SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 23.04.2004

Assinatura: Ana Paula Jampai Freire

Geração automática de aplicações Web para aplicações de captura e acesso

Andrea Rodrigues de Andrade

Orientadora: Profa. Dra. Maria da Graça Campos Pimentel

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP – São Carlos
Abril/2004

A Comissão Julgadora:

Profa. Dra. Maria da Graça Campos Pimentel

Profa. Dra. Maria Cristina Ferreira de Oliveira

Prof. Dr. José Valdeni de Lima

Dedicatória

*Aos meus pais pelo amor e incentivo ao estudo.
Ao meu namorado pelo apoio e paciência.*

Agradecimentos

Primeiramente, agradeço a minha orientadora Graça Pimentel pelo apoio, preocupação e compreensão, além dos tantos ensinamentos passados.

À Alessadra, Toño, Renan, Billy, Patrão, Renato, Valter, Laércio e aos outros amigos do Intermídia, minha sincera gratidão pelo apoio recebido, pela constante disposição em ajudar e pela ótima convivência.

Impossível não citar o esforço dos meus pais e impossível não agradecê-los: muito obrigada!

Sou profundamente grata ao meu namorado, que sempre me confortou e me apoiou nas minhas decisões.

Gostaria também de deixar meus agradecimentos às instituições públicas de pesquisa, CNPq e FAPESP, que apóiam o Projeto InCA-SERVE e à CAPES que apoiou este trabalho.

Finalmente, agradeço a Deus pelo constante cuidado e socorro nas horas mais difíceis. Que Ele abençoe a todos.

Resumo

Um dos principais focos de pesquisa na área de computação ubíqua é o suporte à construção de aplicações de captura e acesso. Tais aplicações automatizam o processo de captura de experiências ao vivo e a geração de hiperdocumentos associados para o acesso à informação que foi capturada. Aplicações de captura e acesso apresentam requisitos que dificultam a sua construção e evolução, tal como a demanda por suporte a variados dispositivos durante a captura assim como diversos formatos para os documentos resultantes e um alto grau de automação na geração desses documentos. Nesse contexto, o projeto InCA-SERVE foi proposto para suportar o desenvolvimento de tais aplicações por meio de um conjunto de infra-estruturas e de serviços. O objetivo do trabalho reportado nesta dissertação é complementar o projeto InCA-SERVE com relação à visualização da informação capturada. Baseada em requisitos de aplicações de captura e acesso em particular, e em conceitos estabelecidos de metodologias de projeto hipermídia em geral, uma nova infra-estrutura chamada wVIEW foi desenvolvida. wVIEW suporta a geração automática de aplicações *Web* que permitem a geração dinâmica de documentos de apresentação para o conteúdo capturado por aplicações de captura e acesso.

Abstract

A main research focus in the area of ubiquitous computing is the support to building capture and access applications, which automate the process of capturing live experiences and the generation of associated hyperdocuments for later review. Capture and access applications present requirements that make their construction and evolution difficult, such as the demand for supporting many devices during capture as well as many formats in the resulting documents and a high degree of automation in the generation of those documents. In this context, the InCA-SERVE Project was proposed, supporting the development of such applications by means of a set of infrastructures and services. The aim of the work reported in this dissertation is to complement the InCA-SERVE Project with respect to the visualization of captured information. Based on requirements of capture and access applications in particular, and on well-known concepts from methods for hypermedia design in general, a new infrastructure called wVIEW was developed. wVIEW supports the automatic generation of Web applications that allow the dynamic generation of documents presenting the contents captured by capture and access applications.

Sumário

Sumário	x
Lista de Figuras	xi
1 Introdução	1
2 Computação Ubíqua	5
2.1 Interfaces Naturais	6
2.2 Computação Ciente de Contexto	6
2.3 Captura e Acesso de Experiências	7
2.3.1 Aplicações de Captura e Acesso	8
2.3.2 Exemplos de Aplicações de Captura e Acesso	11
2.3.3 XML e Aplicações de Captura e Acesso	14
3 Projeto InCA-SERVE	17
3.1 Infra-estruturas e Serviços InCA-SERVE	18
3.1.1 Infra-estrutura INCA	18
3.1.2 Infra-estrutura xINCA	19
3.1.3 Serviço StRES	20
3.1.4 Infra-estrutura iVIEW	21
3.2 Relação entre Infra-estruturas e Serviços InCA-SERVE	24
3.3 Sistema iClass	24
4 Tecnologias e Metodologias para Aplicações Web	31
4.1 Tecnologias para Aplicações Web	33
4.1.1 Framework Cocoon	34
4.2 Metodologias para Aplicações Web	37
4.2.1 Infra-estrutura para Aplicações Web	40
5 Infra-estrutura wVIEW	45
5.1 Visão Geral da Infra-estrutura wVIEW	46
5.2 Processo de Transformação wVIEW	47

5.2.1	Modelo Conceitual (CM)	50
5.2.2	Modelo Navegacional (NM)	54
5.2.3	Modelo de Apresentação (PM)	57
5.3	Aplicação Web Gerada	59
5.4	Pipeline da Infra-estrutura wVIEW	62
5.5	Comparação com Trabalhos Relacionados	62
6	Conclusões	67
6.1	Contribuições	68
6.2	Trabalhos Futuros	69
6.3	Considerações Finais	69
	Referências	76

Lista de Figuras

2.1	Dispositivos ubíquos. (a) <i>Pad</i> . (b) PDA. (c) <i>Tablet</i> . (d) <i>Laptop</i> . (e) Lousa eletrônica. (f) <i>Mímio</i>	5
2.2	eClass. (a) Sala de aula equipada. (b) Hiperdocumento gerado automaticamente.	11
2.3	StuPad. (a) Ambiente de captura. (b) Interface de acesso.	12
2.4	SAAMPad. (a) Projetistas em reunião. (b) Interface de acesso. . . .	13
2.5	DUMMBO. (a) Interface de captura. (b) Interface de acesso.	13
3.1	Entidades e componentes INCA. (a) Locais. (b) Distribuídos.	19
3.2	Comunicação entre componentes xINCA durante duas sessões de captura.	20
3.3	Arquitetura básica do StRES.	21
3.4	Processo de transformação da iVIEW.	22
3.5	Documentos de apresentação do iClass. (a) Interface em SMIL. (b) Interface em XHTML+SMIL. (c) Interface em HTML.	28
3.6	Navegação dos documentos de apresentação do iClass. (a) Índice de disciplinas. (b) Índice de aulas para uma disciplina.	29
5.1	Geração de uma aplicação <i>Web</i> usando a infra-estrutura wVIEW.	46
5.2	Processo de transformação wVIEW que gera os modelos e a aplicação <i>Web</i> correspondente.	48
5.3	Transformações para o modelo conceitual.	51
5.4	Um diagrama de classes como entrada para o processo de transformação wVIEW.	51
5.5	Transformações para o modelo navegacional.	55
5.6	Transformações para o modelo de apresentação.	58
5.7	Execução de uma aplicação <i>Web</i> gerada.	60
5.8	Pipeline da infra-estrutura wVIEW.	63

Introdução

A computação tem sido cada vez mais difundida e tem se tornado presente nos mais diversos ambientes. Esse fato foi vislumbrado por Weiser que ainda previu uma forma menos intrusiva no uso do computador, que exploraria a maneira como o ser humano interage com o mundo (Weiser 1991). Em sua visão, a proliferação de dispositivos computacionais e o paradigma de interação quase transparente demarcariam uma nova fase na evolução da computação, a computação ubíqua.

Um dos focos de pesquisa em computação ubíqua inclui as aplicações de captura e acesso. Essas aplicações auxiliam as atividades humanas registrando a troca de informações que ocorre durante experiências ao vivo como reuniões (Kazman, Abowd, Bass, & Clements 1996) (Brotherton, Abowd, & Truong 1998), por exemplo. Para isso, tais aplicações buscam automatizar o processo de captura de experiências ao vivo e a geração de hiperdocumentos associados que permitem o acesso à informação que foi capturada. Um exemplo pioneiro de aplicação de captura e acesso é o eClass (Abowd 1999b), que captura aulas com dispositivos ubíquos e disponibiliza o conteúdo capturado via *Web* por meio de hiperdocumentos gerados automaticamente.

Existem alguns aspectos que devem ser considerados na construção de aplicações de captura e acesso, principalmente relacionados com sua complexidade. Essas aplicações podem ser consideradas complexas por geralmente possuírem uma natureza distribuída, envolverem dispositivos variados e manipularem informações com características multimídia e evolucionária, provenientes de experiências de diversos domínios (Abowd, Mynatt, & Rodden 2002) (Truong & Abowd 2004). Além disso, um dos principais requisitos para

tais aplicações é a demanda por um alto grau de automação tanto na captura das informações quanto na geração dos documentos associados (Brotherton, Bhalodia, & Abowd 1998).

O ideal seria que, independentemente de sua complexidade e domínio, aplicações de captura e acesso fossem implementadas sem despendar muito esforço e tivessem o conteúdo por elas capturado facilmente armazenado, estendido, recuperado e visualizado.

Nesse contexto, o projeto InCA-SERVE¹ (Pimentel & Abowd 1999) foi proposto para apoiar os processos de captura, acesso, armazenamento, extensão, recuperação e visualização de informação manipulada por aplicações de captura e acesso. Esse projeto suporta a construção de aplicações por meio de um conjunto de infra-estruturas e serviços que fazem uso intensivo de XML e tecnologias afins. A xINCA (Cattelan 2002) é um exemplo de infra-estrutura que suporta o processo de captura de experiências, enquanto que o StRES (Baldochi 2002) é um serviço que permite o armazenamento da informação capturada.

O sistema iClass (Cattelan, Andrade, Penedo, & Pimentel 2002), nova versão do eClass, foi desenvolvido com base no projeto InCA-SERVE fazendo uso da infra-estrutura xINCA e do serviço StRES para capturar e armazenar, respectivamente, informações trocadas em salas de aula. Porém, para a visualização da informação capturada, foram implementados mecanismos específicos para a geração das interfaces de acesso (Andrade 2002).

Considerando que quaisquer outras aplicações baseadas no InCA-SERVE deveriam ter suas interfaces de acesso construídas de modo *ad hoc* como se deu com o iClass, identificou-se a necessidade de uma infra-estrutura que suportasse o processo de visualização da informação capturada. Assim, esta dissertação teve por objetivo principal investigar como suportar a construção de interfaces de acesso para o processo de visualização em aplicações de captura e acesso, do modo mais automático possível.

Uma abordagem interessante para as interfaces de acesso é a utilização de aplicações *Web*. Entretanto, a tarefa de desenvolver aplicações *Web* é bastante complexa e várias metodologias têm sido propostas para suportá-la, tais como HDM (Garzotto, Paolini, & Schwabe 1993), RMM (Isakowitz, Stohr, & Balasubramanian 1995), OOHDM (Rossi, Schwabe, & Guimarães 2001) e UWE (Henicker & Koch 2000).

Geralmente tais metodologias oferecem ferramentas específicas que são associadas a suas primitivas de modelagem e que provêem diferentes níveis de automação ao seu processo de desenvolvimento. Por exemplo, as abordagens

¹Parceria entre o *Georgia Institute of Technology* e o Instituto de Ciências Matemáticas e de Computação, 08/2000–07/2004

de suporte ao projeto hipermidia com o maior nível de automação são Araneus (Merialdo, Atzeni, & Mecca 2003), AutoWeb (Fraternali & Paolini 2000) e WebML (Matera, Maurino, Ceri, & Fraternali 2003).

Considerando pontos comuns de vários métodos de modelagem de aplicações hipermidia da literatura, foi especificada a infra-estrutura wVIEW que, formalizada por uma abordagem declarativa baseada em XML, permite a modelagem e a geração automática de aplicações *Web*. Uma vez que tanto a modelagem quanto a geração ocorrem por meio de transformações XSLT sucessivas, a infra-estrutura wVIEW foi implementada diretamente no *framework* Cocoon (Apache 2003) de publicação de documentos XML. Como resultado, a infra-estrutura wVIEW pode ser utilizada no desenvolvimento de aplicações *Web* de modo geral, e no desenvolvimento de aplicações de captura e acesso de modo particular.

A dissertação está organizada da seguinte maneira: no Capítulo 2, a área de computação ubíqua é apresentada, destacando-se seus principais temas e enfatizando as aplicações de captura e acesso; no Capítulo 3, o projeto InCA-SERVE é abordado juntamente com algumas de suas infra-estruturas e serviços; no Capítulo 4, são discutidas algumas tecnologias e metodologias para o projeto hipermidia de aplicações *Web*; no Capítulo 5, é detalhada a infra-estrutura wVIEW desenvolvida; e, no Capítulo 6 são revisitadas as contribuições e discutidos trabalhos futuros.

Computação Ubíqua

A história dos sistemas computacionais é marcada por transições de tecnologia sempre acompanhadas por novos paradigmas de interação, como se pode constatar na transição do *mainframe* para o computador pessoal. Weiser introduziu o conceito de computação ubíqua ao vislumbrar uma nova transição em um futuro muito próximo (Weiser 1991).

Em sua visão, Weiser previu que o paradigma teclado-*mouse-display* seria quebrado pela proliferação de dispositivos computacionais de variados tamanhos e propósitos: pequenos e médios de uso pessoal e grandes de uso coletivo. De fato, essa proliferação pode começar a ser notada com os dispositivos já relativamente comuns hoje (ver Figura 2.1), como *pads* e PDAs (pequenos), *tablets* e *laptops* (médios) e lousas eletrônicas e *mímios* (grandes).

Figura 2.1: Dispositivos ubíquos. (a) *Pad*. (b) *PDA*. (c) *Tablet*. (d) *Laptop*. (e) Lousa eletrônica. (f) *Mímio*.

Embora esses dispositivos ubíquos estejam cada vez mais presentes, o esforço de interação por parte do usuário vem sendo minimizado pois tem explorado o modo como o ser humano interage com o mundo. Isso denota um novo paradigma de interação, no qual a tecnologia torna-se transparente ao usuário. Assim, recursos computacionais são incorporados aos ambientes

alterando muito pouco o comportamento das pessoas e provendo facilidades como informações e serviços em qualquer momento e em qualquer lugar para auxiliar as atividades humanas.

Weiser define computação ubíqua como a tecnologia integrada ao ambiente, porém de forma não intrusiva (Weiser 1993). Para prover transparência aos dispositivos computacionais, existe a necessidade de novas aplicações, geralmente associadas aos três principais temas apontados na literatura (Abowd & Mynatt 2000): (a) interfaces naturais; (b) computação ciente de contexto; e, (c) captura e acesso de experiências. A pesquisa na área de computação ubíqua aborda principalmente esses três temas que, embora sejam tratados separadamente a seguir, têm forte relação entre si.

2.1 Interfaces Naturais

A computação ubíqua tem trazido um novo paradigma de interação mais natural que substitui a metáfora do *desktop* do computador pessoal. A idéia é aproximar a interação usuário-computador à interação usuário-mundo, que se dá basicamente pela fala, gestos e escrita. As interfaces naturais são interfaces das aplicações em dispositivos ubíquos baseadas nesse novo paradigma.

Alguns trabalhos têm resultado em interfaces com interação por voz (Hauptmann 1995), escrita manual (Schilit, Golovchinsky, & Price 1998) e no uso de objetos físicos para manipular artefatos eletrônicos, criando as interfaces tangíveis (Ishii & Ullmer 1997). O desenvolvimento de interfaces naturais como essas enfrenta desafios relacionados com os tipos dos dados e o tratamento de erros. Por exemplo, na escrita manual nem sempre os traços que são desenhados pelo usuário devem ser convertidos para texto e quando convertidos, corretamente ou não, devem ser interpretados para se identificar erros, por exemplo, de ortografia. Se o reconhecimento for incorreto, provavelmente, os erros identificados não serão verossímeis.

2.2 Computação Ciente de Contexto

Informações contextuais são informações relevantes para o usuário que caracterizem a situação de, por exemplo, uma pessoa, lugar ou objeto (Dey 2001). A computação ciente de contexto investiga o desenvolvimento de aplicações que utilizem informações contextuais para facilitar as atividades humanas. Essas aplicações, chamadas de sensíveis ao contexto, coletam informações contextuais através de dispositivos ubíquos e podem mudar seu comportamento de acordo com o mesmo, possivelmente explorando relacionamentos entre as informações contextuais coletadas.

Active Badge (Want, Hopper, Falcão, & Gibbons 1992) e PARCTab (Want, Schilit, Adams, Gold, Petersen, Ellis, Goldberg, & Weiser 1995) foram aplicações pioneiras sensíveis ao contexto de localização que implementavam facilidades como transferências de chamadas telefônicas e mapas com atualizações automáticas das localizações dos usuários.

As informações contextuais podem ser relacionadas a cinco dimensões, conhecidas como 5 Ws (Abowd & Mynatt 2000): *Who* (identifica as pessoas envolvidas na atividade), *What* (identifica o que as pessoas estão fazendo), *Where* (identifica onde estão as pessoas), *When* (identifica quando ocorreu a atividade) e *Why* (identifica o porquê das ações na atividade).

2.3 Captura e Acesso de Experiências

Várias experiências do dia-a-dia das pessoas, como reuniões ou aulas, tratam-se basicamente de troca de informações. Dada a limitação de memorização das atividades, o ser humano emprega tempo e esforço para registrar o conteúdo dessas experiências através de anotações ou gravação de áudio, por exemplo.

O tema de captura e acesso de experiências¹ diz respeito ao desenvolvimento de aplicações que automatizem o processo de captura e disponibilizem o conteúdo capturado para acesso posterior. Assim, durante a experiência ao vivo as pessoas ficam livres de registrar as informações e podem se dedicar totalmente à experiência em si. A natureza de uma experiência a ser capturada depende de sua formalidade e privacidade. Por exemplo, um encontro casual pode ser considerado uma experiência informal e privada enquanto que uma conferência internacional pode ser considerada uma experiência formal e pública.

Para que ocorra uma sessão de captura, os dispositivos ubíquos devem estar embarcados² no ambiente onde se passará a experiência. Assim, as aplicações coletam simultaneamente diversos fluxos de dados, preservando suas inter-relações. O conteúdo capturado da experiência pode possuir características multimídia, a serem respeitadas na fase de acesso, geralmente por meio de hiperdocumentos *Web*.

