SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 07.06.2005

Assinatura: *Maria Paula Campos Pimentel*

Suporte ao armazenamento, recuperação,
apresentação e extensão de informação
proveniente de atividades de captura

Laércio Augusto Baldochi Júnior

Orientadora: Profa. Dra. Maria da Graça Campos Pimentel

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

USP – São Carlos
Junho de 2005

Aluna: Laércio Augusto Baldochi Junior

A Comissão Julgadora:

Profa. Dra. Maria da Graça Campos Pimentel

Profa. Dra. Renata Pontin de Mattos Fortes

Prof. Dr. Alberto Henrique Frade Laender

Prof. Dr. Ivan Luiz Marques Ricarte

Prof. Dr. José Gonçalves Pereira Filho

Dedicatória

*À minha esposa Hévilla, e ao nosso filho (ou filha) que está a caminho.
À minha mãe, Jaira, in memoriam.*

Agradecimentos

Inicialmente agradeço às minhas irmãs, Cidinha, Sandra e Sílvia, pelo constante apoio e preocupação. Agradeço também à Tereza e à Haline, pela simpatia e a vontade de ajudar sempre.

Agradeço à Profa. Graça Pimentel, pela oportunidade que me concedeu, pelos ensinamentos e pela infinita paciência que teve comigo nesses vários anos de convívio.

A todos os pesquisadores do Laboratório Intermídia, por fazerem do local de trabalho um ambiente agradável e descontraído. Deixo meu agradecimento especial aos colegas Renan, Patrão, Andrea, Hélder, Valter, Alessandra e Toño, pela colaboração no desenvolvimento desse trabalho.

Agradeço o apoio que recebi das Faculdades COC. Deixo meu agradecimento especial ao Prof. Farid C. Mauad e aos colegas Carlos Alberto, Patrícia, Daniel Pires e Daniel Lobato. Impossível deixar de mencionar também o Prof. Drugowitch e o Prof. Cesar Teixeira, os quais não fazem parte mais da instituição, mas, enquanto estiveram por lá, foram amigos de todas as horas.

Finalmente, agradeço a Deus e aos bons espíritos por me manterem firme no caminho, mesmo quando tudo parecia sem solução.

Resumo

A computação ubíqua tem se estabelecido como um novo paradigma de interação usuário-computador; sua essência está em auxiliar atividades humanas de modo minimamente intrusivo, permitindo a integração transparente de tecnologia no cotidiano dos indivíduos (Weiser 1991).

Um tema frequentemente explorado dentro da computação ubíqua é a construção de aplicações de captura acesso (Abowd and Mynatt 2000; Abowd et al. 2002). Gerenciando um conjunto de dispositivos ubíquos que inclui lousas eletrônicas, câmeras de vídeo e microfones, essas aplicações realizam a captura das mais diferentes experiências humanas, desde reuniões de trabalho até atividades educacionais em sala de aula, com o objetivo de permitir o posterior acesso ao conteúdo capturado.

O trabalho desenvolvido nesta tese tem como objetivo geral prover facilidades infra-estruturais e de serviços de modo a facilitar o desenvolvimento dessas aplicações. Trabalhos anteriores identificaram a necessidade de oferecer suporte de comunicação e funcionalidades às aplicações de captura e acesso. Esta tese tem como objetivos específicos tratar problemas associados à representação e ao gerenciamento da informação produzida pelas aplicações. Para tanto, foram definidas abstrações de alto nível capazes de facilitar o intercâmbio da informação capturada, seu armazenamento e sua apresentação na Web.

O efetivo suporte à estruturação e ao gerenciamento dos dados manipulados pelas aplicações de captura e acesso demanda uma modelagem consistente das informações de contexto específicas de cada aplicação. No sentido de apoiar essa modelagem, desenvolveu-se o *Contextractor*, metodologia que permite associar informações presentes no modelo conceitual das aplicações às dimensões de contexto comumente utilizadas por aplicações de captura e acesso. Entre seus resultados, o *Contextractor* produz esquemas XML usados como blocos arquiteturais na definição de uma infra-estrutura de software que oferece abstrações de alto nível para o desenvolvimento de aplicações de captura e acesso. Tais abstrações incluem: um modelo de desenvolvimento de aplicações baseado em componentes de software; um serviço capaz de oferecer facilidades de armazenamento e apresentação de conteúdo capturado de maneira transparente; e uma linguagem para consulta de conteúdo capturado baseada nas informações de contexto da aplicação.

A metodologia de modelagem, o modelo de desenvolvimento e os serviços de gerenciamento de conteúdo, resultados desta tese, são suportados por um conjunto de softwares capazes de apoiar o desenvolvimento de aplicações voltadas a diferentes domínios de captura e acesso.

As abstrações desenvolvidas possibilitaram a construção de diversas aplicações, algumas das quais vêm sendo utilizadas de maneira contínua há mais de três anos. A transformação de protótipos em aplicações robustas consiste em uma importante forma de validação do trabalho desenvolvido.

Abstract

Ubiquitous computing has been establishing itself as a new human-computer interaction paradigm; its essence is in supporting human experiences in a non-intrusive way, allowing the transparent integration of technology in the everyday life (Weiser 1991).

A frequently exploited theme in ubiquitous computing is the construction of capture and access applications (Abowd and Mynatt 2000; Abowd et al. 2002). Managing a set of ubiquitous devices that includes electronic whiteboards, video cameras and microphones, those applications are able to record a varying of human experiences, including domains such as meetings and classes, aiming at providing later access to the captured content.

The work reported in this thesis has the general goal of providing service and infrastructural facilities in order to support the development of those applications. Previous work identified the need to provide communication and functionality abstractions to aid the development of capture and access applications. This thesis aims at addressing specific problems associated to the representation and management of content generated during live capture sessions. To achieve this goal, high level abstractions were defined to facilitate the interchange, storage and presentation of captured information.

The effective support for the structuring and management of captured data requires a consistent approach for modeling the application's contextual information. Towards supporting this modeling, this thesis proposes the *Contextractor*, a methodology that allows the association between entities of the application's conceptual model and the context dimensions commonly used by capture and access applications. Amongst its results, the *Contextractor* generates XML schemas that are used as architectural building blocks of a software infrastructure that provides high level abstractions for the development of applications. These abstractions include: an application development model based on software components; a storage and presentation service that allows the visualization of captured content on the Web; and a language for querying captured content based on the application's contextual information.

The modeling methodology, the application development model and the data management services, results of this thesis, are supported by a set of softwares that allows the development of applications targeted to a wide variety of application domains.

The provided abstractions have been used in the construction of several applications, some of which are being continuously used for more than three years. The transformation of prototypes in robust applications consists in an important validation of the proposed solutions.

Sumário

Lista de Figuras	xiii
Lista de Tabelas	xvii
Lista de Abreviaturas	xix
Trabalhos Publicados pelo Autor	xxi
1 Introdução	1
1.1 Contextualização	1
1.2 Motivação	3
1.3 Problemas	4
1.4 Objetivos	6
1.5 Metodologia	7
1.6 Resultados	9
1.7 Estrutura da tese	12
2 Computação Ubíqua	15
2.1 Considerações Iniciais	15
2.2 Surgimento e Evolução da Computação Ubíqua	16
2.2.1 <i>Calm technology</i>	17
2.3 Interfaces Naturais	18
2.4 Computação Ciente de Contexto	19
2.4.1 Context Toolkit	21
2.4.2 Context Fabric	21
2.4.3 Gaia OS	22
2.4.4 Aura	23
2.5 Captura e Acesso de Experiências ao Vivo	25
2.5.1 eClass	26
2.5.2 StuPad (Student NotePad)	28
2.5.3 SAAMPad (Software Architecture Analysis Method Pad)	28
2.5.4 DUMMBO (Dynamic, Ubiquitous, Mobile Meeting BOard)	30
2.5.5 LiteMinutes	31

