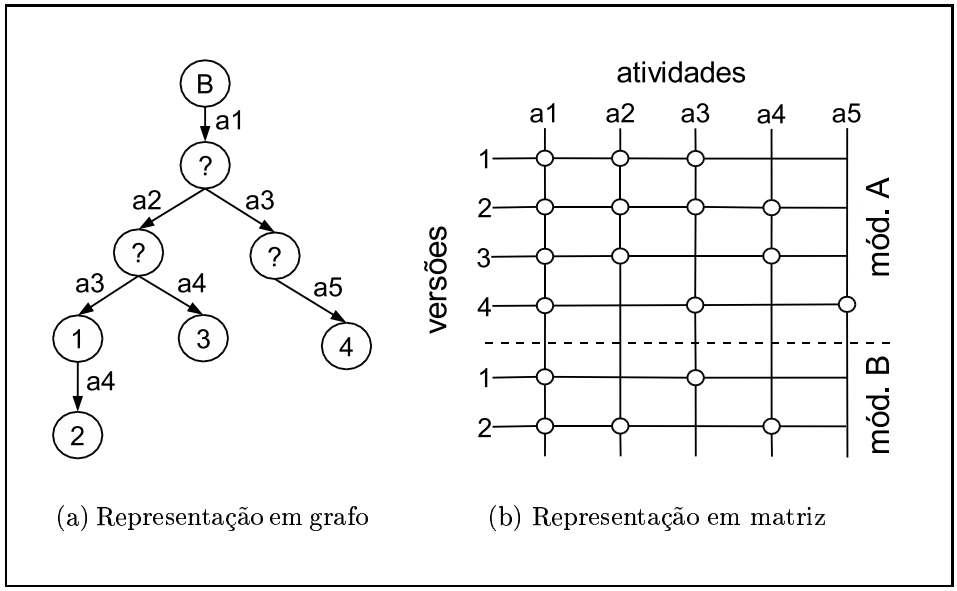
Em sistemas SCM, uma configuração resolvida é formada por uma única versão de cada componente do programa. Note-se que cada versão de um módulo implementa um subconjunto comum de funções, e que um programa não pode ter mais de uma implementação para a mesma função. Logo, uma configuração resolvida só pode ser consistente se for composta por apenas uma versão de cada componente.

Essa exigência não se verifica no domínio de documentos, pois documentos são estruturas hierárquicas, diferentes de programas que são conjuntos de módulos compilados e ligados. Portanto, é permitido que duas ou mais referências para um mesmo objeto versionado sejam resolvidas para versões diferentes, pois elas estão em contextos (posições na hierarquia) diferentes.

2.1.7 Conjuntos de modificações

Na área de SCM, modelos de versões que baseiam-se nos estados dos objetos versionados são chamados *baseados em estados* (*"state-based"*) [CON 98], e são os mais comuns. No entanto, existem modelos chamados *baseados em modificações* (*"change-based"*) [CON 98] que descrevem os estados dos objetos versionados em termos de modificações sobre um estado inicial (configuração resolvida).

Um *conjunto de modificações* é um objeto complexo, que contém alterações (fragmentos de arquivos) realizadas em vários objetos a fim de implementar uma alteração lógica [EST 95]. Também chamado de atividade (*"activity"* [EST 95]) ou tarefa (*"task"* [HAA 94]), ele representa modificações coordenadas em vários objetos versionados para a realização de uma tarefa. Por exemplo, considere-se uma tarefa de tradução de idioma (de inglês para português, por exemplo) de um programa. A realização desta tarefa exigirá a alteração de vários módulos do programa, incluindo



(a) Representação em grafo (b) Representação em matriz

FIGURA 2.5 – Atividades representando versões implicitamente

todos os de interface com o usuário. Todas essas alterações são então agrupadas em um conjunto de modificações.

Modelos baseados em modificações não mantêm versões explícitas dos objetos versionados; as versões são implicitamente construídas pela combinação dos conjuntos de modificações. A figura 2.5 mostra duas representações para atividades criando versões implicitamente.

A figura 2.5(a) mostra atividades sendo combinadas para construir versões de um único objeto versionado. As versões indicadas por um ponto de interrogação (?) são versões anônimas (estados intermediários), possivelmente inconsistentes. As versões são geradas pela aplicação de uma seqüência de atividades sobre uma configuração resolvida B; por exemplo, a versão 3 é gerada pela combinação das atividades a1, a2 e a4, nesta ordem. Note-se que a aplicação de algumas atividades não depende da aplicação de outras; por exemplo, a4 não depende de a3 para gerar a versão 3, mas depende dela para gerar a versão 2.

A figura 2.5(b) mostra atividades em forma de uma matriz: as linhas representam versões (de dois objetos versionados, "mod. A" e "mod. B") e as colunas representam as atividades. Círculos no encontro de uma linha com uma coluna indicam que a atividade é necessária para a geração da versão. Pode-se notar que uma atividade pode ser usada para gerar várias versões, inclusive de objetos versionados diferentes. Entretanto, essa representação não torna explícita a ordem em que as atividades devem ser aplicadas.

Exemplos de sistemas SCM que implementam um modelo baseado em modificações são o COV (*"Change Oriented Versioning"*) [MUN 93] e o ICE (*"Incremental Configuration Environment"*) [ZEL 95, ZEL 97]. Os modelos baseados em modificações e em versões não são mutuamente excludentes. No sistema COV, grafos de versões podem ser gerados a partir do modelo baseado em modificações; isso é obtido através da definição de restrições sobre as combinações das atividades (semelhante à figura 2.5(a)). O inverso também é possível; por exemplo, o sistema SCM Asgard implementa um modelo baseado em modificações sobre um modelo baseado em versões, conforme analisado em [CON 98].

Em sistemas de hipertexto (CoVer [HAA 92, HAA 94]), o conceito de atividade também coexiste com os grafos de versões. No entanto, as atividades são modeladas como conjuntos de versões completas (não só de alterações), possivelmente de objetos versionados diferentes, que foram criadas pela realização de uma tarefa de modificação.

2.1.8 *"Workspaces"*

O controle de versões pode ser usado para suportar o trabalho cooperativo [HAA 93], pois permite que várias pessoas trabalhem simultaneamente no mesmo objeto versionado. Assim, cada usuário gera versões diferentes, que podem então ser fundidas (*"merge"*) em uma única versão que agregue todas as alterações.

No entanto, se as versões não forem alteráveis (como é típico em sistemas de SCM), cada usuário (ou grupo) necessita de um espaço temporário para manter uma cópia da versão a ser modificada, a fim de armazenar as devidas alterações. Este espaço, chamado *"workspace"*, serve para isolar as alterações feitas por um desenvolvedor das alterações paralelas realizadas por outros. Além disso, ele torna a versão em desenvolvimento privada, evitando que outros usuários acesse-na antes que esteja consistente.

"Workspace" é o lugar onde usuários e ferramentas criam, acessam e modificam objetos, normalmente não versionados [EST 95]. Ou seja, enquanto os objetos estão dentro do *"workspace"* as alterações feitas não criam versões. A operação de criar, dentro do *"workspace"*, uma cópia de uma versão do repositório público é chamada *"check-out"*.

Novas versões são criadas somente quando as alterações são propagadas para o repositório público de versões. A operação de copiar um objeto em edição no *"workspace"* de volta para o repositório público é chamada *"check-in"*, e cria uma versão derivada da versão que originalmente sofreu *"check-out"*. O *"workspace"* também é um local adequado para implantar controle de acesso [KAT 90].

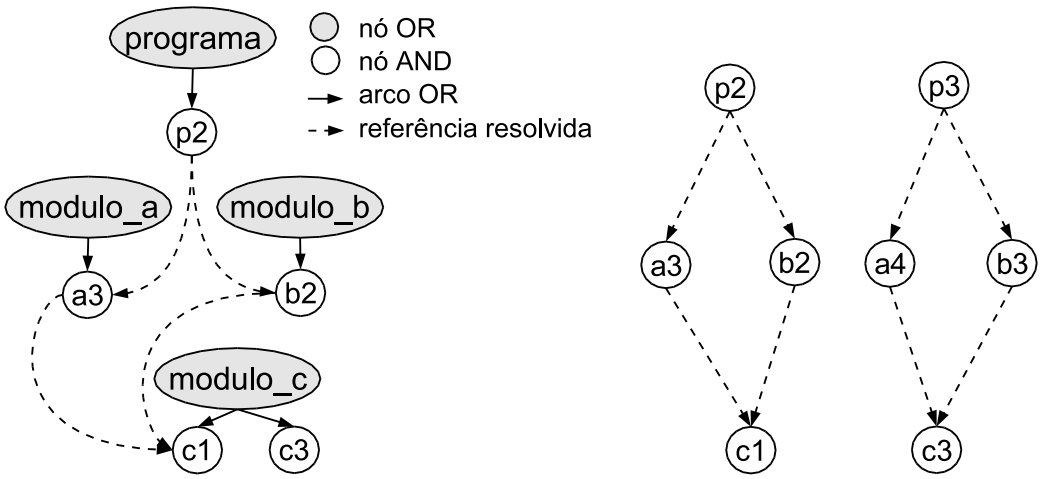
Alguns modelos de hiperdocumentos [HAA 92, ØST 92, NOR 98a] permitem que versões sejam modificadas, dependendo de seu estado (*"status"*). Eles diferenciam versões atualizáveis, que podem ser modificadas, de versões congeladas, que são somente para leitura. Portanto, estes modelos são menos dependentes de um *"workspace"* e assim não o modelam explicitamente.

2.1.9 Notificação/Propagação de mudanças

Uma vez que objetos versionados compostos podem usar componentes versionados, alterações no histórico destes (criação de novas versões) podem requerer atualizações no estado daqueles. Alguns sistemas apresentam um mecanismo de propagação [KAT 90] ou notificação [CHO 86] deste tipo de alteração. Normalmente, estes sistemas criam configurações resolvidas sem usar critérios de configuração.

Propagação de alterações [KAT 90] é o processo que automaticamente incorpora novas versões a configurações (resolvidas). Por exemplo, a figura 2.6(a) mostra a configuração da figura 2.4. Uma nova versão do módulo "c" foi criada (c3) como sucessora da versão "c1"; assim, a figura 2.6(b) mostra a propagação desta alteração. Nesta figura, os objetos versionados foram omitidos para torná-la mais clara.

Uma vez que as versões "a3" e "b2" referenciam "c1", a propagação criou versões sucessoras para elas ("a4" e "b3", respectivamente). Estas novas versões referenciam



(a) configuração resolvida e criação de c3 (b) propagação pela criação de c3

FIGURA 2.6 – Propagação de alterações

a versão "c3". A propagação pode então continuar, pois novas versões foram criadas. Como a versão "p2" referencia "a3" e "b2" e foram criadas sucessoras para as últimas, uma nova versão sucessora de "p2" também é gerada. Assim, a versão "p3" é criada como sucessora de "p2" e referencia "a4" e "b3".

Outra forma de tratar alterações em componentes é a *notificação*. Nesta abordagem, quando uma versão é criada para um componente, todos os usuários deste componente (criadores de um objeto composto que o utiliza) são avisados. Assim, eles podem escolher atualizar ou não a referência para a nova versão. Existem dois tipos de notificação: *baseada em mensagens* (*"message-based"*) e *baseada em sinais* (*"flag-based"*). No primeiro tipo, mensagens são enviadas aos usuários, enquanto que o segundo marca o objeto composto com um sinal, de forma que o usuário só é notificado quando acessa o objeto.

A principal vantagem da propagação é que a atualização das referências é feita automaticamente. No entanto, a principal vantagem da notificação é que não existe proliferação de versões inúteis, pois o usuário atualiza a referência apenas se for necessário. Por outro lado, a notificação é um processo manual e a propagação potencialmente prolifera versões inúteis.

Em sistemas com suporte à configuração automática baseada em critérios de seleção de versões, normalmente tais mecanismos de tratamento de alterações estão ausentes. Estes sistemas confiam nos critérios para escolher as versões, ao invés de dependerem da intervenção humana ou de gerarem versões inúteis; portanto quando uma configuração resolvida é gerada, as novas versões podem ou não ser selecionadas, dependendo dos critérios usados.

2.2 Modelos de versões de documentos

Esta seção analisa alguns modelos de versões para hiperdocumentos e documentos estruturados. A análise destes modelos é feita através de comparações das características dos modelos com os conceitos apresentados na seção 2.1; também são levados em consideração os requisitos relacionados ao versionamento que foram sugeridos na literatura [HAL 88, ØST 92, HAA 92].

Os modelos selecionados para a análise são:

- HyperPro [ØST 92], um dos primeiros modelos de versões de hipertexto a serem publicados. Seu autor sugeriu que, para diminuir o *"overhead"* cognitivo devido à criação de versões, seria necessário criar versões implicitamente. Por *"overhead"* cognitivo entende-se o esforço mental dedicado a aspectos não relacionados com a tarefa de fazer algo [ØST 92];

- CoVer (*"Contextual Version Server"*) [HAA 92, HAA 94], introduziu o conceito de atividades em sistemas hipertexto. Através do conceito de atividades, o CoVer é capaz de criar versões implicitamente;

- NCM (*"Nested Context Model"*) [SOA 95], é um modelo bastante complexo (no sentido de modelar vários conceitos), possuindo muitas características interessantes como a separação de formatação e conteúdo dos hiperdocumentos;

- GDOC (Gestão de Documentos) [GRA 97, GRA 98, NOR 98b, NOR 98, NOR 98a, SAN 99, SAN 2000], é o modelo de um sistema para gerência de documentos estruturados que trata configurações (resolvidas) explicitamente. Como o NCM, é capaz de separar formatação e conteúdo dos documentos;

- CoMa [WES 96], um modelo para documentos estruturados adaptável e independente de domínio que é baseado em grafos.

Uma vez que hipertexto é definido como um conjunto de nós interligados por elos (*"links"*) [SMI 88], os conceitos de nó atômico e elo são comuns a todos os modelos analisados. Nós atômicos são entidades com conteúdo, enquanto que elos estabelecem um caminho de navegação direcional entre dois nós (do origem para o destino). Os modelos selecionados têm ainda o conceito de nó composto, que agrupa elos e outros nós (tanto atômicos quanto nós compostos), formando hierarquias de composição.

Como hipertexto forma uma estrutura hierárquica semelhante a um documento estruturado, os mecanismos necessários ao controle de versões destes dois tipos de documentos também são semelhantes [NOR 98a]. Por essa razão, dois modelos de versões para documentos estruturados (GDOC e CoMa) foram incluídos na comparação entre estes modelos.

A seguir os modelos selecionados são analisados considerando vários aspectos importantes relacionados ao controle de versões, sendo providas citações de outros trabalhos reconhecendo a importância destes aspectos.

1. **Objetos compostos.** É necessária uma forma de representar e manipular grupos de nós e elos como um uma entidade separada de seus componentes [HAL 88].

Todos os modelos analisados suportam objetos compostos, apesar de seus nomes variarem: contexto (*"Context"*, HyperPro), nó composto (*"Composite Node"*, CoVer), contexto do usuário (*"User Context"*, NCM), composto (*"Composite'*, GDOC), grupo de documentos (*"Document Group'*, CoMa).

2. **Manutenção do histórico.** Um bom mecanismo de versionamento permitirá que usuários mantenham e manipulem o histórico das alterações de seu documento [HAL 88].

   Naturalmente, todos os modelos analisados mantém o histórico de versões para os objetos versionados. No HyperPro, o objeto que contém o histórico (objeto versionado) é chamado *"VersionGroup"*, enquanto que o NCM chama esse objeto de *"Version Context"*. O CoVer tem o chamado "mob" (*"multi-state object"*), que armazena objetos "snob" (*"single-state object"*) de apenas um estado (versões).

   O GDOC chama o objeto versionado de documento versionado. A despeito do que o nome possa sugerir, um documento versionado não é necessariamente um documento inteiro, podendo ser também um componente (parte) de outro documento. O CoMa chama o objeto com o histórico simplesmente de objeto; ele é um nó (de grafo, não de hipertexto) que possui vários arcos para nós de histórico; por sua vez, nós de histórico representam relações de derivação (predecessora/sucessora) entre as versões.

3. **Objetos com controle de versões.** Deve ser possível manter os estados de nós, elos e compostos [HAA 92].

   Todos os modelos analisados versionam nós, tanto os atômicos quanto os compostos. Entretanto, o CoVer, o GDOC e o CoMa são capazes também de manter versões de elos. Os autores do HyperPro e do NCM argumentam que o versionamento de elos pode ser obtido criando novas versões dos nós compostos que os hospedam (ancoram).

4. **Estrutura do histórico de versões.** Deve ser possível manter alternativas. Revisores e autores querem manter alternativas explícitas de sub-partes de um documento [HAA 92].

   Todos os modelos analisados suportam versões alternativas. O HyperPro e o GDOC organizam o histórico de versões em forma de árvore. Por sua vez, a estrutura do histórico no CoVer e no NCM é em forma de grafo, possivelmente desconexo (as versões não estão necessariamente ligadas por relações de derivação). O CoVer permite que um objeto não-versionado (*"snob"*) seja transformado em uma versão de um objeto versionado. Isso é feito através da criação de uma relação de derivação entre uma versão do objeto versionado e o objeto não versionado.

   O NCM suporta as chamadas versões correlatas, versões que não são do mesmo tipo (por exemplo, um nó atômico e um nó composto) mas que pertencem ao histórico do mesmo objeto versionado. O histórico no CoMa é organizado em forma de grafo, mas não há informações sobre se este grafo pode ou não ser desconexo, como no CoVer e NCM.

5. **Referências e sua atualização automática.** Uma referência para uma entidade pode se referir a uma versão específica, à versão mais nova dentro de um galho (*"branch"*) específico, ou à (última) versão que case com uma descrição (consulta) particular [HAL 88].

   Modelos que suportam referências dinâmicas resolvem-nas usando algum critério, mesmo que seja um *"default"*. Portanto, todos os modelos analisados têm suporte à atualização automática de referências. Além disso, alguns modelos permitem a especificação de critérios para selecionar versões de um objeto versionado, conforme descrito no item 13. Note-se ainda que, em geral, os modelos empregam resolução dinâmica de referências (sob demanda, conforme subseção 2.1.6).

   O HyperPro tem dois tipos distintos de elos: o comum (referência estática), que liga dois nós (versões) quaisquer, e o *"GenericVersionLink"* (referência dinâmica), que tem um *"VersionGroup"* como destino. O CoMa também tem referências estáticas e dinâmicas, entretanto não há informação em [WES 96] sobre como as referências dinâmicas são resolvidas.

   No CoVer, os elos com destino a um "mob" têm o identificador do "mob". Além disso, há ainda um conjunto (possivelmente unitário) de identificadores de versões específicas do "mob" ou uma expressão de consulta para selecionar versões. Se duas ou mais versões destino são selecionadas (pelo conjunto de identificadores ou pela consulta), elas são consideradas alternativas. Quando é usada uma expressão de consulta, o elo atua como uma referência dinâmica.

   Os elos do NCM podem ter múltiplos destinos, permitindo conexões um-para-muitos. Isso suporta aplicações onde a seleção de um elo leva à exibição simultânea de vários nós. Em outros modelos, o mesmo efeito pode ser obtido com um elo que aponte para um nó composto. Elos com um *"Version Context"* como destino podem ser resolvidos para a versão corrente ou para uma versão selecionada por uma consulta.

   O GDOC tem tanto referências estáticas quanto dinâmicas, sendo que as últimas são resolvidas para a versão corrente de um documento versionado. A exceção a esse comportamento ocorre durante a geração de uma configuração resolvida (estaticamente, conforme subseção 2.1.6), como é explicado no item 13.

6. **Gerência de configurações.** Se queremos suportar desenvolvimento exploratório o problema é como congelar um estado. Toda a estrutura em que o autor está trabalhando deve ser congelada [HAL 88].

   Uma configuração (não ou parcialmente resolvida) no HyperPro é uma versão de um nó composto. Quando esse nó é acessado, suas referências dinâmicas são resolvidas (sob-demanda). No entanto, não existe equivalência para uma configuração resolvida armazenada (ela é transiente, apenas visualizada). Naturalmente, uma versão do nó composto poderia ter apenas referências estáticas, o que é equivalente a uma configuração resolvida; entretanto, estas versões não são diferenciadas de versões comuns.

   Quando os elos usam expressões de consulta, o CoVer é similar ao HyperPro. Nos outros modelos analisados, as configurações não resolvidas são os próprios

nós compostos, como no HyperPro. Para o CoMa, configurações resolvidas são também objetos versionados, sendo mantidos grafos de versões para elas.

No NCM, o controle de configurações é feito através do conceito de bases privadas, que agrupam funcionalidades de atividades e *"workspaces"* como descrito nos itens 7 e 8. Quando o usuário conclui a edição de uma configuração dentro da base privada, ele pode transferi-la para o repositório público. Nesta ocasião, ele pode optar pela geração de uma configuração resolvida, já que a não resolvida é gerada por *"default"*. Apesar disso, as configurações resolvidas não são diferenciadas de documentos com referências estáticas.

As configurações resolvidas são chamadas versões configuradas no GDOC, portanto são diferenciadas de versões comuns. Quando uma versão configurada é criada (estaticamente) para um objeto composto, novas versões configuradas são geradas para cada um de seus componentes. Em outras palavras, uma configuração resolvida tem somente componentes que também são configurações resolvidas e que são referenciados por referências estáticas.

7. **Suporte para alterações coordenadas (atividades)**. Usuários farão alterações coordenadas a um conjunto de entidades no documento em algum momento. O desenvolvedor pode então querer coletar as versões individuais resultantes em um único conjunto de versões para futura referência [HAL 88].

   No HyperPro, GDOC e CoMa não há suporte ao conceito de atividade. No CoVer, as tarefas podem ter sub-tarefas; enquanto uma tarefa não é concluída, os objetos que estão em edição nela somente são visíveis por suas sub-tarefas. Neste ponto, as tarefas do CoVer são similares a *"workspaces"*. As bases privadas do NCM funcionam de forma semelhante a atividades. Como no CoVer, bases privadas podem ser aninhadas, entretanto elas não são "concluídas" e sim exportadas para o repositório público.