Pesquisas relacionadas ao tema de captura e acesso de experiências datam de 1985 com uma aplicação para captura de áudio de chamadas telefônicas que previa acesso posterior com facilidades como resumo de conversas (Sch-

¹No texto é utilizada a forma "captura e acesso" para referenciar ao tema como um todo. O termo *captura* é utilizado para significar o processo de captura de informação em uma experiência. O termo *acesso* corresponde ao acesso posterior à informação capturada de uma experiência.

²O termo utilizado em inglês é *embedded*.

mandt & Arons 1985). No domínio de reuniões, o Tivoli (Minneman, Harrison, Janssen, Kurtenbach, Moran, Smith, & Van Melle 1995) e o Dolphin (Streitz, Geibler, Haake, & Hol 1994) foram trabalhos pioneiros. No domínio educacional, trabalhos pioneiros foram reportados pelo *Georgia Institute of Technology* (Abowd 1999b), como o eClass.

2.3.1 Aplicações de Captura e Acesso

Weiser já previa que o suporte à construção de aplicações seria o ponto central da pesquisa na área de computação ubíqua (Weiser 1993). Entre as aplicações alvo de pesquisa mais relevantes estão as aplicações de captura e acesso por englobarem os principais temas da computação ubíqua: realizam a captura e o acesso de experiências na maioria das vezes explorando o uso de interfaces naturais e informações de contexto.

Aplicações de captura e acesso podem ser definidas como aplicações que automatizam o processo de captura de experiências e a geração de documentos de apresentação associados para o acesso posterior. Por exemplo, a experiência pode ser uma aula, cujas informações a serem capturadas serão áudio e anotações em *slides* utilizando uma lousa eletrônica. Dado o horário, no momento da captura, o sistema infere qual é a disciplina e outras informações contextuais. Logo após a captura, o conteúdo da experiência é disponibilizado para acesso posterior via *Web* através de hiperdocumentos gerados de maneira automática. O acesso por parte do usuário também pode ser baseado em informações contextuais; por exemplo, conhecendo o aluno, o sistema exibe apenas o conteúdo capturado das disciplinas em que está matriculado.

As aplicações de captura e acesso podem capturar as mais diversas experiências e tipos de dados com os mais variados dispositivos ubíquos. Além disso, a captura³ pode ser realizada de diferentes formas – por exemplo, se local ou distribuída – o que acontece também para o acesso⁴.

Apesar das diferentes abordagens possíveis, algumas generalizações podem ser feitas de modo a suportar a construção de aplicações de captura e acesso. Uma generalização que pode ser feita é a definição do espaço de projeto dessas aplicações em cinco dimensões, respondendo as seguintes questões (Truong, Abowd, & Brotherton 2001):

- *Quem* são os usuários na captura e no acesso?
- *O que* é capturado e acessado?

³O termo *captura* é utilizado para significar o processo de captura de informação em uma experiência.

⁴O termo *acesso* corresponde ao acesso posterior à informação capturada de uma experiência.

- *Quando* a captura e o acesso ocorrem?
- *Onde* a captura e o acesso ocorrem?
- *Como* são feitos a captura e o acesso?

Essas questões podem ser respondidas para todas as aplicações de captura e acesso independente de seu contexto pois, por exemplo, a captura e o acesso sempre serão feitos para alguém (*Quem*), mesmo que seja para uma outra aplicação consumidora. Além disso, sempre algum artefato será capturado e posteriormente acessado (*O que*), do contrário não haveria a necessidade de uma aplicação para tal. Ainda tanto a captura como o acesso vão ocorrer em um momento (*Quando*) e em um lugar (*Onde*) possivelmente de modos diferentes (*Como*), por exemplo utilizando-se de variados dispositivos ubíquos.

Embora as aplicações realmente façam uso de diferentes dispositivos, geralmente apresentam funcionalidades recorrentes (Cattelan 2002), como:

- *Áudio*: captura de áudio;
- *Vídeo*: captura de vídeo;
- *Chat*: captura de troca de mensagens textuais entre os usuários;
- *Web browsing*: captura de navegação na *Web* durante uma experiência;
- *Whiteboard*: captura de interação com uma lousa eletrônica por meio de escrita manual, sendo que o usuário desenha traços para representar anotações.

Tomando essas funcionalidades por base, pode-se prever quais fluxos de dados as aplicações suportarão incluindo: fluxo de áudio (*Áudio*), fluxo de vídeo (*Vídeo*), fluxo de texto (*Chat* e *Web browsing*) e fluxo de imagem (*Whiteboard*). Assim, os possíveis tipos de dados capturados pelas aplicações serão áudio, vídeo, texto e imagem. Isso caracteriza a sessão de captura como multimídia, pois envolve mídias contínuas (áudio e vídeo) e discretas (texto e imagem) sincronizadas entre si. A sincronização se refere às relações temporais, espaciais e de conteúdo dos objetos de mídia.

Geralmente cada nova experiência tem relação com outras que já ocorreram, por isso o conteúdo multimídia capturado sempre está em evolução ao longo do tempo. Assim, a informação proveniente da captura de experiências trata-se de *informação evolucionária* (Pimentel, Kerimbaev, Abowd, & Guzdial 2001), que cresce dependendo da aplicação pode crescer com muita frequência e rapidez. Isso pode tornar a tarefa de interpretar os dados lenta e desgastante para o usuário. Para auxiliar a interpretação, existem técnicas de visualização

de informação, como o uso de representações visuais e interativas dos dados, que podem ser usadas para diminuir a sobrecarga cognitiva do usuário (Card, Mackinlay, & Shneiderman 1999).

Para prover suporte aos processos de captura e de acesso a essa informação multimídia e evolucionária, pode-se notar algumas similaridades arquiteturais entre as aplicações (Truong & Abowd 2002), pois partes do sistema são responsáveis por:

- *Capturar a informação;*
- *Armazenar a informação capturada;*
- *Integrar os fluxos de dados;*
- *Converter os dados para diferentes formatos quando necessário;*
- *Prover acesso à informação capturada.*

Com relação à captura e ao acesso de experiências em geral, Abowd sugere uma estruturação do problema em quatro fases (Abowd 1999a): *Pré-produção*, *Gravação ao vivo*, *Pós-produção* e *Acesso*. A primeira fase de Pré-produção está relacionada com a preparação para a sessão de captura. Na fase de Gravação ao vivo, durante a qual a sessão é iniciada e encerrada, ocorre a captura (*Capturar*) e o armazenamento (*Armazenar*) da informação. A fase seguinte de Pós-produção é quando acontece a integração (*Integrar*) e a conversão (*Converter*) dos dados e assim os documentos associados são gerados. Na fase final de Acesso, é provido o acesso (*Prover acesso*) à informação capturada.

As fases de captura e de acesso também podem ser notadas sob a ótica dos usuários. Por exemplo, a sessão a ser capturada é uma aula e, na Pré-produção, o professor prepara o material a ser apresentado. Durante a aula, na fase de Gravação ao vivo, o professor e os alunos anotam sobre o material. Quando a aula é encerrada iniciando a Pós-produção, o professor faz as transcrições de suas anotações capturadas. Em seguida, os alunos visitam o conteúdo discutido em aula na fase de Acesso.

Muitas vezes o termo *acesso* é utilizado em diferentes momentos em aplicações de captura e acesso, ou seja, durante e depois de uma sessão. O acesso intra-sessão é o acesso de informações que ocorre durante a sessão de captura, por exemplo, a *whiteboard* do professor está capturando e a *whiteboard* do aluno está acessando instantaneamente.

Logo depois da captura, quando as informações já estão armazenadas, existem dois significados com diferenças sutis: o acesso ao conteúdo armazenado da sessão e o acesso do usuário através de interfaces. O acesso do usuário se utiliza e depende do acesso ao conteúdo armazenado para a visualização das sessões e ainda envolve a apresentação de uma sessão selecionada.

Geralmente, o acesso do usuário é implementado pelas aplicações de captura e acesso por meio de aplicações *Web*, que acessam o conteúdo armazenado e geram os documentos de apresentação, permitindo a navegação entre os mesmos. Dessa maneira, a informação capturada torna-se mais acessível uma vez que pode ser acessada em qualquer lugar, a qualquer momento e por qualquer pessoa — embora possa haver restrições como a autorização de acesso para apenas alguns usuários, por exemplo.

2.3.2 Exemplos de Aplicações de Captura e Acesso

A seguir, são apresentadas algumas aplicações de captura e acesso: eClass, StuPad, SAAMPad e DUMMBO. As duas primeiras são do domínio de aulas, sendo que o StuPad captura sessões privadas e o eClass, públicas. As duas últimas são do domínio de reuniões, sendo que o DUMMBO captura sessões informais e o SAAMPad, formais.

eClass

O eClass, cujo nome original é Classroom 2000, trata-se de uma aplicação de captura e acesso para o domínio de aulas, sendo que o conteúdo apresentado durante a aula é registrado por dispositivos ubíquos e disponibilizado na *Web* através de hiperdocumentos gerados automaticamente (Abowd 1999b) (Pimentel, Kerimbaev, Abowd, & Guzdial 2001). Um ambiente de computação ubíqua é implementado para permitir a captura de material multimídia em uma sala de aula convencional equipada, como mostrado na Figura 2.2 (a), com uma lousa eletrônica (A), câmera de vídeo (B), microfones no teto (C) e projetores (D e E).

Figura 2.2: eClass. (a) Sala de aula equipada. (b) Hiperdocumento gerado automaticamente.

Após a aula, as informações são armazenadas em uma base de dados e, a

partir da mesma, *scripts* geram hiperdocumentos que podem ser visitados. Um exemplo de um documento gerado é apresentado na Figura 2.2 (b). No frame da direita, existe a sequência de *slides* anotados através da lousa eletrônica, criados previamente ou durante a aula. À esquerda, tem-se uma linha do tempo, contendo ligações que indexam as mídias com relação ao tempo, e abaixo, o vídeo.

StuPad

O StuPad, de **Student Notepad**, é uma aplicação integrada ao eClass, que captura anotações dos alunos e as disponibiliza na *Web* para acesso posterior (Truong, Abowd, & Brotherton 2001). Utilizando *tablets* em rede, os alunos fazem suas anotações particulares sobre o material apresentado na lousa eletrônica do professor, como pode ser visualizado na Figura 2.3 (a). As anotações são armazenadas identificando o seu autor. Assim, no acesso, o aluno consulta o material com suas anotações pessoais. Na Figura 2.3 (b) é mostrada a interface de acesso do StuPad.

(a)

(b)

Figura 2.3: StuPad. (a) Ambiente de captura. (b) Interface de acesso.

SAAMPad

O SAAMPad, de **Software Architecture Analysis Method Pad**, é uma aplicação de captura e acesso para reuniões de projeto (Kazman, Abowd, Bass, & Clements 1996), baseada no sistema SAAM (**Software Architecture Analysis Method**) que documenta decisões de projeto por meio de cenários elaborados pelos projetistas. Utilizando uma lousa eletrônica (ver Figura 2.4 (a)), o SAAMPad permite anotações sobre os cenários, além de também capturar áudio e vídeo da reunião. Assim, a discussão fica ricamente documentada com o material multimídia capturado. A interface de acesso do SAAMPad (ver Figura

2.4 (b)) permite a revisão das interações da lousa sincronizadas com o áudio e o vídeo.

Figura 2.4: SAAMPad. (a) Projetistas em reunião. (b) Interface de acesso.

DUMMBO

O DUMMBO, de **Dynaminc, Ubiquitous, Mobile Meeting Board**, é uma aplicação de captura e acesso para reuniões informais não-programadas (Brotherton, Abowd, & Truong 1998). Com uma lousa eletrônica, sistema de som e sensores para detectar a presença de pessoas acoplados à lousa, o sistema inicia a captura por uma interação com a lousa ou pela aproximação de pelo menos duas pessoas (ver Figura 2.5 (a)). O acesso é por uma interface Web que faz uso de informações contextuais – quem participou, onde e quando – na busca por uma reunião. Como mostrado na Figura 2.5 (b), as reuniões ficam agrupadas em uma linha do tempo e quando exibidas, as interações na lousa são apresentadas de forma sincronizada com o áudio capturado.

Figura 2.5: DUMMBO. (a) Interface de captura. (b) Interface de acesso.

2.3.3 XML e Aplicações de Captura e Acesso

Os exemplos apresentados diferem quanto ao tipo de experiência capturada, dos dispositivos ubíquos utilizados e dos tipos de dados capturados, além das diferenças encontradas nas formas de captura e de acesso. Mesmo as aplicações de mesmo domínio, como o SAAMPad e o DUMMBO, apresentam várias distinções nesses parâmetros. Logo, as informações geradas por cada uma dessas aplicações possuem uma estrutura variável.

Isso é uma questão relevante para aplicações que precisam ser integradas, como o StuPad com relação ao eClass. A estruturação de dados e a integração de aplicações, muitas vezes necessárias para as aplicações de captura e acesso de um modo geral, podem ser facilitadas pela utilização de XML.

XML⁵ (*Extensible Markup Language*), versão simplificada de SGML (*Standard Generalized Markup Language*), foi definida para tratar limitações de HTML, que possui um conjunto reduzido e fixo de marcações para definir como os dados delimitados serão apresentados em um *Web browser*. A linguagem de marcação extensível XML permite criar elementos e atributos próprios, associando significado específico aos dados, em estrutura hierárquica como uma árvore.

Além de ser extensível e permitir estruturação dos dados, XML foi proposto para ser fácil de entender e de manter, independente de plataforma, modular e ainda permite reuso e fácil integração entre aplicações. Portanto, em função das necessidades das aplicações de captura e acesso, como de dados estruturados e integração de diferentes aplicações, XML mostra-se uma boa abordagem.

Em aplicações de captura e acesso, as sessões capturadas podem ser representadas por documentos XML. Além disso, a maneira como os dados capturados são tratados e utilizados reforça essa decisão de projeto, porque XML inclui uma família de tecnologias para apoiá-la. Por exemplo, XML Schema⁶ pode ser usado para definir o esquema do documento XML de sessão. Assim, os documentos XML de sessão podem ser validados de acordo com esse esquema, que consiste na definição de conteúdo, de estrutura e de tipos de dados. O XML Schema tem a mesma sintaxe do XML e possui ainda um mecanismo de herança que possibilita extensão e reaproveitamento de esquemas.

Depois de validados, os documentos XML de sessão podem ser armazenados em um banco de dados XML. Um banco de dados XML possui um modelo de documentos XML, como um modelo de dados, para armazenar e recuperar os documentos. A organização desses documentos é feita utilizando, geralmente, o conceito de coleção, similar aos diretórios em sistemas de arquivos.

⁵<http://www.w3.org/XML/>

⁶<http://www.w3.org/XMLSchema/>

Os bancos de dados XML suportam linguagens, como XPath, que seleciona conjuntos de nós de documentos de acordo com uma consulta. Exemplos de banco de dados XML são o Xindice ⁷ e o eXist ⁸, que armazenam e indexam documentos XML comprimidos em coleções e possibilitam consultas XPath.

Assim que forem armazenados, os documentos XML de sessão podem ser acessados e processados por meio de transformações para a geração de documentos de visualização das sessões e de apresentação de uma sessão. Essas transformações podem ser definidas por folhas de estilo escritas em XSLT⁹ (*Extensible Stylesheet Language Transformations*), uma linguagem baseada em XML para transformar documentos XML em outros documentos XML.

A linguagem XSLT foi projetada como parte de XSL (*Extensible Stylesheet Language*), que inclui também um vocabulário XML para especificar formatação. A transformação de um documento XML em outro se dá basicamente por um conjunto de *templates* que casam com um subconjunto de nós na árvore do documento XML fonte e descrevem a contribuição desses nós na construção da árvore do documento XML resultante.

Como mostram informação evolucionária, seria interessante que os documentos de visualização de sessões explorassem o uso de representações visuais. Um formato bastante adequado para tais documentos de visualização é o SVG¹⁰ (*Scalable Vector Graphics*), que pode ser obtido por transformação, pois é baseado em XML, e permite criar apresentação de gráficos vetoriais. Através das propriedades do SVG é possível gerar diversos tipos de gráficos bidimensionais, como linhas, círculos e polígonos e incluir preenchimento usando gradiente de cores, controle da opacidade, entre outros.

Os documentos de apresentação gerados por transformações devem respeitar as características multimídia da sessão. Nesse contexto, pode ser utilizada a linguagem SMIL¹¹ (*Synchronized Multimedia Integration Language*), uma linguagem baseada em XML para a integração sincronizada de mídias na Web. A sincronização é especificada em documentos que geram apresentações multimídia ao serem tocados em *players*, como o RealOne e o GRiNS, além de outros.

O modelo de tempo proposto pelo SMIL permite hierarquia de tempo, tempo interativo e relativo e transformação de tempo para animações. A estrutura modular e a semântica de integração da linguagem facilitam o reuso de especificações de tempo e de animação do SMIL em outras linguagens. Um exemplo disso é o XHTML+SMIL¹², que é a união do HTML com esses módulos de tempo

⁷<http://xml.apache.org/xindice/>

⁸<http://exist-db.org/>

⁹<http://www.w3.org/TR/xslt/>

¹⁰<http://www.w3.org/Graphics/SVG/>

¹¹<http://www.w3.org/TR/smil20/>

¹²<http://www.w3.org/TR/XHTMLplusSMIL/>

e animação, implementado no Internet Explorer 5.5 ou superior.

No contexto de aplicações de computação ubíqua, uma aplicação de captura e acesso pode ter todas as suas fases suportadas pela utilização de XML. O projeto InCA-SERVE utiliza esse fato como base para suportar a construção de aplicações de captura e acesso por meio de infra-estruturas e serviços genéricos. **Esse projeto, no qual se insere este trabalho de dissertação, e suas infra-estruturas e serviços são descritos no próximo capítulo.**

Projeto InCA-SERVE

Existem alguns fatores que dificultam o desenvolvimento e o uso das aplicações de captura e acesso, referentes, principalmente, a sua natureza distribuída e ao grande número de dispositivos envolvidos (Abowd, Mynatt, & Rodden 2002) (Truong & Abowd 2004). Além disso, as aplicações de captura e acesso geralmente são desenvolvidas sem atentar para sua evolução e possível reuso para outros domínios.

Outra questão importante a ser considerada no desenvolvimento de aplicações de captura e acesso está relacionada à informação multimídia e evolucionária proveniente das experiências. Essa informação, além de ter o requisito de sincronização entre as mídias, ocupa um grande espaço de armazenamento. Além disso, a informação capturada e armazenada é crescente, tanto pelo fato de, geralmente, não ser excluída, quanto por poder ser estendida.