2.5.6	Interactive Workspaces	31
2.5.7	Roomware	33
2.5.8	EasyLiving	34
2.6	Terminologia e Conceitos Relacionados	35
2.6.1	<i>Pervasive Computing</i>	35
2.6.2	<i>Invisible Computing</i>	36
2.6.3	<i>Wearable Computing</i>	36
2.7	Computação no Cotidiano	37
2.7.1	Digital Family Portrait	38
2.7.2	<i>Cook's Collage</i>	39
2.8	Considerações Finais	40
3	O Projeto InCA-SERVE	43
3.1	Considerações Iniciais	43
3.2	Do <i>eClass</i> ao InCA-SERVE	44
3.3	InCA	46
3.4	xInCA: Estendendo a Infra-estrutura InCA	47
3.4.1	Componentes xInCA	48
3.4.2	Funcionamento dos Componentes	50
3.5	SERVE	52
3.5.1	StRES	53
3.5.2	Autore	54
3.5.3	LinkDigger	55
3.6	Considerações Finais	56
4	Modelagem de Informação de Contexto para Aplicações de Captura e Acesso	57
4.1	Considerações Iniciais	57
4.2	Informação de Contexto e Sistemas Legados	59
4.3	Mapeamento das Informações de Contexto	61
4.3.1	Geração do Documento XMI	63
4.4	Visão Geral do Processo de Transformação	63
4.4.1	Geração do Modelo Conceitual	64
4.4.2	Geração do Modelo Conceitual Modificado	66
4.4.3	Geração dos Esquemas XML	69
4.5	Processamento de Consultas CQL	75
4.5.1	Organização do Repositório de Documentos XML	75
4.5.2	O processador CQL	77
4.6	Trabalhos Relacionados	79
4.6.1	wView	79
4.6.2	Modelagem de Contexto na Literatura	83

4.7	Dificuldades Encontradas e Limitações da Metodologia	84
4.8	Considerações Finais	86
5	StRES: Automatizando o Armazenamento e a Recuperação de In-	
	formação	89
5.1	Considerações Iniciais	89
5.2	A Abordagem xInCA-StRES: Visão Arquitetural	91
5.3	Processamento de Documentos	94
5.4	Suporte ao Acesso e à Extensão de Conteúdo Capturado	100
5.4.1	Repositório de Documentos	100
5.4.2	Recuperação de Conteúdo Capturado	102
5.4.3	Apresentação de Conteúdo Capturado	103
5.4.4	Extensão de Conteúdo Capturado	110
5.5	Considerações Finais	111
6	Construção de Aplicações	115
6.1	Considerações Iniciais	115
6.2	Instalação e configuração da infra-estrutura InCA-SERVE	116
6.3	<i>Remote Meeting</i> : Um Estudo de Caso	117
6.3.1	Modelagem da Informação de Contexto	118
6.3.2	Implementação do <i>Front-end</i> da Aplicação	119
6.4	CAT	126
6.5	<i>iClass</i>	128
6.5.1	Pré-produção	128
6.5.2	Gravação ao Vivo	130
6.5.3	Pós-produção	131
6.5.4	Acesso	132
6.5.5	<i>iClass</i> vs. <i>eClass</i> : Uma Análise Comparativa	132
6.6	Caderno On-Line	134
6.7	<i>wiClass</i>	136
6.7.1	Chemistry Lab e-Workbook	137
6.7.2	Mobile Pinhal Digital	138
6.8	Trabalhos Relacionados	139
6.9	Considerações Finais	142
7	Conclusão	145
7.1	Resultados	146
7.2	Trabalhos Futuros	147
7.2.1	Suporte à Construção de Aplicações	147
7.2.2	Suporte ao Gerenciamento de Conteúdo	150
7.3	Considerações Finais	154

Referências	155
Glossário	165
A Documentos produzidos pelo Contextractor	169
B Instalação e configuração da infra-estrutura InCA-SERVE	177

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>	18
2.2 Aplicações cientes de contexto: (a) Leitor de informações de contexto; (b) Uso de informações contextuais no sistema DUMMBO. .	22
2.3 O sistema <i>eClass</i> . (a) Sala de aula instrumentada. (b) Hiperdocumento gerado automaticamente pelo <i>eClass</i>	27
2.4 Stupad auxiliando atividades didáticas	28
2.5 SAAMPad - (a) projetistas participam de uma sessão de projeto de software; (b) tela mostra a interface de acesso de uma sessão SAAM	29
2.6 DUMMBO - (a) interface de captura; (b) interface de acesso	30
2.7 Sistema LiteMinutes. (a) Sala instrumentada. (b) Applet Java para anotações	31
2.8 Exemplo de sala de reuniões equipada com o iRoom.	32
2.9 Equipamentos desenvolvidos no projeto Roomware: <i>ConnecTables</i> , <i>DynaWall</i> , <i>CommChair</i> e <i>InteracTable</i>	34
2.10 EasyLiving: rastreamento de pessoas e objetos através de visão computacional.	35
2.11 Aware Home: (a) Fachada externa; (b) Equipamentos computacionais dão suporte aos dispositivos ubíquos; (c) Câmeras de vídeo e microfones instalados no teto de cada ambiente; (d) Imagem obtida a partir de uma das câmeras.	38
2.12 <i>Digital Family Portrait</i> : (a) Porta retrato com bordas digitais; (b) Interface com informações detalhadas.	39
2.13 Cook's Collage: (a) Quadros representativos das principais atividades realizadas; (b) Escolha dos quadros representativos.	40

3.1	Possíveis configurações da InCA: (a) Todos os componentes residindo na mesma máquina. (b) Componentes distribuídos em várias máquinas.	47
3.2	Componente de <i>whiteboard</i> em uso.	49
3.3	Fluxo de comunicação entre componentes xInCA em duas sessões distintas	51
3.4	Formulário do Autore para reuso de conteúdo.	55
4.1	Diagrama de classes para uma aplicação típica de captura de aulas presenciais.	61
4.2	Extensão feita na interface gráfica da aplicação ArgoUML.	62
4.3	Seqüência de transformações XSLT.	63
4.4	Geração do modelo conceitual modificado.	66
4.5	Geração dos esquemas XML.	69
4.6	Visão geral da abordagem wView.	80
4.7	Geração do modelo conceitual no wView.	81
5.1	Arquitetura xInCA-StRES	94
5.2	Compilação do esquema de sessão pelo JAXB	99
5.3	Documentos de apresentação nos formatos XHTML+SMIL, HTML e SMIL.	104
5.4	Exemplo de <i>pipeline</i> do Cocoon.	106
5.5	Aplicação <i>Web</i> para navegação no repositório de documentos. (A)Visualização dos semestres com aulas capturadas. (B)Visualização dos cursos capturados no primeiro semestre de 2003. (C)Visualização das aulas capturadas no curso SCE-519	108
6.1	Diagrama de classes para uma aplicação de reuniões informais distribuídas.	119
6.2	<i>Remote meeting</i> em ação: componentes executando em janelas independentes.	122
6.3	<i>Remote meeting</i> : formulário de configuração	124
6.4	<i>Remote meeting</i> : visão arquitetural	125
6.5	Interface para configuração de aplicações.	127
6.6	A arquitetura do sistema <i>iClass</i>	129
6.7	<i>iClass</i> : formulário para criação de aulas.	130
6.8	Documento XML produzido pelo sistema <i>iClass</i>	131
6.9	Aplicação Web para apresentação de aulas capturadas pelo <i>iClass</i>	133
6.10	Uso do Caderno On-Line em laboratório de informática.	134
6.11	<i>wiClass</i> suportando aulas de laboratório de química.	137
6.12	<i>wiClass</i> suportando atividades de campo.	139

B.1 Atualização do arquivo incaserve.jar. 179

Lista de Tabelas

2.1	Relação entre o número de pessoas e de computadores para cada uma das eras da computação segundo Weiser & Brown	17
3.1	Funcionalidades presentes em aplicações de captura e acesso . . .	48
6.1	<i>eClass</i> vs. <i>iClass</i> : comparação do código fonte	133
B.1	Organização dos pacotes de <i>software</i> da InCA, da xInCA e do StRES	177
B.2	Softwares de apoio utilizados pela infra-estrutura InCA-SERVE .	181

Lista de Abreviaturas

API — Application Programming Interface

BLOB — Binary Large Object

CAT — Capture and Access Toolkit

CGI — Common Gateway Interface

CM — Conceptual Model

CM-MOD — Modified Conceptual Model

CML — Context Modeling Language

CoWeb — Collaborative Web Site

CQL — Context Query Language

CSL — Context Specification Language

DOM — Document Object Model

DTD — Document Type Declarations

FLWOR — For, Let, Where, Order by, Return (ver glossário)