8. ***"Workspaces".***

   O HyperPro, GDOC e CoMa não suportam o conceito de *"workspace"*. O CoVer tem um conceito de tarefa que engloba parte da funcionalidade de *"workspace"*, pois as versões criadas em uma tarefa não são visíveis fora dela. No entanto, como o CoVer suporta versões atualizáveis (ver item 11), a edição de versões pode ser feita mesmo fora de atividades.

   O NCM tem o conceito de *"workspaces"* sob o nome de bases privadas. Uma vez que as versões do repositório público só podem estar nos estados congelada e obsoleta (conforme item 11), as versões em desenvolvimento são criadas em bases privadas.

9. **Controle de versões de elos.** "Até então, somente versionamento para nós foi considerado. Deve-se também considerar um versionamento separado para elos." [ØST 92].

   O HyperPro e NCM não controlam versões de elos, mas pode-se obter essa funcionalidade através do versionamento de nós compostos. O CoVer, GDOC e CoMa suportam o versionamento de elos. No GDOC, uma versão de um elo seleciona a versão corrente do documento versionado destino (quando não estiver gerando uma configuração resolvida estaticamente).

Sobre o CoVer, não fica claro em [HAA 92, HAA 94] se uma versão de um elo (que usa consultas) sempre seleciona ou não a mesma versão de um "mob" destino. Para o CoMa [WES 96], não há informações de como é escolhida uma versão de um objeto versionado apontado por um elo.

10. **Controle de versões da estrutura.** "É desejável ter uma noção das versões da estrutura do documento" [ØST 92], ou seja saber como os nós estão aninhados.

    Uma vez que os nós compostos representam a estrutura hierárquica dos componentes dos documentos, seu versionamento permite manter a estrutura do documento. Desta forma, todos os modelos que versionam nós compostos suportam o controle de versões da estrutura do documento.

11. **Mutabilidade de versões.** "Em hipertexto é simplista demais ter versões de nós que sejam completamente imutáveis" [ØST 92].

    Todos os modelos analisados possuem pelo menos dois *"status"* de versões: atualizável e congelada. O primeiro permite que a versão seja modificada sem gerar novas versões, enquanto que o segundo não.

    O HyperPro e o NCM permitem que alguns atributos (escolhidos pelo usuário) de uma versão congelada sejam alterados ou que novos atributos sejam adicionados sem gerar novas versões. O HyperPro chama o estado destas versões de "gel". O NCM e o GDOC têm ainda o estado de versão obsoleta; versões neste estado não podem ser usadas para derivação de novas versões.

12. **Pequeno *"overhead"* cognitivo na criação de versões.** "A criação explícita de versões resultará em um grande *"overhead"* cognitivo" [ØST 92].

    O HyperPro, GDOC e CoMa suportam apenas a criação explícita de versões. Além da criação explícita de versões, no CoVer há a criação implícita de versões, que ocorre quando os elementos (congelados) de uma atividade são modificados. No NCM, a criação de versões se dá quando as alterações nos elementos (versões) de uma base privada (*"workspace"*) são publicadas para o repositório público.

13. **Seleção de versões.** "Similar a acessar objetos através de uma consulta, o autor quer acessar versões de objetos hipertexto baseado nos valores de seus atributos" [HAA 92].

    No HyperPro, a seleção de versões, como destino de um elo para um objeto versionado, é feita baseada em critérios de seleção associados ao nó composto que contém o elo. Assim, todos os elos do mesmo nó composto usam os mesmos critérios, o que diminui o *"overhead"* cognitivo, liberando o usuário de especificar critérios para cada elo. O critério *"default"* é a seleção da versão mais recente.

    No Cover, a seleção de versões de um "mob" baseia-se na identificação explícita das versões ou em uma expressão de consulta associada ao elo que referencia o "mob". Por *"default"*, o GDOC resolve as referências dinâmicas para a versão corrente do documento versionado. Entretanto, durante a geração de uma configuração resolvida (estaticamente), o GDOC pode usar critérios especificados pelo usuário para escolher versões. A descrição do CoMa [WES 96] não

comenta como uma versão de um objeto versionado é escolhida para formar uma configuração resolvida.

A versão corrente em um objeto versionado do NCM pode ser definida pelo autor indicando-a diretamente ou através de uma consulta. O critério para a seleção de versões pode ainda estar associado a uma base privada. Quando um elo é criado, se o usuário não especificou a versão corrente para o objeto versionado, o critério da base privada é usado. Se o critério da base privada não foi definido, o elo usa o critério *"default"* do objeto versionado.

14. **Reuso de nós em outros documentos.** "Desde que partes de um documento podem ser reusadas em versões de outro, é necessário registrar as versões com respeito ao seu uso em compostos" [HAA 92].

    Todos os modelos analisados suportam este requisito, pois os nós podem ser compartilhados (referenciados) por vários nós compostos. Entretanto, o suporte ao reuso no NCM e GDOC (quando usam versão corrente para resolver referências dinâmicas) é mais limitado, pois a seleção de versões se baseia no objeto versionado e não no uso que é feito dele.

    Em outras palavras, a versão corrente ou consulta para selecionar versões está associada ao objeto versionado e não ao nó composto ou elo que o referencia. Com isso, todos os nós compostos e elos que referenciam o mesmo objeto versionado selecionam a mesma versão em determinado instante, a despeito das necessidades particulares de cada objeto composto ou elo.

15. **Separação de conteúdo e formatação.** "A maioria dos sites Web é confeccionada utilizando HTML, onde o conteúdo e a formatação estão mesclados em um mesmo arquivo. A separação entre esses componentes simplifica a manutenção das páginas" [MOR 2002].

    Entre os modelos analisados, os que separam a formatação e o conteúdo são o NCM e o GDOC. O NCM armazena a formatação na chamada especificação de apresentação; ela é associada a elos e nós compostos e refere-se aos nós destino dos elos ou aos componentes dos nós compostos, respectivamente.

    O GDOC armazena a formatação separadamente (independente de nós e elos), permitindo inclusive que ela seja reusada por vários documentos. Além disso, um mesmo componente pode ter formatações distintas dependendo do nó composto ao qual esteja vinculado (como no NCM). No entanto, nem o GDOC nem o NCM mantêm versões da formatação.

A tabela 2.1 resume as características dos modelos analisados, conforme descritas nesta seção. Desta forma, estas características dos modelos podem ser comparadas mais facilmente.

2.3 Seleção de versões

A maioria dos modelos analisados na seção 2.2 possui recursos limitados de gerência de configurações. Em especial, nenhum deles permite a especificação de preferências (critérios opcionais) para a seleção das versões. Esta seção analisa

TABELA 2.1 – Resumo das características dos modelos analisados

Característica	**HyperPro**	**CoVer**	**NCM**	**GDOC**	**CoMa**
Objetos compostos	contexto	nó composto	contexto do usuário	composto	grupo de documentos
Histórico de versões	grupo de versões	"mob"	contexto de versões	documento versionado	objeto
Objeto versionado	nós	nós e elos	nós	nós e elos	nós e elos
Estrutura do histórico	árvore	grafo desconexo	grafo desconexo	árvore	grafo
Referências	estática e dinâmica	estática e dinâmica	estática e dinâmica	estática e dinâmica	estática e dinâmica
Configurações	não resolvida	não resolvida	resolvidas e não resolvidas	não resolvidas e resolvidas (versões configuradas)	não resolvidas e resolvidas; podem ter versões
Atividades	não	sim	sim (*"workspace"*)	não	não
"Workspaces"	não	sim (atividade)	sim	não	não
Versões de elos	não	sim	não	sim	sim
Versões da estrutura	sim	sim	sim	sim	sim
"Status" das versões	atualizável e congelada (e "gel")	atualizável e congelada	atualizável, congelada e obsoleda	atualizável, congelada e obsoleta	atualizável e congelada
Criação implícita de versões	não	sim	sim	não	não
Seleção de versões	critérios associados ao nó composto	consulta associada ao elo	critérios associados a referência, nó composto ou *"workspace"*	critérios associados ao nó composto	-
Reuso de nós	sim	sim	sim	sim	sim
Separação conteúdo e formatação	não	não	sim (sem versionamento da formatação)	sim (sem versionamento da formatação)	não

trabalhos relacionados com a gerência de configurações, enfocando principalmente o mecanismo de seleção de versões para a geração de configurações resolvidas.

Os trabalhos analisados são o sistema de SCM Adele [EST 95], os sistemas de gerência de documentos COOP/Orm [MAG 96] e GDOC [SAN 2000] e o método de seleção de versões de Návrat [NÁV 96]. A descrição destes trabalhos aborda as seguintes características:

- tipos de critérios de configuração suportados (restrições, preferências e *"defaults"*);
- suporte a base de critérios e/ou critérios parâmetro (aqueles da descrição de configuração);
- formalismo para expressão dos critérios;
- grau de automação (manual, automático, semi-automático);
- suporte a seleção de mais de uma versão por objeto versionado;
- modo de resolução de referências (estático ou dinâmico);
- algoritmo.

2.3.1 Sistema COOP/Orm

O sistema COOP/Orm [MAG 96] não utiliza critérios de configuração. Quando um autor cria uma referência entre documentos, ele deve escolher uma versão específica para ser o destino da referência; assim, os documentos são sempre configurações resolvidas. A escolha da versão destino da referência é auxiliada pelo sistema por meio de um mecanismo de filtragem baseado no *"status"* da versão.

Quando uma nova versão de um documento componente é criada, a referência do documento composto pode ser atualizada (manualmente, mas com ajuda do sistema) para a nova versão. A atualização de referências gera uma nova versão do documento composto. Esse mecanismo de atualização de referências é semelhante à notificação de atualizações explicada na subseção 2.1.9. Entre os sistemas analisados, o COOP/Orm é o único a permitir que mais de uma versão do mesmo objeto versionado componha a mesma configuração resolvida; logicamente, cada versão é o destino de uma referência diferente.

2.3.2 Sistema GDOC

No sistema GDOC [SAN 2000], a criação de configurações resolvidas é feita (semi-)automaticamente, resolvendo referências dinâmicas estaticamente. O GDOC usa expressões booleanas para especificar tanto os critérios de configuração parâmetro quando os da base de critérios. Existem critérios da base chamados opcionais, que são aplicados apenas aos componentes aos quais se referem. A despeito do nome, todos os critérios associados a um componente devem ser obedecidos, ou seja são obrigatórios (restrições).

Quando existir mais de uma versão candidata que obedeça aos critérios, pode-se utilizar o critério *"default"*, que é a seleção da versão mais recente ou uma escolha manual. No último caso, o processo de configuração é semi-automático. O GDOC

permite ainda o reaproveitamento de sub-configurações: se uma versão participar de alguma configuração resolvida (for uma versão configurada), ela não terá referências dinâmicas. Logo, se o processo de configuração escolhê-la para participar de uma nova configuração resolvida, não necessitará processá-la (resolver referências) novamente.

O processo de configuração do GDOC atravessa o grafo de composição e avalia todos os critérios aplicáveis a cada componente. Se nenhuma versão de um componente for selecionada, o usuário é informado e pode modificar a descrição de configuração. Se mais de uma versão forem selecionadas pode-se: selecionar a versão mais recente, selecionar uma versão manualmente ou alterar a descrição da configuração.

2.3.3 Sistema Adele

A criação de configurações resolvidas no sistema Adele [EST 95] pode ser automática ou semi-automática e as referências são resolvidas estaticamente. São suportados critérios de configuração parâmetro e da base de critérios. Eles são diferenciados explicitamente em restrições, preferências e *"defaults"* e são especificados com expressões booleanas. Como é comum em sistemas SCM, apenas uma versão pode ser selecionada por objeto versionado. Partes de configurações resolvidas já armazenadas podem ser reaproveitadas para gerar novas configurações resolvidas.

O processo de configuração mantém um conjunto de critérios (C) inicializado com os critérios parâmetro. Conforme o grafo AND/OR é percorrido, C é estendido pela adição de critérios da base que estejam associados à cada versão selecionada. A seleção de uma versão só é realizada quando todos os critérios necessário forem conhecidos. Isso ocorre quando todas as versões de objetos que referenciam o componente forem selecionadas.

2.3.4 Método de seleção de versões de Návrat

O método de seleção de versões de Návrat [NÁV 96] não é um sistema completo, apenas uma forma (possivelmente semi-) automática de selecionar uma versão, fornecidos um conjunto de versões e um conjunto de critérios. Portanto, não existe uma base de critérios. Os critérios são expressos como uma lista de funções heurísticas, sendo que a primeira é obrigatória e as seguintes são opcionais.

É reconhecida a necessidade de um critério *"default"*, a ser usado quando não for possível selecionar uma única versão usando apenas a lista de funções; entretanto, tal critério não é especificado. Cada função heurística é avaliada para o conjunto de versões como um todo e não para cada versão individualmente. Isso permite especificar critérios como "selecionar a versão com o maior valor para a propriedade x".

O método de seleção avalia a primeira função heurística (h_0, obrigatória) sobre o conjunto inicial de versões (V) e obtém o conjunto de versões candidatas (V_0). O processo de seleção é interrompido se V_0 for vazio ou unitário, pois, respectivamente, não existe versão admissível ou a única versão selecionável já foi encontrada.

Uma a uma as funções heurísticas opcionais (h_i, onde i varia de um até a quantidade de funções opcionais) são avaliadas sobre um conjunto de versões candidatas V_{i-1}, gerando um novo conjunto de versões candidatas V_i. Entretanto, se V_i for vazio, o efeito da função h_i é descartado ($V_i = V_{i-1}$). Quando V_i tornar-se unitá-

rio, conterá a versão selecionada, e o processo é interrompido. Se, após a execução de todas as funções, ainda existirem mais de uma versão candidata, algum critério *"default"* deve ser utilizado.

2.3.5 Conclusão

Esta seção analisou alguns trabalhos relacionados à seleção de versões para a criação de configurações resolvidas. Pode-se notar que apenas um sistema permite selecionar mais de uma versão por objeto versionado, o que é importante em sistemas de hiperdocumentos. Se duas referências ao mesmo objeto versionado tivessem critérios diferentes de seleção de versões, deveria ser possível selecionar versões diferentes para cada uma delas.

Além dos objetos versionados, a base de critérios de configuração também é um objeto que evolui, portanto deveria estar sob controle de versões [CON 96]. O único trabalho estudado que atende a essa necessidade é o UVM (*"Uniform Version Model"*) [WES 2001], no entanto o versionamento é baseado em tempo, ou seja o histórico de versões tem uma estrutura em forma de lista.

2.4 WebDAV

A seção 2.2 descreveu modelos de versões implementados em alguns sistemas de hipertexto e documentos estruturados. Entretanto, estes modelos (e sistemas) não podem ser facilmente integrados a outros aplicativos de hipertexto. Outros sistemas foram desenvolvidos [HAA 96, HIC 98] com a preocupação de reusar a infra-estrutura de controle de versões. Mesmo assim, é difícil reusar essa infra-estrutura enquanto não existir uma forma unificada e padrão de tratar o controle de versões.

Existe um grupo de trabalho que busca padronizar o controle de versões na Web, chamado WebDAV (*"Web Distributed Authoring and Versioning"*) [STE 2002]. Esta seção explica um protocolo de rede padrão, de mesmo nome, gerado por este grupo. O protocolo WebDAV estende o HTTP (*"Hypertext Transfer Protocol"*) [FIE 99] a fim de prover uma infra-estrutura para autoria assíncrona em equipe (*"collaborative"*) através da Internet [WHI 98].

Usando o protocolo WebDAV, é possível publicar arquivos em um servidor Web como se ele fosse um sistema de arquivos compartilhado. Dessa forma, o servidor Web deixa de ser somente para leitura e permite também a gravação de arquivos e criação de diretórios (obviamente resguardado por um sistema de segurança com recursos de autenticação e controle de permissões).

O protocolo HTTP [FIE 99] define regras detalhadas para a comunicação entre o servidor Web e um aplicativo cliente, especificando inclusive que seqüência de bytes deve ser enviada pela rede em cada ocasião. Além disso, o WebDAV ainda define conceitos relacionados à autoria remota e ao versionamento de arquivos, bem como sua semântica, formando o chamado modelo de dados do protocolo. O foco desta seção é justamente este modelo de dados. Entretanto, a comunicação usando o protocolo HTTP é introduzida a seguir.

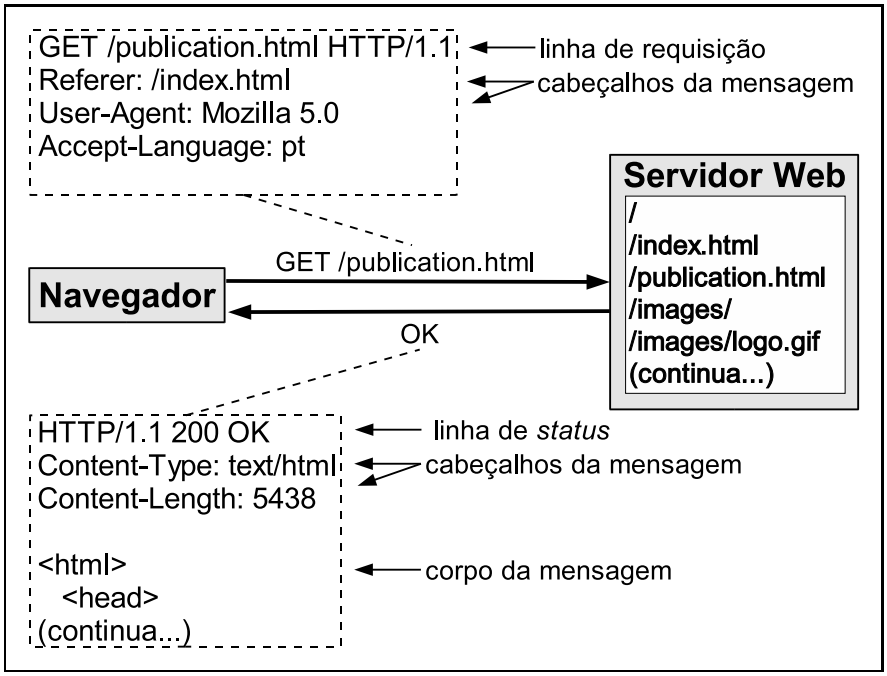


FIGURA 2.7 – Comunicação HTTP entre um navegador e um servidor Web

2.4.1 Comunicação usando o protocolo HTTP

A figura 2.7 mostra a comunicação entre um navegador e um servidor Web usando o protocolo HTTP, que é baseado em mensagens. O navegador inicia a comunicação enviando uma requisição (detalhada na caixa tracejada acima) solicitando o conteúdo da página "/publication.html". A mensagem de requisição consiste de: linha de requisição, conjunto de cabeçalhos e corpo. A linha de requisição informa o método (comando) a ser aplicado a um recurso (arquivo) e o URI (*"Uniform Resource Identifier"*) [BER 98] do recurso. O conjunto de cabeçalhos informa metadados da requisição e/ou do corpo da mensagem. O corpo da mensagem é usado para transportar dados de uma entidade e está vazio, no caso desta requisição.

O cabeçalho *"User-Agent"* informa ao servidor Web qual navegador está sendo usado e o *"Accept-Language"* indica o(s) idioma(s) aceito(s)/preferido(s) pelo usuário. O cabeçalho *"Referer"*[sic] informa qual o recurso de onde foi obtido o URI do recurso indicado na linha de requisição. Na prática, esse cabeçalho indica a página que está sendo atualmente visualizada no navegador.

Em resposta à requisição do usuário, o servidor Web aplica o método sobre o recurso, neste caso o método GET. Esse método é usado para obter o conteúdo do recurso. O servidor Web então envia uma mensagem de resposta ao navegador (caixa tracejada abaixo). A mensagem de resposta possui uma linha de *"status"*, contendo um código que indica o resultado da operação (no exemplo, sucesso). O restante da mensagem de resposta é semelhante à mensagem de requisição.

Entretanto, o corpo da resposta é ocupado pelo conteúdo do recurso solicitado. Os cabeçalhos *"Content-Type"* e *"Content-Length"* informam o tipo e tamanho (quantidade de bytes) do corpo da mensagem, respectivamente. Com essas informações, o navegador pode tratar o corpo da mensagem adequadamente.

Se a página recebida referenciar recursos multimídia (como figuras), o navegador deve obtê-los para gerar a visualização da página. Assim, para cada referência

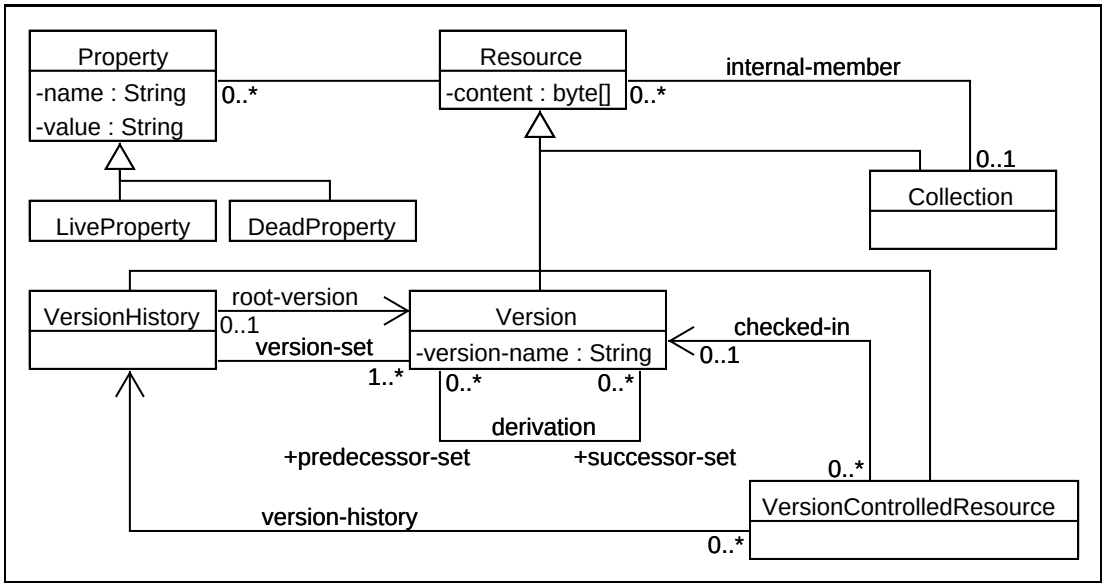


FIGURA 2.8 – Conceitos básicos do modelo de dados do WebDAV (I)

a um recurso multimídia, o navegador envia uma requisição GET ao servidor com o URI do recurso.