O ideal seria que, independente do domínio, as aplicações de captura e acesso fossem implementadas e mantidas sem despendar muito esforço e tivessem seu conteúdo facilmente armazenado, estendido, recuperado e visualizado. Nesse contexto, com o intuito de investigar os problemas relacionados à construção de aplicações de captura e acesso, surgiu o projeto InCA-SERVE (InCA, de **I**nfrastructure for **C**apture and **A**ccess e SERVE, de **I**nfrastructure for **S**ore, **E**xtend, **R**etrieve and **V**isualize **E**volutionary **I**nformation) (Pimentel & Abowd 1999).

O projeto InCA-SERVE é uma parceria entre o *Georgia Institute of Technology* (GATech), sob supervisão do Dr. Gregory Abowd, e o Instituto de Ciências Matemáticas e de Computação (ICMC), sob supervisão da Dra. Maria da Graça Campos Pimentel. Esse projeto é financiado por um programa de cooperação

internacional entre a NSF/CISE nos EUA e o CNPq/ProTeM-CC no Brasil (Pimentel, Fortes, & Moreira 2001a), tendo contado também no Brasil com o apoio da FAPESP e da CAPES (Pimentel, Fortes, & Moreira 2001b).

Pode-se dizer que o objetivo principal das aplicações de captura e acesso é facilitar a vida das pessoas que as utilizam. Por sua vez, o objetivo principal do projeto InCA-SERVE é facilitar a vida das pessoas que as constroem. Assim, para oferecer funcionalidades e resolver limitações de aplicações de quaisquer domínios, o projeto InCA-SERVE provê um conjunto de infra-estruturas e serviços de captura, acesso, armazenamento, extensão, recuperação e visualização da informação gerada. A seguir, são apresentadas algumas dessas infra-estruturas e serviços construídas no contexto do projeto.

3.1 *Infra-estruturas e Serviços InCA-SERVE*

Todas as infra-estruturas e serviços InCA-SERVE fazem uso de XML, o que traz uma padronização para o projeto, além das vantagens já discutidas para as aplicações de captura e acesso, como estruturação dos dados e integração das aplicações. Com as infra-estruturas e os serviços, o projeto InCA-SERVE procura atender muitas das necessidades encontradas na construção de aplicações de captura e acesso variadas.

Para preparação de sessões de captura, por exemplo, existe o serviço de autoria AutorE (Sante 2003), que permite criar sessões, inclusive a partir de sessões previamente existentes. No caso de captura de sessões privadas, existe a infra-estrutura INDU que trabalha com plataformas PDAs (Rocha 2004). Com relação a aplicações sensíveis ao contexto, há o serviço Context Kernel baseado nas dimensões de contexto para oferecer armazenamento, extensão e recuperação de informação contextual (Arruda 2002).

Para extensão do conteúdo capturado, existem dois serviços genéricos: o GroupNote (Izeki 2001), para anotações colaborativas, e o WLS (Bulcão 2001), para prover funcionalidades hipermídia, por meio de ligações, a aplicações que não as possuam. O InCA-SERVE ainda conta com outras infra-estruturas e serviços, alguns descritos a seguir.

3.1.1 *Infra-estrutura INCA*

A INCA (**I**nfrastructure for **C**apture and **A**ccess Applications) é uma infra-estrutura genérica escrita em linguagem Java com arquitetura cliente/servidor, baseada nas similaridades das aplicações de captura e acesso: captura, armazenamento, integração, conversão e acesso à informação (Truong & Abowd 2002) (Truong & Abowd 2004).

Estando em uma camada mais baixa, a INCA promove suporte por meio de interfaces e módulos, como *Capturer* e *CaptureModule* no caso da captura. Além disso, dado que as aplicações possuem natureza distribuída e envolvem vários dispositivos, a INCA provê dois níveis de abstração de:

- Comunicação: A entidade de registro (*Registry*), que pode compartilhar informação com outras e cuja localização é bem conhecida, gerencia automaticamente a transferência de dados entre os módulos, mantendo uma lista de quais estão disponíveis em um dado momento. Na Figura 3.1 (Truong & Abowd 2002) são mostradas duas configurações possíveis de comunicação, sendo em (a) entidade e componentes locais e em (b) entidades e componentes distribuídos;
- Informação: Os dados são tratados como grandes objetos binários (*Data-Objects*) com pares de atributos nome-valor que expressam meta-informação e permitem uma representação uniforme sobre os dados.

Figura 3.1: Entidades e componentes INCA. (a) Locais. (b) Distribuídos.

3.1.2 Infra-estrutura xINCA

A xINCA (*Extended Infrastructure for Capture and Access Applications*) é também uma infra-estrutura genérica escrita em linguagem Java com arquitetura cliente/servidor que estende a INCA com base nas funcionalidades recorrentes das aplicações de captura e acesso (Cattelan 2004). Utilizando módulos e abstrações da INCA, a xINCA provê componentes de software reutilizáveis responsáveis pelas funcionalidades de áudio, vídeo, *chat*, *Web browsing* e *whiteboard*.

Assim, para implementar uma aplicação basta combinar componentes desejados, sendo que o desenvolvedor não deixa de ter acesso direto aos recursos da camada INCA, podendo inclusive estender a xINCA ao adicionar novos componentes. Os componentes combinados comunicam-se usando um identificador de sessão único, sob qual os módulos INCA são registrados em tempo de execução em uma entidade *Registry* local ou distribuída. Na Figura 3.2 (Cattelan 2002) é ilustrada a comunicação entre componentes durante duas sessões simultâneas com três aplicações.

Figura 3.2: Comunicação entre componentes xINCA durante duas sessões de captura.

3.1.3 Serviço StRES

Para que a xINCA tenha uma arquitetura extensível, cada componente deve ter um esquema de dados estruturando toda a informação produzida. Além disso, a aplicação deve possuir também um esquema de dados para a sessão que combina todos os esquemas dos componentes. Os esquemas são compilados para um conjunto de classes Java quando submetidos ao mecanismo JAXB (**J**ava **A**rchitecture for **X**ML **B**inding), que permite manipular e criar documentos XML facilmente.

Esse conjunto faz parte do módulo *Storage Module*, que recebe os dados produzidos, os associa aos respectivos objetos das classes e gera um documento XML representando a sessão capturada. Cada documento é enviado a outro módulo, o *Document Manager*, responsável por armazená-lo em um banco de dados XML.

Ambos *Storage Module* e *Document Manager* pertencem ao serviço StRES (**S**toring, **R**etrieving and **E**xtending **S**ervice) (Baldochi 2002), que provê módulos para manipulação de forma transparente da informação produzida por aplicações de captura e acesso. Na Figura 3.3 é ilustrada sua arquitetura.

Cada aplicação é associada a um *Storage Module* (SM) e, opcionalmente, a um *Extension Module* (EM), para extensão do conteúdo com anotações e ligações. O *Document Manager* gerencia todas as operações com o repositório. Dessa maneira, o StRES é um elo entre a captura e o acesso ao disponibilizar um repositório com o conteúdo capturado que pode ser acessado e estendido.

Figura 3.3: Arquitetura básica do StRES.

3.1.4 Infra-estrutura iVIEW

A infra-estrutura iVIEW provê a visualização de informação evolucionária — tal qual a informação capturada por aplicações de captura e acesso — extraída a partir de documentos XML, usando representações gráficas em SVG (Oshiro 2003). As estruturas gráficas da iVIEW permitem que os usuários identifiquem as relações intrínsecas da informação visualizada em grandes quantidades.

Para tanto são utilizadas técnicas de representações visuais e interativas, implementadas por meio de um processo de transformação dos dados. O processo de transformação da iVIEW utiliza um conjunto de documentos e possui alguns passos para processá-los. Esse processo é baseado em XML e usa algumas linguagens relacionadas para:

1. definir a estrutura dos documentos de entrada contendo a informação sobre o *layout* da visualização (XML Schema);
2. descrever os algoritmos que executam os passos necessários para obter a representação gráfica da informação (XSLT); e,
3. prover a representação gráfica da informação (SVG).

O processamento dos documentos ocorre em duas fases. A primeira fase permite que um formato de transformação intermediário seja obtido para uma dada aplicação (*Step 1a* na Figura 3.4). Na segunda fase, a especificação intermediária é processada a fim de gerar uma especificação final da visualização (*Step 2* na Figura 3.4). Para que a representação visual em SVG seja interativa, a iVIEW possui ainda outro passo na primeira fase para gerar um documento JavaScript com funções de interação (*Step 1b* na Figura 3.4).

O processo de transformação é independente do contexto da aplicação e isso só é possível pelo processamento em duas fases. O esquema geral dos documentos e dos passos de processamento da iVIEW é mostrado na Figura 3.4 — os retângulos numerados representam os documentos de entrada e saída que são processados pelos três passos.

Figura 3.4: Processo de transformação da iVIEW.

O desenvolvedor da aplicação é responsável pelo documento 1 (application.xml) que trata-se da fonte de informação da aplicação. Além disso, o desenvolvedor deve especificar o documento 2 (layout.xml) para definir a estrutura da visualização da informação. Os documentos 3 (layout.xsd), 4 (template.xslt) e 7 (templateJS.xslt) são partes fixas da iVIEW e os documentos 5 (application.xslt), 6 (application.svg) e 8 (application.js) são gerados automaticamente. A seguir, os documentos envolvidos no processo de transformação da iVIEW são descritos:

- 1 (application.xml): é um documento XML específico da aplicação contendo a informação a ser visualizada. Trata-se do documento fonte para o qual deseja-se obter uma representação em SVG;
- 2 (layout.xml): é um documento XML contendo a especificação do *layout* da visualização, que deve ser criado com base na estrutura do documento 1 (application.xml);
- 3 (layout.xsd): é um documento XML Schema que contém as regras de construção para definir o documento 2 (layout.xml), fazendo parte da infra-estrutura iVIEW;
- 4 (template.xslt): é um documento XSLT genérico que faz parte da infra-estrutura iVIEW e contém instruções para gerar outro documento XSLT, específico para a aplicação. Para tanto, o documento XSLT genérico toma como entrada a especificação do documento 2 (layout.xml);
- 5 (application.xslt): é um documento XSLT obtido automaticamente e que contém as instruções para gerar uma representação SVG, processando o documento 1 (application.xml);
- 6 (application.svg): é um documento SVG gerado automaticamente e que contém uma representação gráfica do documento 1 (application.xml), de acordo com as definições do documento 2 (layout.xml);
- 7 (templateJS.xslt): é um documento XSLT genérico que faz parte da infra-estrutura iVIEW para gerar um documento JavaScript específico para a aplicação. Para tanto, este documento toma como entrada a especificação do documento 2 (layout.xml);
- 8 (application.js): é um documento JavaScript gerado automaticamente e que contém funções para prover interação do usuário com o documento 6 (application.svg);

No *Step 1a*, a especificação (documento 2) é processada por meio do documento XSLT genérico (documento 4) a fim de gerar as instruções XSLT específicas da aplicação (documento 5). Ainda na mesma fase, o *Step 1b* processa a especificação (documento 2) com o documento XSLT genérico (documento 7), gerando as funções de interação da aplicação (documento 8). Finalizando o processo de transformação, o *Step 2* processa o documento fonte (documento 1) com o documento XSLT específico (documento 5) para gerar a representação gráfica final (documento 6).

3.2 *Relação entre Infra-estruturas e Serviços InCA-SERVE*

As infra-estruturas e serviços têm forte relação entre si e pode-se dizer que são complementares com respeito à construção de aplicações de captura e acesso. Por exemplo, a infra-estrutura xINCA, que estende a infra-estrutura INCA com funcionalidades específicas, é complementada pelo serviço StRES para possibilitar que o conteúdo capturado seja armazenado. No caso, as infra-estruturas e os serviços InCA-SERVE podem ser associados às fases de captura e acesso de experiências como segue:

- Pré-produção: AutorE;
- Gravação ao vivo: INCA, xINCA, INDU, StRES e Context Kernel;
- Pós-produção: GroupNote e WLS;
- Acesso: iVIEW.

Embora todas as fases possuam pelo menos uma infra-estrutura ou serviço que apóie a sua implementação, a fase de acesso demanda serviços adicionais. A única infra-estrutura dessa fase, a iVIEW, fornece apenas a visualização de um conjunto de sessões, não atentando para a apresentação de uma sessão específica e para a navegação entre as sessões. Tanto a apresentação quanto a navegação são necessidades primordiais para o acesso em aplicações de captura e acesso.

Assim, existe uma lacuna no projeto InCA-SERVE — que este trabalho de dissertação visa preencher — quanto à fase de acesso, uma vez que o projeto não oferece suporte para a geração de documentos de apresentação e para a navegação referente a esses documentos. Esse fato pode ser melhor observado com a descrição do sistema iClass a seguir, que foi implementado com base no projeto InCA-SERVE.

3.3 *Sistema iClass*

O sistema iClass trata-se de uma nova versão para o sistema eClass. O iClass está sendo utilizado para capturar aulas no ICMC desde o 2º semestre de 2002. Além disso, foi implantado no GATech e nas Faculdades COC em Ribeirão Preto no ano de 2003. O desenvolvimento do sistema iClass é baseado no projeto InCA-SERVE pois são utilizadas algumas de suas infra-estruturas e serviço: a xINCA e a INCA, para implementar a captura, e o StRES, para implementar o armazenamento (Cattelan, Andrade, Penedo, & Pimentel 2002).

Assim, a captura se dá com os componentes xINCA por meio da infra-estrutura INCA e o armazenamento ocorre com os módulos do serviço StRES.

No caso, o acesso posterior às sessões capturadas e armazenadas pelo iClass é disponibilizado por meio de interfaces que foram desenvolvidas de forma *ad hoc* especificamente para o sistema em questão (Andrade 2002). Embora essas interfaces não sejam baseadas em uma infra-estrutura genérica do projeto InCA-SERVE, elas foram construídas explorando-se o uso de XML dos documentos de sessão para apresentar a aula capturada.

No iClass, o documento XML de sessão é estruturado como ilustrado a seguir.

```
<session id="173" type="2" date="2002.10.03 AD at 11:20:47 BRT">
  <context_info>
    <person>demo</person>
    <place>Lab-4</place>
    <eclass_context>
      <term>2o_2002</term>
      <course_name>Hipermidia</course_name>
      <class_title>www2002_mm_cg_web</class_title>
    </eclass_context>
  </context_info>
  <chatml/>
  <whiteboardml board_width="640" board_height="480">
    ...
  </whiteboardml>
</session>
```

O elemento raiz *session* possui o atributo *id* que identifica a sessão e seus filhos podem ser divididos em elemento de contexto (*context_info*) e elementos referentes aos componentes xINCA. Nesse exemplo, há dois componentes implementados, sendo que a informação apresentada e anotada na lousa eletrônica foi capturada pelo componente *whiteboard* (*whiteboardml*). Como não há *chat* síncrono durante a sessão, o componente *chat* não foi utilizado, portanto o elemento *chatml* está vazio. O elemento *context_info* possui um elemento filho, *eclass_context*, específico da aplicação de captura.

A seguir, é ilustrado o elemento *whiteboardml* expandido. Esse elemento possui uma seqüência de elementos *slide* e *visit*. O elemento *visit* indica a visita de um *slide* referenciado pelo atributo *slide_id*, possuindo ainda um atributo referente a sua duração, *duration*. Por sua vez, os elementos *slide* possuem elementos *title* e *url* e os elementos *visit* possuem elementos *stroke*, que representam as anotações feitas na lousa eletrônica durante uma visita.

```
<whiteboardml board_width="640" board_height="480">
  <slide slide_id="0">
    <title>Slide0</title>
```

```

    <url>http://iclass.icmc.usp.br/..._mm_cg_web/slide0.jpg</url>
</slide>
...
<visit visit_id="0" duration="34729" slide_id="0">
  <stroke stroke_id="1" person_id="demo" ... width="0">
    <min_x>182</min_x>
    <min_y>307</min_y>
    <max_x>230</max_x>
    <max_y>340</max_y>
    <points>
      ...
    </points>
  </stroke>
  ...
</visit>
...
</whiteboardml>

```

O documento XML de sessão é gerado pelo serviço StRES logo após a captura, sendo armazenado em seguida no banco de dados XML eXist. Como as sessões são representadas por documentos XML, formas de apresentação são obtidas pelo processamento dos documentos XML juntamente com folhas de estilo XSLT, respeitando as características multimídia da informação produzida. Por isso, antes de ser armazenado, o documento XML é processado com três folhas de estilo XSLT que geram documentos SMIL, XHTML+SMIL e HTML, este último para impressão ou revisão rápida de todos os *slides* em uma única página. A seguir, é ilustrada parte da folha de estilo XSLT que gera SMIL.

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" media-type="application/smil"/>
  <xsl:template match="/">
    ...
  </xsl:template>
  <xsl:template match="session">
    ...
  </xsl:template>
  <xsl:template match="whiteboardml">
    ...
  </xsl:template>
  <xsl:template match="visit">
    <xsl:variable name="dur" select="@duration div 1000"/>
    <xsl:variable name="slideid" select="@id"/>

```



```

        <xsl:variable name="url" select="../../slide[@slide_id=$id]/url"/>
        <xsl:variable name="title" select="../../slide[@slide_id=$id]/title"/>
        <par id="visit{@visit_id}" dur="{ $dur }">
            ...
        </par>
    </xsl:template>
</xsl:stylesheet>

```

O documento é estruturado em quatro *templates*, sendo que a última, *visit*, está expandida e se refere à geração de uma sequência de elementos *par* que representam uma visita da sessão e permitem a transição de *slides* da apresentação. Para tanto, utiliza-se a duração da visita (atributo *duration* de *visit*) e a *url* do *slide* correspondente (elemento *url* de *slide* referenciado pelo atributo *slide_id* de *visit*).

Os três documentos de apresentação, gerados a partir do processamento do documento XML de sessão com as folhas de estilo XSLT, são diferentes opções de visualização no momento do acesso. A seguir, é ilustrado o documento gerado pela folha de estilo XSLT referente ao SMIL.