GUI — Graphical User Interface

HTML — Hypertext Markup Language

IHC — Interação Usuário-Computador

InCa — Infrastructure for Capture and Access

JAXB — Java Architecture for XML Binding

JSP — Java Server Pages

MPEG — Motion Picture Experts Group

NM — Navigation Model

ORM — Object-Role Modeling

PC — Personal Computer

PDA — Personal Digital Assistant

PM — Presentation Model

RTP — Real-time Transfer Protocol

SAX — Simple API for XML Processing

SERVE — infrastructure for Store, Extend, Retrieve and Visualize Evolutionary information

SMIL — Synchronized Multimedia Integration Language

SOAP — Simple Object Access Protocol

SQL — Structured Query Language

StRES — Storing, Retrieving and Extending Service

UML — Unified Modeling Language

URL — Universal Resource Locator

SAX — Simple API for XML processing

XHTML — Extensible Hypertext Markup Language

xlnCa — extended InCA

XMI — XML Metadata Interchange

XML — Extensible Markup Language

XPath — XML Path Language

XQUERY — XML Query Language

XSD — XML Schema

XSL — Extensible Stylesheet Language

XSLT — Extensible Stylesheet Language Transformation

XSP — XML Server Pages

Trabalhos Publicados pelo Autor¹

Baldochi, L. A., R. G. Cattelan, M. G. C. Pimentel, e K. N. Truong. Automatic generation of capture and access applications. In Proceedings of the 8th Brazilian Symposium on Multimedia and Hypermedia Systems, pp. 100-115, outubro de 2002.

Cattelan, R. G., **L. A. Baldochi**, e M. G. C. Pimentel. Processing and storage middleware support for capture and access applications. In Companion Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference, pp. 315, junho de 2003.

Baldochi, L. A., R. G. Cattelan, e M. G. Pimentel (2003). Building a middleware infrastructure for capture and access applications. In Proceedings of the XXX SEMISH - Integrated Seminar on Software and Hardware, pp. 299-313, agosto de 2003.

Cattelan, R. G., **L. A. Baldochi**, M. G. C. Pimentel, e K. N. Truong. Experiences on building capture and access applications. In Proceedings of the 9th Brazilian Symposium on Multimedia and Hypermedia Systems, pp. 112-127, outubro de 2003.

Baldochi, L. A., A. R. Andrade, R. G. Cattelan, e M. G. C. Pimentel. Architecture and components for capture and access applications. In WebMedia & LAWeb 2004 Joint Conference, pp. 150-157, IEEE CS Press, outubro de 2004.

Macedo, A. A., **L. A. Baldochi**, J. A. Camacho-Guerrero, R. G. Cattelan, e M. G. C. Pimentel. Automatically linking live experiences captured with a ubiquitous infrastructure. Multimedia Tools and Applications (MTAP) Journal (to appear), 39p.

Pimentel, M. G. C., **L. A. Baldochi**, e R. G. Cattelan. Prototyping applications to document the human experience. Submitted to the IEEE Pervasive Computing Magazine. Issue on Rapid Prototyping for Ubiquitous Computing, 8p.

Baldochi, L. A., R. G. Cattelan, e M. G. C. Pimentel. Xslt-based gathering of contextual information from UML models. Submitted to the ACM Symposium on Document Engineering - DocEng 2005, 10p.

¹Inclui artigos sob revisão.
Artigos disponíveis para download em <http://iclass.icmc.usp.br/baldochi/publications>

Introdução

1.1 Contextualização

Uma abordagem largamente utilizada pelos professores de cursos introdutórios à ciência da computação consiste em apresentar o histórico da computação nos primeiros dias de aula. O aluno ingressante descobre, então, que o computador que ele usa em casa como se fosse um eletrodoméstico representou um divisor de águas: antes dele havia um computador grande e caro utilizado por várias pessoas; depois, essa relação passou a ser de um para um, dando início à fase da computação pessoal. Ele ainda aprende que uma segunda revolução aconteceu entre o final da década de 80 e o início da década de 90, quando a Internet se popularizou, a *World Wide Web* surgiu e as aplicações distribuídas, baseadas no paradigma cliente-servidor, tornaram-se comuns.

Essa forma de apresentar a evolução da computação aos seus futuros profissionais está no limiar de uma mudança. Em breve os professores estarão apresentando a seus alunos uma terceira revolução, a qual, no entanto, já está em curso e recebe o nome de computação ubíqua. Mark Weiser, responsável por cunhar esse nome, define essa nova revolução como uma fase em que a relação de um para um entre usuários e computadores passa a ser um para vários: cada usuário passa a interagir com vários dispositivos computacionais, muitos dos quais integrados aos ambientes de trabalho, estudo e até

doméstico (Weiser and Brown 1997). Dessa forma, a essência da computação ubíqua está em auxiliar as atividades humanas de maneira minimamente intrusiva, permitindo a integração transparente de tecnologia no cotidiano das pessoas (Weiser 1991; Weiser 1993).

Um termo ainda mais recente, porém capaz de representar a essência da computação ubíqua em uma única palavra, é *everyware* (Cahill et al. 2004). Mistura das palavras *software*, *hardware* e *everywhere*, o termo indica que a computação ubíqua permeia por todas as partes — ambientes de trabalho, ensino, residências — dispositivos computacionais gerenciados por programas.

Apesar do termo computação ubíqua ter surgido em 1991, aplicações consideradas ubíquas datam de 1985, quando foi desenvolvido o *PhoneSlave*, programa capaz de capturar seqüências de áudio obtidas a partir de perguntas exibidas em uma interface gráfica (Schmandt and Arons 1985).

O *PhoneSlave* é o precursor de um tipo particular de aplicação, cujo objetivo principal consiste em *capturar* detalhes de um evento ou experiência vivida, com o objetivo de *acessá-los* no futuro. Tais aplicações, chamadas de *captura e acesso*, passaram a ser alvo de intensa pesquisa a partir da metade da década de 90.

Motivados sobretudo pela ineficiência do cérebro humano em registrar adequadamente as experiências do cotidiano, seja no trabalho, na escola ou mesmo no ambiente doméstico, grupos de pesquisa passaram a investigar o uso de ambientes instrumentados de computação ubíqua para registrar atividades em grupo tais como aulas (Abowd 1999; Mukhopadhyay and Smith 1999; Mueller and Ottmann 2000; Shi et al. 2003) e reuniões (Moran and van Melle 2000; Chiu et al. 2001; Richter et al. 2001; Johanson et al. 2002).

Explorando o uso de câmeras de vídeo e microfones, bem como dispositivos computacionais como lousas eletrônicas, *tablets* e PDAs, as aplicações de captura e acesso permitem registrar diferentes fluxos de informação, gerados de maneira passiva (áudio e vídeo), ou por meio da interação explícita com os dispositivos de captura. Na fase de acesso, os diversos fluxos de informação registrados permitem diferentes formas de revisão do conteúdo capturado. Dessa forma, a captura automática de uma experiência libera o participante da necessidade de se preocupar com anotações ou qualquer outra forma de preservar o evento, o que lhe permite participar mais ativamente do evento em si. Ainda assim, um número muito maior de detalhes ficam registrados para posterior revisão.

1.2 Motivação

As aplicações de computação ubíqua de maneira geral, e as de captura e acesso de maneira particular, exibem um conjunto de particularidades que as fazem essencialmente diferentes das aplicações computacionais tradicionais (Streitz and Nixon 2005).

Ao longo das últimas décadas, as aplicações tradicionais — por exemplo, sistemas corporativos — têm contado com um número expressivo e sempre crescente de ferramentas que auxiliam as diferentes fases do seu desenvolvimento. Tais ferramentas asseguram produtividade no desenvolvimento dessas aplicações, contribuindo para a produção de aplicações cada vez mais eficientes e menos custosas.

As aplicações de computação ubíqua, por outro lado, contam com um suporte de ferramentas de modelagem e desenvolvimento ainda incipiente (Ligkas et al. 2004; Johanson 2002). Particularmente no caso das aplicações de captura e acesso, esse fato pode ser facilmente constatado na literatura, uma vez que as principais aplicações construídas utilizam abordagens *ad hoc* de desenvolvimento.