2.4.2 Introdução ao modelo de dados do WebDAV

Introduzido o funcionamento geral do protocolo HTTP, o modelo de dados do WebDAV é descrito. O WebDAV foi especificado em dois documentos, um para autoria distribuída [GOL 99a] e outro para controle de versões [CLE 2002].

O primeiro provê características como propriedades e bloqueios (*"lock"*) de arquivos para a escrita, além de métodos para manipular os arquivos (copiar, mover, apagar, etc.). As propriedades armazenam meta-dados sobre os arquivos. Os bloqueios evitam que um arquivo, em edição por um autor, seja atualizado simultaneamente por outro, o que faria com que as alterações fossem perdidas (sobrescritas). O segundo documento define conceitos como versão, histórico de versões, recurso versionado, etc., que serão detalhados a seguir.

Como o modelo de dados do WebDAV envolve muitos conceitos, eles foram divididos em dois diagramas de classes UML (*"Unified Modeling Language"*) [LAR 2000] (figuras 2.8 e 2.10). Além disso, são mostrados somente os conceitos do modelo de dados básico, pois existem as chamadas características avançadas de versionamento [CLE 2002] que definem outros conceitos. Além disso, os diagramas mostram apenas as propriedades mais importantes (aquelas que relacionam conceitos). Uma relação mais completa das propriedades dos conceitos é fornecida pelas tabelas 2.2 e 2.3 no final do capítulo. A seguir os conceitos da figura 2.8 são explicados.

2.4.3 Recursos e propriedades

Um recurso (`Resource`), como definido por [FIE 99], é um objeto de dados (*"data object"*) ou serviço de rede que pode ser identificado por um URI [BER 98]. É a entidade mais geral do modelo e pode receber métodos (comandos, ações).

Para fins práticos, o recurso pode ser visto como uma entrada de diretório (arquivo, diretório, *"link"*, etc.) em um sistema de arquivos. Similarmente a um arquivo, o recurso tem uma seqüência de bytes que forma seu conteúdo (atributo `content` na figura 2.8).

Recursos podem ter informações descritivas (meta-dados) associadas, dadas em forma de pares nome/valor. Cada um destes pares é chamado propriedade (`Property` [GOL 99a]) e é representado por documentos XML [BRA 2000, BRA 2002], podendo assim ser (semi-)estruturado. As propriedades podem ser classificadas em:

- propriedade viva (`LiveProperty`) - é aquela que tem sua sintaxe e semântica imposta (reforçada) pelo servidor, ou seja o servidor conhece a propriedade e é capaz de validá-la. O conjunto de propriedades vivas pode ser diferente dependendo do tipo do recurso;

- propriedade morta (`DeadProperty`) - é aquela que o servidor apenas armazena seu nome e valor, e a verificação de sua consistência é de responsabilidade do cliente.

Por exemplo, a propriedade viva *"getcontentlength"* [GOL 99a] representa o tamanho de um recurso (número de bytes de seu conteúdo), o que é calculado automaticamente pelo servidor. Não há uma forma direta de definir o valor desta propriedade, sendo possível sua modificação somente alterando-se o conteúdo do recurso.

Um usuário pode definir suas próprias propriedades para um recurso. Por exemplo, uma propriedade "aprovado" poderia assumir os valores "sim" e "não", conforme o recurso fosse ou não aprovado para publicação, respectivamente. Como um servidor comum não é projetado para tratar essa propriedade, ele não é capaz de validá-la, logo essa responsabilidade fica a cargo do autor. Assim, todas as propriedades criadas pelo usuário são propriedades mortas.

As propriedades mortas e o conteúdo formam o estado de um recurso. Cada vez que o conteúdo ou valor de uma propriedade morta é atualizado, ou quando uma propriedade é criada (definida) ou removida, o estado do recurso muda. Esse conceito de estado do recurso é importante ao controle de versões, como será discutido adiante.

2.4.4 Coleções

Análoga à forma como os arquivos são organizados em diretórios em um sistema de arquivos, o modelo do WebDAV organiza os recursos em coleções. Portanto, uma coleção é um recurso que "contém" outros recursos.

Formalmente, uma coleção (`Collection` [GOL 99a]) é um recurso cujo estado consiste de pelo menos uma lista de URIs dos seus membros internos, além de seu conteúdo e propriedades. Todos os recursos são membros internos de alguma coleção (associação `internal-member`); a exceção a essa regra é a coleção raiz ("/") da hierarquia de coleções (exatamente igual a uma árvore de diretórios).

2.4.5 Recursos versionados e versões

Tanto recursos comuns quanto coleções têm estados transitórios, pois quando são modificados o estado anterior é perdido. Diferente destes tipos de recursos, um

recurso versionado (`VersionControlledResource` [CLE 2002]) é um recurso que tem todos os seus estados armazenados pelo servidor, similar ao conceito de objeto versionado visto na subseção 2.1.1.

Cada estado de um recurso versionado é chamado de versão, e é ele próprio um recurso. Assim, um recurso versão (`Version` [CLE 2002]) contém uma cópia de um estado (propriedades mortas e conteúdo) particular de um recurso versionado. Cada versão tem uma propriedade viva *"version-name"* (mostrada como o atributo `version-name` na figura 2.8), que é uma string definida pelo servidor para diferenciar as versões do recurso versionado. O recurso versionado tem uma propriedade viva *"checked-in"* (associação `checked-in`) que refere-se a sua versão corrente.

2.4.6 Histórico de versões

Todas as versões de um mesmo recurso versionado são armazenadas em um histórico de versões. Portanto, o recurso histórico de versões (`VersionHistory` [CLE 2002]) é um recurso que contém todas as versões de determinado recurso versionado. Essa relação composto/componente é realizada pela propriedade viva *"version-set"* (associação `version-set`) do histórico de versões, que aponta para todas as versões do recurso versionado. O histórico de versões tem ainda uma propriedade *"root-version"* (associação `root-version`) que indica a primeira versão de um recurso versionado.

Os históricos de versões de todos os recursos versionados são colocados em um espaço de nomes reservado (coleção). Ou seja, eles não ficam "misturados" com os outros recursos. Por sua vez, as versões também estão isoladas, pois ficam dentro (como em uma coleção) de um histórico. Por exemplo, suponha-se que o espaço de nomes para históricos seja "/his/". Assim, os históricos criados para recursos versionados seriam "/his/1", "/his/14", etc. e suas respectivas versões seriam "/his/1/ver/1", "/his/1/ver/6", etc. e "/his/14/ver/3", "/his/14/ver/8", etc.

Apesar de as versões estarem armazenadas de forma plana (sem hierarquia) dentro de um histórico de versões, elas são logicamente organizadas em forma de grafo. O grafo é modelado pelas propriedades vivas *"predecessor-set"* e *"successor-set"* das versões (associação `derivation`); essas propriedades referem-se a suas versões predecessoras e sucessoras, respectivamente. Embora o WebDAV modele versões de forma a suportar uma estrutura de grafo, ele obriga o servidor a suportar apenas uma estrutura de lista, ou seja o suporte a grafos é opcional.

Um recurso versionado tem uma propriedade viva *"version-history"* (associação `version-history`) que aponta para seu histórico de versões. Uma vez que podem existir vários recursos versionados para o mesmo histórico de versões, cada um deles pode ter o mesmo estado (estar ligado à mesma versão pela associação `checked-in`). Naturalmente, é possível que cada recurso versionado do mesmo histórico tenha uma versão corrente diferente, escolhida pelo usuário.

2.4.7 Edição de recursos versionados

A figura 2.9, inspirada em diagramas de seqüência da UML, mostra a edição de um recurso versionado usando uma seqüência típica de operações. Primeiramente o recurso versionado ("/publication.html") recebe um método CHECK-OUT [CLE 2002], que habilita-o a ser modificado e substitui sua propriedade *"checked-in"* e por uma *"checked-out"* com o mesmo valor. Um autor pode então

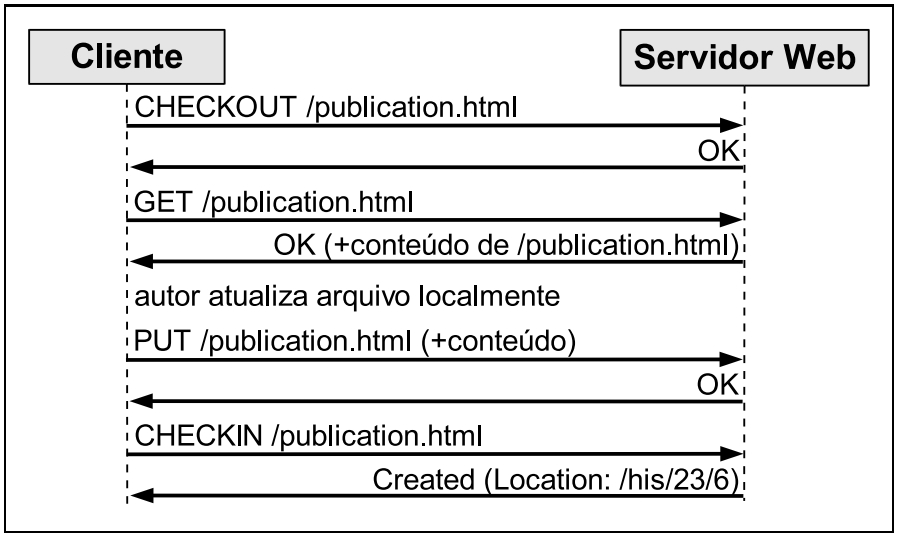


FIGURA 2.9 – Seqüência de operações para edição de um recurso versionado

obter o conteúdo e as propriedades do recurso versionado, a fim de modificá-los, usando os métodos GET e PROPFIND, respectivamente. No exemplo, apenas o conteúdo é modificado.

Após editar o conteúdo e/ou propriedades do recurso versionado localmente, o autor deve enviar as modificações ao servidor. Os métodos PUT e PROPPATCH são usados para atualizar o conteúdo e as propriedades de um recurso, respectivamente. PUT também é usado para criar novos recursos simples (não versionados e nem coleções).

Para finalizar o processo de edição, o autor envia um método CHECKIN. Em resposta a esse método, uma nova versão é criada com o estado atualizado do objeto versionado; o URI da nova versão é informado pelo cabeçalho *"Location"* da mensagem de resposta. A nova versão é derivada da versão apontada pela propriedade *"checked-out"* (antiga versão corrente); esta propriedade é novamente substituída pela *"checked-in"*, mas seu valor passa a ser o URI da nova versão.

2.4.8 Recursos *"checkedout"*

Quando um recurso tem a propriedade *"checked-out"*, ele é chamado de recurso *"checkedout"* (`CheckedoutResource` [CLE 2002], na figura 2.10). Um recurso *"checkedout"* é o que pode ser editado (modificado), e que criará uma nova versão após a edição ser concluída. Sua propriedade *"checked-out"* indica qual versão está sendo usada como base para a edição.

A especificação do WebDAV [CLE 2002] define que um recurso pode ter dois tipos (por exemplo, recurso versionado e *"checkedout"*). Entretanto, o tipo de recurso *"checkedout"* é ganho ou perdido (efeito dos métodos CHECKOUT e CHECKIN, respectivamente), o que não é comum em modelos usuais.

2.4.9 Recursos em trabalho

A seqüência de edição apresentada na subseção 2.4.7 também pode ser usada para editar versões específicas, não só a corrente (recurso versionado). Neste caso, o método CHECKOUT aplicado sobre uma versão cria um recurso em traba-

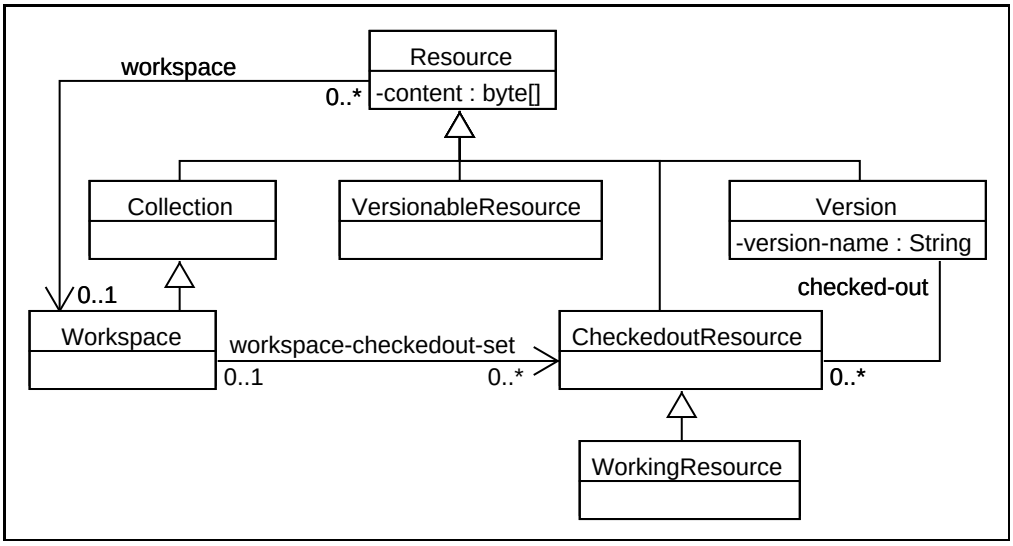


FIGURA 2.10 – Conceitos básicos do modelo de dados do WebDAV (II)

lho (`WorkingResource` [CLE 2002]) com uma cópia do estado da versão. O recurso em trabalho serve como um repositório temporário para as alterações do autor, uma vez que versões não são atualizáveis.

De forma similar aos históricos, o recurso em trabalho também é criado em um espaço de nomes reservado ("/wr/", por exemplo). Por isso, junto à resposta ao método CHECKOUT, o servidor envia um cabeçalho *"Location"* indicando o URI do recurso em trabalho criado. Assim, é possível ao autor editar (alterar) o recurso em trabalho. Ao final da edição, o recurso em trabalho recebe um método CHECKIN e uma nova versão é criada com seu estado; o método CHECKIN ainda remove o recurso em trabalho.

2.4.10 *"Workspaces"*

Outra forma de criar versões é usando o conceito de *"workspace"*. O *"workspace"* (`Workspace` [CLE 2002]) é uma coleção que serve de espaço temporário para a edição de recursos versionados e não versionados que sejam relacionados. Isso também pode ser feito usando recursos em trabalho, entretanto é necessário que o cliente mantenha um controle sobre os recursos em trabalho. Usando *"workspaces"*, todo o controle de recursos em edição é feito pelo servidor (nenhuma informação é armazenada no cliente).

Um *"workspace"* tem uma propriedade viva *"workspace-checkedout-set"* (associação "`workspace-checkedout-set`" na figura 2.10) que referencia os recursos *"checkedout"* contidos no *"workspace"*. Os recursos que são membros de um *"workspace"*, mesmo que indiretamente (por estarem em uma coleção que está no *"workspace"*), têm a propriedade *"workspace"* (associação `workspace`) referenciando o respectivo *"workspace"*.

2.4.11 Recurso versionável

O WebDAV ainda especifica um recurso versionável (`VersionableResource` [CLE 2002]), que é um recurso não versionado que pode ser colocado sob o controle de versões. Em outras palavras, o recurso

TABELA 2.2 – Principais métodos definidos pelo protocolo WebDAV

Método	**Efeito**	**Definição original**
CHECKIN	cria uma nova versão com o estado de um recurso *checkedout*	WebDAVe
CHECKOUT	prepara um recurso para ser modificado, criando um recurso *checkedout*	WebDAVe
COPY	copia um recurso ou árvore de recursos	WebDAV
DELETE	remove um recurso ou árvore de recursos	HTTP
GET	obtém o conteúdo de um recurso e alguns metadados simples	HTTP
LOCK	bloqueia para escrita um recurso ou árvore de recursos	WebDAV
MKCOL	cria uma nova coleção	WebDAV
MOVE	move um recurso ou árvore de recursos para uma nova posição na hierarquia de coleções	WebDAV
PROPFIND	retorna propriedades de um recurso ou árvore de recursos	WebDAV
PROPPATCH	define ou remove propriedades do recurso	WebDAV
PUT	modifica um recurso existente ou cria um novo recurso	HTTP
REPORT	obtém informações sobre um recurso	WebDAVe
UNCHECKOUT	desfaz uma operação de CHECKOUT	WebDAVe
UNLOCK	desbloqueia para escrita um recurso ou árvore de recursos	WebDAV
UPDATE	muda o estado de um recurso versionado para o de uma outra versão (muda a versão corrente)	WebDAVe
VERSION-CONTROL	transforma um recurso versionável em versionado ou cria em um *workspace* um novo recurso versionado para um histórico existente	WebDAVe

versionável é um recurso que pode ser transformado em um recurso versionado através da aplicação do método VERSION-CONTROL [CLE 2002]. Neste caso, um histórico de versões é criado e a primeira versão assume o estado do recurso versionável.

2.4.12 Principais métodos e propriedades do WebDAV

A tabela 2.2 resume os principais métodos definidos pelo protocolo WebDAV. Para cada método existe uma descrição curta de seu efeito e a indicação de qual protocolo o definiu originalmente: HTTP [FIE 99], WebDAV (original) [GOL 99a] e WebDAVe (WebDAV estendido com controle de versões) [CLE 2002].

As tabelas 2.3 e 2.4 resumem as principais propriedades dos recursos básicos do WebDAV. Para cada tipo de recurso básico, são fornecidas uma lista de propriedades e uma descrição curta. As propriedades são descritas apenas para o recurso mais geral da hierarquia de conceitos (conforme figuras 2.8 e 2.10), não sendo repetidas para os recursos especializados.

TABELA 2.3 – Principais propriedades do modelo do WebDAV

Tipo de recurso	**Propriedade**	**Descrição**
Resource (WebDAV original)	creationdate	data e hora em que o recurso foi criado
	displayname	nome do recurso para apresentação ao usuário
	getcontentlanguage	idioma do conteúdo do recurso
	getcontentlength	tamanho (em bytes) do conteúdo do recurso
	getcontenttype	tipo de dados do conteúdo do recurso (imagem, texto, etc.)
	getlastmodified	data e hora da última modificação do recurso (inclusive em propriedades mortas)
	lockdiscovery	descrição dos bloqueios (*locks*) ativos sobre o recurso
	resourcetype	tipo do recurso (coleção, *workspace*, etc.)
	source	recurso que contém o conteúdo não processado de um *link*, usado para obtenção de programas que geram recursos (*scripts*, CGIs, etc.)
	supportedlock	lista de tipos de bloqueio (*lock*) suportados pelo recurso
Resource	comment	comentário sobre o recurso para apresentação ao usuário
	creator-displayname	nome do criador do recurso para apresentação ao usuário
	supported-method-set	métodos que podem ser executados sobre o recurso em algum de seus estados
	supported-live-property-set	propriedades para as quais a semântica é entendida pelo servidor
	supported-report-set	relatórios que são suportados pelo recurso
	workspace	*workspace* no qual o recurso está (mesmo que indiretamente)
Version-Controlled Resource (*checked-in*)	auto-version	indica se operações de CHECKOUT/CHECKIN são executadas automaticamente quando uma alteração é tentada (útil para clientes que desconhecem controle de versões)
	checked-in	versão que tem o mesmo conteúdo e propriedades mortas
	version-history	histórico de versões do recurso
Checked-Out Resource	checked-out	versão identificada pela propriedade checked-in quando recurso sofreu CHECKOUT
	checkin-fork	valor da propriedade *checkin-fork* da versão criada pela operação CHECKIN
	checkout-fork	valor da propriedade *checkout-fork* da versão criada pela operação CHECKIN
	predecessor-set	versões que serão predecessoras da versão criada por CHECKIN
Version-History	root-version	versão raiz do grafo de versões
	version-set	versões do histórico de versões

TABELA 2.4 – Principais propriedades do modelo do WebDAV (cont.)

Tipo de recurso	**Propriedade**	**Descrição**
Version	checkedout-set	recursos *checkedout* com esta versão apontada pela propriedade *checked-out*
	checkin-fork	indica se a operação CHECKOUT é permitida se a versão já tem sucessora ou se já sofreu CHECK-OUT
	checkout-fork	indica se a operação CHECKIN é permitida se a versão já tem sucessora
	label-name-set	rótulos que selecionam a versão dentro do histórico de versões
	predecessor-set	versões predecessoras
	sucessor-set	versões sucessoras (as quais têm a propriedade *predecessor-set* indicando esta versão)
	version-history	histórico de versões que contém a versão
	version-name	string definida pelo servidor que diferencia versões do mesmo histórico
Working Resource	auto-update	recurso versionado que será atualizado (propriedade *checked-in* alterada para apontar para a nova versão) quando este recurso sofrer CHECKIN
Workspace	workspace-checkedout-set	recursos *checkedout* que estão no *workspace*

2.5 Conclusão

Este capítulo apresentou conceitos e mecanismos usados no controle de versões e na gerência de configurações. Alguns trabalhos sobre controle de versões de hiperdocumentos foram analisados. Apesar dos muitos recursos interessantes implementados por estes modelos, foi mostrado que eles não são capazes de controlar versões da formatação dos documentos.

Além disso, estes modelos possuem mecanismos limitados para a geração de configurações resolvidas. Portanto, foram analisados também alguns trabalhos da área de Gerência de Configuração de Software. Para a criação de configurações, estes trabalhos baseiam a escolha de versões em critérios definidos pelo usuário. Assim, é possível criar configurações resolvidas automaticamente.

O protocolo de rede WebDAV, que padroniza o controle de versões na Web, foi explicado. Esse protocolo possui um modelo de dados que apresenta muitos conceitos gerais relacionados ao controle de versões e gerência de configurações. Entretanto, é um modelo genérico e requer uma especialização ou extensão para adequar-se a determinados domínios de aplicação.

3 Infra-estrutura Proposta

Este trabalho propõe uma infra-estrutura para controle de versões e suporte a adaptação de páginas de acordo com seu contexto de requisição. Esta infra-estrutura foi projetada para ser implementada em um servidor Web, tornando o controle de versões e a adaptação transparentes aos visitantes do site.