```

<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
    <head>
        ...
    </head>
    <body>
        <par>
            <a href="#visit0">
                
            </a>
            ...
            <seq>
                <par id="visit0" dur="34.729">
                    <text region="title" src="data:,Slide0"/>
                    
                    
                </par>
                ...
            </seq>
        </par>
    </body>
</smil>

```

O elemento *par* filho de *body* indica que seus filhos aparecerão na apresentação paralelamente. Os elementos *a* são referentes à linha do tempo

que referencia as visitas aos *slides*, as quais são definidas no elemento *seq* e apresentadas em sequência. Na Figura 3.6 são mostrados os documentos de apresentação para uma mesma sessão. Os documentos de apresentação em SMIL (a) e em XHTML+SMIL (b) possuem *slides* e uma linha do tempo na qual cada marcador refere-se a uma visita de um *slide* e corresponde a uma ligação hipertexto que pode ser seleccionada para a apresentação do *slide* correspondente.

Figura 3.5: Documentos de apresentação do iClass. (a) Interface em SMIL. (b) Interface em XHTML+SMIL. (c) Interface em HTML.

Para permitir a navegação referente aos documentos de apresentação gerados, foi implementada uma aplicação *Web* com *scripts* PHP que fazem consultas aos documentos de sessão armazenados. Essa aplicação *Web* permite a navegação entre os documentos de apresentação por meio de estruturas de acesso, como por exemplo um índice de disciplinas e um índice de aulas para uma dada disciplina. Na Figura 3.6 é mostrado o índice de disciplina em (a) e

o índice de aulas de uma disciplina em (b).

infra-estrutura também genérica que suportasse o desenvolvimento de aplicações *Web* para visualização e apresentação do conteúdo capturado.

Desenvolver aplicações *Web* trata-se de uma tarefa bastante complexa, principalmente, em decorrência de sua inerente característica hipermídia. O projeto hipermídia de aplicações *Web* tem sido um importante foco de pesquisa na área de Hipermídia. Isso pode ser notado, por exemplo, pela quantidade de metodologias que existem para dar suporte ao projeto hipermídia no desenvolvimento de aplicações *Web*.

Geralmente, essa tarefa de desenvolver aplicações *Web* também faz parte do desenvolvimento de quaisquer aplicações de captura e acesso. Como o projeto InCA-SERVE pretende inclusive fornecer facilidades relacionadas à implementação da visualização, é necessário que haja mecanismos de apoio ao desenvolvimento das aplicações *Web* associadas às aplicações de captura e acesso em construção. Isso poderia ser disponibilizado pelo projeto com uma nova infra-estrutura genérica baseada em elementos da área de Hipermídia.

No capítulo seguinte, são apresentadas algumas metodologias de suporte a projeto hipermídia. Além disso, há discussões sobre possíveis tecnologias de implementação para aplicações *Web* quanto à utilização no contexto de uma infra-estrutura de suporte à visualização em aplicações de captura e acesso.

Tecnologias e Metodologias para Aplicações Web

A maioria das aplicações de captura e acesso tem utilizado a *Web* como o principal meio para disponibilizar o conteúdo que é capturado. Assim, a disponibilidade desse conteúdo pode ser constante e bastante abrangente. Geralmente, são utilizadas aplicações *Web* para gerar os documentos de apresentação referentes ao conteúdo capturado, permitindo a navegação entre esses documentos gerados.

O conteúdo que é capturado por uma aplicação de captura e acesso, na verdade, faz parte do espaço de informação da aplicação *Web* associada. O espaço de informação de uma aplicação *Web* qualquer compreende todos os dados que serão apresentados pela aplicação — dados sobre os quais é feita a navegação.

No caso de aplicações de captura e acesso, o espaço de informação é complementado por outras informações referentes ao domínio da aplicação. Muitas vezes, essas informações não são obtidas em ambientes de captura e existem independentemente do conteúdo capturado. Além disso, elas permitem contextualizar a navegação sobre o conteúdo capturado em si. Por exemplo, no domínio de captura de aulas, existem os dados referentes às disciplinas e aos docentes além das sessões capturadas durante as aulas. Nesse caso, provavelmente será possível se navegar entre as sessões capturadas de uma determinada disciplina ou entre as sessões que foram ministradas por um determinado docente, por exemplo.

As informações não capturadas relacionadas ao domínio da aplicação são

geralmente armazenadas em bancos de dados relacionais — o que é bastante apropriado. Porém, isso não se trata de uma regra uma vez que diferentes fontes de dados podem ser utilizadas, como bancos de dados XML, *Web Services*, entre outras.

Há também várias opções para o armazenamento do conteúdo que é capturado. Por exemplo, para as aplicações de captura e acesso baseadas no projeto InCA-SERVE, as sessões capturadas representadas por documentos XML são armazenadas em um banco de dados XML. Nesse caso, se as informações do domínio estiverem em um banco de dados relacional, as aplicações *Web* associadas deverão acessar essas duas fontes de dados diferentes.

As aplicações *Web* associadas a aplicações de captura e acesso geralmente não necessitam de entrada de dados na *Web* uma vez que os dados referentes ao conteúdo proveniente de captura são atualizados automaticamente. No caso das informações referentes ao domínio, estas não mudam com muita frequência podendo ser persistentes por um período razoavelmente longo, como um semestre para o domínio de captura de aulas.

Sendo assim, as aplicações *Web* associadas a aplicações de captura e acesso podem ser caracterizadas basicamente como aplicações de apresentação. Essa situação é verdadeira para muitos casos, sendo explorada no restante deste trabalho para fins de simplificação.

Existem outros fatores importantes a serem considerados para as aplicações *Web* nesse contexto, principalmente com relação à geração dos documentos de apresentação. Por exemplo, é interessante que a geração dos documentos de apresentação seja feita dinamicamente conforme a requisição do usuário. Isso evita que se ocupe espaço de armazenamento desnecessário, uma vez que pode haver diferentes formatos de apresentação para um mesmo conteúdo. Além disso, documentos de apresentação não ficariam desatualizados caso houvesse alguma alteração no formato da apresentação.

Além de permitir a geração dinâmica dos documentos de apresentação, também é interessante que as aplicações *Web* permitam diferentes formatos para os documentos. Por exemplo, enquanto se navega entre as sessões utilizando HTML, cada sessão capturada poderia ser visualizada também em SMIL, já que se trata de uma apresentação multimídia. Ou ainda, pode-se querer que os documentos também sejam disponibilizados em WML para ser visualizados por meio de dispositivos móveis, além de outros cenários. Portanto, algumas suposições podem ser feitas para aplicações *Web* associadas a aplicações de captura e acesso:

- tratam-se de aplicações de apresentação sem entrada de dados;
- possuem o espaço de informação composto por informações referentes

ao domínio e ao conteúdo capturado;

Além disso, essas aplicações possuem alguns requisitos nesse contexto. Resumindo, uma aplicação *Web* associada a uma aplicação de captura e acesso deve:

- acessar diferentes fontes de dados;
- gerar dinamicamente os documentos de apresentação; e,
- permitir diferentes formatos para os documentos de apresentação.

A seguir, é discutido como aplicações *Web* de modo geral têm sido implementadas com diferentes tecnologias. Além disso, é eleita uma abordagem para as aplicações *Web* que devem ser associadas a aplicações de captura e acesso, com base nas suposições e nos requisitos apontados acima.

4.1 *Tecnologias para Aplicações Web*

A maneira como as aplicações *Web* são implementadas têm mudado consideravelmente desde que se tornaram populares a partir da década de 90. As aplicações *Web* passaram de simples coleções de páginas estáticas para oferecer soluções dinâmicas e personalizadas. Além disso, o acesso à *Web* atualmente não está restrito apenas a *browsers* em *desktops*, sendo disponível também em PDAs, por exemplo.

Páginas HTML estáticas são adequadas para publicar informação que não muda com frequência, porém não são apropriadas para a demanda por informação dinâmica — como acontece por parte das aplicações de captura e acesso. Dessa forma, há a necessidade de infra-estruturas que permitam publicar páginas dinamicamente e que ofereçam componentes e interfaces necessárias para servir informações.

Para atender essa demanda por informação dinâmica, inicialmente surgiu o CGI (*Common Gateway Interface*), cuja interface poderia ser suportada por programas em diversas linguagens, como Perl ou C. Assim, era possível gerar sob demanda a página HTML que seria servida em seguida. Cada componente tinha um propósito específico e controlava um conjunto determinado de requisições.

Como os servidores *Web* não proviam padrões, geralmente, os desenvolvedores especificavam seus próprios procedimentos para a geração das páginas. Desse modo, apenas seus próprios componentes eram integrados, além de só poderem executar em um servidor específico. Para resolver essas questões, foram desenvolvidas novas tecnologias como ASP (*Active Server Pages*) e JSP

(*Java Server Pages*). Além dessas tecnologias permitirem a integração de componentes, os servidores passaram a prover linguagens de *scripts* para a geração *on the fly* das páginas. Essas linguagens tornaram-se bastante populares e muitas aplicações *Web* atuais as utilizam.

No caso, os *scripts* devem ser adequados a cada nova versão de *browser*, uma tarefa difícil de estender e manter. Além disso, a abordagem de *scripting* funciona bem para servir informação em um único formato para um determinado dispositivo ou aplicação, não sendo portanto adequada para aplicações de captura e acesso. Por exemplo, se há a necessidade de prover um novo formato para o mesmo conteúdo, como WML para telefones celulares, o *script* deve ser simplesmente reescrito.

Outra desvantagem é que essa abordagem de *scripts* não permite separar a apresentação e o conteúdo a ser apresentado. As mesmas pessoas responsáveis pela apresentação devem também saber, por exemplo, como acessar o conteúdo. Uma tecnologia alternativa para a abordagem de *scripts* bastante interessante é o *framework* Cocoon (Apache 2003), que se baseia na separação de interesses¹ para implementar aplicações *Web*. No caso de aplicações de captura e acesso, esse *framework* é uma tecnologia adequada para as aplicações *Web* associadas, conforme pode ser observado pela sua descrição a seguir.

4.1.1 Framework Cocoon

Cocoon é um *framework* Java que permite a forte separação entre lógica, conteúdo e estilo em aplicações *Web* (Apache 2002). Esse *framework* publica dinamicamente conteúdo XML que pode ser gerado a partir de fontes variadas usando transformações XSLT: XML descreve conteúdo e XSLT transforma esse conteúdo em múltiplos formatos. Com Cocoon pode-se:

- servir arquivos estáticos, assim como respostas geradas automaticamente;
- mapear requisições de forma transparente para recursos físicos;
- executar transformações XSLT simples e com múltiplos passos;
- passar parâmetros dinamicamente para transformações XSLT; e,
- gerar saída com diferentes formatos, como HTML, WML, PDF, TXT, XML, SMIL, VRML, SVG, PNG, JPEG, entre outros.

Por permitir a geração dinâmica de documentos com variados formatos a partir de diferentes fontes, o *framework* Cocoon mostra-se bastante apropri-

¹O termo utilizado em inglês é *separation of concerns*.

ado para implementar aplicações Web associadas a aplicações de captura e acesso, conforme as características apontadas anteriormente.

Cocoon utiliza o conceito de *pipeline* para processar documentos XML: os documentos avançam na *pipeline* por meio de eventos SAX. Cada passo da *pipeline* é tratado com um tipo particular de componente. Por exemplo, a entrada pode ser tratada com o componente *File Generator*, associado a um documento XML.

Seguindo, o processamento pode ser tratado com um componente *XSLT Transformer*, associado a uma folha de estilo que gera HTML. No final da *pipeline*, a saída pode ser tratada com um componente *HTML Serializer*, que serializa o resultado de todo o processamento em um documento HTML.

Como resposta a uma requisição do usuário, a *pipeline* correta pode ser identificada por um componente *Wildcard Matcher* para apresentar o documento apropriado. A declaração dos componentes a serem utilizados e as definições da *pipeline* são feitas em um arquivo de gerenciamento chamado *sitemap*, cuja estrutura segue:

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
  <map:components>
    <!-- declaração dos componentes -->
    <map:generators/>
    <map:transformers/>
    <map:serializers/>
    <map:matchers/>
  </map:components>
  <map:pipelines>
    <!-- definições da pipeline -->
  </map:pipelines>
</map:sitemap>
```

Por exemplo, o processo de transformação da infra-estrutura iVIEW, detalhado na Seção 3.1.4, pode ser mapeado em uma *pipeline* do Cocoon, que ao final do processamento serializaria a representação SVG correspondente. O código a seguir é uma *pipeline* definida de acordo com os passos de processamento da iVIEW:

```
<map:pipeline>
  <!-- Step 1a -->
  <map:match pattern="application.xslt">
    <map:generate src="layout.xml"/>
    <map:transform src="template.xslt"/>
    <map:serialize type="xml"/>
  </map:match>
```

```

<!-- Step 1b -->
<map:match pattern="application.js">
  <map:generate src="layout.xml"/>
  <map:transform src="templateJS.xslt"/>
  <map:serialize type="text"/>
</map:match>
<!-- Step 2 -->
<map:match pattern="application.svg">
  <map:generate src="application.xml"/>
  <map:transform src="cocoon:/application.xslt"/>
  <map:serialize type="svg"/>
</map:match>
</map:pipeline>

```

Iniciando os passos de processamento, a requisição do usuário *application.svg* pela representação SVG (documento 6 na Figura 3.4) é atendida ativando o código relacionado com o *Step 2*. Esse código inicia e finaliza todo o processo de transformação. O documento de entrada *application.xml* (documento 1) é processado por meio do documento XSLT específico *application.xslt* (documento 5), que, por sua vez, deve ser gerado.

Nesse momento, o código relacionado com o *Step 1a* é invocado, usando o protocolo *cocoon:/* que permite a requisição interna por *application.xslt*. Seguindo, o documento *application.xslt* é obtido com o processamento do arquivo *layout.xml* (documento 2) com a folha de estilo específica *template.xslt* (documento 4). O resultado é serializado em XML e será utilizado para dar continuidade ao *Step 2*.

Quando o *Step 1a* acaba, o *Step 1b* é executado ativando seu código relacionado no *sitemap*. Nesse código, o processamento do *layout.xml* com a folha de estilo *templateJS.xslt* (documento 7) produz o documento JavaScript (documento 8), que será associado ao documento SVG. O *Step 2* continua e, finalmente, gera e serializa o documento SVG, finalizando todo o processo de transformação.

No caso do documento de entrada da *pipeline*, este pode ser um documento XML que existe no sistema de arquivos — como no exemplo da infra-estrutura iVIEW — ou pode ser gerado dinamicamente a partir de uma consulta a um banco de dados ou a um *Web Service*, entre muitas outras fontes. Para tanto, o Cocoon fornece a tecnologia de páginas XSP que permite a criação de dados XML dinâmicos para servir como entrada em *pipelines*.

Uma página XSP é processada por um componente especializado, o *Server Pages Generator*, e pode conter elementos customizados, definidos em *logicsheets*. Uma *logicsheet* é como uma biblioteca, sendo implementada por meio de transformações XSLT que introduzem código em uma página XSP.

Existem *logicsheets* de ambiente, para manipulação de dados e de utilidade geral pré-definidas no Cocoon; além disso, é possível criar novas *logicsheets* para necessidades específicas.

Como mencionado, Cocoon é bastante apropriado para o contexto de aplicações de captura e acesso, pois as aplicações nele baseadas podem ter seu espaço de informação em diferentes fontes de dados de forma transparente. Além disso, Cocoon permite que os documentos de apresentação das aplicações sejam gerados dinamicamente em formatos diferentes com um bom desempenho — uma vez que utiliza eventos SAX e faz *cache* dos documentos gerados.

Tecnologias como Cocoon, JSP e outras afins permitem gerar nós e ligações para aplicações *Web*. Porém, definir quais nós devem ser gerados e como devem ser ligados é uma tarefa bastante complexa, delegada ao desenvolvedor. Muitos esforços da área de Hipermídia têm sido direcionados para suportar essa tarefa de projeto hipermídia, resultando em várias metodologias de apoio ao desenvolvimento de aplicações *Web*. A seguir, algumas dessas metodologias são apresentadas, discutindo-se possíveis soluções para o contexto de aplicações de captura e acesso.

4.2 Metodologias para Aplicações Web

Existem muitos aspectos a serem considerados no desenvolvimento de aplicações *Web*, principalmente relacionados à complexidade do projeto hipermídia dessas aplicações. Desde meados da década de 90, muitas metodologias têm sido propostas pela área de Hipermídia para dar suporte ao desenvolvimento de tais aplicações. Essas metodologias apresentam diferentes abordagens para cumprir a tarefa de suportar o projeto hipermídia.

Geralmente, cada metodologia possui um conjunto específico de primitivas de modelagem, associadas a um processo específico de desenvolvimento de aplicações *Web*. Além disso, algumas metodologias oferecem ferramentas, também específicas, baseadas em suas primitivas de modelagem e seu processo de desenvolvimento. As ferramentas provêem diferentes níveis de automação para o desenvolvimento das aplicações — desde nenhuma automação até a completa geração de aplicações *Web*.

É importante que as metodologias sejam claras o suficiente para não tornar a tarefa de desenvolver aplicações *Web* ainda mais complexa. Além disso, seria bom que o desenvolvedor não tivesse que desviar muito o foco de sua atenção da aplicação *Web* em si em função do processo de desenvolvimento ou das ferramentas da metodologia adotada — o que geralmente não acontece.

HDM (Garzotto, Paolini, & Schwabe 1993) é uma das propostas pioneiras

para o suporte ao projeto hipermídia. Com HDM, um esquema da aplicação é definido, sendo organizado em duas partes: *hyperbase*, que descreve os elementos navegacionais básicos (*entity types* e *link types*); e de acesso, que descreve a organização das estruturas de acesso (como índices e *guided tours*). A notação do HDM é baseada no modelo entidade-relacionamento para expressar as abstrações utilizadas. Além disso, HDM possui duas dimensões semânticas para a definição do esquema: *in-the-large*, para fornecer características gerais do projeto; e *in-the-small*, para fornecer detalhes do projeto.

O sistema JWeb é um ambiente de projeto baseado na metodologia HDM com funcionalidades de prototipagem rápida (Bochicchio, Paiano, & Paolini 1999). O JWeb suporta o projeto *hyperbase* e o projeto das estruturas de acesso permitindo várias configurações. Além disso, o JWeb promove uma clara separação *in-the-large* e *in-the-small* para o projeto de uma aplicação.