Outro fato que demonstra o estágio embrionário no suporte ao desenvolvimento de aplicações é a questão dos protótipos. Entre as iniciativas de desenvolvimento realizadas, um número expressivo é constituída por protótipos de laboratório, os quais, na maioria das vezes, não possuem robustez suficiente para se transformar em produto (Davies and Gellersen 2002).

Soluções proprietárias, construídas em função da habilidade individual de alguns poucos desenvolvedores, obviamente não colaboram para o desenvolvimento da área de uma maneira ampla, uma vez que não provêem uma metodologia clara para o reuso das soluções implementadas. A ausência de reuso, faz com que a solução de um mesmo problema seja implementada várias vezes.

No domínio das aplicações de captura e acesso essa ausência de reuso é particularmente grave em função das similaridades apresentadas pelos sistemas desenvolvidos. De maneira geral, tais sistemas fazem uso extensivo de comunicação via rede, produzem e manipulam dados convencionais, metadados e dados multimídia, e utilizam serviços de armazenamento. Além disso, apresentam também recorrência de funcionalidades: quase a totalidade das aplicações voltadas para os domínios de ensino e reunião fazem uso de *whiteboards*, por exemplo.

Todo esse conjunto de similaridades sugere um suporte infra-estrutural para o desenvolvimento de aplicações de captura e acesso, de forma a prover aos desenvolvedores um modelo de desenvolvimento capaz de acelerar a prototipação de aplicações.

1.3 Problemas

O desenvolvimento de aplicações de captura e acesso requer suporte de diferentes áreas da computação. Engenharia de software, sistemas distribuídos, sistemas hipermídia e bancos de dados são algumas das áreas que colaboram para compor o cenário multidisciplinar no qual se insere esse tipo de aplicação. Dessa forma, a construção de uma infra-estrutura de suporte ao desenvolvimento dessas aplicações apresenta um número significativo de desafios de pesquisa, os quais foram investigados por diferentes pesquisadores no contexto do Projeto InCA-SERVE¹.

O primeiro problema investigado no contexto desse projeto refere-se à distribuição transparente da informação manipulada pelos diversos *peers* que compõem uma aplicação de captura e acesso. A solução para esse problema foi alcançada com a disponibilização de uma entidade de registro que oferece suporte ao mecanismo *publish/subscribe*, juntamente com módulos de *software* capazes de abstrair as tarefas básicas de captura e de acesso da informação produzida (Truong and Abowd 2004).

O segundo problema tratado refere-se à recorrência de funcionalidades encontrada nas aplicações. Captura de áudio, vídeo, texto, *Web logging* (registro de páginas *Web* visitadas) e interações com dispositivos ubíquos, tais como lousas eletrônicas e PDAs, ocorrem de maneira recorrente na maioria das aplicações de captura e acesso. A solução encontrada para tratar esse problema consistiu no desenvolvimento de componentes de *software* que implementam essas funcionalidades. Utilizando esses componentes, o desenvolvedor é capaz de construir uma aplicação de captura e acesso simplesmente agrupando os componentes desejados (Cattelan, Baldochi, and Pimentel 2003; Cattelan, Baldochi, Pimentel, and Truong 2003).

As duas abstrações providas são capazes de acelerar substancialmente o processo de desenvolvimento de aplicações. Porém, deixam a cargo do desenvolvedor questões relacionadas ao gerenciamento da informação produzida.

¹O InCA-SERVE foi um projeto da área de computação ubíqua realizado por meio de uma parceria formal entre o *Georgia Institute of Technology* e o ICMC-USP (<http://coweb.icmc.usp.br/incaserve>).

Entretanto, essas questões também apresentam similaridades nas diferentes aplicações, podendo se beneficiar tanto de abstrações infra-estruturais, quanto de serviços de suporte ao armazenamento e à apresentação de conteúdo capturado.

Franklin (2001) destaca o suporte à mobilidade, a ciência de contexto e o suporte à colaboração como os três principais tópicos da computação ubíqua nos quais existem desafios de pesquisa importantes relacionados ao gerenciamento de dados. Entre esses tópicos, o suporte à colaboração é o que está mais diretamente associado às aplicações de captura e acesso, uma vez que essas aplicações são inerentemente colaborativas. Não menos importante é a exploração de informações de contexto: a identificação dos diferentes usuários, seus papéis e a caracterização das experiências no tempo e no espaço são fundamentais para facilitar o posterior acesso às informações capturadas. Já o suporte à mobilidade, apesar de ter uma importância inquestionável para diversas aplicações, possui uma importância secundária no contexto das aplicações de captura e acesso, uma vez que essas aplicações, na maioria dos casos, restringem-se a ambientes instrumentados que limitam a mobilidade do usuário.

Dessa forma, o suporte às questões associadas ao gerenciamento dos dados produzidos e manipulados por aplicações de captura e acesso envolvem o tratamento adequado das informações de contexto associadas à cada aplicação, bem como facilidades para a representação e intercâmbio de dados de maneira a suportar adequadamente o desenvolvimento de aplicações colaborativas.

É importante frisar que as informações de contexto nas aplicações de captura e acesso são utilizadas tanto no suporte à captura das experiências, como também no posterior acesso ao conteúdo capturado. Nesse segundo momento, as informações de contexto são essenciais a fim de permitir flexibilidade na recuperação desse conteúdo.

No sentido de atender esse conjunto de requisitos associados à representação e ao gerenciamento da informação produzida, é necessário prover tanto abstrações de ordem infra-estrutural, a fim de oferecer suporte a atividades colaborativas, quanto serviços de alto nível que facilitem os procedimentos de armazenamento, recuperação e apresentação de conteúdo capturado, com base em informações de contexto providas pelo usuário.

Fica claro, portanto, que o *problema* associado ao gerenciamento de informação é o mesmo que vem sendo tratado desde o início do Projeto InCA-SERVE e refere-se à elevação do grau de abstração necessário para a constru-

ção de aplicações de captura e acesso. Por se tratar de um problema amplo e multidisciplinar, as soluções resolvem subproblemas específicos, colaborando, em conjunto, para a solução do problema como um todo.

1.4 Objetivos

Suportar a captura de atividades colaborativas, estruturar a informação capturada de maneira a facilitar seu intercâmbio, permitir a recuperação de conteúdo baseada em informações de contexto e prover mecanismos capazes de apresentar o conteúdo capturado na *Web*, com diferentes opções de visualização, são os objetivos gerais dessa tese.

Para alcançá-los, é necessário oferecer um modelo integrado de desenvolvimento que contemple as necessidades das aplicações de captura e acesso em todas as suas fases. Isso significa suportar a modelagem das aplicações, utilizar transparentemente a informação modelada na captura de experiências ao vivo e contar com serviços capazes de explorar a semântica dessa informação para armazenar e recuperar conteúdo capturado.

No sentido de disponibilizar um modelo de desenvolvimento baseado no suporte infra-estrutural e de serviços, é necessário alcançar os seguintes objetivos específicos:

1. implementar um mecanismo capaz de permitir a modelagem das informações de contexto associadas a cada aplicação de captura e acesso;
2. definir um formato para representação das informações que seja ao mesmo tempo adequado para o intercâmbio de dados durante a fase de captura e para o armazenamento do conteúdo capturado para posterior acesso. O formato definido deve ser capaz de encapsular as informações de contexto obtidas no item 1, de modo a permitir a recuperação de conteúdo com base nessas informações;
3. prover facilidades infra-estruturais na forma de operações capazes de manipular transparentemente as informações estruturadas no item 2, inclusive no que diz respeito ao armazenamento da informação produzida;
4. prover um mecanismo capaz de apresentar na *Web* conteúdo capturado sem nenhum tipo de intervenção humana;
5. definir uma linguagem de consulta baseada nas informações de contexto modeladas no item 1, e disponibilizar um processador capaz de processar as consultas elaboradas na linguagem modelada; e

6. disponibilizar uma interface de programação que permita a construção de aplicações de acesso. As operações de recuperação suportadas devem incluir as consultas discutidas no item 5.

1.5 Metodologia

A análise das primeiras aplicações implementadas no contexto do Projeto InCA-SERVE demonstravam que as abstrações de comunicação e funcionalidades providas permitiam a rápida prototipação de aplicações. Entretanto, havia dificuldade na fase de acesso, uma vez que a informação gerada não apresentava uma estruturação comum capaz de permitir a criação de aplicações de acesso reutilizáveis.