A adaptação é feita através da configuração da página, ou seja um processo de configuração resolve referências dinâmicas escolhendo versões específicas de cada componente da página. Critérios de seleção de versões são especificados por autores de páginas e pelo próprio servidor Web, guiando as escolhas do processo de configuração.

O servidor Web também fornece ao processo de configuração informações sobre o contexto da requisição da página. Essas informações podem ser usadas na especificação dos critérios de seleção de versões, podendo ajudar o processo de configuração na escolha das versões mais adequadas ao visitante.

Portanto, a infra-estrutura proposta consiste de:

- um modelo de versões para as páginas Web e seus componentes, que modela inclusive os critérios de seleção de versões necessários ao processo de configuração;
- um processo de configuração automático que, baseado em critérios de configuração, pode ser usado para adaptar páginas Web.

O modelo de versões proposto é uma extensão ao modelo de dados do WebDAV, o que permite padronizar os termos e conceitos básicos utilizados. Assim, todos os conceitos do modelo proposto são recursos ou participam do estado dos recursos. Uma vez que existirão produtos implementando o padrão WebDAV, seu uso como base do modelo proposto facilita a implementação deste. Por outro lado, o uso de um modelo existente impõe algumas restrições ao novo modelo, como será comentado no decorrer deste capítulo.

Este capítulo apresenta a infra-estrutura proposta por este trabalho. A seção 3.1 apresenta uma visão geral do funcionamento da infra-estrutura proposta. A seção 3.2 apresenta detalhadamente o modelo de versões para as páginas Web. O processo de configuração é explicado na seção 3.3. A seção 3.4 conclui o capítulo.

3.1 Visão geral da infra-estrutura proposta

Esta seção apresenta uma visão geral de como a infra-estrutura proposta funciona. Os conceitos principais do modelo de versões e o funcionamento do processo de configuração são introduzidos de maneira informal, através de exemplos. Os exemplos são ilustrados por figuras que mostram os passos no desenvolvimento de uma página Web de um site de notícias. Em cada figura, os elementos (versões, recursos, etc.) que não mudam em relação aos exemplos anteriores são omitidos.

Nos exemplos a seguir, os seguintes aspectos da página são adaptados:

- idioma do conteúdo - existindo duas alternativas, conteúdo em inglês ou em português;

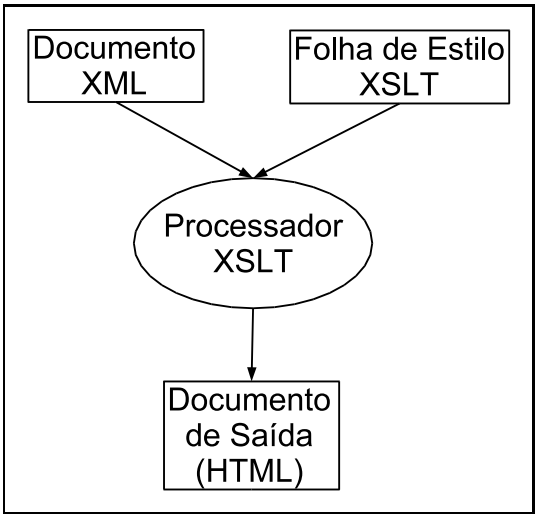


FIGURA 3.1 – Aplicando uma transformação XSLT a um documento XML

- formato da página resultante - existindo duas alternativas, formato WML (*"Wireless Markup Language"*) [OPE 2002] para telefones celulares ou formato HTML para navegadores comuns.

A tradução de documentos XML para o formato HTML usando XSLT é uma forma comum de apresentar documentos XML em um navegador Web [KAY 2000]. Essa tradução é mostrada na figura 3.1, onde um processo chamado "processador XSLT" recebe como entrada pelo menos um documento XML e uma folha de estilo XSLT. Usando as regras de transformação contidas na folha de estilo, o processador transforma o documento XML em algum formato de saída (HTML na figura). Portanto, uma página HTML pode ser gerada a partir de (ao menos) dois elementos: um documento XML e uma folha de estilo XSLT.

3.1.1 Página

O modelo de versões proposto segue o conceito de objeto versionado composto, que é ligado aos componentes através de referências, de forma similar ao objeto composto explicado na subseção 2.1.4. Uma *página* é o recurso (na terminologia do WebDAV) raiz da hierarquia de composição e tem duas referências: uma para o componente de conteúdo XML e outro para o componente de formatação XSLT.

Portanto, uma página, como modelada por este trabalho, representa a descrição de uma página Web, ou seja a visão que um desenvolvedor tem de uma página Web e não a visão de um visitante. Tanto a página quanto seus componentes são recursos versionados.

A figura 3.2 mostra um exemplo de uma página de notícia e seus dois componentes. A notação da figura é a mesma usada no capítulo 2; no entanto, o fundo dos recursos versionados é preenchido com um padrão diferente dependendo do tipo de recurso (página, conteúdo ou formatação). Pode-se notar também que o recurso histórico de versões não é representado explicitamente, permitindo uma visualização mais clara.

Apenas algumas propriedades das versões, necessárias aos exemplos, são mostradas na figura 3.2 (e seguintes). Na lista de propriedades, as versões de um recurso versionado são representadas pelo nome do recurso versionado e pelo número

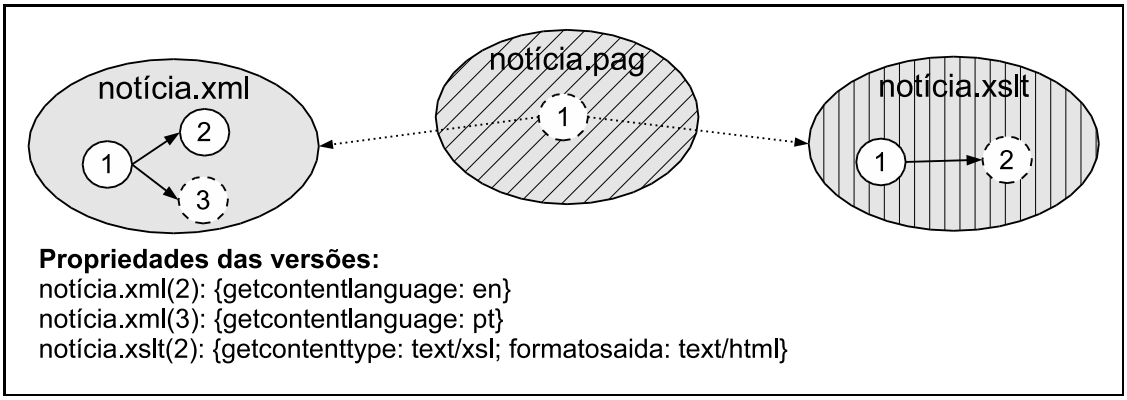


FIGURA 3.2 – Exemplo de uma página e seus componentes

da versão; por exemplo, a versão 2 do recurso versionado `notícia.xml` é denotada por `notícia.xml(2)`. Essa notação também será usada durante o restante deste trabalho. Outros detalhes da notação usada serão explicados à medida que forem necessários.

3.1.2 Referências estáticas e dinâmicas

O modelo suporta *referências estáticas* que apontam para qualquer tipo de recurso, mas que não são processadas pelo processo de configuração, ou seja não são resolvidas. As *referências dinâmicas* podem apontar apenas para recursos versionados. Como a configuração (e adaptação, potencialmente) automática de páginas é feita através da resolução de referências dinâmicas, estas são normalmente preferidas.

Na figura 3.2, referências dinâmicas são usadas para ligar a versão 1 da página `notícia.pag` a seus componentes de conteúdo (`notícia.xml`) e de formatação (`notícia.xslt`). Algumas propriedades das versões também são mostradas. As propriedades *"getcontentlanguage"* e *"getcontenttype"* são definidas pelo WebDAV (ver tabela 2.3). O texto da notícia (conteúdo) nas versões 2 e 3 de `notícia.xml` está escrito em inglês (en) e português (pt), respectivamente. Neste exemplo, a propriedade definida pelo autor *"formatosaida"* indica o formato (HTML no caso de `notícia.xslt(2)`) para o qual a folha de estilo da formatação transforma o documento XML de conteúdo.

3.1.3 Processo de configuração e critérios

Quando um visitante requisita uma página, o processo de configuração é executado para resolver as referências dinâmicas e gerar uma configuração resolvida (página na visão do visitante), como mostra a figura 3.3. Primeiramente, o visitante requisita uma página ao servidor Web (passo 1), que repassa a requisição ao processo de configuração.

A partir da página, o processo de configuração obtém as referências aos componentes de conteúdo e formatação (passo 2). O processo obtém o conteúdo dos recursos de conteúdo e formatação, resolvendo referências dinâmicas se necessário. Em seguida o processo utiliza o processador XSLT para transformar o conteúdo XML em uma página HTML para o visitante (passo 3). Então a página gerada na

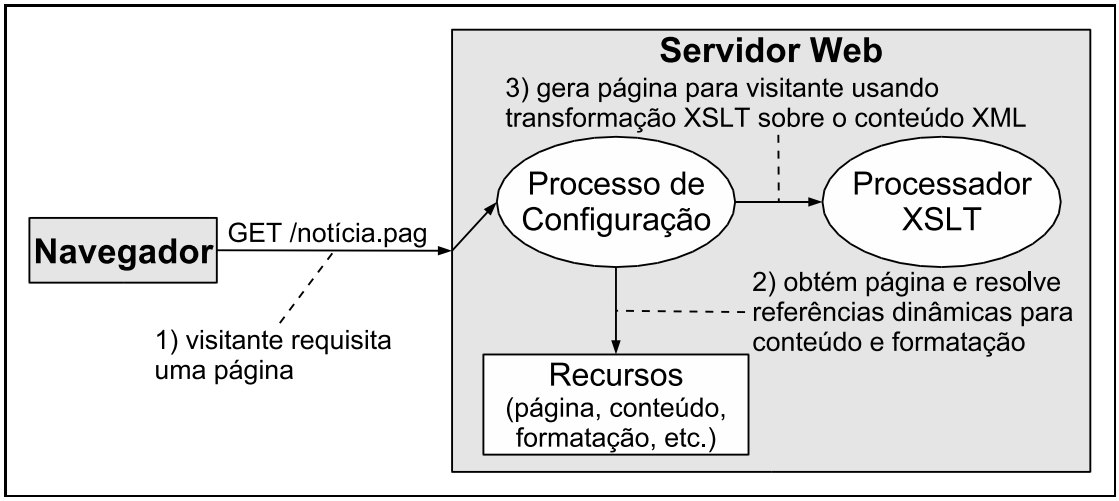


FIGURA 3.3 – Configuração de uma página para um visitante

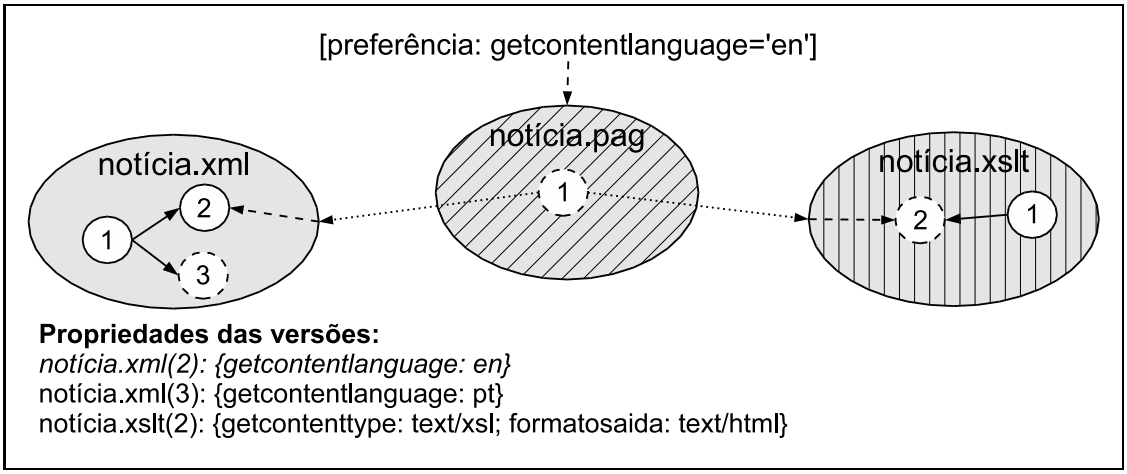


FIGURA 3.4 – Processo de configuração resolvendo referências dinâmicas

saída do processador XSLT é enviada ao visitante.

Para resolver as referências dinâmicas automaticamente, o processo de configuração utiliza critérios de seleção de versões (de agora em diante chamados apenas critérios), como explicado na subseção 2.1.6. Os critérios podem ser definidos pelo autor, sendo armazenados e formando a base de critérios. Eles também podem ser gerados dinamicamente pelo servidor Web, formando os *critérios parâmetro*, ou seja os critérios da descrição de configuração. Se não houver critérios para resolver uma referência dinâmica, ou se mais de uma versão candidata atende aos critérios existentes, o processo seleciona a versão mais recente (esse é o critério *"default"*).

A figura 3.4 mostra o processo de configuração resolvendo referências dinâmicas (passo 2 da figura 3.3). Quando a página é requisitada pelo visitante, o processo de configuração recebe o URI da página a ser configurada e os critérios gerados pelo servidor Web, ou seja uma descrição de configuração. Isso é representado pela seta tracejada no topo da figura 3.4. Neste exemplo, a página requisitada é `notícia.pag` e o critério gerado indica uma preferência por versões em idioma inglês (o prefixo "`preferência:`" indica que é um critério opcional, conforme subseção 2.1.6).

Caso o URI da página indique um recurso versionado, o processo de configu-

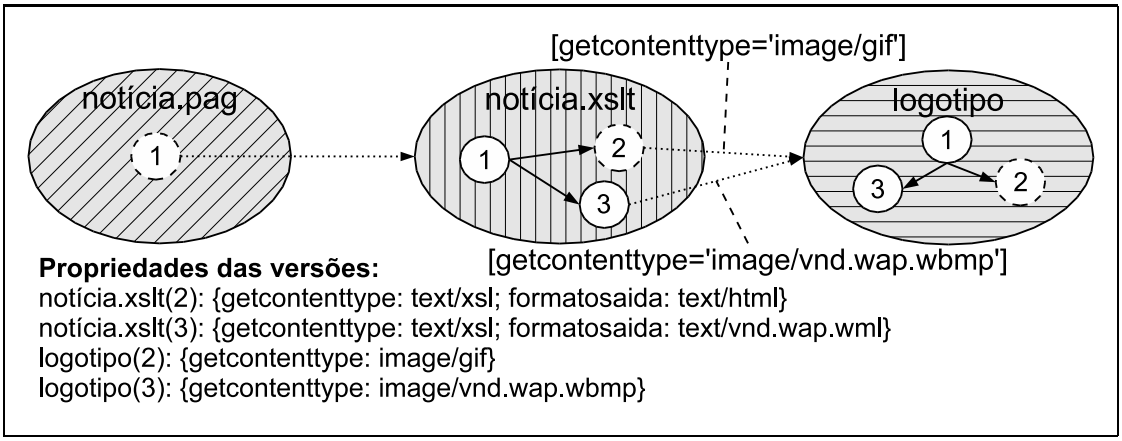


FIGURA 3.5 – Critérios locais e referências entre componentes

ração utiliza a versão corrente como ponto de partida (no caso da figura 3.4, é a única). Em seguida, o processo obtém o componente de conteúdo referenciado pela página, resolvendo a referência se ela for dinâmica. Neste exemplo, `notícia.xml` representa o conteúdo da página e, como a versão 2 é a única que está em inglês, ela é selecionada (a seta tracejada indica que a referência para o recurso versionado foi resolvida para a versão).

O processo também obtém a formatação da página, resolvendo a referência dinâmica para `notícia.xslt`. Como nenhuma versão de `notícia.xslt` tem uma propriedade *"getcontentlanguage"* com valor "en", a versão 2 é selecionada usando o critério *"default"*. Com estes componentes da página, e usando o processador XSLT, o processo de configuração gera a página configurada para o visitante. Destaca-se que a página gerada (configuração resolvida) não é armazenada pelo servidor Web, apenas enviada ao visitante.

De maneira similar à adaptação de idioma do conteúdo da página, exemplificada na figura 3.4, pode-se adaptar também a formatação da página usando o processo de configuração. Dependendo do dispositivo e/ou aplicativo usado para visualizar as páginas (ou ainda de outros critérios), uma versão adequada da formatação pode ser selecionada. Por exemplo, duas versões diferentes podem formatar a página para navegadores diferentes ou ainda para um mini-navegador WAP (*"Wireless Application Protocol"*) [OPE 2002] de um telefone celular.

A figura 3.5 mostra uma nova versão (3) de `notícia.xslt` que gera WML, o que é indicado pela propriedade *"formatosaida"*. WML é o formato de páginas reconhecido por telefones celulares. Por ora, assume-se que o servidor Web gere critérios para selecionar a versão apropriada do componente de formatação (`notícia.xslt`), de forma similar àquele gerado para a adaptação do idioma da página.

Também foi criado um recurso versionado `logotipo`, que é uma figura contendo o logotipo do site. A versão `logotipo(3)` está no formato WBMP (*"WAP bitmap"*), o formato de imagem reconhecido por telefones celulares. Por sua vez, `logotipo(2)` está no formato GIF (*"Graphics Interchange Format"*), que é aceito por navegadores Web comuns.

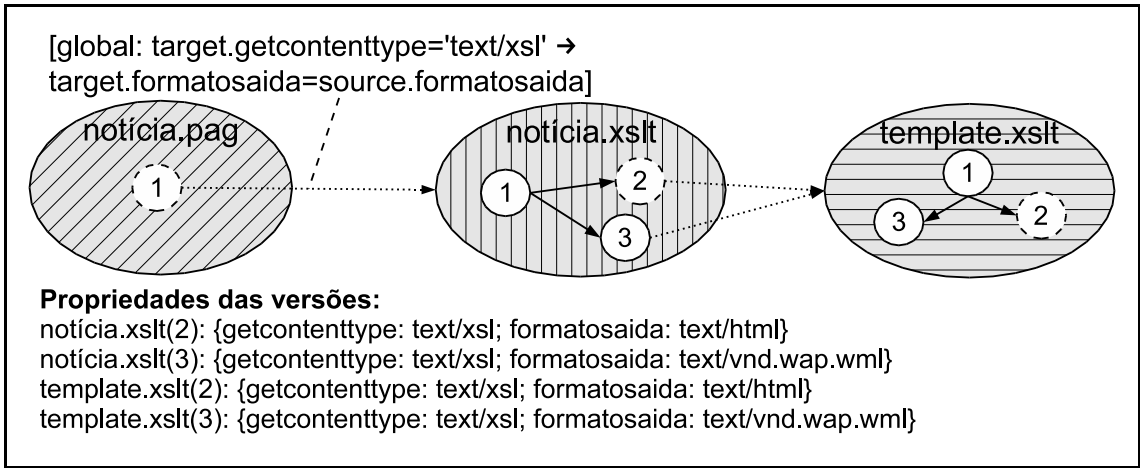


FIGURA 3.6 – Critérios globais e referências entre componentes

3.1.4 Referências em componentes e critérios locais

Tanto os componentes de formatação quanto os de conteúdo podem ter referências para outros recursos. Por exemplo, as versões 2 e 3 de `notícia.xslt` têm uma referência para o componente `logotipo` (figura 3.5). Para selecionar versões adequadas do logotipo, critérios definidos pelo autor são associados a essas referências, e por essa razão são chamados de *critérios locais*. Critérios globais serão abordados mais adiante.

Por exemplo, considere-se `notícia.xslt(3)`, que gera formatação WML. Sua referência dinâmica para `logotipo` possui um critério local, exigindo que o conteúdo da versão selecionada esteja no formato WBMP. Note-se que o critério é obrigatório, pois não tem o prefixo "`preferência:`"; assim, ele representa uma restrição de integridade, pois telefones celulares aceitam apenas páginas WML e figuras WBMP.

Neste caso, uma referência estática para a versão correspondente poderia ter sido usada. Entretanto, usando-se uma referência dinâmica, pode-se selecionar automaticamente uma versão melhorada do logotipo quando ela for criada. O critério de seleção para a referência de `notícia.xslt(2)` é análogo ao de `notícia.xslt(3)`.

3.1.5 Critérios globais

A figura 3.6 mostra um novo recurso versionado `template.xslt`, que representa o padrão de formatação de todas as páginas. Assim, todas as páginas devem utilizá-lo, a fim de manter a identidade visual do site. Por isso, as versões de `notícia.xslt` referenciam-no. As versões 2 e 3 de `template.xslt` geram HTML e WML, respectivamente.

Para expressar as combinações possíveis entre as versões de `notícia.xslt` e `template.xslt`, critérios locais poderiam ser utilizados, como na figura 3.5. Neste caso, entretanto, é necessário especificar critérios para cada referência originada em versões de `notícias.xslt` (como na figura 3.5). Para evitar isso, pode-se empregar critérios globais. Os *critérios globais* são associados à página, mas são usados pelo processo de configuração para resolver todas as referências das versões selecionadas.

Existem conjuntos distintos de critérios globais para formatação e para conteúdo. Na figura 3.6, um critério global para formatação aparece associado à referência da página para o componente de formatação. Apesar disso, o prefixo "`global:`"

indica que trata-se de um critério global.

Assim, é necessário especificar os critérios globais de maneira geral, pois eles devem ser aplicáveis a todos os componentes (de formatação, neste caso). Se um critério global (representando uma restrição) for especificado de forma que não selecione nenhuma versão de um recurso versionado, a página não poderá ser configurada. Neste caso, o processo de configuração deve indicar um erro. Para o exemplo da figura 3.6, a regra geral de consistência para combinar versões dos componentes de formatação poderia ser "o formato gerado pelas versões origem e destino da referência deve ser o mesmo".

Note-se a necessidade de denotar os recursos origem e destino da referência, o que é dependente da linguagem de especificação de critérios. Entretanto, será usada uma sintaxe (comum a várias linguagens) em que o papel do recurso precede o nome de sua propriedade, e é separado desta por um ponto (.). Por exemplo, o critério global supracitado seria: `target.formatosaida=source.formatosaida`.