RMM (Isakowitz, Stohr, & Balasubramanian 1995) é uma evolução do HDM por utilizar os conceitos de projeto hipermídia em uma metodologia estruturada. O processo de desenvolvimento do RMM é dividido em sete passos com uma série de *guidelines* para as tarefas envolvidas. No modelo de dados do RMM, a navegação é definida dentro e através das entidades utilizando-se relações associativas e ligações estruturais. A ferramenta RMC foi construída baseada na metodologia RMM (Diaz, Isakowitz, Maiorana, & Gilabert 1995), e utiliza um repositório interno para gerar páginas HTML pré-compiladas.

OOHDM (Schwabe & Rossi 1995) também se trata de uma evolução do HDM, com uma abordagem orientada a objetos baseada em modelos. Os modelos são divididos com base na separação de interesses da aplicação *Web* em conceitual, navegacional e de interface. O modelo conceitual representa os objetos do domínio, suas relações e as funcionalidades da aplicação. A navegação é definida por objetos navegacionais baseados no modelo conceitual.

Os objetos navegacionais são, na verdade, os nós da aplicação com atributos e âncoras, compostos ou não. O modelo navegacional é composto por: um esquema navegacional, que representa a estrutura navegacional da aplicação; e, por um esquema de contexto que complementa o esquema navegacional com contextos e estruturas de acesso. O modelo de interface é baseado em *Abstract Data Views* (Cowan & Lucena 1995) para indicar as interfaces abstratas dos objetos navegacionais.

OOHDM-Web é um sistema baseado em uma biblioteca específica para implementar aplicações *Web* projetadas com a metodologia OOHDM (Schwabe, Pontes, & Moura 1999). Esse sistema assiste o desenvolvedor em alguns passos do processo de desenvolvimento da aplicação sem aliviá-lo totalmente de esforços de programação.

UWE (Baumeister, Koch, & Mandel 1999) propõe uma abordagem meto-

dológica similar ao OOHDM, uma vez que inclui a modelagem separada de aspectos conceituais, navegacionais e de apresentação de aplicações Web. A grande diferença do UWE é a utilização da UML em todo o processo de modelagem por meio de extensões. ArgoUWE é uma ferramenta CASE que suporta a fase de projeto do processo de desenvolvimento da metodologia UWE (Zhang, Knapp, Koch, Kraus, & Hennicker 2003). Essa ferramenta, na verdade, é um módulo de *plug-in* para a ferramenta CASE de modelagem ArgoUML (Tritis.org 2003).

Strudel (Fernández, Suciu, & Tatarinov 1999) é um sistema para implementar aplicações Web. Esse sistema provê a linguagem declarativa de consulta StruQL que permite tanto a especificação do conteúdo quanto a especificação da estrutura da aplicação. As consultas são realizadas sobre um modelo de dados semi-estruturado: um grafo com legendas direcionado, cujos elementos conectados são nós — complexos ou não e com atributos. Strudel provê uma linguagem de *templates* HTML para descrever a apresentação da aplicação. Além disso, Strudel permite a utilização de diferentes fontes de dados por meio do mapeamento das fontes em grafos do sistema.

AutoWeb (Fraternali & Paolini 2000) propõe uma metodologia e um ambiente de desenvolvimento para aplicações Web. O processo de desenvolvimento de uma aplicação é dividido em: definição da aplicação, geração de uma base de dados de suporte e implementação e execução da aplicação. A definição da aplicação é baseada no modelo de dados HDM-lite, que permite a especificação de um *hyperschema* — uma descrição em alto nível da estrutura dos dados, da navegação e da apresentação da aplicação.

HDM-lite é diretamente baseado no HDM, tratando-se de uma evolução específica para o contexto de aplicações Web. No AutoWeb, um ambiente gráfico assiste o desenvolvedor em cada passo do processo de definição do *hyperschema*. Assim que o *hyperschema* é definido, o AutoWeb automaticamente gera a aplicação: um esquema de base dados relacional é gerado e a base de dados correspondente é utilizada para a geração das páginas HTML.

WebML (Matera, Maurino, Ceri, & Fraternali 2003) pode ser considerado como descendente do AutoWeb uma vez que herda boa parte de sua abordagem. WebML provê uma representação baseada em XML para suas especificações que também podem ser representadas por diagramas gráficos. A ferramenta CASE WebRatio (WebModels 2004) faz o processamento automático de especificações WebML gerando de forma completa aplicações Web.

Araneus (Merialdo, Atzeni, & Mecca 2003) propõe uma metodologia baseada em modelos composta de um conjunto de passos e transformações que conduz de uma especificação conceitual à implementação de uma aplicação Web. A ferramenta HOMER suporta os vários passos da metodologia e, ba-

seada nos artefatos produzidos durante o processo, automatiza a geração do código necessário para implementar a aplicação.

Como os modelos do Araneus são baseados em uma extensão do modelo relacional aninhado, o mapeamento entre eles é feito por álgebra relacional para estruturas aninhadas. Os modelos são divididos em diferentes níveis de abstração: de dados, de hipertexto e de apresentação. O modelo do nível de hipertexto é baseado no modelo de dados ADM-d, que abstrai as características lógicas de páginas de uma aplicação *Web*. O ADM-d mapeia o conceito de páginas em esquemas *page-scheme*, que contém atributos inclusive do tipo *link*, e um conjunto de *page-scheme* compõe um *site-scheme* referente à estrutura da aplicação como um todo.

As metodologias apresentadas estão entre as mais relevantes dentre as disponíveis — que incluem além dessas, abordagens como OOH-Method (Gomez, Cachero, & Pastor 2000), OO-Method (Pastor, Abrahao, & Fons 2001) e Weave (Yagoub, Florescu, Issarny, & Valduriez 2000), entre muitas outras.

No contexto comercial, apenas o Oracle WebDB (Oracle 2004) se baseia em modelagem de alto nível para desenvolver aplicações *Web*. Para tanto, são utilizadas algumas ferramentas: Web Generator gera uma aplicação específica para o DBMS da Oracle, a partir de uma especificação desenhada no Designer 2000. Existem ainda outras abordagens mais focadas em aspectos tecnológicos e de implementação que também geram aplicações *Web*, como o WebJinn (Kojarski & Lorenz 2003) e a arquitetura proposta em (Milosavljevic, Vidakovic, & Konjovic; 2002).

Além disso, assuntos como personalização e entrada de dados para as aplicações *Web* têm despertado muitos esforços de pesquisa na área. Por exemplo, OOHDM (Rossi, Schwabe, & Guimarães 2001), WebML (Fraternali & Paoletti 1998), OntoWeaver (Lei, Motta, & Domingue 2003) e WSDM (De Troyer & Leune 1997) têm reportado trabalhos com relação à personalização das aplicações. Com relação à modelagem aplicações que incluem entrada de dados, pode-se citar o Wizard (Turau 2002), OOWS (Pastor, Abrahao, & Fons 2001), Araneus (Merlaldo, Atzeni, & Mecca 2003) e WebML (Matera, Maurino, Ceri, & Fraternali 2003).

4.2.1 Infra-estrutura para Aplicações Web

O principal objetivo do projeto InCA-SERVE, apresentado no capítulo anterior, é suportar o desenvolvimento de aplicações de captura e acesso. Como, em termos de acesso à informação capturada, desenvolver aplicações de captura e acesso também inclui desenvolver aplicações *Web*, faz-se necessária uma infra-estrutura que suporte o desenvolvimento de aplicações *Web* nesse contexto. Essa infra-estrutura poderia ser baseada em uma das metodolo-

gias de projeto hipermidia que foram apresentadas. Entretanto, existem características desejáveis para uma metodologia a ser utilizada no contexto de aplicações de captura e acesso, em especial dentro do projeto InCA-SERVE. Em particular, a metodologia ideal deve envolver poucos artefatos, de modo a exigir pouco esforço tanto para aprendê-la quanto para utilizá-la de fato.

Para que a metodologia adotada se case completamente com os objetivos do projeto, é interessante que esta ofereça um alto grau de automação, uma vez que se espera que as aplicações tenham a visualização de seu conteúdo implementada de forma direta e transparente. Sendo assim, a metodologia deve permitir que tanto a modelagem quanto as aplicações *Web* propriamente ditas sejam geradas completamente de forma automática.

Além disso, como as aplicações de captura e acesso baseadas no projeto podem ser as mais variadas — por exemplo, realizando a captura de experiências para diferentes domínios — é necessário que a metodologia seja bastante flexível, tanto com relação ao espaço de informação da aplicação, quanto com relação às tecnologias utilizadas. Por exemplo, a metodologia deve permitir que as aplicações geradas possam usar diferentes fontes de dados, além de diferentes formatos de apresentação.

Outra característica importante a ser considerada para a metodologia ideal é que esta seja extensível, uma vez que o projeto InCA-SERVE está em constante evolução. Além disso, é interessante que a metodologia adotada também seja facilmente estendida para utilizar novas fontes de dados, novas formas de acesso e novos formatos de apresentação.

Assim, como apresentado, existem algumas características especiais para uma metodologia a ser adotada pela infra-estrutura InCA-SERVE responsável pelo suporte ao desenvolvimento de aplicações *Web*. Resumindo essas características específicas, a metodologia ideal deve:

- permitir a geração automática e completa de aplicações *Web*;
- ser simples para facilitar a integração ao projeto InCA-SERVE;
- ser flexível para atender diferentes domínios e necessidades; e,
- ser extensível para permitir o uso de novas tecnologias.

Entre as abordagens apresentadas, as que oferecem o maior nível de automação são AutoWeb (Fraternali & Paolini 2000), Araneus (Merialdo, Atzeni, & Mecca 2003) e WebML (Matera, Maurino, Ceri, & Fraternali 2003). Porém, elas são bastante complexas. Essas abordagens exigem do desenvolvedor a preparação do ambiente de desenvolvimento e as aplicações geradas geralmente dependem de algumas condições específicas para executarem. Por exemplo, o

AutoWeb e o Araneus geram um esquema para a aplicação que é armazenado em uma base de dados e as páginas são geradas por meio de consultas à base.

Além disso, essas abordagens exigem que o desenvolvedor necessariamente aprenda a utilizar suas ferramentas e esteja totalmente consciente do processo de desenvolvimento e das primitivas de modelagem envolvidas. Adicionalmente, essas abordagens oferecem poucas opções para as aplicações geradas. Por exemplo, sendo pouco flexíveis quanto a fontes de dados e formatos de apresentação. Na maioria dos casos a única fonte de dados possível são bases de dados relacionais e o único formato de apresentação possível é o HTML.

Recentemente WebML introduziu uma interface com *Web Services* para fonte de dados e a geração em WML como formato de apresentação, porém essas opções ainda são insuficientes para o contexto de aplicações de captura e acesso. Além disso, nenhuma abordagem discute profundamente possíveis extensões — por exemplo, o Araneus alega que poderia ser estendido facilmente para gerar PHP, porém não há detalhes em como fazê-lo (Merialdo, Atzeni, & Mecca 2003).

De acordo com o exposto acima, nenhuma das metodologias apresentadas possui todas as características desejadas para o contexto de aplicações de captura e acesso de modo geral e do projeto InCA-SERVE em particular. Embora nenhuma metodologia seja completamente adequada, a nova infra-estrutura do projeto InCA-SERVE pode se basear em pontos comuns às mesmas e nos pontos positivos de algumas delas.

No caso, apesar das diferenças encontradas entre as metodologias — principalmente quanto às suas primitivas de modelagem, seus processos de desenvolvimento e ferramentas associadas — várias similaridades podem ser identificadas. Ao longo dos anos, essas similaridades têm se mostrado boas práticas de desenvolvimento de aplicações *Web*.

Embora com notações diferentes, a maioria das metodologias tem utilizado a separação de interesses na modelagem das aplicações, geralmente dividindo seus modelos de acordo com os aspectos conceitual, navegacional e de apresentação da aplicação. Exemplos de abordagens que fazem uso da separação de interesses na modelagem das aplicações incluem o OOHDM (Schwabe & Rossi 1995), UWE (Baumeister, Koch, & Mandel 1999), Araneus (Merialdo, Atzeni, & Mecca 2003) e o WebML (Matera, Maurino, Ceri, & Fraternali 2003).

Além disso, o conceito de nós e estruturas de acesso pode ser notado em todas as metodologias que suportam o projeto hipermídia. Na maioria das vezes os nós podem ser compostos ou não e as estruturas de acesso são associadas a índices, menus e *guided tours*, entre outras.

Ainda com relação à modelagem, as metodologias têm adotado como base

para seus modelos a orientação a objetos ou o modelo entidade-relacionamento. Por exemplo, o OOHDM (Schwabe & Rossi 1995), UWE (Baumeister, Koch, & Mandel 1999), OOH-Method (Gomez, Cachero, & Pastor 2000), OO-Method (Pastor, Abrahao, & Fons 2001) utilizam uma abordagem baseada em orientação a objetos, enquanto que o RMM (Isakowitz, Stohr, & Balasubramanian 1995), Araneus (Merialdo, Atzeni, & Mecca 2003), AutoWeb (Fraternali & Paolini 2000) e WebML (Matera, Maurino, Ceri, & Fraternali 2003) se baseiam no modelo entidade-relacionamento.

O processo de desenvolvimento das metodologias geralmente envolve os passos que compreendem desde a modelagem até a implementação da aplicação *Web* em si. Salvo em algumas abordagens específicas, primeiramente é feita a modelagem conceitual da aplicação no nível dos dados. Em seguida é definido um novo esquema por meio de visões sobre o esquema conceitual. Esse novo esquema representa o espaço de informação sobre o qual realmente será feita a navegação.

Com base no novo esquema, são definidos os nós e a estrutura organizacional dos mesmos, geralmente em um diagrama indicando: como os nós serão aninhados ou compostos e como serão ligados — no caso utilizando estruturas de acesso (como índices, menus e outras estruturas). Depois de definir os nós e a estrutura associada a eles, é iniciado o passo de definição da apresentação, basicamente com relação ao *layout* dos nós.

Depois de modelada a aplicação, começam os passos de sua implementação, que geralmente são bem diferentes entre as diversas metodologias. Porém, incluem geralmente ao menos a geração dos *scripts* para a criação do espaço de informação, por exemplo, da base de dados. No caso, os passos de implementação dependem muito do tipo de aplicação *Web* que será gerada e da abordagem tecnológica utilizada.

Resumindo, os principais pontos em comum entre as metodologias são a separação de interesses na modelagem, o conceito de nós e estruturas de acesso e o processo de desenvolvimento envolvendo:

1. modelagem conceitual;
2. definição do espaço de navegação;
3. definição da estrutura navegacional;
4. definição da apresentação dos nós de navegação; e,
5. implementação da aplicação.

A abordagem para a modelagem dos dados baseada no modelo entidade-relacionamento — que envolve entidades e relações entre entidades — tem a

vantagem de ser a mais próxima da estrutura da navegação — que envolve nós e ligações entre nós. Porém, um ponto positivo da abordagem orientada a objetos é que esta é facilmente mapeada para a estrutura navegacional e ainda pode contar, por exemplo, com os avanços no campo da engenharia de requisitos relacionados à UML.

Outro ponto positivo de algumas metodologias é a utilização de representações XML em suas abordagens. Exemplos incluem as especificações da abordagem WebML (Matera, Maurino, Ceri, & Fraternali 2003), a linguagem OOHDM-ML da metodologia OOHDM (Medeiros & Schwabe 2001) e a abordagem UWEXML da metodologia UWE (Kraus & Koch 2002), que utilizam XML para representar seus modelos. Com a utilização de XML, o processo de desenvolvimento das aplicações pode ser baseado em simples transformações XSLT de documentos.

Além disso, a utilização de XML poderia ser associada a uma abordagem declarativa para especificar o processo de desenvolvimento das aplicações. Essa abordagem declarativa também poderia ser baseada em XML e em transformações XSLT. Dessa forma, seria simples de ser executada e ainda bastante extensível, dada a utilização de XML e tecnologias afins.

Para permitir a automação completa nessa abordagem, podem ser utilizadas algumas regras pré-definidas de mapeamento entre os modelos, além de valores padrões para a configuração das aplicações. Além disso, a abordagem poderia se utilizar do *framework* Cocoon para permitir diferentes fontes de dados e formatos de apresentação, garantindo sua flexibilidade.

Essa abordagem declarativa foi implementada nesses termos no contexto da infra-estrutura wVIEW, que é apresentada no próximo capítulo. Essa infra-estrutura, parte do projeto InCA-SERVE, permite a geração automática de aplicações Web associadas a aplicações de captura e acesso. Porém, dada sua generalidade, a wVIEW também pode ser utilizada no desenvolvimento de aplicações Web independente do contexto de aplicações de captura e acesso, como ocorrem com as metodologias apresentadas neste capítulo.

Infra-estrutura wVIEW

Aplicações de captura e acesso podem ser construídas a partir do reuso das infra-estruturas e serviços do projeto InCA-SERVE. A fim de complementar esse projeto quanto ao suporte à visualização da informação capturada, foi desenvolvida a *wVIEW*: uma infra-estrutura genérica que permite a geração automática de aplicações *Web* para acessar e apresentar conteúdo capturado.

Embora tenha sido desenvolvida como parte do projeto InCA-SERVE, a *wVIEW* pode ser utilizada no desenvolvimento de quaisquer aplicações *Web*, independente do contexto de aplicações de captura e acesso. A infra-estrutura *wVIEW* é baseada em conceitos estabelecidos da área de Hipermissão, tal qual a separação em modelos conceitual, navegacional e de apresentação para o projeto de uma aplicação. Basicamente, uma aplicação *Web* é gerada a partir de transformações desses modelos.

Até o momento, a infra-estrutura *wVIEW* gera aplicações *Web* de apresentação que não fazem atualização no espaço de informação. Isso não se trata de um problema, dado que as aplicações *Web* associadas a aplicações de captura e acesso geralmente não necessitam de entrada de dados. Porém, a *wVIEW* pode ser futuramente estendida para permitir a entrada de dados nas aplicações.

Além disso, por enquanto só é possível gerar aplicações baseadas no *framework* Cocoon — o que é bastante apropriado para o contexto de aplicações de captura e acesso, uma vez que permite utilizar diversas fontes de dados e gerar dinamicamente documentos em diferentes formatos. A *wVIEW* também pode ser estendida para outras tecnologias, por exemplo, JSP ou PHP.