Percebia-se que a abordagem *ad hoc* utilizada na modelagem das informações forçava soluções *ad hoc* para o acesso a essas informações. Dessa forma, a primeira ação no sentido de alcançar os objetivos enumerados na Seção 1.4 foi estabelecer uma representação homogênea para os dados capturados, de maneira a facilitar o posterior acesso à essa informação.

Esse procedimento envolveu a definição de esquemas XML para os componentes de software disponibilizados e a utilização de um mecanismo de *data binding* para manipular os dados modelados nos esquemas de maneira programática. As classes geradas a partir da compilação dos esquemas pelo mecanismo de *data binding* passaram, então, a integrar a infra-estrutura implementada. Dessa forma, o intercâmbio de dados entre as diversas partes de cada aplicação suportada passa a ser feito por meio dos objetos resultantes das classes geradas pelo processo de *binding*.

Além dos esquemas para os componentes de captura e acesso, dois outros foram definidos: o primeiro com a função de modelar as informações de contexto da aplicação e o segundo, denominado esquema de sessão, com a função de agrupar todos os demais esquemas. As classes geradas a partir desses esquemas, também pelo mecanismo de *binding*, são utilizadas na construção de um serviço denominado StRES — *Storing, Retrieving and Extending Service* — o qual automatiza o armazenamento das informações capturadas pelas aplicações. O StRES utiliza as abstrações de comunicação implementadas para receber em tempo real os objetos produzidos pelos componentes de captura, os quais são integrados a uma estrutura de dados representativa da sessão em curso. Essa estrutura de dados é utilizada, ao final da sessão, para produzir um documento XML representativo da experiência capturada, o qual fica armazenado em um repositório de documentos gerenciado pelo StRES.

Como resultado dessa abordagem, o conteúdo produzido por qualquer aplicação passa a ser sempre um documento XML, estruturado de acordo com os componentes de captura utilizados. Isso permitiu criar mecanismos de apresentação baseados na transformação de documentos. A simples implementação de uma folha de estilo capaz de transformar um documento de sessão XML em outro HTML passa a ser suficiente para apresentar o conteúdo capturado na *Web*.

Dessa forma, a implementação da parte infra-estrutural do StRES contempla os objetivos 2, 3 e 4 listados na Seção 1.4. A solução inicialmente implementada, entretanto, impunha uma estrutura fixa para o esquema XML responsável por modelar as informações de contexto. A medida que novas aplicações foram sendo desenvolvidas com o suporte infra-estrutural provido, percebeu-se que a utilização de um esquema fixo não contemplava adequadamente a necessidade de uma parcela significativa dos domínios que se desejava suportar. A solução encontrada foi permitir que o desenvolvedor definisse explicitamente o esquema com as informações de contexto.

Entretanto, essa abordagem traz uma sobrecarga considerável ao desenvolvedor da aplicação, uma vez que o obriga (i) a conhecer a especificação *XML Schema*, (ii) conhecer o mecanismo de *data binding* utilizado, (iii) realizar a compilação do esquema e atualizar a infra-estrutura com as classes produzidas. Além da sobrecarga ao desenvolvedor, esse procedimento traz também a desvantagem de obrigar o desenvolvedor a alterar a própria infra-estrutura.

Uma forma de contornar todos esses problemas consistiu em permitir que o usuário defina o esquema de contexto de maneira implícita. Para tanto, desenvolveu-se uma metodologia capaz de permitir a geração do esquema de contexto de maneira semi-automática. Essa metodologia, denominada *Contextractor*, consiste em permitir a extração das informações de contexto a partir da modelagem UML — *Unified Modeling Language* — estendida das aplicações.

O *Contextractor* é formado por um ferramenta de modelagem UML estendida com funcionalidades que permitem a associação de dimensões de contexto a entidades do diagrama de classes das aplicações. Essa ferramenta é capaz de exportar a modelagem como um documento XMI — *XML Metadata Interchange* — o qual é utilizado como entrada de um *pipeline* que realiza transformações sucessivas, com interação mínima por parte do usuário, as quais resultam no esquema de contexto da aplicação. O *Contextractor* provê ainda uma ferramenta responsável por compilar o esquema gerado e integrar as classes produzidas ao pacote de distribuição da infra-estrutura.

Além do esquema de contexto, o *Contextractor* produz ainda dois outros

esquemas XML responsáveis por definir uma linguagem de consulta baseada nas informações de contexto modeladas, denominada CQL — *Context Query Language*. Esses dois esquemas são utilizados por um programa capaz de processar expressões CQL, retornando referências aos documentos que atendem às restrições impostas em cada consulta.

Dessa forma, as facilidades providas pelo *Contextractor* contemplam as metas 1 e 5 do trabalho.

O StRES, apesar do nome, é parte infra-estrutura e parte serviço. A parte infra-estrutural, já discutida, cuida da geração automática de documentos que representam sessões capturadas. A parte serviço explora as facilidades de um banco de dados XML nativo a fim de suportar o gerenciamento de um repositório de documentos XML referentes às sessões capturadas. O serviço de armazenamento provê operações habituais de armazenagem e recuperação de documentos, bem como consultas, as quais podem ser formuladas na linguagem XQuery, suportada pelo banco, ou utilizando a linguagem CQL. O serviço StRES contempla a sexta e última meta listada na seção de objetivos.

1.6 Resultados

Os objetivos discutidos na Seção 1.4 convergem para um mesmo propósito: a elevação do grau de abstração no desenvolvimento de aplicações de captura e acesso.

Cada ação no sentido de atingir as metas propostas encerra esse propósito. Entretanto, mensurar a efetividade dessas ações no sentido de facilitar a atividade de desenvolvimento é uma tarefa complexa.

O primeiro esforço de validação realizado consistiu na implementação do CAT — *Capture and Access Toolkit* (Baldochi et al. 2002). Baseado em uma interface *Web* na qual o usuário pode configurar uma aplicação de captura e acesso por meio da seleção dos componentes de captura desejados e da especificação de permissões de acesso às sessões, o CAT teve o objetivo de demonstrar a simplicidade do modelo de desenvolvimento de aplicações desenvolvido, o qual permite a implementação de uma aplicação funcional apenas instanciando os componentes desejados e implementando suas interfaces associadas.

Aplicações simples, tais como um *chat*, podem ser implementadas utilizando esse modelo de desenvolvimento em cerca de 15 linhas de código Java. Evidentemente, essas 15 linhas referem-se ao *front-end* do *chat*, uma vez que

o servidor responsável por fazer a distribuição das mensagens encontra-se embutido no suporte infra-estrutural.

Apesar de mostrar a efetividade do modelo de desenvolvimento implementado, o CAT era mais uma prova de conceito que uma aplicação de captura e acesso real. Havia a necessidade de verificar se a solução de gerenciamento da informação produzida e sua respectiva apresentação na *Web* atendia os requisitos de aplicações reais.

Um resultado importante do trabalho consiste na automatização do processo que envolve a geração de documentos XML, seu armazenamento em um repositório de documentos e sua apresentação na *Web* em diferentes formatos. Verificar a efetividade dessa solução envolve produzir uma quantidade expressiva de sessões capturadas a fim de popular o repositório com seus respectivos documentos XML, os quais devem ser acessados ao longo do tempo.

As atividades educacionais, sobretudo aquelas que acontecem no ambiente de sala de aula, apresentam as características adequadas para a construção de aplicações de captura e acesso envolvendo a produção de um grande número de sessões acessadas por um grande número de pessoas. Esse domínio, aliás, vem sendo explorado por aplicações de computação ubíqua desde o início da década de noventa (Abowd and Mynatt 2000).

A fim de explorar as experiências em sala de aula, foi desenvolvido o *iClass*, aplicação de captura de aulas presenciais que se beneficia do suporte infra-estrutural e de serviços desenvolvido no contexto do Projeto InCA-SERVE (Baldochi et al. 2003). Ao longo de três anos, já foram capturadas mais de 300 aulas de diferentes cursos de graduação e pós-graduação ministradas no ICMC-USP, e esse número cresce a cada dia.