Deve-se considerar ainda o fato de que as versões de `notícia.xslt` têm referências para recursos que não são folhas de estilo XSLT, como por exemplo o `logotipo` (figura 3.5). Neste caso, nenhuma das versões de `logotipo` obedece à restrição expressa pelo critério global, pois elas não têm uma propriedade *formatosaida*. Portanto, esse critério deve ser aplicado apenas quando a versão destino da referência for uma folha de estilo XSLT; isso pode ser expresso pelo bem conhecido operador lógico de implicação ($\rightarrow$).

O critério global na figura 3.6 expressa a restrição sobre as versões de formatação usando o operador de implicação. Este critério pode ser lido como "se a versão destino da referência for uma folha de estilo XSLT, então ela deve gerar o mesmo formato de saída da folha de estilo (versão) origem da referência".

3.1.6 Propriedades do ambiente

A figura 3.4 mostra um exemplo onde o servidor Web gera critérios globais para adaptar o conteúdo da página. Além de critérios globais, o servidor Web pode fornecer ao processo de configuração informações sobre o contexto da requisição da página. Essas informações, chamadas *propriedades do ambiente*, podem ser usadas na especificação de critérios, conforme explicado a seguir.

As propriedades do ambiente podem ser extraídas dos cabeçalhos da requisição do cliente, como por exemplo o idioma preferido do usuário e o dispositivo utilizado (cabeçalhos *"Accept-Language"* e *"User-Agent"* da requisição HTTP, conforme figura 2.7). Se o servidor Web conhece características do usuário, ele pode ainda extrair propriedades das preferências particulares do usuário. Por exemplo, o servidor Web pode gerar propriedades do ambiente indicando a cor preferida do usuário ou se este prefere notícias resumidas ou completas.

A figura 3.7 mostra um novo recurso versionado `grafbarras` que é uma figura de um gráfico de barras mostrado na página da notícia. Esse gráfico, como o `logotipo`, tem versões específicas tanto para telefones celulares quando para navegadores Web. As versões de `notícia.xml` referenciam-no, pois o gráfico faz parte do conteúdo e não da formatação da notícia.

Nesta situação, surge uma dificuldade na especificação de critérios que garantam um conteúdo consistente para a página, pois não é possível especificar critérios como os da figura 3.5. Nesta figura, os critérios selecionam versões da figura (logotipo) que têm formato compatível com a respectiva versão da formatação (formato

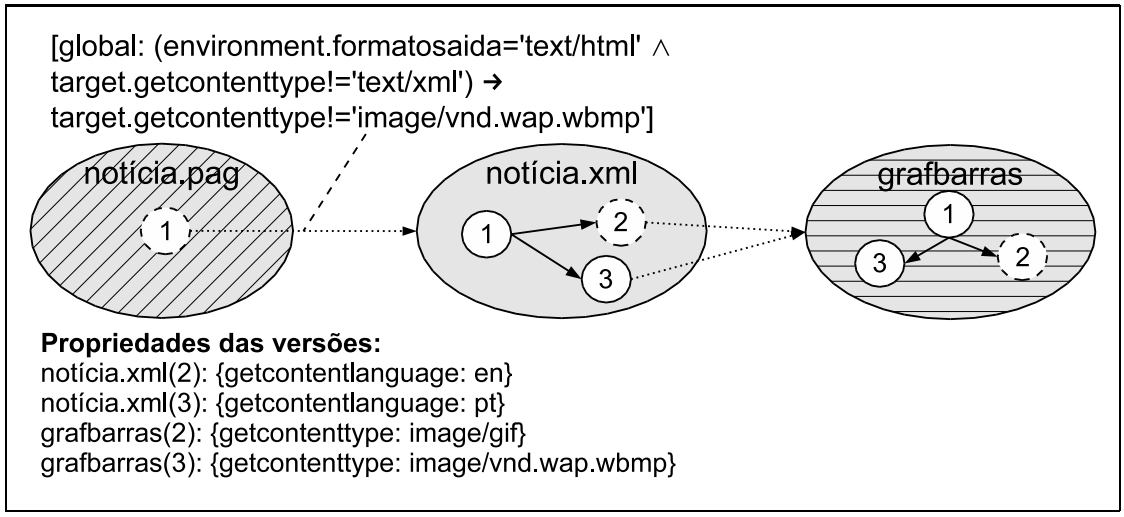


FIGURA 3.7 – Critérios usando propriedades do ambiente

gerado pela formatação). Entretanto, na figura 3.7, a figura (gráfico) é referenciada pelo componente de conteúdo, que é independente do dispositivo do usuário. Portanto, o conteúdo não tem um formato de saída definido.

Uma forma de contornar essa dificuldade é usar as propriedades do ambiente. Nos exemplos seguintes, essas propriedades são denotadas como se fossem de um recurso especial (`environment`). A figura 3.7 mostra um critério global para o conteúdo, que utiliza a propriedade do ambiente *"formatosaida"*.

Neste critério, a expressão `environment.formatosaida` indica o formato de saída requerido pelo ambiente, ou seja pelo dispositivo do usuário. Assim, devido ao operador de implicação, a restrição aplica-se apenas quando o dispositivo do usuário for um navegador Web e a versão destino da referência não for um documento XML (e sim uma imagem). Neste caso, o formato da imagem (versão) referenciada não pode ser WBMP, mas sim GIF ou outro formato reconhecido por navegadores Web.

As propriedades do ambiente também podem ser usadas para especificar preferências na escolha das versões. Logo, um autor pode especificar preferências destinadas à adaptação das páginas, caso o servidor Web não gere critérios para realizar algum tipo de adaptação. Na figura 3.4, por exemplo, se o servidor Web não pudesse gerar o critério (`getcontentlanguage='en'`), o autor poderia usar propriedades do ambiente (`getcontentlanguage`) para especificar uma preferência equivalente (`target.getcontentlanguage=environment.getcontentlanguage`).

3.1.7 Conclusão

Resumindo, a configuração (e adaptação, potencialmente) das páginas é feita através da escolha de versões dos componentes das páginas (resolução de referências dinâmicas). Uma página tem duas referências para seus componentes principais (diretos) de conteúdo e formatação. Por sua vez, estes componentes podem referenciar outros componentes, formando uma hierarquia de composição (e dependência).

Um autor pode especificar critérios locais às referências dinâmicas, que são usados para resolver apenas a referência à qual estão associados. Existem ainda critérios globais, usados para resolver todas as referências dinâmicas (em qualquer

nível da hierarquia de composição da página). Os critérios globais são associados à página e dividem-se em dois conjuntos, destinados a resolver referências originadas em componentes de conteúdo e formatação, respectivamente.

Os critérios expressam restrições ou preferências para a escolha de versões; eles são especificados pelo relacionamento das propriedades das versões e das propriedades do ambiente (geradas pelo servidor Web). O processo de configuração resolve uma referência dinâmica baseado nos critérios, tanto locais quanto globais, presentes no contexto da referência. Os critérios que representam restrições são avaliados primeiramente; em seguida, são avaliados os critérios que representam preferências. Assim, a ordem de avaliação dos critérios independe de eles serem locais ou globais.

3.2 Modelo de versões de páginas

Esta seção apresenta o modelo de versões de páginas de forma mais detalhada e exata do que a seção anterior. Para tanto, diagramas de classes da UML [LAR 2000] são usados para modelar as associações entre os conceitos do modelo. Em todos estes diagramas, os conceitos sombreados são os do modelo de dados do WebDAV.

Note-se que, no modelo de dados do WebDAV, o conceito que o servidor Web manipula é o recurso. Portanto, toda a informação existente no modelo participa do estado de um recurso (como conteúdo ou propriedade). Para modelar essa característica do WebDAV, os diagramas de classes desta seção usam associações de composição. Assim, uma composição indica que o conceito componente participa do estado do composto.

Como pode ser visto nestes diagramas, o conceito componente nunca é um tipo especial de recurso; por outro lado, o conceito composto sempre é um recurso ou um componente de um recurso. Isso pode ser observado na figura 3.8, que mostra um diagrama de classes UML modelando todos conceitos do modelo proposto. Além deste diagrama completo, diagramas parciais são apresentados conforme os conceitos do modelo são explicados.

Todos os conceitos do modelo proposto (que são recursos) estendem o recurso versão do WebDAV. Como o objeto versionado assume o estado de alguma versão, as características das versões estão também presentes no objeto versionado. Por isso, o conceito de versão é um ponto adequado do modelo para aplicar uma extensão. Outros modelos de versões (HyperPro [ØST 92], CoVer [HAA 92] e GDOC [NOR 98, NOR 98a]) também especializam o conceito versão a fim de atribuir funções específicas a um objeto versionado.

3.2.1 Fragmentos de conteúdo e formatação

O conteúdo e a formatação das páginas são chamados genericamente de fragmentos, como mostra a figura 3.9. Um *fragmento* (`Fragment`) é um recurso que pode ser um componente direto de uma página. O fragmento pode ser tanto um recurso completo quanto um recurso incompleto, que requer outros fragmentos para compor um recurso completo; conseqüentemente, ele também pode ser incluído em outro(s) fragmento(s).

Por exemplo, um fragmento de conteúdo pode conter um documento XML completo ou requerer a inclusão de outro(s) fragmento(s) para formar um documento completo. Isso é normalmente implementado por referências a entidades

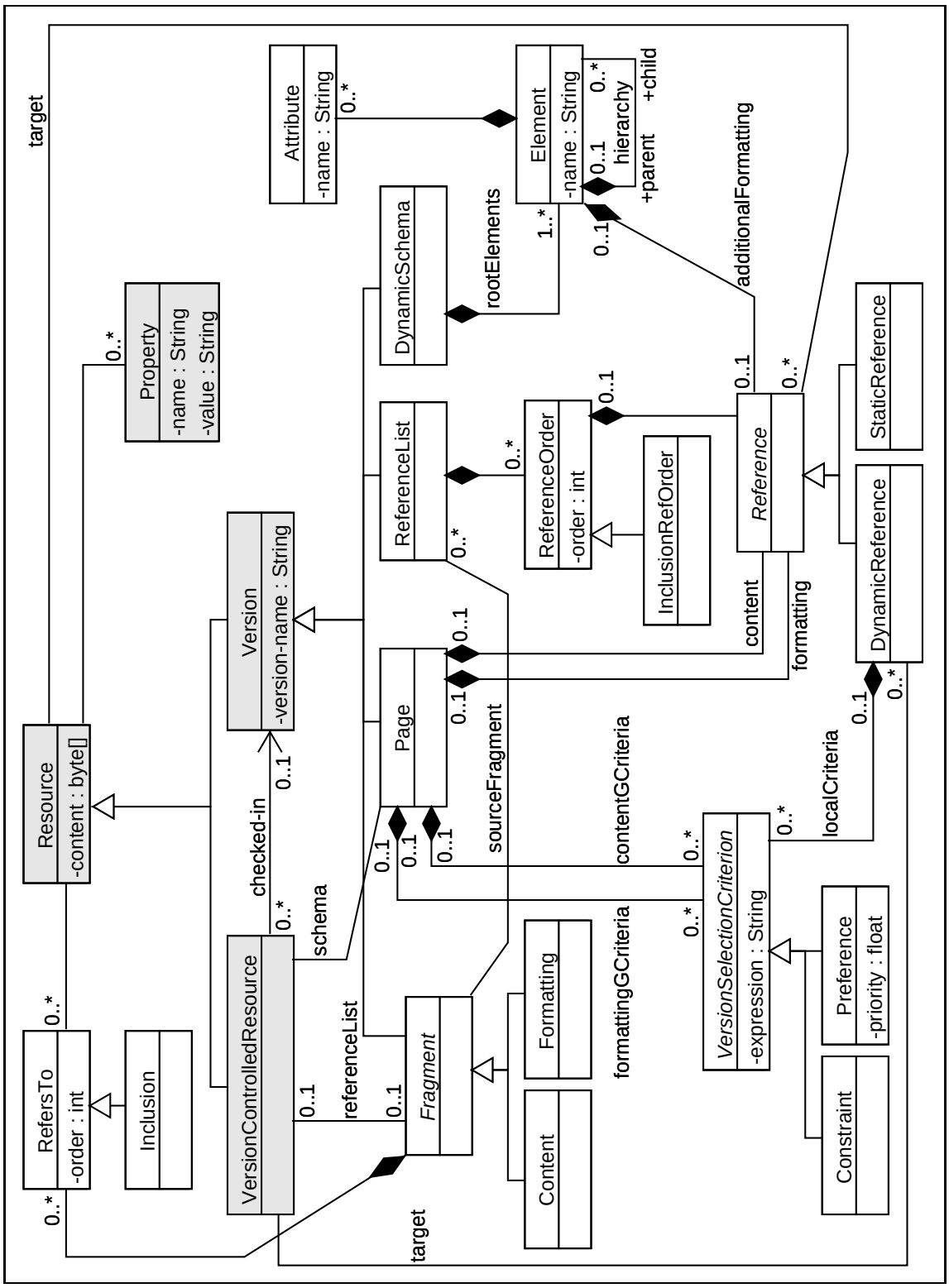


FIGURA 3.8 – Diagrama completo do modelo de versões de páginas proposto

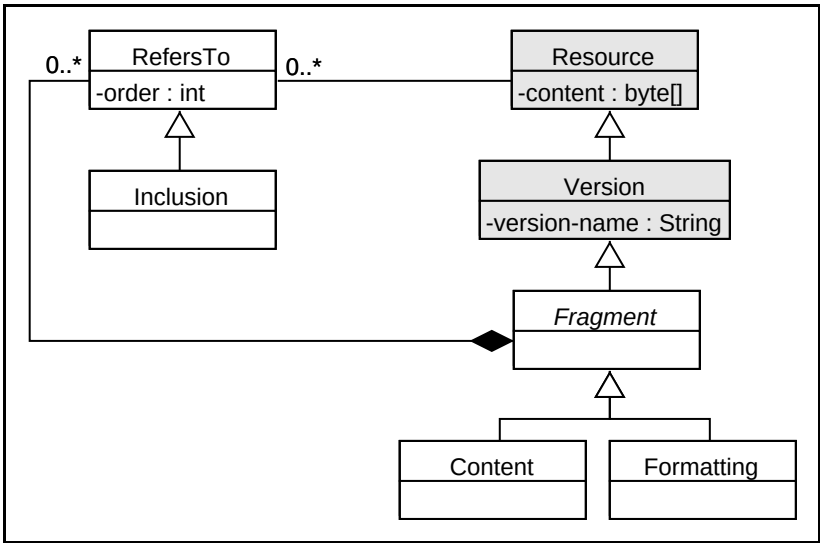


FIGURA 3.9 – Diagrama parcial: fragmentos de conteúdo e formatação

externas [BRA 2000]. Similarmente, fragmentos de formatação podem conter uma folha de estilo XSLT completa ou incluir outras, o que é realizado por elementos *xsl:include* e *xsl:import* [CLA 99].

Assim, o fragmento pode conter referências a outros recursos, o que é representado pelo conceito associativo `RefersTo` na figura 3.9. A ordem em que as referências aparecem no conteúdo do fragmento é indicada pelo atributo `order`. Quanto à função que uma referência exerce em um fragmento, ela pode ser classificada em dois tipos:

- *dependência*, significando que os recursos origem e destino estão relacionados, ou seja um depende do outro; uma referência de dependência pode representar, por exemplo, um *"link"* HTML [RAG 99] usado para navegação entre páginas ou ainda uma referência a um recurso binário (como uma imagem) que o navegador deve carregar no momento da apresentar a página;

- *inclusão* (conceito `Inclusion`), é um tipo especial de dependência que representa uma relação de composto/componente entre dois recursos, ou seja o recurso destino da referência é parte do recurso origem da referência. Para obter o conteúdo completo do recurso origem (em tempo de configuração), é necessário inserir o conteúdo do recurso destino dentro do recurso origem, substituindo a referência.

Os *fragmentos de conteúdo* (`Content`) são documentos XML que representam pedaços de conteúdo que podem ser usados em páginas. Por sua vez, *fragmentos de formatação* (`Formatting`) são folhas de estilo XSLT que representam pedaços de formatação da página. Os fragmentos de conteúdo podem incluir apenas recursos XML, enquanto que fragmentos de formatação podem incluir apenas folhas de estilo XSLT. Entretanto, fragmentos podem ter referências de dependência para qualquer tipo de recurso.

As referências são expressas em uma sintaxe própria do conteúdo dos fragmentos, ou seja o servidor Web não deve impor uma sintaxe ou estrutura aos fragmentos. Assim, eles são criados livremente, usando as ferramentas de edição preferidas do

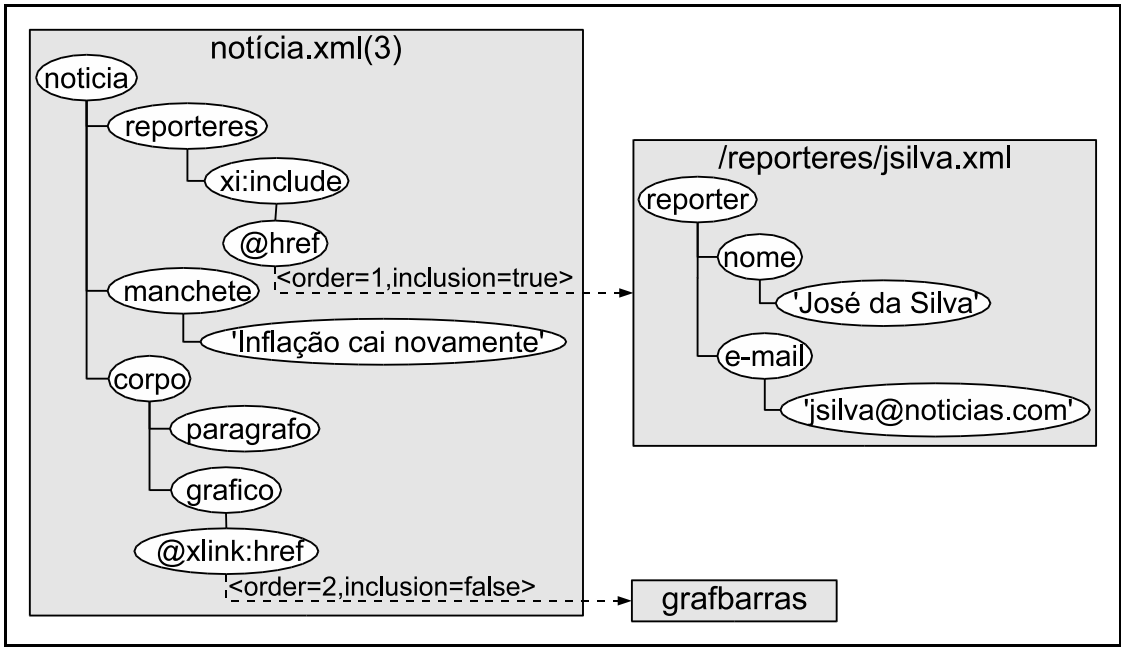


FIGURA 3.10 – Uma instância de fragmento de conteúdo e suas referências

autor. Além disso, essa liberdade na expressão de referências permite ao autor usar formatos de arquivos (baseados em XML) sobre os quais ele não tenha controle. Por exemplo, um formato que poderia ser usado é o SMIL (*"Synchronized Multimedia Integration Language"*) [AYA 2001], que é empregado na modelagem de apresentações multimídia.

No entanto, o servidor Web deve ser capaz de reconhecer as referências de um fragmento, a fim de criar os recursos "lista de referências" (explicados na subseção 3.2.5). Como não é imposta uma sintaxe para as referências dos fragmentos, não é possível identificar se uma referência é estática ou dinâmica. Essa questão é tratada por recursos "lista de referências".

Para expressar referências de dependência, sugere-se o uso de *"links"* simples do padrão XLink [DER 2001], que foi definido para expressar e descrever relações explícitas entre recursos. As referências de inclusão podem ser expressas por elementos *xsl:include* e *xsl:import* (conforme padrão XSLT [CLA 99]) em fragmentos de formatação.

Por sua vez, inclusões em documentos XML são normalmente realizadas por referências a entidades externas. Entretanto, só é possível usar estas referências com uma DTD (*"Document Type Definition"*) [BRA 2000], ou seja um esquema que determina a estrutura do documento. Como o modelo proposto não requer DTDs, sugere-se o uso do padrão XInclude [MAR 2002] na expressão de referências de inclusão em fragmentos de conteúdo.

A figura 3.10 mostra um exemplo de um fragmento de conteúdo de uma notícia. Documentos XML são mostrados como árvores, em que os elementos, nós texto (entre aspas simples) ou atributos (com prefixo "@") são representados por elipses. O elemento "reporteres" do fragmento `notícia.xml(3)` possui uma referência de inclusão (no padrão XInclude) para o fragmento `/reporteres/jsilva.xml`. Por sua vez, este contém dados sobre o autor do texto da notícia.

Na figura, as referências são denotadas por setas tracejadas, acompanhadas

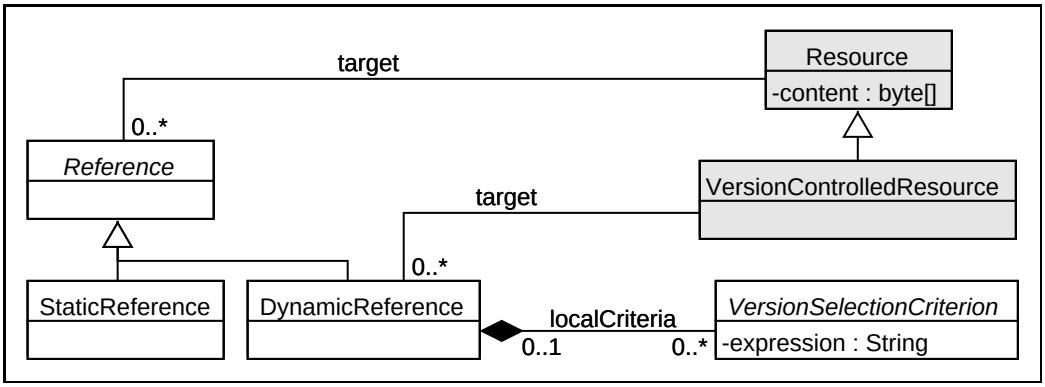


FIGURA 3.11 – Diagrama parcial: referências

de rótulos contendo os valores dos atributos de `RefersTo`. O elemento "grafico" possui uma referência de dependência (no padrão XLink) para um gráfico de barras `grafbarras`. Este gráfico deve ser carregado pelo navegador Web no momento da apresentação da página ao visitante do site.