Dado o escopo desta dissertação, a infra-estrutura *wVIEW* foi inicialmente

implementada contemplando poucas opções para a sua primeira versão. Nessa versão inicial, a única opção para fonte de dados é base de dados relacional e para formato de apresentação é HTML. Porém, o processo de desenvolvimento das aplicações foi totalmente construído e as novas opções que se façam necessárias — por exemplo, base de dados XML para fonte de dados e SMIL para formato de apresentação — podem ser adicionadas de modo sistemático sem muito esforço. A próxima seção apresenta uma visão geral da versão atual da wVIEW.

5.1 Visão Geral da Infra-estrutura wVIEW

A infra-estrutura wVIEW adota uma abordagem declarativa baseada em XML para automatizar a geração de aplicações *Web*. wVIEW é baseada em conceitos hipermídia, tais como a separação de interesses utilizada para modelagem, além de nós e estruturas de acesso. A geração de aplicações ocorre por meio de um processo automático de transformação de modelos. Na Figura 5.1 é ilustrada a geração de uma aplicação *Web* usando a infra-estrutura wVIEW.

Figura 5.1: Geração de uma aplicação *Web* usando a infra-estrutura wVIEW.

O desenvolvedor usuário da infra-estrutura wVIEW deve estar familiarizado com orientação a objetos, uma vez que a wVIEW utiliza um diagrama de classes da UML como entrada — de forma similar a outras metodologias que suportam projeto hipermídia. No caso da wVIEW, o diagrama de classes deve estar representado em um documento XMI, como mostrado na Figura 5.1.

Isso é baseado no fato de que diagramas de classes UML podem ser intercambiados entre ferramentas de modelagem e aplicações pela utilização do formato XMI (*XML Metadata Interchange*) (OMG 2003). Atualmente, muitas ferramentas de modelagem são capazes de exportar especificações UML nesse formato XMI — exemplos incluem IBM Rational Rose (IBM 2003) e ArgoUML (Tritis.org 2003).

O desenvolvedor usuário da wVIEW pode especificar um digrama de classes em qualquer ferramenta de modelagem e em seguida exportar a especificação como um documento XMI para ser usado como entrada no processo de transformação da wVIEW. Embora o formato XMI esteja em sua segunda versão, a wVIEW no momento aceita apenas a primeira versão, uma vez que a maioria das ferramentas de modelagem ainda não exportam a segunda versão.

O documento XMI é a fonte para a geração do modelo conceitual da aplicação, que representa o nível dos dados da mesma. Se o desenvolvedor não deseja utilizar uma ferramenta de modelagem, ele pode prover diretamente o modelo conceitual da aplicação sem ter que necessariamente oferecer o documento XMI. Isso pode ser feito editando o documento XML referente ao modelo conceitual, detalhado na Seção 5.2.1.

O processo de transformação para a geração de aplicações *Web*, ocorre por meio de transformações XSLT sucessivas de documentos e é representado por wVIEW na Figura 5.1. Apesar da geração automática, o desenvolvedor pode interferir — se desejar — em qualquer ponto de todo o processo de transformação. Existem valores padrões para os modelos intermediários e para as especificações de configuração. Porém, o desenvolvedor pode especializar a modelagem, assim como configurar a aplicação *Web* a ser gerada, de forma declarativa editando alguns poucos documentos XML.

Quando todo o processo de transformação se encerra, uma aplicação *Web* é gerada, representada por *Web Application* na Figura 5.1. O desenvolvedor deve ser capaz de colocar em execução a aplicação *Web* gerada, que é composta por: *scripts* para criar o espaço de informação e documentos XML para gerar nós e estruturas de acesso quando requisitado. Até o momento, a wVIEW gera aplicações baseadas no *framework* Cocoon que consultam uma base de dados relacional e apresenta os nós em HTML.

5.2 Processo de Transformação wVIEW

A infra-estrutura wVIEW utiliza a separação de interesses como base para a modelagem das aplicações *Web* a serem geradas — assim como outras metodologias para projeto hipermídia o fazem. Assim, a wVIEW organiza a modelagem de forma a separar a aplicação em níveis diferentes de abstração. Isso se dá por meio dos modelos referentes às camadas dos dados, da estrutura hipermídia da aplicação e da apresentação dos nós de navegação.

Transformações sucessivas desses modelos definem o processo de transformação wVIEW. No caso, essas transformações ocorrem em duas fases. Primeiramente, os modelos são gerados. Em seguida, a wVIEW gera uma aplicação *Web* baseada nos modelos obtidos na fase anterior.

São três os modelos gerados na primeira fase: modelo conceitual, modelo navegacional e modelo de apresentação. No caso, o modelo conceitual é referente à camada dos dados e representa o espaço de informação da aplicação Web a ser gerada. Por sua vez, o modelo navegacional, referente à camada da estrutura hipermídia da aplicação, define a navegação sobre o espaço de informação por meio de nós e estruturas de acesso. O último modelo, o modelo de apresentação, determina como os nós devem ser apresentados e como as ligações das estruturas de acesso devem ser implementadas, uma vez que se refere à camada de apresentação dos nós de navegação.

Na verdade, o modelo navegacional corresponde a visões do modelo conceitual e o modelo de apresentação é praticamente uma tradução do modelo navegacional em termos de apresentação. Assim, o modelo navegacional deve ser gerado com base no modelo conceitual, enquanto que o modelo de apresentação pode ser obtido diretamente do modelo navegacional. Vale ressaltar que a geração dos modelos é independente das configurações da aplicação Web com respeito aos tipos utilizados de fontes de dados, formatos de apresentação ou tecnologia para forma acesso.

Figura 5.2: Processo de transformação wVIEW que gera os modelos e a aplicação Web correspondente.

A Figura 5.2 ilustra todo o processo de transformação wVIEW, incluindo a primeira fase relacionada à geração dos modelos. O documento XMI (XMI na Figura 5.2), contendo a representação do diagrama de classes referente à aplicação Web desejada, é usado como fonte para a geração do modelo conceitual (CM na Figura 5.2). Em seguida, o modelo navegacional (NM na Figura 5.2) é gerado com base no modelo conceitual. Como último passo dessa fase, gera-se o modelo de apresentação (PM na Figura 5.2) a partir do modelo navegacional.

Quando a geração dos modelos é concluída, a geração da aplicação Web é iniciada com o processamento dos modelos gerados. Uma aplicação Web gerada pela wVIEW é composta por alguns documentos. Como a wVIEW gera,

até momento, apenas aplicações *Web* baseadas no Cocoon que consultam bases de dados relacionais, os documentos a serem gerados para uma aplicação são: *scripts* para criar o espaço da informação, um *sitemap* para o gerenciamento da aplicação e documentos XML capazes de gerar nós sob demanda — quando associados a folhas de estilo apropriadas.

Assim, transformando o modelo conceitual (*CM* na Figura 5.2), a *wVIEW* gera um *script* DDL SQL para criar o espaço de informação, outro para adicionar as chaves estrangeiras e ainda outro para popular a base de dados, que pode ser usado para prototipagem rápida (*Scripts* na Figura 5.2).

Para coordenar a navegação sobre o espaço de informação, a *wVIEW* gera um *sitemap* (*Sitemap* na Figura 5.2) a partir de um documento de configuração (*Config* na Figura 5.2) provido pela *wVIEW*, que configura a aplicação *Web* a ser gerada. Se desejado, este documento pode ser editado pelo desenvolvedor de acordo com a linguagem declarativa em XML chamada CONFIG.

CONFIG permite a configuração de informações gerais sobre a aplicação, tal qual seu nome, a tecnologia utilizada, as fontes de dados e os formatos de apresentação para os nós, além de qual deve ser o primeiro nó na navegação. Existem valores padrões para todas as informações: o primeiro nó padrão é o *MainMenu*, geralmente adicionado na aplicação *Web*; o nome padrão da aplicação é *wVIEW*; a forma de acesso ou tecnologia padrão é Cocoon; a fonte de dados padrão é uma base de dados relacional; e o formato de apresentação padrão é HTML.

A seguir, é listado um exemplo do documento de configuração. Assim que a *wVIEW* passar a oferecer novas opções, o desenvolvedor poderá, por exemplo, configurar diferentes fontes de dados e formatos de apresentação para nós específicos. Novas opções para fontes de dados poderiam ser bases de dados XML, *Web Services*, além de outras fontes. No caso de formatos de apresentação, poderiam ser PDF, SMIL entre muitos outros.

```
<config>
  <web-application name="wVIEW"/>
  <first-node name="MainMenu"/>
  <access name="cocoon"/>
  <source default="rdb"/>
  <format default="html"/>
</config>
```

Durante a navegação, a aplicação *Web* gerada deve ser capaz de consultar corretamente a base de dados a fim de obter o conteúdo a ser apresentado especificamente para o nó visitado. Assim, como uma aplicação *Web* baseada no Cocoon, a aplicação *Web* gerada deve executar uma página XSP, consultando a base de dados para cada nó. Todas as páginas XSP poderiam ser

previamente geradas. Porém, além das dificuldades para manutenção, isto repete código — como o código de conexão à base de dados — que poderia ser adicionado no momento em que uma requisição por um nó é atendida.

A wVIEW permite a geração de páginas XSP assim que requisitadas por meio de um documento de conteúdo (*Content* na Figura 5.2). Este documento é gerado a partir do modelo navegacional, separando que conteúdo é necessário para cada nó. Assim, esse documento contém a especificação de todas as consultas para todos os nós.

A geração sob demanda de uma página XSP ocorre associando a folha de estilo XSP.XSL — uma folha de estilo estática provida pela wVIEW — com um documento de conteúdo gerado. Assim, de acordo com as consultas contidas neste documento, a folha de estilo XSP.XSL gera o código de uma página XSP para um dado nó. Quando a página XSP é executada, é gerado o conteúdo necessário para o nó.

Um processo similar ocorre para a folha de estilo XSLT que deve gerar a apresentação do nó. A wVIEW gera, a partir do modelo de apresentação, um documento de estilo (*Style* na Figura 5.2) que contém a especificação para formatar cada nó. Quando um nó é requisitado, esse documento é processado por meio da folha de estilo XSL.XSL — outra folha de estilo estática provida pela wVIEW. Assim, uma folha de estilo XSLT específica é gerada para transformar o conteúdo obtido com a página XSP.

É importante notar que a geração dos nós e páginas XSP e folhas de estilo XSLT associadas ocorrem em tempo de execução — ativado pelo *sitemap* da aplicação Web gerada. Isto não é um processamento demorado, uma vez que o *framework* Cocoon faz *cache* dos documentos gerados.

5.2.1 Modelo Conceitual (CM)

O modelo conceitual representa um diagrama de classes, contendo somente classes conceituais com atributos e relações entre as classes conceituais. Esse modelo é gerado a partir de um documento XMI e, por sua vez, é a base para a geração do modelo navegacional e para a geração dos *scripts* de criação do espaço de informação da aplicação Web. Na Figura 5.3 é detalhado como essas gerações são executadas no processo de transformação wVIEW.

Uma versão inicial do modelo conceitual (CM na Figura 5.3) é obtida apenas filtrando o documento XMI. O documento XMI é processado por meio da folha de estilo CM.XSL, uma folha de estilo estática da wVIEW, que gera: classes conceituais para todas as classes criadas pelo desenvolvedor; atributos com tipos específicos de acordo com os atributos do desenvolvedor; e relações para qualquer associação, composição e agregação especificada pelo desenvolvedor. Na versão atual da wVIEW, relações de herança e classes associativas não são

Figura 5.3: Transformações para o modelo conceitual.

consideradas.

Figura 5.4: Um diagrama de classes como entrada para o processo de transformação wVIEW.

Na Figura 5.4 é mostrado um simples diagrama de classes, que modela o domínio de aulas, para ilustrar a geração do modelo conceitual. Esse diagrama de classes foi especificado usando a ferramenta CASE ArgoUML e foi exportado para um documento no formato XMI. O documento XMI gerado foi usado como entrada para o processo de transformação wVIEW. A seguir, é listado o modelo conceitual inicial gerado ao se filtrar o documento XMI.

```

<conceptual-model>
  <conceptual-class name="Course">
    <attribute name="name" type="string"/>
    <attribute name="homepage" type="url"/>
  </conceptual-class>
  <conceptual-class name="Student">
    <attribute name="name" type="string"/>
    <attribute name="email" type="string"/>
  </conceptual-class>
  <conceptual-class name="Instructor">
    <attribute name="name" type="string"/>
    <attribute name="email" type="string"/>
  </conceptual-class>
  <conceptual-class name="Lecture">
    <attribute name="title" type="string"/>
    <attribute name="date" type="date"/>
  </conceptual-class>
  <relationship name="has" type="composition">
    <from conceptual-class="Course" multiplicity="1"/>
    <to conceptual-class="Lecture" multiplicity="*"/>
  </relationship>
  <relationship name="matriculates" type="association">
    <from conceptual-class="Student" multiplicity="*"/>
    <to conceptual-class="Course" multiplicity="*"/>
  </relationship>
  <relationship name="teaches" type="association">
    <from conceptual-class="Instructor" multiplicity="1"/>
    <to conceptual-class="Course" multiplicity="*"/>
  </relationship>
</conceptual-model>

```

Durante a navegação, nem toda a informação modelada inicialmente deve sempre aparecer e, algumas vezes, o desenvolvedor também pode estar interessado em adicionar outras informações. A fim de permitir essas operações sobre o modelo conceitual, a wVIEW provê uma linguagem declarativa baseada em XML — chamada CM-INSTRUCTIONS — para explicitamente omitir algumas classes conceituais, atributos e/ou relações, além de explicitamente adicionar algumas relações e/ou atributos de classes conceituais omitidas em outras classes conceituais.

O padrão é que nenhum elemento conceitual seja omitido ou adicionado. Então, o documento original de instruções sobre o modelo conceitual possui os elementos *omit* e *add* vazios. Se desejado, o desenvolvedor pode editar este documento omitindo ou adicionando elementos conceituais de acordo com a

CM-INSTRUCTIONS. A seguir, é listado um exemplo para o documento de instruções (*CM-INSTRUCTIONS* na Figura 5.3).

```
<cm-instructions>
  <omit>
    <conceptual-class name="Student"/>
    <conceptual-class name="Instructor"/>
  </omit>
  <add>
    <attributes conceptual-class="Course">
      <attribute name="name">
        <of conceptual-class="Instructor"/>
        <through relationship="teaches"/>
      </attribute>
    </attributes>
  </add>
</cm-instructions>
```

De acordo com este documento, as classes conceituais *Student* e *Instructor* devem ser omitidas. Além disso, o atributo *name* da classe conceitual *Instructor* deve ser adicionado na classe conceitual *Course* através da relação *teaches*. Se a multiplicidade da classe conceitual de origem é muitos (*), o atributo a ser adicionado deve ser um conjunto (*set*) na classe conceitual de destino.

O documento contendo as instruções é processado por meio da folha de estilo CM-INSTRUCTIONS.XSL, uma folha de estilo estática provida pela wVIEW, que gera outra folha de estilo, a CM-MOD.XSL. Se o desenvolvedor não especificar nenhuma instrução, então a CM-MOD.XSL somente copia o modelo conceitual original.

A folha de estilo CM-MOD.XSL é uma folha de estilo específica para a primeira versão do modelo conceitual. Esta folha de estilo modifica o modelo conceitual, gerando uma nova versão para o mesmo (*CM-MOD* na Figura 5.3). O modelo conceitual modificado é gerado de acordo com as instruções que foram especificadas pelo desenvolvedor. Assim, esse documento contém exatamente os elementos conceituais desejados e representa o espaço de informação que, de fato, será navegado.

Além de executar as instruções, a folha de estilo CM-MOD.XSL adiciona identificadores para as classes conceituais e mapeia as relações no modelo conceitual modificado. Isso se trata de uma preparação para a navegação que inclusive facilita a geração dos *scripts* de criação do espaço de informação.

No caso, a geração desses *scripts* é executada por meio de folhas de estilo fornecidas pela wVIEW: CREATE.XSL, ALTER.XSL e INSERT.XSL. A folha de

estilo CREATE.XSL mapeia as classes conceituais com seus atributos e relações para tabelas com colunas em uma base de dados relacional, enquanto que a folha de estilo ALTER.XSL adiciona as chaves estrangeiras e a folha de estilo INSERT.XSL gera um *script* para popular a base de dados.

A seguir, é listado o modelo conceitual modificado baseado no documento de instruções apresentado como exemplo. O atributo *name* foi adicionado e as classes conceituais *Instructor* e *Student* foram omitidas, como especificado. É importante notar que as relações com classes conceituais omitidas também são omitidas. Além disso, as classes conceituais receberam identificadores — atributos *id* — e as relações foram mapeadas — por exemplo, incluindo o atributo *idCourse* na classe conceitual *Lecture* dada a multiplicidade da relação *has*.

```
<conceptual-model>
  <conceptual-class name="Course">
    <attribute name="name" type="string"/>
    <attribute name="homepage" type="url"/>
    <attribute name="nameInstructor" type="string" >
    <attribute name="id" type="integer"/>
  </conceptual-class>
  <conceptual-class name="Lecture">
    <attribute name="title" type="string"/>
    <attribute name="date" type="date"/>
    <attribute name="id" type="integer"/>
    <attribute name="idCourse" type="integer"/>
  </conceptual-class>
  <relationship name="has" type="composition">
    <from conceptual-class="Course" multiplicity="1"/>
    <to conceptual-class="Lecture" multiplicity="*/>
  </relationship>
</conceptual-model>
```

5.2.2 Modelo Navegacional (NM)

O modelo navegacional define a navegação sobre o espaço de informação — modificado ou não — através de nós e ligações. Este modelo é gerado a partir do modelo conceitual modificado. Por sua vez, o modelo navegacional é a base para a geração do modelo de apresentação e para a geração do documento que especifica o conteúdo para os nós. Na Figura 5.5 é ilustrado como essas gerações ocorrem dentro do processo de transformação wVIEW.

Os elementos navegacionais podem ser derivados a partir dos elementos conceituais. Por exemplo, todas as classes conceituais podem ser mapeadas

Figura 5.5: Transformações para o modelo navegacional.

para nós contendo seus atributos — classes navegacionais — e as relações podem ser mapeadas para ligações implementadas por estruturas de acesso — tais como índices e menus — dependendo da multiplicidade associada. No caso, apenas índices e menus são as estruturas de acesso disponíveis nessa versão atual da wVIEW.