Graças ao suporte oferecido pelo StRES, os alunos são capazes de visualizar o conteúdo de uma aula capturada na *Web* segundos após seu término. O suporte oferecido ao pós-processamento de conteúdo capturado cuida de todo o processo que envolve encapsular o conteúdo capturado em um documento XML, armazenar esse documento e prover sua visualização sob demanda por meio de uma aplicação *Web*. O mais importante, entretanto, é que essa facilidade não foi feita sob medida para o *iClass* — funciona para qualquer aplicação de captura e acesso produzida com o suporte infra-estrutural disponibilizado, conforme discutido em Baldochi et al. (2004).

O sucesso do *iClass* inspirou o desenvolvimento de aplicações que estendem suas funcionalidades básicas, como é o caso do Caderno On-Line, extensão desenvolvida nas Faculdades COC de Ribeirão Preto. O Caderno On-Line

explora a interação dos alunos no material gerado pelos professores, possibilitando que ao final de cada aula o aluno tenha sua própria versão anotada do conteúdo ministrado (Silva, Lobato, and Baldochi 2004).

Outra extensão relevante refere-se ao *wiClass*, versão do *iClass* capaz de ser executada em dispositivos móveis, tais como *tablet PCs*. Nessa aplicação as facilidades de captura já sedimentadas do *iClass* são aliadas à mobilidade, o que permite a exploração de novas atividades, em ambientes pouco convencionais, incluindo laboratórios experimentais de química, museus e zoológicos (Pimentel, Baldochi, and Cattelan 2005).

A visualização de conteúdo capturado obedece a estruturação do repositório de documentos produzidos, a qual, por sua vez, é determinada na fase de modelagem da aplicação. Dessa forma, as vantagens trazidas pela automatização do processo de apresentação de conteúdo tem o preço da falta de flexibilidade no acesso: apenas uma forma de recuperação é provida, na qual é necessário navegar pelas coleções de documentos até o conteúdo desejado.

Vale notar, entretanto, que essa navegação possui semântica associada. No caso do *iClass*, por exemplo, o usuário inicia a navegação escolhendo o semestre que deseja acessar. A escolha de um semestre resulta na exibição dos cursos capturados naquele semestre. A escolha de um curso mostra as aulas capturadas daquele curso, com hiperligações para os possíveis formatos de visualização.

No sentido de prover um mecanismo de recuperação de informação mais flexível, desenvolveu-se o *Contextractor*, o qual permite especificar as informações de contexto na fase de modelagem da aplicação, e definir, com base nessas informações, uma linguagem de consulta contextual. Para processar consultas na linguagem especificada, desenvolveu-se um processador que retorna referências aos documentos que atendem às restrições especificadas em cada expressão de consulta (Baldochi et al. 2005).

Com base nesse processador, foi possível construir uma extensão à aplicação Web responsável por apresentar conteúdo capturado. Essa extensão constrói a mesma página que permite selecionar diferentes formatos de visualização para uma dada sessão. Entretanto, as sessões passíveis de serem visualizadas são obtidas a partir de uma expressão de consulta contextual fornecida pelo usuário.

Esse mecanismo permite flexibilizar o acesso ao conteúdo capturado, uma vez que é disponibilizado às aplicações na forma de uma interface XML-RPC. Dessa forma, diferentes aplicações que suportem esse protocolo são capazes

de submeter consultas contextuais ao repositório de documentos (Baldochi et al. 2005).

Vale salientar, por fim, que os repositórios de documentos XML construídos e mantidos pelo StRES constituem uma fonte rica de informação passível de ser explorada por um vasto conjunto de ferramentas capazes de processar informação XML. Um exemplo desse tipo de ferramenta é o *LinkDigger*, ferramenta desenvolvida no contexto do Projeto InCA-SERVE que permite identificar relações envolvendo conteúdo textual entre diferentes documentos. Usado no contexto de aplicações de captura e acesso tais como o *iClass*, o *LinkDigger* permite gerar um conjunto de hiperligações entre documentos relacionados, o que possibilita navegar entre sessões que possuem semântica associada (Macedo, Baldochi, Camacho-Guerrero, Cattelan, and Pimentel 2005).

1.7 Estrutura da tese

Além da introdução, esta tese conta com outros seis capítulos, brevemente resumidos a seguir.

- Capítulo 2: Computação Ubíqua — apresenta uma visão geral do tema, com enfoque nas aplicações de captura e acesso e na ciência de contexto, dois dos três temas centrais dessa área.
- Capítulo 3: Projeto InCA-SERVE — apresenta os objetivos e resultados alcançados pelo projeto, enfatizando a relação do trabalho descrito nesta tese com outros desenvolvidos no contexto do projeto.
- Capítulo 4: Modelagem de Informação de Contexto para Aplicações de Captura e Acesso — apresenta o *Contextextractor*, metodologia desenvolvida para suportar a modelagem de informações de contexto para aplicações de captura e acesso.
- Capítulo 5: StRES: Automatizando o Armazenamento e a Recuperação de Informação — apresenta o StRES, serviço responsável pelo armazenamento, recuperação e extensão de conteúdo capturado que agrupa diversos resultados importantes no contexto desta tese.
- Capítulo 6: Construção de Aplicações — mostra como o conjunto de abstrações desenvolvido no contexto desta tese facilita o procedimento de construção de aplicações de captura e acesso.

- Capítulo 7: Conclusão — traz as considerações finais, lembrando os problemas existentes e mostrando as soluções desenvolvidas. Apresenta ainda considerações referentes a trabalhos futuros.

Computação Ubíqua

2.1 *Considerações Iniciais*

Os inúmeros benefícios trazidos pela difusão dos sistemas computacionais têm seu custo: na maioria das vezes as pessoas são obrigadas a mudar hábitos e métodos de trabalho a fim de se adaptarem à onipresença do computador. Weiser (1991) vislumbrou uma forma menos intrusiva de fazer uso do computador no cotidiano das pessoas. Segundo ele, facilidades computacionais devem ser incorporadas aos ambientes a fim de auxiliar atividades humanas mudando minimamente a forma como tais atividades são realizadas. Essa forma transparente de integrar tecnologia às atividades diárias das pessoas foi denominada computação ubíqua.

Em sua visão seminal de 1991, Weiser previa para um futuro próximo a proliferação de dispositivos computacionais ubíquos de diversos tamanhos: pequenos e pessoais (*inch-scale*), de médio porte (*foot-scale*) e grandes e de uso coletivo (*yard-scale*). De fato, isso acabou acontecendo: dispositivos pessoais de pequeno e médio porte tais como PDA's, *tablets* digitais e *laptops* tornaram-se comuns no final da década de 90. Da mesma forma, dispositivos maiores, tais como lousas eletrônicas, passaram a fazer parte de ambientes de uso comum, tais como salas de reuniões, salas de aula e laboratórios.

Um outro aspecto vislumbrado por Weiser também se confirmou: novas aplicações computacionais surgiriam para explorar o uso desses novos dis-

positivos. O desenvolvimento de tais aplicações está diretamente associado a três temas, que são hoje os principais focos de pesquisa na área de computação ubíqua. São eles: interfaces naturais, computação ciente de contexto e captura e acesso de atividades humanas (Abowd and Mynatt 2000).

À medida que ambientes ubíquos tornam-se uma realidade e a computação não intrusiva passa a fazer parte da vida das pessoas, uma nova questão torna-se relevante: a dimensão temporal das interações humanas. Essa questão fez surgir um novo tema que se associa aos três anteriormente citados: a computação no cotidiano (*everyday computing*), segundo a qual se faz necessário prover disponibilidade de serviço vinte e quatro horas por dia, sete dias por semana, de forma a auxiliar computacionalmente atividades informais e não estruturadas comuns no dia a dia das pessoas.

Este capítulo tem como objetivo dar uma visão geral sobre computação ubíqua e está organizado da seguinte forma: a Seção 2.2 discute o surgimento e a evolução dessa nova área da computação, apresentando alguns dos dispositivos computacionais usados em suas aplicações. As Seções 2.3, 2.4 e 2.5 apresentam os principais temas de pesquisa relacionados à computação ubíqua, a saber: interfaces naturais, computação ciente de contexto e aplicações de captura e acesso. Os dois últimos temas são enfatizados pois possuem relação direta com os objetivos de investigação dessa tese. A Seção 2.6 apresenta a terminologia e os principais conceitos relacionados ao tema em estudo. A Seção 2.7 aborda a computação do cotidiano. Finalmente, a Seção 4.8 apresenta as considerações finais do capítulo.