3.2.2 Referências estáticas e dinâmicas

Uma vez que as referências contidas nos fragmentos são expressas em uma sintaxe própria, não se pode esperar que referências estáticas e dinâmicas sejam diferenciadas explicitamente (como pode ser percebido na figura 3.10). Além disso, esses tipos de referência são também usados no contexto de outros tipos de recursos do modelo. Portanto, um conceito diferente (de `RefersTo`) foi definido para classificar referências em estáticas e dinâmicas. A classificação das referências dos fragmentos (`RefersTo`) em referências estáticas e dinâmicas é realizada pelos recursos "lista de referências", como é explicado na seção 3.2.5.

A figura 3.11 mostra a modelagem de referências estáticas e dinâmicas. Uma *referência* (`Reference`) representa um ponteiro para um recurso destino (associação `target`). Assim, ela liga o recurso ao qual pertence, chamado seu *recurso origem* (não mostrado na figura), a seu *recurso destino*. Portanto, uma referência não existe isoladamente e sim no contexto do recurso que a contém, que pode ser uma página (`Page`), lista de referências (`ReferenceList`) ou um esquema dinâmico (`DynamicSchema`), conforme figura 3.8. Esses tipos de recurso são explicados nas seções 3.2.4, 3.2.5 e 3.2.6, respectivamente.

Uma referência pode ser de dois tipos específicos: estática (`StaticReference`) e dinâmica (`DynamicReference`). O destino de uma *referência estática* pode ser qualquer tipo de recurso, como recursos versionáveis, versionados, versões, etc. Por sua vez, as *referências dinâmicas* podem apontar apenas para recursos versionados. Note-se a existência de duas associações homônimas `target`; a associação ligando referência dinâmica a recurso versionado foi mostrada para tornar graficamente explícita a restrição de destino de uma referência dinâmica.

As referências dinâmicas podem ser usadas para adaptar páginas, pois são resolvidas em tempo de configuração, ou seja uma versão do recurso versionado destino é escolhida no momento da resolução da referência. Assim, a versão mais adequada ao contexto do uso do recurso versionado é selecionada.

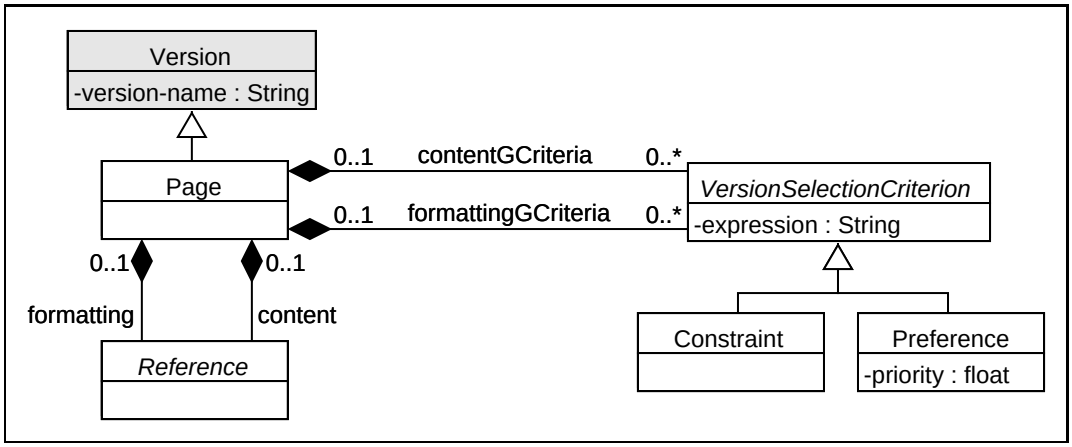


FIGURA 3.12 – Diagrama parcial: página e critérios globais

3.2.3 Critérios de seleção de versões

Para determinar qual versão de um componente da página é a mais adequada para ser selecionada, o processo de configuração utiliza um conjunto de critérios de seleção de versões. Um *critério de seleção de versões* (`VersionSelectionCriterion`, figura 3.11) consiste de uma expressão (atributo `expression`) especificando características que a versão a ser selecionada deve (necessariamente ou preferencialmente) apresentar.

Quando os critérios são associados a uma referência dinâmica, eles são chamados de *critérios locais* (associação `localCriteria`). Esse nome deve-se ao escopo do critério, ou seja critérios locais são usados pelo processo de configuração para resolver apenas a referência a qual estão associados.

Como mostrado na figura 3.12, os critérios podem ser de dois tipos: restrição (`Constraint`) e preferência (`Preference`). Uma *restrição* expressa características da versão selecionada que são necessárias à consistência da página gerada. Por sua vez, uma *preferência* indica características desejáveis da versão selecionada para gerar páginas adaptadas.

As preferências têm uma prioridade (atributo `priority`) que indica seu grau de importância. Ela é usada pelo processo de configuração para determinar a ordem em que as preferências são avaliadas sobre o conjunto de versões de um recurso versionado. O funcionamento do processo de configuração é detalhado na seção 3.3.

3.2.4 Página e critérios globais de seleção de versões

A figura 3.12 também mostra a modelagem de uma página. A *página* (`Page`) representa uma página Web. Ela é o conceito principal do modelo, pois é o ponto de partida (descrição de configuração) para o processo de configuração. Seu conteúdo e formatação estão separados, e são indicados por duas referências uma para o fragmento de formatação (associação `formatting`) e outra para o fragmento de conteúdo (associação `content`). Como mostrado na figura 3.11, essas referências podem ser estáticas ou dinâmicas, e as últimas podem ter critérios locais de seleção de versões.

Uma página pode ter critérios globais de seleção de versões. Os *critérios globais* diferenciam-se dos locais em seu escopo: são usados para resolver todas as

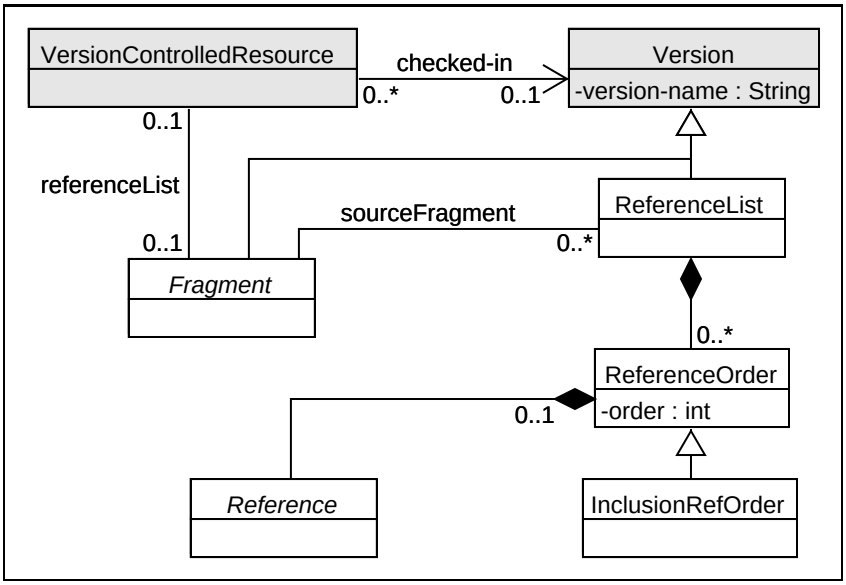


FIGURA 3.13 – Diagrama parcial: fragmentos e lista de referências

referências da hierarquia de composição de uma página. Existem dois conjuntos de critérios globais distintos: um para resolver referências dinâmicas com origem em componentes de conteúdo e outro para as de componentes de formatação (associações `contentGCriteria` e `formattingGCriteria`, respectivamente).

Convém lembrar que as associações de composição indicam que os conceitos que não são recursos (componentes) participam do estado dos que são recursos (compostos). Por exemplo, uma página (que é um recurso) é composta (associação `content`) de uma referência dinâmica (que não é um recurso). Neste caso, a referência dinâmica (assim como seus critérios locais e os critérios globais da página) participam do estado da página.

Assim, uma nova versão do recurso composto deve ser criada para que o componente seja modificado. No exemplo supracitado, uma nova versão da página deve ser criada para que a referência (e os critérios) seja(m) modificada(os). Portanto, é possível controlar versões de referências e de critérios de seleção de versões, de forma similar ao controle de versões de *"links"* nos modelos de versões de hiperdocumentos HyperPro e NCM, como discutido na seção 2.2.

3.2.5 Lista de Referências

Como explicado na seção 3.2.1, referências (conceito `RefersTo` da figura 3.9) contidas em fragmentos são expressas usando uma sintaxe própria. Entretanto, o processo de configuração deve diferenciar referências estáticas e dinâmicas (e obter critérios de seleção de versões para resolver as últimas). Para este fim, foi definido o conceito lista de referências (`ReferenceList`), como modelado na figura 3.13.

O recurso *lista de referências* é uma coleção ordenada de referências (especializadas em estáticas e dinâmicas, conforme figura 3.11) contidas em um fragmento. Para cada referência (`RefersTo`) contida em um fragmento, existe uma referência (`Reference`) correspondente no recurso lista de referências. A ordem das referências em uma lista de referências é a mesma que ocorre no respectivo fragmento.

Na figura 3.13, os elementos da lista de referências são representados pelo

conceito associativo `ReferenceOrder`, onde a ordem das referências é indicada pelo atributo `order`. Análogo ao conceito `Inclusion`, o conceito `InclusionRefOrder` indica que a referência é de inclusão, ou seja que o recurso destino da referência deve ser incluído no recurso origem (o fragmento, neste caso). Essa informação é importante para o processo de configuração, pois ele deve resolver recursivamente as referências dinâmicas dos fragmentos incluídos (detalhes na seção 3.3).

Quando um fragmento for criado (ou seja, uma nova versão), o servidor Web deve criar automaticamente um recurso versionado lista de referências. As referências da primeira versão deste recurso (correspondentes às do fragmento) devem ser estáticas. Dessa forma, o processo de configuração não irá resolvê-las, o que é o comportamento esperado por um autor que desconheça essa capacidade do servidor Web.

O servidor Web deve ainda associar o recurso versionado lista de referências ao fragmento recém criado (associação `referenceList`). Essa associação é usada pelo processo de configuração para localizar a lista de referências correspondente ao fragmento, como detalhado na seção 3.3.

Similarmente, o servidor Web deve associar a lista de referências (versão) ao respectivo fragmento origem (associação `sourceFragment`). Aparentemente as associações `referenceList` e `sourceFragment` são redundantes. Entretanto, `sourceFragment` é necessária para localizar o fragmento origem de uma lista de referências (versão). Neste caso, a associação `referenceList` não pode ser usada, pois a associação `checked-in` é direcional no sentido da lista de referências (versão).

Como ocorre em uma página, as referências (e critérios locais) compõem o estado de uma lista de referências. No contexto de uma lista de referências, o destino de uma referência não pode ser modificado (pois deve ser o mesmo de sua referência correspondente no fragmento origem). Entretanto, é possível alterar o tipo da referência para estática ou dinâmica.

Portanto, um fragmento tem uma lista de referências (recurso versionado) capaz de evoluir. Como um recurso versionado, a lista de referências pode ter um histórico de versões com estrutura em forma de grafo. Assim, pode-se controlar versões dos critérios locais separadamente das versões de fragmentos.

3.2.6 Esquema dinâmico

Autores de folhas de estilo XSLT costumam usar uma DTD para conhecer a estrutura dos documentos XML a serem transformados. Dessa forma, eles constroem folhas de estilo que transformem corretamente qualquer documento XML que obedeça à DTD. Entretanto, obrigar um autor a definir uma DTD para o conteúdo da página não é prático. Como a estrutura das páginas pode mudar com freqüência, a DTD deveria ser constantemente atualizada para refletir estas alterações. Para auxiliar estes autores em seu trabalho, o servidor Web deve construir automaticamente um esquema ou resumo da estrutura do conteúdo de uma página.

O esquema é gerado a partir de instâncias (documentos XML), semelhante a um *"DataGuide"* [GOL 97, GOL 99]. Um *"DataGuide"* é um esquema dinâmico gerado a partir de um banco de dados semi-estruturados, onde objetos são relacionados em uma estrutura hierárquica semelhante a um documento XML. Assim, a explicação a seguir foi adaptada para o contexto de documentos XML.

Um *"DataGuide"* é um documento XML que resume a estrutura de outro documento, através da eliminação de caminhos (seqüências de nomes de elementos

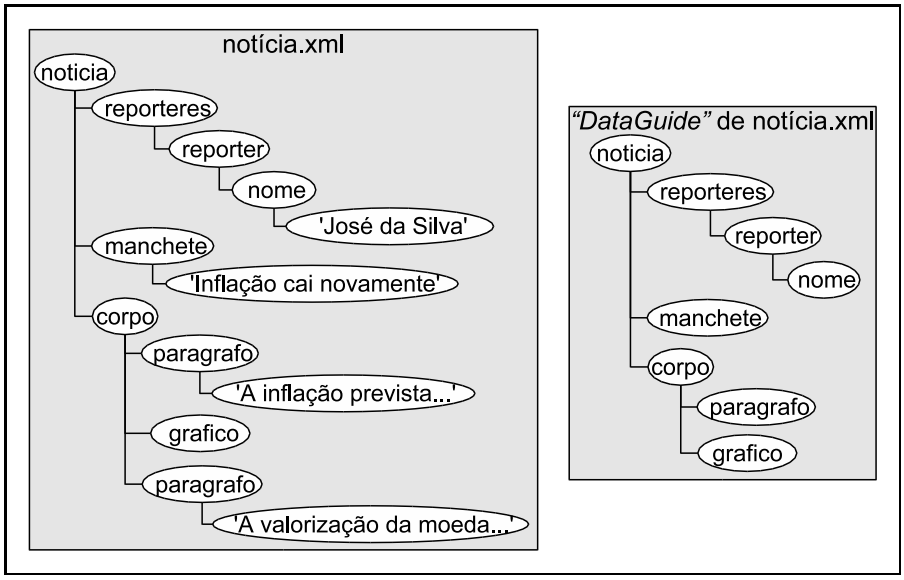


FIGURA 3.14 – Documento XML e *"DataGuide"* correspondente

aninhados) duplicados, como mostra a figura 3.14. Essa figura mostra um documento XML e seu *"DataGuide"* correspondente. Note-se a dupla ocorrência do caminho `\noticia\corpo\paragrafo` no documento XML, mas sua ocorrência simples no *"DataGuide"*. Além disso, os nós texto (entre aspas simples no documento XML) não aparecem no *"DataGuide"*, pois este reflete apenas a estrutura do documento, não seus dados.

Quando o documento XML é modificado, o *"DataGuide"* deve ser atualizado para refletir as mudanças. Os algoritmos para criação e manutenção incremental de *"DataGuides"*, bem como detalhes adicionais sobre eles, podem ser encontrados em [GOL 97, GOL 99].

Um *"DataGuide"* resume apenas um documento XML. Por sua vez, o esquema gerado pelo servidor Web deve refletir a estrutura de todas as versões de um mesmo fragmento de conteúdo (recurso versionado). Isso é necessário porque qualquer versão pode ser selecionada para a resolução de uma referência dinâmica, e a formatação deve ser capaz de transformá-la corretamente. Para isso, o servidor Web pode criar um *"DataGuide"* para cada versão e fazer a união de todos os *"DataGuides"* (removendo caminhos duplicados) e gerando assim o esquema.

A figura 3.15 mostra um exemplo de geração de um esquema dinâmico a partir de dois *"DataGuides"*. Os elementos comuns aos dois *"DataGuides"* são mantidos (sem duplicação) no esquema dinâmico. Os elementos que não são comuns (como `manchete` ou `titulo`) também aparecem no esquema gerado.

Note-se que o esquema gerado, assim como um *"DataGuide"*, não indica a cardinalidade dos elementos. Além disso, a ordem em que elementos irmãos aparecem não é relevante, pois elementos de mesmo nome podem aparecer intercalados com outros no documento (como os elementos `paragrafo` na figura 3.14). Portanto, os autores devem estar atentos a essas características do esquema dinâmico, a fim de produzirem folhas de estilo corretas.

A figura 3.16 mostra a modelagem do esquema dinâmico. O *esquema dinâmico* é um documento XML, gerado pelo servidor Web, que representa a estrutura do conteúdo de uma página. Como nenhuma imposição é feita quanto a estrutu-

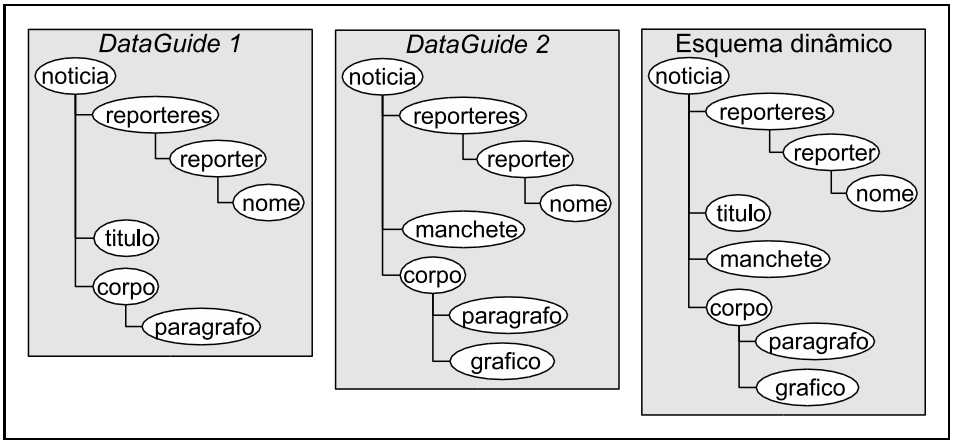


FIGURA 3.15 – Dois *"DataGuides"* e o esquema dinâmico resultante

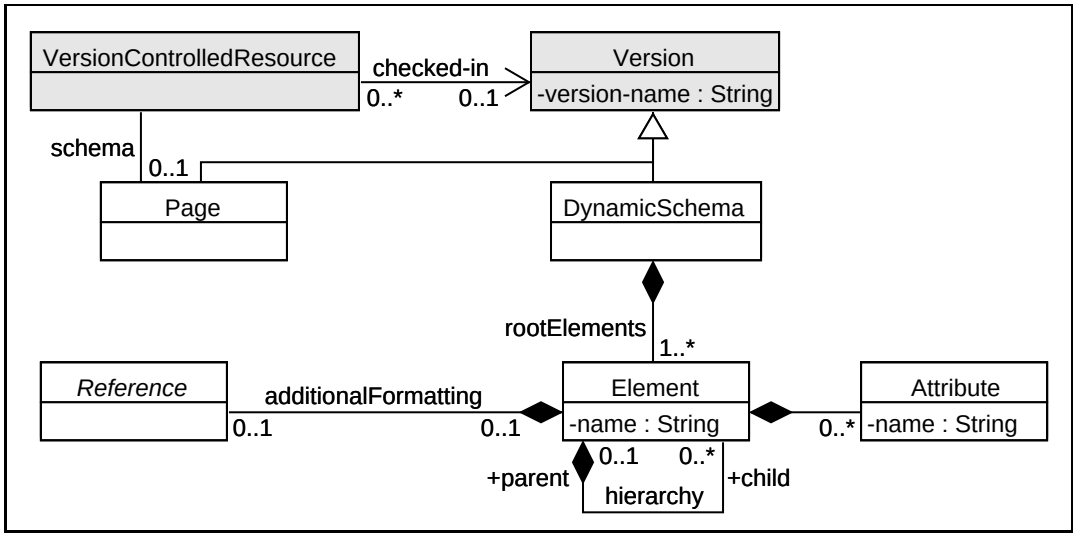


FIGURA 3.16 – Diagrama parcial: página e esquema automático

ra dos fragmentos, versões do mesmo fragmento de conteúdo (recurso versionado) podem ter elementos raízes diferentes. Assim, o esquema deve indicar esses elementos (`Element`), o que é modelado pela associação `rootElements`. Elementos podem ser aninhados (associação `hierarchy`), e podem ter atributos (`Attribute`).

Como visto até então, o conteúdo e a formatação da página são completamente independentes. Inclusive, eles são configurados como entidades isoladas (com critérios globais distintos), que são mescladas apenas ao final do processo de configuração. Entretanto, uma aproximação entre estes dois componentes da página pode ser importante em alguns casos.

Por exemplo, considere-se uma página com um fragmento de formatação para formatar (qualquer versão de) seu fragmento de conteúdo. Se uma nova versão do fragmento de conteúdo for criada, contendo um novo elemento (não existente no esquema dinâmico), o esquema deve ser atualizado. Portanto, provavelmente o fragmento de formatação não poderá formatar corretamente a nova versão. Neste caso, seria necessária uma nova versão do fragmento de formatação para tratar a modificação ocorrida no fragmento de conteúdo.

Entretanto, o modelo oferece uma alternativa: pode-se criar um novo fragmento de formatação que trate especificamente a alteração ocorrida. Esse fragmento é então incluído na formatação da página antes de aplicá-la ao conteúdo. Essa inclusão é feita pelo processo de configuração apenas se o novo elemento estiver presente no conteúdo, conforme detalhado na seção 3.3. Quando usados desta forma, os fragmentos de formatação compõem a *formatação adicional* da página, que é gerada de acordo com a estrutura do conteúdo da página.

Logo, o processo de configuração necessita saber qual fragmento de formatação incluir quando o novo elemento ocorrer no conteúdo da página. Para isso, deve-se associar ao elemento (no esquema dinâmico) uma referência para o fragmento. Isso é modelado pela associação `additionalFormatting` na figura 3.16. Note-se que esse mecanismo representa uma alternativa para a geração da formatação da página. A geração da formatação, usando esse mecanismo, é dirigida pela estrutura do conteúdo da página, de forma mais próxima ao funcionamento das folhas de estilo XSLT (que são baseadas em expressões de caminho).