Existem algumas opções de mapeamento que podem ser escolhidas por meio de outra linguagem declarativa baseada em XML provida pela wVIEW, a CM2NM. As regras de mapeamento para uma aplicação *Web* particular são especificadas em um documento de mapeamento (CM2NM na Figura 5.5) baseado na CM2NM. Essas regras podem ter exceções para mapear um elemento específico de uma forma diferente. Se assim desejado, o documento pode ser editado pelo desenvolvedor especificando suas próprias regras de mapeamento.

Além disso, elementos pré-definidos podem ser adicionados. Até o momento, a wVIEW provê apenas um elemento pré-definido, o elemento *main-menu*. Este elemento inclui um índice para cada classe conceitual e um menu, intitulado *MainMenu*, que aponta estes índices. Na wVIEW, o padrão é que esse menu seja o ponto inicial da aplicação *Web*. Vale lembrar que o desenvolvedor pode configurar qualquer outro nó para ser o nó raiz da aplicação.

A seguir, é listado o documento de mapeamento padrão. As classes conceituais são diretamente mapeadas para classes navegacionais e as relações são mapeadas de acordo com o valor da multiplicidade da classe conceitual de destino. Se essa multiplicidade for 1, mapeia-se a relação para um menu

dentro da classe de origem que aponta para a classe de destino. Do contrário, se a multiplicidade for muitos, a relação é mapeada para um menu dentro da classe de origem que aponta para um índice que, por sua vez, aponta para a classe de destino.

```
<cm2nm>
  <map conceptual-element="conceptual-class">
    <to navigational-element="navigational-class"/>
  </map>
  <map conceptual-element="relationship-1">
    <to navigational-element="menu-in"/>
  </map>
  <map conceptual-element="relationship-*">
    <to navigational-element="index"/>
  </map>
  <add>
    <main-menu/>
  </add>
</cm2nm>
```

O documento que contém as regras de mapeamento (CM2NM na Figura 5.5) é processado por meio da folha de estilo CM2NM.XSL, outra folha de estilo provida pela wVIEW. Essa folha de estilo gera outra folha de estilo, chamada NM.XSL, específica para o modelo conceitual modificado (CM-MOD na Figura 5.5). A folha de estilo NM.XSL mapeia os elementos conceituais para elementos navegacionais de acordo com as regras de mapeamento especificadas, gerando o modelo navegacional correspondente (NM na Figura 5.5).

A seguir, é listado o modelo navegacional para as regras de mapeamento apresentadas. Como pode ser notado, cada classe navegacional recebeu um parâmetro para se identificar especificamente um *Course* ou uma *Lecture*. Além disso, a relação *has* foi corretamente mapeada para um menu, intitulado *has*, dentro da classe navegacional *Course*. Esse menu aponta para o índice *LectureByTitleByCourse*, que lista as aulas ordenadas por título de um determinado curso.

```
<navigational-model>
  <navigational-class name="Course" conceptual-class="Course">
    <attribute name="name" type="string"/>
    <attribute name="homepage" type="url"/>
    <attribute name="nameInstructor" type="string"/>
    <with attribute="id" type="integer" equal-parameter="id"/>
  </navigational-class>
  <navigational-class name="Lecture" conceptual-class="Lecture">
```

```

    <attribute name="title" type="string"/>
    <attribute name="date" type="date"/>
    <navigation type="sequential">
        <order-by attribute="date"/>
    </navigation>
    <with attribute="id" type="integer" equal-parameter="id"/>
</navigational-class>
<index name="LectureByTitleByCourse" conceptual-class="Lecture">
    <item attribute="title" type="string">
        <target name="Lecture">
            <parameter name="id" value-of="id"/>
        </target>
    </item>
    <order-by attribute="title"/>
    <relationship name="has" associative="false"/>
    <with attribute="idCourse" type="integer" equal-parameter="idCourse"/>
</index>
<menu name="has">
    <item value="Lecture" type="string">
        <target name="LectureByTitleByCourse">
            <parameter name="idCourse" value-of="id"/>
        </target>
    </item>
    <relationship name="has" associative="false"/>
    <with attribute="id" type="integer" equal-parameter="id"/>
    <in navigational-class="Course">
        <parameter name="id" value-of="id"/>
    </in>
</menu>
...
</navigational-model>

```

O modelo navegacional gerado (NM na Figura 5.5) associado ao documento de configuração (*Config* na Figura 5.5) é usado para gerar o documento de conteúdo (*Content* na Figura 5.5). Esta associação é realizada para incluir informação sobre as fontes de dados dos nós. A geração do documento de conteúdo ocorre por meio da folha de estilo CONTENT.XSL, uma folha de estilo provida pela wVIEW, que separa o conteúdo necessário para cada nó.

5.2.3 Modelo de Apresentação (PM)

O modelo de apresentação especifica como os elementos navegacionais devem ser apresentados para o usuário final. Este modelo é gerado a partir do

modelo navegacional e é a base para a geração do documento que contém regras de estilo para os nós. Na Figura 5.6 é mostrado como ocorrem essas gerações, que fazem parte do processo de transformação wVIEW.

Figura 5.6: Transformações para o modelo de apresentação.

Geralmente os elementos navegacionais possuem representações específicas. Por exemplo, um índice será sempre uma coleção de itens com âncora. Assim, o mapeamento de elementos navegacionais em elementos de apresentação é praticamente uma tradução para as primitivas de modelagem da apresentação. As primitivas básicas estão relacionadas com mídias — como texto e imagem — além de âncora e coleção. Até o momento, apenas essas primitivas de modelagem da apresentação foram implementadas na wVIEW.

Os elementos navegacionais — tanto classes navegacionais, quanto estruturas de acesso — são mapeados com a mesmas primitivas de modelagem da apresentação. Por exemplo, atributos de classes navegacionais e itens de estruturas de acesso são mapeados de acordo com seus tipos. Além disso, classes navegacionais e estruturas de acesso são mapeadas para classes de apresentação, já que possuem a mesma estrutura em termos de apresentação.

O modelo navegacional (NM na Figura 5.6) gera o modelo de apresentação (PM na Figura 5.6) por meio da folha de estilo PM.XSL, uma folha de estilo estática provida pela wVIEW. Esta folha de estilo aplica o mapeamento entre elementos navegacionais e de apresentação. Vale ressaltar que apenas informações que impactam no estilo e ligações do modelo navegacional são consideradas no modelo de apresentação.

A seguir, é listado um fragmento do modelo de apresentação gerado para o modelo navegacional mostrado anteriormente. Pode-se notar que as classes navegacionais e as estruturas de acesso foram mapeadas para classes de apresentação. Além disso, o atributo *name* da classe navegacional *Course* foi mapeada para um elemento de apresentação texto, em função do seu tipo *string*. No caso do atributo *homepage*, esse foi mapeado para um texto com âncora, dado que seu tipo é url. *LectureByTitleByCourse* como um índice, possui uma coleção de texto com âncora referente ao atributo título, cujos itens apontam para aulas correspondentes.

```
<presentational-model>
  <presentational-class name="Course">
    <text attribute="name"/>
    <anchor>
      <text attribute="homepage"/>
    </anchor>
    <text attribute="nameInstructor"/>
    <add>
      <presentational-class name="has"/>
    </add>
  </presentational-class>
  ...
  <presentational-class name="LectureByTitleByCourse">
    <anchor>
      <target name="Lecture">
        <parameter name="id" value-of="id"/>
      </target>
      <collection attribute="title" type="text"/>
    </anchor>
  </presentational-class>
  ...
</presentational-model>
```

Em seguida, o modelo de apresentação associado ao documento de configuração (*Config* na Figura 5.6) gera o documento de estilo (*Style* na Figura 5.6) por meio da folha de estilo STYLE.XSL — um folha de estilo estática provida pela wVIEW. Neste caso, a associação é feita para incluir informação sobre os formatos dos nós.

5.3 Aplicação Web Gerada

Uma aplicação *Web* é o resultado de todo o processo de transformação wVIEW. Dado um diagrama de classes, a wVIEW gera documentos que des-

crevem o espaço de informação, a navegação e a apresentação dos nós. Ao mesmo tempo, a wVIEW disponibiliza duas folhas de estilo para montar os nós sob demanda, conforme a navegação flui. A Figura 5.7 mostra a execução de uma aplicação *Web* gerando um nó sob demanda.

Figura 5.7: Execução de uma aplicação *Web* gerada.

Alguns documentos — mostrados na Figura 5.7 — são gerados compondo a aplicação *Web* resultante: *Scripts*, *Content*, *Style* e *Sitemap*. Além disso, as folhas de estilo XSP.XSL e XSL.XSL participam na composição da aplicação, totalizando 7 documentos. A abordagem da geração sob demanda dos nós implica nestes poucos documentos para manter. Quando o desenvolvedor recebe esses 7 documentos, deve seguir alguns passos de instalação da aplicação — antes ou depois de executar os *Scripts* para criar a base de dados e, se desejar, também populá-la.

Primeiramente, o desenvolvedor deve instalar o *framework* Cocoon para permitir a execução da aplicação *Web* gerada, criando um diretório apropriado para a aplicação. Em seguida, um *pool* deve ser criado no Cocoon para permitir a conexão à base de dados — este procedimento é simples e bastante documentado nos arquivos do Cocoon. O desenvolvedor também deve copiar os documentos *Content*, *Style*, *Sitemap*, XSP.XSL e XSL.XSL no diretório apropriado da aplicação. Neste momento, a aplicação *Web* é capaz de executar.

Depois de colocada em execução, um usuário final pode navegar através dos nós da aplicação *Web*. Então, cada nó visitado é gerado atendendo uma requisição do usuário por esse nó. Como já mencionado, em uma aplicação *Web* baseada no Cocoon, as requisições do usuário são atendidas de acordo com um *sitemap*. A seguir, é listada a *pipeline* contida no *Sitemap* para atender requisições do usuário por qualquer nó na aplicação *Web* gerada.

```
<map:pipeline>
  <map:match pattern="*.xsp">
    <map:generate type="file" src="content.xml"/>
    <map:transform src="xsp.xsl">
```

```

        <map:parameter name="node" value="{1}"/>
    </map:transform>
    <map:serialize type="xml"/>
</map:match>
<map:match pattern="*.xsl">
    <map:generate type="file" src="style.xml"/>
    <map:transform src="xsl.xsl">
        <map:parameter name="node" value="{1}"/>
    </map:transform>
    <map:serialize type="xml"/>
</map:match>
<map:match pattern="*.html">
    <map:generate src="cocoon://{1}.xsp"/>
    <map:transform src="cocoon://{1}.xsl"/>
    <map:serialize type="html"/>
</map:match>
<map:match pattern="">
    <map:redirect-to uri="cocoon:/MainMenu.html"/>
</map:match>
</map:pipeline>

```

Quando o usuário faz a requisição pela página inicial da aplicação *Web*, o *Sitemap* redireciona para o primeiro nó configurado pelo desenvolvedor. Isto invoca uma requisição interna pelo nó, que é tratado como uma requisição normal por um nó. Uma requisição por um nó é composta pelo nome do nó com a extensão relacionada ao seu formato e, se for o caso, parâmetros. No caso da aplicação *Web* gerada, qualquer requisição por um nó casa com **.html* no *Sitemap* — uma vez que apenas o formato HTML é possível até o momento. Assim, o *Sitemap* internamente invoca a página XSP para o nó (XSP na Figura 5.7).

A fim de gerar a página XSP, o *Content* é processado por meio da folha de estilo XSP.XSL, passando o nome do nó requisitado. Depois de gerada, a página XSP é executada causando uma consulta à base de dados pelo conteúdo necessário descrito no *Content*. Assim, um documento XML é gerado para o nó (XML na Figura 5.7).

Em seguida, a folha de estilo XSLT (XSL na Figura 5.7), que deve transformar o conteúdo XML obtido, é gerada em um processo similar: o *Sitemap* internamente invoca a folha de estilo XSLT para o nó requisitado e o *Style* is processado por meio da folha de estilo XSL.XSL, passando o nome do nó. Finalmente, o nó é gerado e apresentado quando o processamento do conteúdo XML por meio da folha de estilo XSLT termina.

5.4 Pipeline da Infra-estrutura wVIEW

O processo de transformação wVIEW é uma seqüência de transformações XSLT definida por uma *pipeline* com entrada (documento XML), processamento (transformações dos modelos) e saída (documentos da aplicação Web gerada). Esse processo poderia ser implementado de várias maneiras, por exemplo, utilizando uma linguagem de programação como Java ou mesmo uma *pipeline* do *framework* Cocoon. Nesse último caso, seria uma aplicação Web baseada no Cocoon gerando outras aplicações Web também baseadas no Cocoon. Essa é uma abordagem bastante interessante, pois, uma vez a wVIEW sendo disponibilizada na Web, qualquer pessoa poderia gerar aplicações sem ter que, por exemplo, fazer nenhuma instalação para utilizar essa infra-estrutura. Dessa forma, o processo de transformação wVIEW foi implementado em uma *pipeline* Cocoon.

Foi especificado um documento SVG para ilustrar e ativar o processo e a aplicação Web gerada ¹. Na Figura 5.8 pode ser visualizado esse documento, que contém representações gráficas com ligações para todos os documentos que são providos ou gerados pela wVIEW, incluindo ligações para seus esquemas. O desenvolvedor pode acompanhar todo o processo e ainda visualizar a aplicação Web selecionando os documentos correspondentes.

5.5 Comparação com Trabalhos Relacionados

Quando abordagens para suporte ao projeto hipermídia são utilizadas, elas geralmente exigem muita atenção do desenvolvedor para suas primitivas de modelagem, processos de desenvolvimento e ferramentas associadas — algumas vezes desviando o desenvolvedor do seu objetivo principal que é obter uma aplicação Web.

A principal vantagem da wVIEW com relação a outras abordagens é sua simplicidade, resultante da estreita associação com XML e tecnologias afins — completamente modelando e gerando aplicações Web por meio de transformações XSLT sucessivas. Ao utilizar a wVIEW, o desenvolvedor não precisa instalar nenhuma ferramenta específica, ao contrário do que acontece com as abordagens que disponibilizam ferramentas. Assim, o desenvolvedor não precisa aprender a usar nenhuma ferramenta específica. Na verdade, o desenvolvedor sequer precisa conhecer o processo de transformação wVIEW, uma vez que existem valores padrões para tudo. Apesar disso, não é uma tarefa difícil interferir durante o processo para ajustá-lo a requisitos particulares.

A wVIEW foi construída baseada em conceitos da área de Hipermídia que

¹Disponível em <http://coweb.icmc.usp.br/coweb/mostra.php?ident=5.15>

representar modelos, como faz a wVIEW, também é explorada pelas abordagens WebML (Matera, Maurino, Ceri, & Fraternali 2003), OOHDM (Medeiros & Schwabe 2001) e UWE (Kraus & Koch 2002). No WebML as especificações são processadas automaticamente gerando de uma vez um conjunto de *templates* e descritores dos dados das páginas — separando conteúdo e estilo como faz a wVIEW. Porém, dependendo do tamanho da aplicação esse conjunto pode ser muito grande dificultando sua manutenção.

No caso do OOHDM, foi especificada a linguagem OOHDM-ML para representar de forma declarativa seus modelos conceitual, navegacional e de interface — assim como é feito na wVIEW. Porém, a especificação baseada em OOHDM-ML referente a um projeto OOHDM deve ser completamente escrita pelo desenvolvedor sem suporte algum. Além disso, o desenvolvedor deve diretamente especificar tanto o mapeamento para a base de dados quanto os objetos de interface. A partir de uma especificação OOHDM-ML, o ambiente OOHDM-XWeb gera automaticamente a descrição da aplicação para o sistema OOHDM-Web. Para a geração é utilizada uma folha de estilo XSLT de tradução.

A metodologia UWE disponibiliza um método, o UWEXML, para a transição de seus modelos para uma aplicação *Web* baseada no *framework* Cocoon, inclusive utilizando um documento XMI. A transição não é automática — na wVIEW é totalmente automática — e a aplicação *Web* é baseada na primeira versão do Cocoon — a wVIEW utiliza a segunda versão. A primeira versão do Cocoon possui problemas de desempenho por utilizar DOM — na segunda versão passou a utilizar SAX — e de gerenciamento, já que o processamento dos documentos é definido nos próprios documentos — na segunda versão o *site-map* passou a ser utilizado para gerenciamento. Além disso, diferentemente da wVIEW, o documento XMI é utilizado apenas para gerar os esquemas das instâncias dos modelos.

As abordagens AutoWeb (Fraternali & Paolini 2000) e Araneus (Merialdo, Atzeni, & Mecca 2003) permitem a geração automática das aplicações *Web*, assim como a wVIEW o faz. Uma aplicação gerada pelo AutoWeb ou pelo Araneus trata-se, na verdade, de um esquema em uma base de dados relacional, enquanto que a wVIEW gera documentos XML — de conteúdo e estilo — que podem ser associados a folhas de estilo para gerar código para a aplicação em diferentes tecnologias. Até o momento, a wVIEW gera somente aplicações baseadas no Cocoon, mas pode ser estendida para gerar aplicações também em JSP, PHP, e outras tecnologias, apenas especificando-se folhas de estilo apropriadas.

Além disso, o AutoWeb e o Araneus geram apenas aplicações que utilizam bases de dados relacionais e apresentam o conteúdo somente em HTML. Atu-

almente, a wVIEW gera aplicações apenas nessas condições também, porém pode ser estendida para outras fontes de dados e formatos sem demandar muito esforço, apenas seguindo o que feito para as opções atuais.

No caso, a abordagem WebML (Matera, Maurino, Ceri, & Fraternali 2003) também permite a geração automática de aplicações *Web*, que, além de utilizarem base de dados relacional e apresentarem o conteúdo em HTML, podem utilizar *Web Services* e apresentar em WML. Porém, essas aplicações dependem de uma infra-estrutura específica do WebML para executarem. Assim como o WebML, o Strudel também permite mais de uma fonte de dados, como páginas de uma outra aplicação *Web*. Isso é possível por meio de *wrappers* que mapeiam fonte de dados externas para o modelo do Strudel. Assim, inclusive as bases de dados são tratadas como elementos externos, o que adiciona um passo a mais para sua utilização.