2.2 Surgimento e Evolução da Computação Ubíqua

Weiser e Brown (1997) dividem os últimos cinquenta anos da computação nas eras do *mainframe* e do computador pessoal. Para eles, a Internet e a difusão da computação distribuída estão conduzindo para o paradigma da computação ubíqua. A era do *mainframe* é caracterizada por muitas pessoas compartilhando um único e raro recurso computacional. Na era da computação pessoal, essa relação muda para um computador para cada pessoa. A Internet e a difusão da computação distribuída representam uma fase de transição, agregando componentes das eras do *mainframe* e da computação pessoal através do uso massivo do modelo cliente-servidor, em que os computadores pessoais são os clientes *Web* e os *mainframes*, os servidores *Web*. Essa transição conduz a uma terceira era: a da computação ubíqua, com vários computadores disponíveis para cada pessoa. Esses computadores es-

tariam embutidos em praticamente tudo (nas paredes, na mobília, em nossas roupas, etc.) e, principalmente, estariam interligados à Internet, formando, assim, uma gigantesca rede e possibilitando as mais diversas formas de utilização. A Tabela 2.1 ilustra as eras da computação segundo essa visão.

Era	Pessoa(s)	Computador(es)
Mainframe	n	1
Computação Pessoal	1	1
Internet/Comp. Distribuída	transição	transição
Computação Ubíqua	1	n

Tabela 2.1: Relação entre o número de pessoas e de computadores para cada uma das eras da computação (Weiser and Brown 1997).

2.2.1 Calm technology

A computação ubíqua requer uma nova abordagem que integre essas novas tecnologias ao cotidiano das pessoas. Weiser e Brown (1997) chamam essa abordagem de *calm technology*, dividindo a percepção humana em periferia e centro. O termo periferia é usado para fazer alusão aquilo de que as pessoas estão cientes mas sem explicitamente estarem prestando atenção. O centro diz respeito aos objetos que concentram o foco de atenção das pessoas. Por exemplo, ao dirigir, a atenção do motorista está concentrada na estrada, no rádio, na conversa com o passageiro, mas não necessariamente no barulho do motor. No entanto, um barulho anormal produzido pelo motor é imediatamente percebido, mostrando que o motorista estava sintonizado com o barulho ao seu redor (sua periferia) e que é capaz de mudar seu foco de atenção para um objeto periférico rapidamente, tornando-o, dessa forma, um objeto central.

Essa possibilidade de movimentação entre centro e periferia e vice-versa é o mais puro conceito de *calm technology*. Ao terem sua periferia enriquecida, as pessoas são capazes de controlar muito mais coisas do que se tivessem que colocá-las todas no centro. Uma grande porção do cérebro humano é dedicada ao processamento sensorial, que é fundamentalmente periférico (Weiser and Brown 1997). Portanto, a periferia das pessoas é capaz de informá-las sem sobrecarregá-las.

O enriquecimento da periferia pode ser feito através da incorporação de dispositivos computacionais ao ambiente a fim de auxiliar as atividades humanas mudando minimamente a forma como são realizadas. As visões de Mark Weiser sugeriam a disseminação, no ambiente físico, de dispositivos computacionais das mais variadas escalas (Weiser 1991):

- *Inch scale*: são os dispositivos cujas medidas são expressas em uma escala de polegadas. Por exemplo, *pads* e *PDA*s (Figura 2.1(a)(b)).
- *Foot scale*: são os dispositivos cujas medidas são expressas em uma escala de pés. Por exemplo, *tablets* e *laptops* (Figura 2.1(c)(d)).
- *Yard scale*: são os dispositivos cujas medidas são expressas em uma escala de jardas. Por exemplo, lousas eletrônicas (Figura 2.1(e)).
- *Wall-sized*: São os dispositivos cujas medidas extrapolam as escalas anteriores. Por exemplo, paredes inteiras cujas superfícies são monitoradas por *mimios* (Figura 2.1(f)).

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

2.3 Interfaces Naturais

A computação ubíqua inspira o desenvolvimento de aplicações não centradas no paradigma teclado/mouse/display. A premissa básica nesse tipo de aplicações é adaptar as interfaces à forma segundo a qual as pessoas interagem com o mundo físico.

Interfaces computacionais que suportam formas mais naturais de comunicação, tais como fala, escrita manual e expressão corporal, estão começando a substituir ou complementar elementos do paradigma de interfaces gráficas convencionais. Essas novas interfaces facilitam o aprendizado e a usabilidade dos sistemas, uma vez que não modificam drasticamente a forma como as atividades são realizadas.

A comunidade científica da área de interação homem-computador (IHC) vem trabalhando há vários anos no desenvolvimento de interfaces cada vez mais amigáveis. Esse trabalho tem resultado em interfaces que suportam paradigmas de interação baseados em voz (Hauptmann 1995), em escrita manual (Schilit et al. 1998), e, mais recentemente, no uso de objetos do

mundo físico com o objetivo de manipular artefatos eletrônicos, criando o conceito de interfaces tangíveis (Ishii and Ullmer 1997). Há ainda pesquisas relacionadas ao uso de sensores em dispositivos computacionais, de forma a permitir que a manipulação física desses dispositivos seja corretamente interpretada pelas aplicações que os controlam (Harrison et al. 1998).

2.4 Computação Ciente de Contexto

A computação ciente de contexto consiste na ligação entre atividades humanas e serviços computacionais no sentido de expandir e facilitar tais atividades. Os serviços computacionais devem ser capazes de inferir noções de espaço e tempo, bem como realizar relacionamentos em um dado domínio desejado de forma a auxiliar atividades humanas.

A primeira informação contextual usada computacionalmente foi a localização espacial. Pesquisadores de dois grandes centros de pesquisa em computação, o Olivetti Research Lab's, com seu Active Badge (Want et al. 1992), e o XEROX PARC, com seu PARCTab (Want et al. 1995), foram os pioneiros no desenvolvimento de aplicações sensíveis ao contexto de localização, possibilitando a implementação de facilidades tais como transferências automáticas de chamadas telefônicas e atualizações automáticas de mapas com as localizações de usuários em um escritório.

Aplicações mais recentes usam as facilidades providas pelo sistema GPS — *Global Positioning System* — para prover navegação de carros em cidades e rodovias. O sistema GPS também tem sido utilizado no desenvolvimento de guias turísticos inteligentes, capazes de escolher um vídeo ou áudio para ser exibido de acordo com informações de localização providas através do GPS (Abowd et al. 1997).

Uma outra forma de explorar informação contextual está nas técnicas de reconhecimento de objetos. Usando tais técnicas é possível reconhecer áreas em vídeos e imagens de forma a entender o contexto de uma cena ou de uma fotografia. Isso possibilita enriquecer apresentações com informações contextuais e realizar busca baseada em conteúdo em fluxos de mídia contínua. O projeto Informedia, por exemplo, utiliza técnicas de reconhecimento de faces e legendas em vídeos de forma a estender vídeos com meta-informação contextual (Wactlar et al. 1996).

Entretanto, há muitas outras informações contextuais além de localização e identificação de pessoas e objetos que não são exploradas. A maioria dos sis-

temas cientes de contexto ainda não incorpora noções de tempo, história, outras pessoas além do próprio usuário, e muitas outras informações disponíveis nos ambientes. Abowd e Mynatt (2000) descrevem como obter informações contextuais com base em cinco dimensões, as quais são chamados de "cinco W's":

- *Who* (quem): Deve-se prover informações contextuais de todas as pessoas envolvidas em uma dada atividade assistida por computador. Como seres humanos, nós nos lembramos de eventos do passado baseado na presença das pessoas que os vivenciaram.
- *What* (o quê): Tem como função identificar o que usuário está fazendo. Alguns sistemas de propósito geral assumem a atividade do usuário em função do próprio projeto do sistema. No entanto, em sistemas onde atividades variadas podem ser desenvolvidas, identificar o que um usuário está fazendo em um determinado momento pode ser uma tarefa complexa. Sistemas como os de captura e acesso, assunto da Seção 2.5, são exemplos de sistemas que lidam com atividades variadas dos usuários nas quais é altamente desejável obter informações contextuais a respeito dessas atividades. Sendo assim, formas de explorar o contexto das atividades dos usuários são objetos de pesquisa em computação ubíqua.
- *Where* (onde): Como já discutido, o contexto de localização é o mais utilizado pelos sistemas sensíveis a contexto. Pesquisas apontam a utilização dessa informação contextual associada a outras como, por exemplo, *quando*, indicando que é interessante mesclar os contextos temporais e espaciais no sentido de prover novas funcionalidades às aplicações.
- *When* (quando): O contexto temporal é usado para indexar um registro capturado ou para informar por quanto tempo um usuário esteve em um determinado local (onde + quando).
- *Why* (porque): Mais complexo que perceber o que o usuário está fazendo é entender o porque de sua ação. Obter informações contextuais que possam caracterizar o estado de espírito de uma pessoa é talvez o maior desafio da computação sensível a contexto. Um ponto de partida pode ser o monitoramento dos sinais vitais do usuário, tais como batimento cardíaco, pressão arterial, etc.