3.3 Processo de configuração

O processo de configuração (nesta seção chamado apenas de processo) é responsável por gerar páginas configuradas (e adaptadas, potencialmente) em resposta a uma requisição de um visitante do site. Para isso, ele é auxiliado pelo autor das páginas, que cria versões alternativas para os fragmentos de conteúdo e formatação da página, bem como para outros componentes (como figuras, por exemplo). Assim, o processo pode escolher versões de fragmentos que sejam mais adequadas a determinado visitante.

A escolha das versões é guiada por critérios de seleção de versões, que especificam características necessárias ou desejáveis que versões selecionadas devem possuir. Novamente, o processo é auxiliado por autores que especificam estes critérios quando criam as versões. O servidor Web também pode gerar critérios globais automaticamente. Assim, o processo pode gerar páginas configuradas automaticamente, ou seja uma vez que o processo inicia sua execução, ele não interage com o usuário,

pois todas as informações necessárias são conhecidas.

Cabe salientar que o processo não armazena nenhuma informação no repositório de versões, ou seja todas as operações são feitas em memória. A página resultante é apenas enviada ao visitante e não é armazenada. Eventualmente, uma implementação da infra-estrutura proposta pode manter as páginas configuradas em *cache* para melhorar o desempenho. Entretanto, essa é uma questão de implementação que não é tratada pela presente proposta.

Esta seção explica detalhadamente o funcionamento do processo de configuração. Primeiramente as entradas (parâmetros) do processo são explicados e a execução do processo é dividida em fases. A subseção 3.3.1 explica a resolução de uma referência dinâmica, enquanto que as fases do processo são explicadas da subseção 3.3.2 à 3.3.5.

As informações necessárias para o início da execução do processo (fornecidas como parâmetros pelo servidor Web) são:

- URI da página a ser configurada. Esse URI pode indicar um recurso versionado ou uma versão específica;
- dois conjuntos de critérios globais de seleção de versões, gerados pelo servidor Web (que podem ser vazios), para a configuração dos fragmentos de conteúdo e formatação, respectivamente;
- um conjunto de *propriedades do ambiente*, que podem ser usadas na especificação dos critérios. Esse conjunto pode conter propriedades que expressem características ou preferências do visitante (como idade ou assuntos de interesse), caso o servidor Web as conheça. Entretanto, este conjunto de propriedades deve conter pelo menos as informações obtidas dos cabeçalhos da requisição do visitante (conforme explicação da figura 2.7).

A fim de facilitar o entendimento, a execução do processo é dividida nas seguintes fases:

1. processamento da página, onde os fragmentos principais e critérios globais são obtidos;
2. configuração do fragmento de conteúdo principal da página;
3. configuração do fragmento de formatação principal da página;
4. geração da formatação adicional para o conteúdo da página, usando os fragmentos de formatação referenciados pelo esquema dinâmico (associação `additionalFormatting` figura 3.16);
5. geração da página configurada, transformando o conteúdo XML com a formatação XSLT, e envio desta ao visitante.

3.3.1 Resolução de uma referência dinâmica

Antes de explicar cada fase do processo, a resolução de uma referência dinâmica é explicada, uma vez que essa operação é necessária em várias fases do processo. O resultado desta operação é o URI da versão escolhida ou uma indicação de que

nenhuma versão satisfaz as restrições impostas. Para realizar a resolução de uma referência dinâmica, é necessário ter-se (como parâmetros para a operação):

- o recurso origem da referência dinâmica;
- a referência dinâmica propriamente dita;
- um conjunto de critérios globais;
- um conjunto de propriedades do ambiente; esse conjunto é sempre o mesmo (não muda) durante a mesma execução do processo.

Os critérios podem relacionar propriedades do recurso origem e da versão (candidata) destino da referência, e ainda as propriedades do ambiente (conforme explicado na seção 3.1). Conseqüentemente, todas estas propriedades devem ser obtidas para a avaliação dos critérios. Com base nos parâmetros, a operação de resolução consegue obter estas propriedades e também os próprios critérios.

A partir do recurso origem da referência dinâmica, é possível obter seu conjunto de propriedades (P_O). O conjunto de critérios locais da referência são obtidos navegando-se na associação `localCriteria` (figura 3.8); o recurso versionado destino da referência é localizado através da associação `target`.

Os conjuntos de critérios locais e globais são unidos em um novo conjunto de critérios do escopo da referência (C_R). Então C_R é ordenado, formando uma seqüência, de maneira que as restrições estejam no início e as preferências no fim. A ordem nesta seqüência indica em que ordem os critérios serão avaliados sobre as versões candidatas a destino da referência. A ordem relativa das restrições não é relevante (a não ser por otimização), pois a versão escolhida deve atender a todas elas. No entanto, a ordem entre as preferências é importante e é a ordem decrescente de suas prioridades (atributo `priority`).

Portanto, os autores das páginas e fragmentos devem estar atentos à atribuição de prioridades às preferências. Sugere-se a definição de uma política de atribuição de prioridades às preferências que deve ser seguida pelos autores. Em particular, as prioridades dos critérios globais (preferências) gerados pelo servidor Web devem estar afinadas com essa política. Por exemplo, considere-se um site em que a adaptação de idioma do conteúdo da página seja mais importante do que a adaptação de sua extensão (se o conteúdo é ou não resumido). Neste caso, essa política deveria dar maior prioridade aos critérios que adaptem o idioma.

Além disso, se o servidor Web gera critérios globais, provavelmente ele deva conhecer a semântica das propriedades associadas às versões dos componentes da página. Portanto, uma política para atribuição de propriedades às versões dos componentes das páginas também deve ser estabelecida.

O histórico de versões do recurso versionado referenciado (associação `version-history`, figura 2.8) é localizado. Tendo-se o histórico, pode-se acessar as versões (associação `version-set`) e assim obter o conjunto de propriedades de cada versão (P_{Vj}, onde j é o nome da versão). Neste ponto, tem-se todas as informações necessárias para avaliar os critérios de seleção: P_O, C_R, o conjunto de propriedades do ambiente, e os P_{Vj}.

Os critérios de seleção de versões funcionam como consultas (filtros) sobre o conjunto de versões candidatas (V) a destino da referência. Dependendo da linguagem na qual os critérios estão especificados, eles podem ser avaliados sobre cada

versão individualmente ou sobre V como um todo. Por exemplo, se a linguagem permite expressar um critério como "selecionar a versão com o maior valor para a propriedade x", ele deve ser avaliado sobre V para que se possa comparar o valor da propriedade "x" de todas as versões.

O algoritmo para a escolha de uma versão de V é bastante semelhante ao método de seleção de versões de Návrat [NÁV 96] (explicado na subseção 2.3.4). Assim, apenas as diferenças ou especificidades são descritas aqui. Primeiramente, V é inicializado com todas as versões do histórico de versões do recurso versionado referenciado. Entretanto, diferente do método de Návrat que possui apenas uma função heurística obrigatória, o modelo proposto permite várias restrições.

Assim, o conjunto de versões candidatas V_0 é obtido apenas após a avaliação de todas as restrições de C_R. A avaliação das preferências de C_R segue o algoritmo do método de Návrat (avaliação de funções heurísticas opcionais). Como o processo deve ser automático, pois o visitante não deve tomar conhecimento do controle de versões, o critério *"default"* é a escolha da versão candidata mais recente.

3.3.2 Fase 1: processamento da página

Quando o processo é iniciado, a primeira ação a ser tomada é obter o estado da página a ser configurada. A partir dela, localiza-se seus fragmentos de conteúdo e formatação principais, além dos critérios globais de seleção de versões que estiverem associados a ela. Se a URI da página requisitada for um recurso versionado, sua versão corrente é usada.

Os conjuntos de critérios globais são obtidos através das associações `contentGCriteria` e `formattingGCriteria` da figura 3.8. Esses conjuntos são unidos aos respectivos conjuntos de critérios globais fornecidos pelo servidor Web. Os conjuntos unificados podem então ser usados para resolver referências dinâmicas. Através das associações `content` e `formatting`, chega-se às referências aos fragmentos principais de conteúdo e formatação da página, respectivamente.

Quando uma referência para um fragmento principal for estática, o estado do objeto destino (associação `target`) é obtido. Este fragmento pode ser um recurso versionado ou uma versão específica. Se for um recurso versionado, sua versão corrente é obtida.

Por outro lado, quando uma referência para um fragmento principal for dinâmica, ela deve ser resolvida conforme a operação de resolução de referências dinâmicas (explicada na subseção 3.3.1). Os parâmetros informados à operação são: o recurso origem da referência é a página; o conjunto de critérios globais é vazio, pois os critérios globais associados à página são usados apenas quando a origem de uma referência for um fragmento.

Para tornar mais clara a explicação desta fase do processo e das próximas, a seguinte convenção será adotada. Quando uma referência dinâmica tiver de ser resolvida, será indicado que a operação da subseção 3.3.1 deverá ser usada. Como as propriedades do ambiente não mudam durante a execução do processo, elas são sempre passadas como parâmetro, mesmo quando não for mencionado explicitamente.

Outra informação passada implicitamente à operação é a referência dinâmica a ser resolvida, que poderá ser percebida pelo contexto da explicação. O conjunto de critérios globais (unificado) passado à operação será sempre o correspondente ao tipo de fragmento que for informado como origem da referência.

3.3.3 Fases 2 e 3: configuração dos fragmentos principais

Obtidos os fragmentos principais de conteúdo e formatação da página, sua configuração é feita nas fases 2 e 3, respectivamente. Uma vez que a configuração de fragmentos de conteúdo e formatação são análogos, esta seção descreve a configuração de um fragmento em geral.

Tendo-se um fragmento, é necessário obter suas referências acessando sua lista de referências (associação `ReferenceList`), ou seja a versão corrente da lista de referências. Se o fragmento não tiver uma lista de referências associada, então ele não possui referências, ou seja já está configurado.

As referências da lista de referências são processadas, sendo que as referências dinâmicas são resolvidas pela operação de resolução (subseção 3.3.1). Como recurso origem da referência, é passado como parâmetro à operação o fragmento que está sendo configurado (e não a lista de referências).

Se uma referência dinâmica for de dependência, o destino da referência (`RefersTo`) no fragmento origem será substituído pela URI resultante da operação de resolução. Assim, o navegador Web poderá requisitar o recurso referenciado quando necessário. Por exemplo, considere-se uma figura referenciada por uma referência dinâmica (de dependência); quando esta for resolvida, uma versão específica da figura será referenciada a partir do fragmento. Assim, quando receber a página configurada, o navegador poderá requisitar uma versão da figura através de seu URI contido na página.

Por outro lado, se uma referência (mesmo estática) for de inclusão, o fragmento destino da referência será processado recursivamente como descrito nesta subseção. Após sua configuração, seu conteúdo substituirá a referência (`Inclusion`) no fragmento origem.

Ao final das fases 2 e 3, tem-se os fragmentos principais da página configurados. Antes de gerar a página final (configurada) que será enviada ao visitante, deve-se obter os fragmentos de formatação referenciados pelo esquema dinâmico da página.

3.3.4 Fase 4: geração da formatação adicional

A fim de gerar a formatação adicional, o esquema dinâmico (associação `schema`) da página deve ser obtido. O conteúdo (configurado) da página é então analisado: para cada elemento XML distinto (de nome e/ou aninhamento diferente) encontrado, localiza-se seu correspondente (E_S) no esquema dinâmico. Se E_S tiver uma referência para um fragmento de formatação (associação `additionalFormatting`), essa referência deve ser processada. Logo, não serão processadas as referências de elementos do esquema que não estiverem presentes no conteúdo da página.

Se a referência de E_S for dinâmica, ela é resolvida pela operação de resolução (subseção 3.3.1). Neste caso, o fragmento de conteúdo principal da página é informado como recurso origem da referência. Assim, as propriedades do fragmento de conteúdo (e não as do esquema dinâmico) são usadas para avaliar os critérios de seleção de versões.

O fragmento de formatação destino da referência é configurado conforme explicado na subseção 3.3.3. Os fragmentos de formatação obtidos através das referências do esquema dinâmico formam a formatação adicional. Ela é incluída na formatação da página (configurada na fase 3), obtendo-se a formatação completa a ser usada na fase 5.

3.3.5 Fase 5: geração da página configurada

De posse do conteúdo e formatação da página completamente definidos, o processo usa um processador XSLT para aplicar a transformação (formatação) ao conteúdo. Essa transformação produz uma página configurada gerada dinamicamente, que é então enviada ao visitante que a requisitou.

Alternativamente, uma implementação do processo pode fornecer como saída o conteúdo e a formatação da página. Neste caso, o servidor Web pode aplicar a transformação XSLT ao conteúdo XML ele próprio. Como os navegadores mais recentes suportam XML, o servidor Web pode deixar que o navegador usado pelo visitante realize essa transformação, reduzindo assim a carga de trabalho do servidor Web.

3.4 Conclusão

Este capítulo apresentou a proposta de uma infra-estrutura para um servidor Web que realiza controle de versões e suporte a adaptação de páginas Web, tornando-os transparentes aos visitantes do site. O controle de versões obedece a um modelo de versões proposto, que separa a página em vários componentes, incluindo o conteúdo e a formatação. Estes componentes em particular são expressos nos padrões XML e XSLT, respectivamente, e têm suas versões controladas independentemente.

O modelo de versões proposto modela explicitamente os critérios de seleção de versões, que são usados para gerar automaticamente configurações resolvidas das páginas. Como estes critérios participam do estado de recursos versionados, pode-se também controlar suas versões.

A adaptação de páginas pode ser obtida através de sua configuração, na qual os autores e o próprio servidor Web podem interferir especificando critérios de seleção de versões. Além de uma descrição de configuração, o processo recebe do servidor Web um conjunto de informações referentes ao contexto da requisição da página. Assim, os critérios podem ser especificados tanto para garantir a consistência da página quanto para adaptá-la de acordo com o contexto.

4 Conclusão

Este trabalho apresentou uma infra-estrutura geral para um servidor Web que implementa o controle de versões e suporte a adaptação de páginas Web, tornando-os transparentes aos visitantes do site. Essa infra-estrutura visa atender, por um lado, às necessidades de gerência de conteúdo dos sites, e por outro, à demanda por adaptação do site a características particulares de cada visitante.

Para tanto, foi projetado um modelo de versões de páginas Web, que separa formatação e conteúdo, controlando suas versões independentemente. O modelo provê esquemas gerados dinamicamente a partir das versões do conteúdo da página. Esses esquemas auxiliam autores a criar a formatação da página, pois refletem a estrutura do conteúdo da página. Critérios de seleção de versões são modelados explicitamente e auxiliam na configuração de páginas. Como eles participam do estado de recursos versionados, pode-se manter um controle de versões destes critérios.

A adaptação de páginas é suportada pelo processo de configuração, que utiliza critérios de seleção de versões para resolver referências dinâmicas. Esses critérios indicam características de versões dos componentes que sejam adequadas para formar a página configurada. Estes critérios podem ser especificados pelo autor da página ou gerados pelo próprio servidor Web.

A contribuição deste trabalho é uma infra-estrutura geral que serve de base para a implementação de sistemas específicos de adaptação de páginas Web. O mecanismo básico de adaptação baseia-se na configuração de páginas com componentes versionados. Em particular, o modelo de versões de páginas proposto apresenta as seguintes características:

- separação do conteúdo e formatação da página em componentes distintos, mantendo controle de versões sobre eles de maneira independente;
- modelagem explícita dos critérios de seleção de versões usados na configuração de páginas, sendo que também há controle de versões sobre estes critérios;
- é uma extensão ao WebDAV, portanto pode servir de exemplo para outras extensões a esse protocolo.

Os processos de configuração analisados (seção 2.3) utilizam apenas a estrutura de composição (ou dependência) dos objetos e meta-dados para a geração de configurações resolvidas. Por sua vez, o processo de configuração da infra-estrutura proposta permite que a estrutura interna dos objetos (conteúdo da página) influencie na configuração resolvida gerada (pela inclusão de formatação adicional).

Uma deficiência que pode ser notada no modelo de versões proposto é que o esquema dinâmico é baseado em *DataGuides*. Para serem concisos [GOL 97], *DataGuides* possuem uma única ocorrência de um caminho do documento XML origem. Dessa forma, tem-se apenas a estrutura de aninhamento dos elementos, mas não a ordem em que elementos irmãos ocorrem no documento. Entretanto, no domínio de documentos, e dependendo da forma como as folhas de estilo XSLT são construídas, a ordem em que os elementos ocorrem pode ser importante.

Portanto, é necessário investigar uma maneira de gerar o esquema dinâmico a partir de versões do conteúdo da página, mantendo a ordem de elementos irmãos.

Além disso, o esquema deve ainda ser conciso, caso contrário ele será de difícil manipulação por um autor de folhas de estilo XSLT.

Os recursos lista de referências mantêm apenas critérios locais de seleção de versões. Eles poderiam também manter critérios globais, a serem usados para resolver referências no contexto de um fragmento e seus componentes (como o sistema Adele, subseção 2.3.3). De maneira similar, o servidor Web não gera critérios globais para resolver referências contidas em páginas (apenas para as de fragmentos). Poder-se-ia então modificar o processo para receber tais critérios do servidor Web.

No entanto, não foram encontradas situações práticas onde essas características fossem necessárias. Por isso, elas não foram incluídas na infra-estrutura proposta. Entretanto, elas podem ser necessárias em alguma aplicação da infra-estrutura, podendo-se então estender esta para incluí-las.

Outra melhoria que pode ser realizada na infra-estrutura proposta diz respeito à escolha de uma versão do conjunto de critérios de seleção de versões (associados a página ou lista de referências). O processo de configuração usa como critério *default* a escolha da versão corrente do conjunto de critérios (recursos versionados página e lista de referências). Entretanto, poder-se-ia estabelecer algum critério, definido pelo autor, para escolher uma versão do conjunto de critérios a ser usada.

Ao longo deste trabalho, o tipo de adaptação exemplificado normalmente dependia de informações providas através dos cabeçalhos da requisição HTTP. Entretanto, deve-se investigar outras formas de adaptação. Em especial, deve-se investigar adaptações que dependam de informações contidas em um perfil de usuário, pois espera-se que estas sejam as mais interessantes ao sites Web em geral.

Em particular, pode-se desenvolver um sistema de adaptação baseada em perfis do usuário para validar a solução proposta neste trabalho e verificar eventuais modificações necessárias. Para isso, será necessário desenvolver uma política de atribuição de prioridades aos critérios de seleção de versões e de associação de metadados (propriedades) aos componentes das páginas.

Foi desenvolvido um protótipo (ver anexo) que implementa a infra-estrutura proposta. O objetivo desse protótipo é permitir experimentações com os conceitos modelados. O protótipo não foi desenvolvido com o intuito de avaliar o desempenho da solução proposta; inclusive, várias de suas características de implementação levam a uma performance baixa, como por exemplo o uso da linguagem Java [SUN 2002]. Mesmo em uma implementação otimizada, o servidor Web pode ter problemas de desempenho, uma vez que é necessário realizar transformações XSLT e os processadores XSLT têm baixa performance por natureza. Portanto, é importante investigar formas de melhorar o desempenho do servidor Web.

Uma idéia preliminar para melhorar a performance do servidor Web seria a seguinte. Quando um usuário obtiver uma página, o servidor Web pode verificar quais páginas podem ser acessadas a partir dela e configurá-las *a priori*; pode-se ainda tentar prever qual página será acessada [SCH 98] e configurá-la. As páginas configuradas *a priori* são colocadas em *cache*. Dessa forma, quando o visitante acessar uma página, poderá recebê-las rapidamente, pois ela estará previamente configurada na *cache*.

Anexo Arquitetura do servidor Web protótipo

Um servidor Web que utiliza a infra-estrutura proposta neste trabalho foi implementado. Este protótipo foi criado usando a linguagem Java [SUN 2002]. A descrição geral da arquitetura do protótipo e de sua implementação é apresentada neste anexo.

A figura A.1 mostra a arquitetura interna do servidor Web e os aplicativos cliente que se comunicam com ele. As setas indicam comunicação entre aplicativos cliente e módulos do servidor Web, bem como comunicação inter-módulos; a origem da seta indica qual aplicativo ou módulo inicia uma comunicação.

Aplicativos Cliente

Os visitantes do site acessam o servidor Web normalmente, usando um navegador Web para visualizar as páginas. Por outro lado, os autores utilizam requisições (explicado figura 2.7) usando os métodos do protocolo WebDAV (tabela 2.2) para manipular os recursos. Portanto, os autores utilizam um aplicativo especial para acessar o servidor Web, chamado importador/exportador.

Existem alguns aplicativos clientes que implementam o protocolo WebDAV, destinados ao acesso e atualização dos recursos, como por exemplo o DAV Explorer [KAN 2002]. Como o protocolo WebDAV em si (métodos e formato das informações que trafegam na rede) não foi modificado pela proposta do presente trabalho, um destes aplicativos pode ser usado por um autor.

Entretanto, o servidor Web protótipo não implementou todos os métodos do protocolo (como por exemplo o OPTIONS, que permite a um cliente descobrir as capacidades de um servidor Web). Isso inviabilizou o uso do DAV Explorer. Logo, foi implementado um aplicativo simples que envia arquivos para o servidor Web, permitindo que o protótipo seja testado. Assim, as mensagens do protocolo devem ser colocadas manualmente em arquivos.