Além de similaridades e vantagens, a wVIEW também possui limitações com relação às outras abordagens para projeto hiperídia. Por exemplo, não provê mecanismos de personalização, como os providos pelas abordagens OOHD (Rossi, Schwabe, & Guimarães 2001), WebML (Fraternali & Paolini 1998), OntoWeaver (Lei, Motta, & Domingue 2003) e WSDM (De Troyer & Leune 1997). Além disso, não suporta entrada de dados, ao contrário do que fazem as abordagens Wizard (Turau 2002), OOWS (Pastor, Abrahao, & Fons 2001), Araneus (Merialdo, Atzeni, & Mecca 2003) e WebML (Matera, Maurino, Ceri, & Fraternali 2003). Outra limitação é que a wVIEW não oferece visualização gráfica para seus modelos, o que é feito por abordagens como OOHD, UWE, Araneus, AutoWeb e WebML. Todas essas limitações são candidatas a trabalhos futuros, que também são apresentados no próximo capítulo de conclusões.

Conclusões

A maioria das aplicações de captura e acesso utiliza aplicações *Web* para prover a visualização da informação que é capturada. Assim, desenvolver aplicações de captura e acesso também envolve o desenvolvimento de aplicações *Web*. Independente do contexto de captura e acesso de experiências, a tarefa de desenvolver aplicações *Web* é complexa, principalmente pela estrutura hipermídia de tais aplicações.

Várias metodologias têm sido propostas para o projeto hipermídia de aplicações *Web*, cada uma com seu conjunto de primitivas de modelagem, processo de desenvolvimento e ferramentas associadas. Como todas tratam o mesmo problema, alguns conceitos são recorrentes entre as metodologias, embora possuam diferentes notações. Por exemplo, nós e estruturas de acesso podem ser identificados em praticamente todas as metodologias. Além disso, algumas metodologias apresentam aspectos particulares interessantes, como o uso de XML em suas representações.

Com base nos conceitos recorrentes e algumas características específicas de metodologias de suporte ao projeto hipermídia de um modo geral, foi desenvolvida a infra-estrutura wVIEW apresentada nesta dissertação. Essa infra-estrutura é resultado da investigação do processo de visualização de aplicações de captura e acesso por meio de interfaces de acesso providas por aplicações *Web*. Embora tenha sido construída como parte do projeto InCA-SERVE, a wVIEW permite a geração automática de aplicações *Web* independente do contexto de aplicações de captura e acesso. A geração das aplicações ocorre por meio do processo de transformação wVIEW: transformações XSLT sucessivas sobre representações XML de modelos.

6.1 Contribuições

Diferentes contribuições deste trabalho de dissertação podem ser identificadas sob o ponto de vista do projeto InCA-SERVE, da área de Hipermedia e dos desenvolvedores de aplicações *Web* de um modo geral. As principais contribuições relacionadas ao projeto InCA-SERVE são:

- uma infra-estrutura genérica, chamada wVIEW, que permite a geração automática de aplicações *Web* no contexto de aplicações de captura e acesso;
- a infra-estrutura wVIEW pode ser facilmente integrada ao projeto InCA-SERVE pela sua abordagem totalmente baseada em XML; e,
- a infra-estrutura wVIEW, sendo baseada em XML, é bastante extensível podendo ser adequada a novas tecnologias e necessidades do projeto InCA-SERVE.

Para a área de Hipermedia, poderiam ser citadas as seguintes contribuições deste trabalho de dissertação:

- uma abordagem declarativa totalmente baseada em XML para suportar o projeto hipermedia de aplicações *Web*;
- representações XML para uma aplicação *Web* independente da tecnologia a ser utilizada que, quando associadas a folhas de estilo XSLT apropriadas, geram a implementação da aplicação; e,
- como a wVIEW utiliza vários conceitos comuns às outras metodologias, ela poderia ser adaptada para ser utilizada em outra abordagem de projeto hipermedia.

Com relação aos desenvolvedores de aplicações *Web* de um modo geral, as principais contribuições são:

- a geração automática de uma aplicação *Web* completa no tempo de algumas transformações XSLT;
- a não exigência de instalação de um ambiente de desenvolvimento, além de não exigir que se conheça profundamente sobre as primitivas de modelagem e processo de desenvolvimento; e,
- a total liberdade para interferir em qualquer ponto do processo de desenvolvimento da aplicação.

6.2 Trabalhos Futuros

A primeira versão da infra-estrutura wVIEW por questões de simplicidade foi implementada com apenas algumas opções básicas de modelagem e de configuração da aplicação. A preocupação maior deste trabalho de dissertação foi obter o processo de desenvolvimento completo e funcional baseado em transformações XSLT, pois assim novas opções poderiam ser acopladas conforme foi feito para as opções básicas. Além disso, limitações da wVIEW com relação a outras abordagens para projeto hipermídia podem ser consideradas em suas versões futuras. Assim, entre os trabalhos futuros para estender a wVIEW pode-se citar:

- com relação à modelagem, a definição de outras estruturas de acesso além de índices e menus, como *guided tour* e *query*, e de outros elementos pré-definidos além de *main-menu*, como histórico de navegação (para indicar o caminho até o nó sendo visitado);
- com relação à configuração, a definição de outras formas de acesso além de Cocoon, como JSP e PHP; de outras fontes de dados além de bases de dados relacionais, como bases de dados XML e *Web Services*; e, de outros formatos além de HTML, como PDF e SMIL;
- a wVIEW poderia ser estendida para permitir a geração de aplicações *Web* com entrada de dados, partindo da investigação na literatura do que tem sido feito por outras metodologias neste sentido;
- a wVIEW também poderia ser estendida para permitir a personalização em aplicações *Web* a serem geradas;
- definição de representações gráficas para os modelos, por exemplo em documentos SVG, que também poderiam ser obtidos por transformações XSLT, já que a wVIEW é totalmente baseada em XML.

6.3 Considerações Finais

Nesta dissertação foi apresentada a infra-estrutura wVIEW, uma abordagem declarativa para automatizar a geração de aplicações *Web*. Além disso, foram abordados alguns conceitos da área de Hipermídia, sobre os quais se baseiam tanto os modelos quanto o processo de desenvolvimento da wVIEW. Como essa infra-estrutura faz parte do projeto InCA-SERVE, algumas de suas infra-estruturas e serviços foram considerados, assim como a área de computação ubíqua, com ênfase em aplicações de captura e acesso.

Referências

- Abowd, G. A. (1999a). Software engineering issues for ubiquitous computing. In *Proceedings of the 21st international conference on Software engineering*, Los Angeles, California, United States, pp. 75–84. IEEE Computer Society Press.
- Abowd, G. D. (1999b). Classroom 2000: an experiment with the instrumentation of a living educational environment. *IBM Systems Journal* 38(4), 508–530.
- Abowd, G. D. & E. D. Mynatt (2000, March). Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI)* 7(1), 31–58.
- Abowd, G. D., E. D. Mynatt, & T. Rodden (2002, January-March). The human experience. *IEEE Pervasive Computing* 1(1), 48–57.
- Andrade, A. R. (2002, Outubro). Acesso e apresentação de sessões capturadas e armazenadas pelo iclass. Monografia de Projeto de Graduação ICMC-USP.
- Apache (2002). The Apache Software Foundation: Apache Cocoon 2.0. URL:<http://cocoon.apache.org/2.0/>.
- Apache (2003). The Apache Software Foundation: Apache Cocoon. URL:<http://cocoon.apache.org/>.
- Arruda, C. R. E. (2002). Investigação, formalização e desenvolvimento de núcleo de operações para suporte a aplicações conscientes de contexto. Monografia de Qualificação de Mestrado ICMC-USP.
- Baldochi, L. A. (2002). Suporte ao armazenamento, recuperação e extensão de informação resultante de atividades de captura. Monografia de Qualificação de Doutorado ICMC-USP.
- Baumeister, H., N. Koch, & L. Mandel (1999). Towards a uml extension for hypermedia design. In *Proceedings of the Unified Modeling Language*

- Conference*, pp. 614–629. Springer Verlag.
- Bochicchio, M. A., R. Paiano, & P. Paolini (1999). Jweb: An hdm environment for fast development of web applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems Volume II-Volume 2*, pp. 809. IEEE Computer Society.
- Brotherton, J. A., G. D. Abowd, & K. N. Truong (1998). **Supporting capture and access interfaces for informal and opportunistic meetings**. GATECH GIT-GVU-99-06.
- Brotherton, J. A., J. Bhalodia, & G. D. Abowd (1998, Julho). Automated capture, integration and visualization of multiple media streams. In *Proceedings of IEEE Multimedia*, pp. 54–63.
- Bulcão, R. F. (2001, Outubro). Wls: um serviço aberto de ligações hiper-mídia para web baseado em xml. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação da USP. São Carlos – SP.
- Card, S., J. Mackinlay, & B. Shneiderman (1999). Readings in information visualization using vision to think. pp. 686. Morgan Kaufmann Publishers.
- Cattelan, R. (2004). Construção de aplicações de captura e acesso baseada em recorrência de funcionalidades. Dissertação de Mestrado, Universidade de São Paulo.
- Cattelan, R. G. (2002, Maio). xınca: construção de aplicações de captura e acesso baseada em montagem de componentes de software. Plano de mestrado submetido à FAPESP.
- Cattelan, R. G., A. R. Andrade, C. F. R. Penedo, & M. G. C. Pimentel (2002). iclass: um sistema para captura e acesso de sessões em ambiente educacional. In *REIC - Revista Eletrônica de Iniciação Científica*. SBC.
- Cowan, D. D. & C. J. P. Lucena (1995). Abstract data views, an interface specification concept to enhance design for reuse. *IEEE Transactions on Software Engineering* 21(3), 207–225.
- De Troyer, O. & C. Leune (1997). Wsdm: A user-centered design method for web sites. In *Proceedings of the 7th International World Wide Web Conference*.
- Dey, A. (2001). Understanding and using context. *Personal UbiComp* 5(1), 4–7.
- Diaz, A., T. Isakowitz, V. Maiorana, & G. Gilabert (1995). RMC: A tool to design WWW applications. In *Fourth Int. WWW Conf.*, pp. 11–14.

- Fernández, M., D. Suciu, & I. Tatarinov (1999). Declarative specification of data-intensive web sites. In *Proceedings of the 2nd conference on Domain-specific languages*, pp. 135–148. ACM Press.
- Fraternali, P. & P. Paolini (1998). A conceptual model and a tool environment for developing more scalable, dynamic, and customizable web applications. In *Proceedings of the 6th International Conference on Extending Database Technology*, pp. 421–435. Springer-Verlag.
- Fraternali, P. & P. Paolini (2000). Model-driven development of web applications: the AutoWeb system. *ACM Trans. Inf. Syst.* 18(4), 323–382.
- Garzotto, F., P. Paolini, & D. Schwabe (1993). Dm-a model-based approach to hypertext application design. *ACM Transactions on Information Systems (TOIS)* 11(15), 1–26.
- Gomez, J., C. Cachero, & O. Pastor (2000). Extending a conceptual modeling approach to web application design. In *Proceedings of Conference on Advanced Information Systems Engineering*, pp. 79–93. Springer Verlag.
- Hauptmann, A. (1995, November). Speech recognition in the informedia digital video library: Uses and limitations. *IEEE International Conference on Tools with AI (ICTAI)*.
- Hennicker, R. & N. Koch (2000). A uml-based methodology for hypermedia design. In *Proceedings of the Unified Modeling Language Conference*, pp. 410–424. Springer Verlag.
- IBM (2003). Rational software. <http://www-306.ibm.com/software/rational/>.
- Isakowitz, T., E. A. Stohr, & P. Balasubramanian (1995). RMM: a methodology for structured hypermedia design. *Commun. ACM* 58(8), 34–44.
- Ishii, H. & B. Ullmer (1997, March). Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*.
- Izeki, C. A. (2001). Anotações colaborativas como hiperdocumentos de primeira classe na web semântica. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação da USP, São Carlos – SP.
- Kazman, R., G. D. Abowd, L. Bass, & P. Clements (1996). Scenario-based analysis of software architecture. In *IEEE Software*, pp. 47–55.
- Kojarski, S. & D. H. Lorenz (2003). Domain driven web development with webjinn. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 53–65. ACM Press.

- Kraus, A. & N. Koch (2002). Generation of web applications from uml models using an xml publishing framework. In *Proceedings of the sixth World Conference on Integrated Design and Process Technology*.
- Lei, Y., E. Motta, & J. Domingue (2003). Design of customized web applications with ontoweaver. In *Proceedings of the international conference on Knowledge capture*, pp. 54–61. ACM Press.
- Matera, M., A. Maurino, S. Ceri, & P. Fraternali (2003). Model-driven design of collaborative web applications. *Softw. Pract. Exper.* 33(8), 701–732.
- Medeiros, A. P. & D. Schwabe (2001). Especificação declarativa de aplicações web em oohdm. In *Anais do Simpósio Brasileiro de Sistemas Multimídia e Hiperídia - SBMídia*, pp. 122–138. Sociedade Brasileira de Computação.
- Merialdo, P., P. Atzeni, & G. Mecca (2003). Design and development of data-intensive web sites: The Araneus approach. *ACM Trans. Inter. Tech.* 3(1), 49–92.
- Milosavljevic, B., M. Vidakovic, & Z. Konjovic (2002). Automatic code generation for database-oriented web applications. In *Proceedings of the inaugural conference on the Principles and Practice of programming, 2002 and Proceedings of the second workshop on Intermediate representation engineering for virtual machines, 2002*, pp. 59–64. National University of Ireland.
- Minneman, S., S. Harrison, B. Janssen, G. Kurtenbach, T. Moran, I. Smith, & B. Van Melle (1995, November). A confederation of tools for capturing and accessing collaborative activity. In *Proceedings of the 3rd International Conference on Multimedia*, San Francisco.
- OMG (2003). Xml Metadata Interchange. <http://www.omg.org/technology/documents/formal/xmi.htm>.
- Oracle (2004). Oracle webdb. <http://www.oracle.com>.
- Oshiro, A. K. (2003, Janeiro). Visualização de informação evolucionária. Relatório de Atividades Realizadas de Bolsa de Mestrado CNPq.
- Pastor, O., S. Abrahao, & J. Fons (2001). Building e-commerce applications from object-oriented conceptual models. *SIGecom Exch.* 2(2), 28–36.
- Pimentel, M. G. C. & G. D. Abowd (1999). Development and understanding of automated capture environments to support long-term use. Projeto de cooperação internacional aprovado junto ao ProTeM-Cc-Cnpq/Brasil e ao NSF/EUA.
- Pimentel, M. G. C., R. P. M. Fortes, & E. S. Moreira (2001a). Development and understanding of automated capture environments to support long-

- term use: Proposal 1999. Technical report, Instituto de Ciências Matemáticas e de Computação da USP.
- Pimentel, M. G. C., R. P. M. Fortes, & E. S. Moreira (2001b). Development and understanding of automated capture environments to support long-term use: Report 1/2001. Technical report, Instituto de Ciências Matemáticas e de Computação da USP.
- Pimentel, M. G. C., Y. Kerimbaev, G. D. Abowd, & M. Guzdial (2001). Supporting long-term educational activities through dynamic web interfaces. *Interacting with Computers Journal* 13, 353–374.
- Rocha, C. F. P. (2004, Abril). Anotações com pdas: extensão da área de escrita e integração com o projeto inca-serve. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação da USP, So Carlos – SP.
- Rossi, G., D. Schwabe, & R. Guimarães (2001). Designing personalized web applications. In *Proceedings of the tenth international conference on World Wide Web*, pp. 275–284. ACM Press.
- Sante, D. (2003, Maio). Autore: suportando autoria evolucionária em ambientes de captura. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação da USP, São Carlos – SP.
- Schilit, B., G. Golovchinsky, & M. Price (1998). Beyond paper: supporting active reading with free form digital ink annotations. In *CHI 98 Conference Proceedings*, pp. 249–256. ACM Press.
- Schmandt, C. & B. Arons (1985). Phoneslave: a graphical telecommunications interface. In *Proc. Soc. Inf. Display* 26, pp. 79–82.
- Schwabe, D., R. S. Pontes, & I. Moura (1999). OOHDM-Web: An environment for implementation of hypermedia applications in the WWW. *SigWeb Newsletter* 8(2).
- Schwabe, D. & G. Rossi (1995). The object-oriented hypermedia design model. *Commun. ACM* 58(8), 45–46.
- Streitz, N., J. Geibler, J. Haake, & J. Hol (1994). Dolphin: integrated meeting support across local and remote desktop environments and liveboards. In *Proceedings of CSCW' 94*, pp. 345–358.
- Tritis.org (2003). Argouml: A uml design tool with cognitive support. <http://argouml.tigris.org/>.
- Truong, K. N. & G. D. Abowd (2002). Enabling the generation, preservation & use of records and memories of everyday life. In *Proceeding of CHI*.

- Truong, K. N. & G. D. Abowd (2004, April). Inca: A software infrastructure to facilitate the construction and evolution of ubiquitous capture & access applications. In *Proceedings of Pervasive 2004: The Second International Conference on Pervasive Computing*.
- Truong, K. N., G. D. Abowd, & J. A. Brotherton (2001). Who, what, when, where, how: Design issues of capture & access applications. In *Proceedings of the Ubicomp 2001*, Atlanta, Georgia, United States, pp. 209–224.
- Turau, V. (2002). A framework for automatic generation of web-based data entry applications based on xml. In *Proceedings of the 2002 ACM symposium on Applied computing*, pp. 1121–1126. ACM Press.
- Want, R., A. Hopper, V. Falcão, & J. Gibbons (1992, January). The active badge location system. *ACM Trans. Inf. Syst.* 10(1), 91–102.
- Want, R., B. Schilit, N. Adams, R. Gold, K. Petersen, J. Ellis, D. Goldberg, & M. Weiser (1995). The parctab ubiquitous computing environment. Technical report, Xerox PARC.
- WebModels (2004). WebRatio - model driven code generation of web applications. URL:<http://www.webratio.com/>.
- Weiser, M. (1991, September). The computer for the 21st century. In *Scientific American* 265, 3, pp. 94–104.
- Weiser, M. (1993, July). Some computer science issues in ubiquitous computing. *Communications of the ACM* 6(7), 75–84.
- Yagoub, K., D. Florescu, V. Issarny, & P. Valduriez (2000). Caching strategies for data-intensive web sites. In *Proceedings of the 26th International Conference on Very Large Databases*, pp. 188–199.
- Zhang, G., A. Knapp, N. Koch, A. Kraus, & R. Hennicker (2003). ArgoUWE - case tool for modelling web applications. URL:<http://www.pst.informatik.uni-muenchen.de/projekte/argouwe/>.