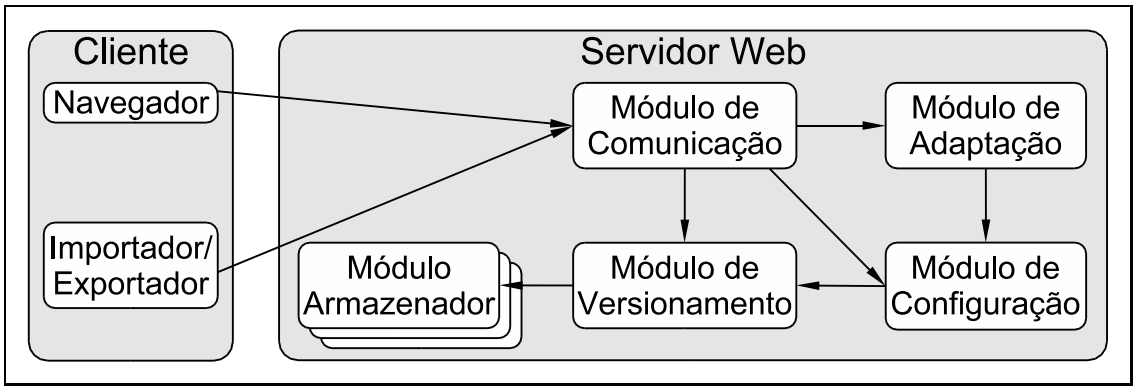


FIGURA A.1 – Arquitetura do servidor Web protótipo

Módulos Armazenadores

Quando se considera o controle de versões, uma questão a ser resolvida é o armazenamento das versões. Uma vez que é necessário registrar o histórico de versões, existem muitos arquivos (versões) onde normalmente existiria apenas um, e isso aumenta a necessidade de espaço de armazenamento. Para atender a este requisito, foi definida uma camada de armazenamento, composta por módulos armazenadores.

Os módulos armazenadores são responsáveis por armazenar o conteúdo de recursos. Cada um deles é especialista em armazenar determinado(s) tipo(s) de conteúdo, podendo empregar técnicas específicas para reduzir o espaço ocupado pelos recursos. No entanto, apenas um armazenador foi implementado: ele armazena o conteúdo dos recursos em arquivos sem nenhum pré-processamento. Isso permite que ele seja de uso geral, podendo armazenar o conteúdo de qualquer recurso.

Módulo de Versionamento

Os módulos armazenadores são acessados apenas pelo módulo de versionamento. Este módulo implementa o modelo de dados do WebDAV e coordena o controle de versões, usando os módulos armazenadores para armazenar o conteúdo dos recursos. Ele ainda é responsável pelas propriedades dos recursos, podendo usar um módulo armazenador para mantê-las. Na implementação realizada, as propriedades foram armazenadas em um banco de dados relacional.

Apenas um subconjunto das características do WebDAV básico (sem características avançadas de versionamento [CLE 2002]) foi implementado:

- métodos: CHECKIN, CHECKOUT, GET, HEAD, MKCOL, PROPFIND, PROPPATCH, PUT, UPDATE, VERSION-CONTROL (tabela 2.2);
- recursos: todos, com exceção de `Workspace` e `VersionableResource` (figura 2.10). O recurso versionável não foi necessário pois todos os recursos criados foram versionados automaticamente; note-se que esse comportamento é permitido pelo WebDAV;
- propriedades: por serem muitas, não serão enumeradas; em geral, as que relacionam recursos (como *checked-in*) e algumas simples de maior interesse (como *getcontentlanguage*, tabela 2.3) foram implementadas.

Módulo de Configuração

Como o módulo de versionamento implementa o modelo de dados do WebDAV, outros módulos podem implementar extensões sobre ele, interceptando suas requisições e usando suas funcionalidades. É desta forma que o módulo de configuração implementa o modelo proposto e faz as validações necessárias conforme descrito no capítulo 3. Portanto, este é o módulo principal da arquitetura.

Como seu nome sugere, o módulo de configuração gera páginas configuradas, executando o processo de configuração. Durante a execução do processo, o módulo de versionamento é contactado para a obtenção de informações (propriedades e conteúdo) das versões e recursos relacionados (conforme seção 3.3).

Na implementação realizada, as informações mantidas pelo módulo de configuração são armazenadas em um banco de dados relacional (como ocorre com o módulo de versionamento). Por isso, todas as informações necessárias ao processo (incluindo as de responsabilidade do módulo de versionamento) são obtidas diretamente do banco de dados. As validações sobre os recursos do modelo são realizadas em resposta a mensagens PUT (quando um novo recurso é criado) e CHECKIN (tabela 2.2).

Entretanto, a associação `additionalFormatting` (figura 3.8) não foi implementada, logo não há configuração de formatação adicional. Outra diferença da implementação em relação ao modelo é a existência de um esquema dinâmico para cada histórico de fragmentos de conteúdo, não apenas para o fragmento principal da página. Assim, quando um fragmento F_R é reusado (incluído em outro fragmento) em várias páginas, pode-se incluir o esquema dinâmico de F_R na posição adequada do esquema dinâmico da página.

Dessa forma, quando o esquema de F_R for alterado, essa alteração não será propagada imediatamente para os esquemas das páginas que usam F_R. Essa alteração é propagada sob-demanda quando os esquemas afetados (pela alteração) forem requisitados por autores. Portanto, essa implementação auxilia a controlar a proliferação de versões dos esquemas das páginas.

Um aspecto que merece destaque na implementação realizada é a avaliação de critérios para resolver uma referência. As propriedades das versões origem e destino da referência e as propriedades do ambiente são agrupadas em um documento XML. Um destes documentos é construído para cada versão candidata. Os critérios são especificados como expressões XPath [CLA 99a], assim eles podem ser avaliados diretamente sobre os documentos contendo as propriedades. Se a expressão retornar alguma informação, considera-se que a versão candidata atende ao critério.

Módulo de Adaptação

Além dos critérios definidos pelo autor, o módulo de configuração conta com critérios providos pelo servidor Web para guiar a escolha das versões durante a resolução de referências dinâmicas. O fornecimento destes critérios é função do módulo de adaptação, e é uma forma de interferir na execução do processo de configuração.

O módulo de adaptação ainda informa ao módulo de configuração as propriedades do ambiente extraídas da requisição e do contexto (por exemplo do perfil do visitante, caso o conheça). Este módulo também é um ponto adequado da arquitetura para a validação das políticas de atribuição de prioridades a critérios e de propriedades dos componentes das páginas, conforme comentado na subseção 3.3.1.

Essas funções são realizadas pelo módulo de adaptação de maneira específica em cada servidor Web. Na implementação realizada, este módulo fornece apenas propriedades do ambiente extraídas da requisição, não gera nenhum critério global e não realiza validações de propriedades definidas pelo autor.

Módulo de Comunicação

O ponto de entrada de requisições para o servidor Web é o módulo de comunicação. Ele é responsável pela implementação de protocolos de rede para a comunicação com aplicativos cliente. Além do WebDAV, ele pode implementar outros

protocolos como o WAP [OPE 2002], assim o servidor Web pode responder diretamente a telefones celulares. As requisições recebidas pelo módulo de comunicação devem ser distribuídas para os módulos internos adequados.

A implementação deste módulo converte a requisição HTTP (WebDAV) em uma estrutura de dados interna que é conhecida e manipulada pelos outros módulos. Este módulo passa essa estrutura de dados para cada módulo, perguntando se este é capaz de responder à requisição. A ordem da consulta aos módulos é: o de adaptação, o de configuração e o de versionamento. O primeiro módulo a responder positivamente recebe a requisição para ser processada.

Bibliografia

[ADB 99] ADBELZAHER, T. F.; BHATTI, N. Web Content Adaptation to Improve Server Overload Behavior. **Computer Networks**, Amsterdam, v.31, n.11-16, p.1563–1577, May 1999.

[ARN 2000] ARNOLD-MOORE, T. et al. Architecture of a Content Management Server for XML Documents Applications. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, WISE, 1., 2000, Hong Kong, China. **Proceedings...** [Hong Kong: Steering Commitee of Wise Conferences], 2000. v.1, p.97–108.

[AYA 2001] AYARS, J. et al. **Synchronized Multimedia Integration Language (SMIL 2.0) - W3C Recommendation**. Disponível em: <http://www.w3.org/TR/smil20/>. Acesso em: out. 2002.

[BER 88] BERNARD, Y. et al. Configuration Management in an Open Environment. In: EUROPEAN SOFTWARE ENGINEERING CONFERENCE, 1., 1988. **Proceedings...** Berlin: Springer, 1988. p.35–43. (Lecture Notes in Computer Science, v.289).

[BER 98] BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. **RFC-2396**: uniform resource identifiers (URI): generic syntax. Disponível em: <ftp://ftp.isi.edu/in-notes/rfc2396.txt>. Acesso em: set. 2002.

[BIE 98] BIELIKOVÁ, M.; NÁVRAT, P. Modelling Versioned Hypertext Documents. In: SYSTEM CONFIGURATION MANAGEMENT SYMPOSIUM, 8., 1998, Brussels, Belgium. **Proceedings...** Berlin: Springer-Verlag, 1998. p.188–197. (Lecture Notes in Computer Science, v.1439).

[BRA 2002] BRADLEY, N. **The XML Companion**. 3rd ed. Boston: Addison-Wesley, 2002. 834p.

[BRA 2000] BRAY, T. et al. **Extensible Markup Language (XML) 1.0 (Second Edition) - W3C Recommendation**. Disponível em: <http://www.w3.org/TR/REC-xml>. Acesso em: out. 2001.

[CHI 2001] CHIEN, S.-Y. et al. Storing and Querying Multiversion XML Documents Using Durable Node Numbers. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, WISE, 2., 2001, Kyoto, Japan. **Proceedings...** [S.l.]: IEEE Computer Society, 2001. v.1, p.232–244.

[CHI 2002] CHIEN, S.-Y. et al. Efficient Complex Query Support for Multiversion XML Documents. In: CONFERENCE ON EXTENDING DATABASE TECHNOLOGY, EDBT, 8., 2002, Prague, Czech Republic. **Advances in Database Technology:** proceedings... Berlin: Springer-Verlag, 2002. p.161–178. (Lecture Notes in Computer Science, v.2287).

[CHI 2001a] CHIEN, S.-Y.; TSOTRAS, V. J.; ZANIOLO, C. Efficient Management of Multiversion Documents by Object Referencing. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 27., 2001, Roma, Italy. **Proceedings...** San Francisco: Morgan Kaufmann, 2001. p.291–300.

[CHO 86] CHOU, H. T.; KIM, W. A Unifying Framework for Version Control in a CAD Environment. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, VLDB, 12., 1986, Kyoto, Japan. **Proceedings...** San Francisco: Morgan Kaufmann, 1986. p.336–344.

[CLA 99] CLARK, J. **XSL Transformations (XSLT) Version 1.0 - W3C Recommendation**. Disponível em: <http://www.w3.org/TR/xslt>. Acesso em: out. 2001.

[CLA 99a] CLARK, J.; DEROSE, S. **XML Path Language (XPath) Version 1.0 - W3C Recommendation**. Disponível em: <http://www.w3.org/TR/xpath>. Acesso em: out. 2002.

[CLE 2002] CLEMM, G. et al. **RFC-3253**: versioning extensions to WebDAV (Web distributed authoring and versioning). Disponível em: <http://www.ietf.org/rfc/rfc3253.txt>. Acesso em: set. 2002.

[CON 96] CONRADI, R.; WESTFECHTEL, B. Configuring Versioned Software Products. In: INTERNATIONAL WORKSHOP ON SOFTWARE CONFIGURATION MANAGEMENT, ICSE, 6., 1996, Berlin, Germany. **Proceedings...** Berlin: Springer, 1996. p.88–109. (Lecture Notes in Computer Science, v.1167).

[CON 98] CONRADI, R.; WESTFECHTEL, B. Version Models for Software Configuration Management. **ACM Computing Surveys (CSUR)**, New York, v.30, n.2, p.232–282, June 1998.

[DEL 87] DELISLE, N.; SCHWARTZ, M. Context - a Partitioning Concept for Hypertext Enviroment. **Transactions on Office Information Systems**, [S.l.], v.15, n.2, p.168–186, Apr. 1987.

[DER 2001] DEROSE, S.; MALER, E.; ORCHARD, D. **XML Linking Language (XLink) Version 1.0 - W3C Recommendation**. Disponível em: <http://www.w3.org/TR/xlink/>. Acesso em: out. 2001.

[EST 95] ESTUBLIER, J.; CASALLAS, R. The Adele Configuration Manager. In: TICHY, W. F. (Ed.). **Trends in Software**. England: John Wiley & Sons, 1995. p.99-139.

[FIE 99] FIELDING, R. et al. **RFC-2616**: hypertext transfer protocol – HTTP/1.1. Disponível em: <ftp://ftp.isi.edu/in-notes/rfc2616.txt>. Acesso em: dez. 2001.

[GOL 97] GOLDMAN, R.; WIDOM, J. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In: INTER-

NATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 23., 1997, Athens, Greece. **Proceedings...** San Francisco: Morgan Kaufmann, 1997. p.436–445.

[GOL 99] GOLDMAN, R.; WIDOM, J. Approximate DataGuides. In: WORKSHOP ON QUERY PROCESSING FOR SEMISTRUCTURED DATA AND NON-STANDARD DATA FORMATS, 1999, Jerusalem, Israel. **Proceedings...** [S.l.: s.n.], 1999.

[GOL 95] GOLENDZINER, L. G. **Um Modelo de Versões para Bancos de Dados Orientados a Objetos**. 1995. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

[GOL 99a] GOLLAND, Y. et al. **RFC-2518**: HTTP extensions for distributed authoring - WebDAV. Disponível em: <ftp://ftp.isi.edu/in-notes/rfc2518.txt>. Acesso em: dez. 2001.

[GRA 98] GRALA, A. **GDOC**: Um Sistema para Armazenamento e Autoria de Documentos Através de Web Browsers. 1998. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

[GRA 97] GRALA, A.; HEUSER, C. A. GDOC: A System for Storage and Authoring of Documents Through Web Browsers. In: INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY, SCCC, 17., 1997, Valpariso, Chile. **Proceedings...** Los Alamitos: IEEE Computer Society, 1997. p.115–124.

[HAA 92] HAAKE, A. CoVer: A Contextual Version Server for Hypertext Applications. In: EUROPEAN CONFERENCE ON HYPERTEXT TECHNOLOGY, ECHT, 8., 1992, Milan, Italy. **Proceedings...** New York: ACM Press, 1992. p.43–52.

[HAA 94] HAAKE, A. Under CoVer: The Implementation of a Contextual Version Server for Hypertext Applications. In: EUROPEAN CONFERENCE ON HYPERTEXT TECHNOLOGY, ECHT, 9., 1994, Edinburgh, Scotland. **Proceedings...** New York: ACM Press, 1994. p.81–93.

[HAA 93] HAAKE, A.; HAAKE, J. M. Take CoVer: Exploiting Version Support in Cooperative Systems. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, INTERCHI, 1993, Amsterdam, Netherlands. **Proceedings...** New York: ACM Press, 1993. p.406–413.

[HAA 96] HAAKE, A.; HICKS, D. VerSE: Towards Hypertext Versioning Styles. In: ACM CONFERENCE ON HYPERTEXT, 7., 1996, Washington, DC, USA. **Proceedings...** New York: ACM Press, 1996. p.224–234.

[HAL 88] HALASZ, F. G. Reflections on Notecards: seven issues for the next generation of hypermedia systems. **Communications of the ACM**, New York, v.31, n.7, p.836–852, July 1988.

[HIC 98] HICKS, D. L. et al. A Hypermedia Version Control Framework. **ACM Transactions on Information Systems, TOIS**, New York, v.16, n.2, p.127–160, Apr. 1998.

[KAN 2002] KANOMATA, Y.; FEISE, J. **DAV Explorer**. Disponível em: <http://www.ics.uci.edu/webdav/>. Acesso em: set. 2002.

[KAT 90] KATZ, R. H. Toward a Unified Framework for Version Modeling in Engineering Databases. **ACM Computing Surveys (CSUR)**, New York, v.22, n.4, p.375–409, Dec. 1990.

[KAY 2000] KAY, M. **XSLT Programmer's Reference**. UK: Wrox Press, 2000. 778p.

[LAR 2000] LARMAN, C. **Utilizando UML e Padrões**: uma introdução à análise e ao projeto orientados a objetos. Porto Alegre: Bookman, 2000. 494p.

[MAG 96] MAGNUSSON, B.; ASKLUND, U. Fine Grained Version Control of Configurations in COOP/Orm. In: INTERNATIONAL WORKSHOP ON SOFTWARE CONFIGURATION MANAGEMENT, ICSE, 6., 1996, Berlin, Germany. **Proceedings...** Berlin: Springer, 1996. p.31–48. (Lecture Notes in Computer Science, v.1167).

[MAN 2000] MANBER, U.; PATEL, A.; ROBISON, J. Experience with Personalization on Yahoo! **Communications of the ACM**, New York, v.43, n.8, p.35–39, Aug. 2000.

[MAR 2002] MARSH, J.; ORCHARD, D. **XML Inclusions (XInclude) Version 1.0 - W3C Candidate Recommendation**. Disponível em: <http://www.w3.org/TR/xinclude/>. Acesso em: out. 2002.

[MOR 2002] MORO, R. G.; HEUSER, C. A. Uma Arquitetura para Versionamento de Conteúdo e Formatação de Sites Web. In: WORKSHOP IBEROAMERICANO DE INGENIERÍA DE REQUISITOS Y DESARROLO DE AMBIENTES DE SOFTWARE, 5., 2002, La Havana, Cuba. **Memórias...** La Habana: Universidad de la Habana, 2002. p.36–44.

[MOR 2001] MORO, R. G.; HEUSER, C. A.; SANTOS, C. S. dos. Usando Versões e XML no Gerenciamento de Conteúdo Web. In: SEMINÁRIO INTERNACIONAL DE GESTÃO DO CONHECIMENTO/GESTÃO DE DOCUMENTOS, 4., 2001, Curitiba, Brasil. **Anais...** Curitiba: PUC-PR/CITS, 2001. v.2, p.117–134.

[MUN 93] MUNCH, B. P. et al. Uniform Versioning: The Change-Oriented Model. In: INTERNATIONAL WORKSHOP ON SOFTWARE CONFIGURATION MANAGEMENT (PREPRINT), 4., 1993, Baltimore, Maryland, USA. **Proceedings...** [S.l.: s.n.], 1993. p.188–196.

[NÁV 96] NÁVRAT, P.; BIELIKOVÁ, M. Knowledge-Controlled Version Selection in Software Configuration Management. **Software - Concepts and Tools**, [S.l.], v.17, n.1, p.40–48, Mar. 1996.

[NOR 98] NORONHA, M. A.; GOLENDZINER, L. G.; SANTOS, C. S. dos. Extending a Structured Document Model with Version Control. In: INTERNATIONAL DATABASE ENGINEERING AND APPLICATIONS SYMPOSIUM, IDEAS, 1998, Cardiff, Wales, U.K. **Proceedings...** Los Alamitos: IEEE Computer Society, 1998. p.234–242.

[NOR 98a] NORONHA, M. **Uma Proposta de Suporte a Versões em Documentos Estruturados**. 1998. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

[NOR 98b] NORONHA, M.; GOLENDZINER, L.; SANTOS, C. S. dos. Compartilhamento de Componentes com Versões em Documentos Estruturados. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 12., 1998, Maringá. **Anais...** Maringá: Universidade Estadual de Maringá, 1998. p.319–333.

[OPE 2002] OPEN MOBILE ALLIANCE LTD. **WAP White Papers**. Disponível em: <http://www.wapforum.org/what/whitepapers.htm>. Acesso em: maio 2002.

[ØST 92] ØSTERBYE, K. Structural and Cognitive Problems in Providing Version Control for Hypertext. In: EUROPEAN CONFERENCE ON HYPERTEXT TECHNOLOGY, ECHT, 8., 1992, Milan, Italy. **Proceedings...** New York: ACM Press, 1992. p.33–42.

[RAG 99] RAGGETT, D.; HORS, A. L.; JACOBS, I. **HTML 4.01 Specification - W3C Recommendation**. Disponível em: <http://www.w3.org/TR/html4/>. Acesso em: out. 2001.

[SAN 2000] SANTOS, C. P. **Configuração de Documentos Versionados**. 2000. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

[SAN 99] SANTOS, C. P.; SANTOS, C. S. dos. Configuration of Versioned Documents. In: INTERNATIONAL CONFERENCE OF THE CHILEAN COMPUTER SCIENCE SOCIETY, SCCC, 19., 1999, Talca, Chile. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p.53–61.

[SCH 98] SCHECHTER, S.; KRISHNAN, M.; SMITH, M. D. Using Path Profiles to Predict HTTP Requests. **Computer Networks**, Amsterdam, v.30, n.1-7, p.457–467, Apr. 1998.

[SMI 88] SMITH, J.; WEISS, S. Hypertext. **Communications of the ACM**, New York, v.31, n.7, p.816–819, July 1988.

[SOA 95] SOARES, L. F. G.; RODRIGUES, N. L. R.; CASANOVA, M. A. Nested Composite Nodes and Version Control in an Open Hypermedia System. **Information Systems**, Oxford, v.20, n.6, p.501–519, Sept. 1995.

[STE 2002] STEIN, G. **WebDAV Resources**. Disponível em: <http://www.webdav.org>. Acesso em: out. 2002.

[SUN 2002] SUN MICROSYSTEMS INC. **Java Tutorial**. Disponível em: <http://java.sun.com/docs/books/tutorial/>. Acesso em: jul. 2002.

[TIC 85] TICHY, W. F. RCS - A System for Version Control. **Software - Practice and Experience**, Sussex, England, v.15, n.7, p.637–654, July 1985.

[WES 96] WESTFECHTEL, B. A Graph-Based System for Managing Configurations of Engineering Design Documents. **International Journal of Software Engineering and Knowledge Engineering**, [S.l.], v.6, n.4, p.549–583, Dec. 1996.

[WES 2001] WESTFECHTEL, B.; MUNCH, B. P.; CONRADI, R. A Layered Architecture for Uniform Version Management. **IEEE Transactions on Software Engineering**, New York, v.27, n.12, p.1111–1133, Dec. 2001.

[WHI 98] WHITEHEAD, E. J.; WIGGINS, M. WebDAV: IETF Standard for Collaborative Authoring on the Web. **IEEE Internet Computing**, Los Alamitos, v.2, n.5, p.34–40, Sept. 1998.

[ZEL 95] ZELLER, A. A Unified Version Model for Configuration Management. **SIGSOFT Software Engineering Notes**, New York, v.20, n.4, p.151–160, Oct. 1995.

[ZEL 97] ZELLER, A.; SNELTING, G. Unified Versioning Through Feature Logic. **ACM Transactions on Software Engineering and Methodology, TOSEM**, New York, v.6, n.4, p.398–441, Oct. 1997.