2.4.1 Context Toolkit

No sentido de facilitar a utilização de informações de contexto por aplicações de computação ubíqua, pesquisadores do GATech desenvolveram o Context Toolkit (Dey and Abowd 2000), ferramenta composta por *context widgets* e uma infra-estrutura distribuída que oferece suporte aos *widgets*. *Context widgets* são componentes de software que provêem às aplicações informações de contexto em alto nível, sem que as aplicações tenham que lidar com o gerenciamento de sensores que coletam tais informações. Da mesma forma que GUI *widgets* facilitam o desenvolvimento de aplicações que utilizam interfaces gráficas, *context widgets* facilitam o desenvolvimento de aplicações que utilizam informações de contexto. De forma resumida, os serviços providos pelo Context Toolkit são:

- Encapsulamento de sensores;
- Acesso a dados de contexto através de uma API de rede;
- Compartilhamento de dados de contexto através de uma infraestrutura distribuída;
- Armazenamento de dados de contexto, incluindo históricos;
- Controle de acesso visando proteção de privacidade.

Inferindo Presença e Identidade

Uma aplicação simples que utiliza as facilidades providas pelo Context Toolkit é a que infere a presença e identifica pessoas através de informações coletadas através de sensores. Esta aplicação é baseada nos *iButtons*, dispositivos na forma de botão dotados de um *chip* interno. Cada *iButton* é usado para identificar uma pessoa. Ao conectar seu *iButton* a um leitor, as informações pessoais do usuário são passadas ao sistema, permitindo identificá-lo (Figura 2.2(a)). Na Figura 2.2(b), o leitor encontra-se montado no sistema DUMMBO (descrito adiante, na Seção 2.5.4). Quando dois usuários se conectam ao sistema, a aplicação assume que uma reunião está começando e começa a capturar as interações com a lousa eletrônica.

2.4.2 Context Fabric

Similarmente ao Context Toolkit, o Context Fabric (Hong and Landay 2001) define uma infra-estrutura que simplifica a tarefa de criar e manter aplicações

Figura 2.2: Aplicações cientes de contexto: (a) Leitor de informações de contexto; (b) Uso de informações contextuais no sistema DUMMBO.

cientes de contexto. Essencialmente, ele propõe um modelo distribuído para armazenamento e recuperação de informação de contexto e uma linguagem declarativa para especificação e processamento dessas informações.

O modelo de armazenamento é dividido em modelo *lógico* e *físico* de dados. O *modelo lógico* fornece uma representação para entidades, atributos, relacionamentos e agregações. As *entidades* podem ser, por exemplo, pessoas, lugares ou coisas, e possuem um controle de acesso associado que restringe o acesso aos seus dados internos. Os *atributos* descrevem as propriedades das entidades, enquanto que os *relacionamentos* são tipos especiais de atributos que ligam duas ou mais entidades. As *agregações* correspondem a uma forma de agrupar ou definir conjuntos de entidades. O *modelo físico*, por sua vez, gerencia como os dados realmente são armazenados, distribuindo e replicando informações, de forma que elas fiquem próximas aos lugares onde são geradas e utilizadas.

A linguagem de especificação de contexto definida pelo Context Fabric é denominada CSL (*Context Specification Language*) (Hong 2002) e oferece uma abstração semelhante ao SQL, permitindo executar consultas em alto nível baseadas no padrão XML. As instruções são processadas localmente por um *Local Context Service* associado a cada dispositivo ou de maneira distribuída, consultando outros *Context Services* caso necessário.

2.4.3 Gaia OS

O Gaia OS (Roman et al. 2002; Hess et al. 2002) consiste de uma infraestrutura de *middleware* que suporta o desenvolvimento e a execução de apli-

cações nos chamados *Active Spaces* – espaços físicos equipados com dispositivos computacionais em rede coordenados por um sistema baseado em contexto, que se reconfiguram de acordo com a interação dos usuários. A idéia principal do Gaia é oferecer um ambiente de computação ubíqua no qual os usuários interajam com diversos dispositivos simultaneamente e as aplicações se adaptem às mudanças relacionadas ao ambiente, usuários e dispositivos.

De maneira semelhante a um sistema operacional, o Gaia OS gerencia as tarefas comuns às aplicações construídas para os *Active Spaces*, oferecendo serviços para identificação, localização e utilização dos recursos existentes. Os componentes que disponibilizam tais funcionalidades são implementados sobre uma arquitetura distribuída, baseada em CORBA, e provêm cinco serviços básicos (Roman et al. 2002):

- *Event Manager Service*: responsável pela notificação de eventos nos *Active Spaces*, implementa um modelo de comunicação baseado em produtores, consumidores e canais;
- *Presence Service*: é responsável por detectar a presença física ou digital de uma entidade no *Active Space*, sendo que a implementação atual define quatro tipos de entidades: usuário, dispositivo, serviço e aplicação;
- *Context Service*: permite às aplicações executarem consultas e se registrarem em canais de eventos para serem notificadas sobre alguma informação de contexto em particular;
- *Space Repository Service*: armazena informação sobre os softwares e hardwares existentes no *Active Space*, provendo funcionalidades de busca e recuperação de entidades baseadas em atributos específicos;
- *Context File System*: incorpora contexto ao modelo tradicional de sistema de arquivos, de forma a suportar a mobilidade de usuários, a heterogeneidade de dispositivos e as diferentes organizações dos dados.

2.4.4 Aura

Um dos desafios mais importantes da computação ciente de contexto é o suporte a mobilidade do usuário. Idealmente, uma infra-estrutura ubíqua deveria permitir aos seus usuários (ou aplicações) mover-se facilmente de um ambiente (ou dispositivo) a outro, suportando de maneira robusta a entrada e saída de recursos computacionais no ambiente. O projeto Aura (Garlan et al.

2002), desenvolvido por pesquisadores da Universidade Carnegie Mellon, apresenta uma solução à essas questões, propondo uma abordagem que lida com a heterogeneidade e a variabilidade dinâmica de recursos nos ambientes. A intenção é que o Aura atue como um *proxy* para o usuário móvel, de modo que quando esse adentre um novo ambiente, o *proxy* identifique as capacidades e restrições do contexto e busque (ou adapte) o recurso apropriado para a execução de uma determinada tarefa.

Basicamente, o Aura define um *framework* arquitetural para aplicações de computação ubíqua em que as tarefas do usuário se tornam entidades de primeira classe, sendo definidas explicitamente pelos ambientes e representadas como uma coleção de serviços abstratos. Nela, os ambientes também são capazes de se auto-monitorar e renegociar o suporte a uma tarefa quando houver variação de capacidade ou recurso. Esse conjunto de características são implementados no Aura por quatro componentes principais (Sousa and Garlan 2002):

- *Service Suppliers*: responsáveis por fornecer os serviços abstratos que compõem uma determinada tarefa. Na prática, esses serviços encapsulam aplicações existentes para deixá-las em conformidade com as APIs do Aura. Por exemplo, *Emacs*, *MS-Word* e *Notepad* podem ser encapsulados para suprir serviços de *edição de texto*.
- *Context Observer*: fornece informações sobre o contexto físico e reporta eventos aos demais componentes como, por exemplo, localização de usuários, atividades sendo realizadas, etc. A implementação do *Context Observer* pode ser tão sofisticada quanto necessário, dependendo do número de sensores disponíveis no ambiente.
- *Environment Manager*: tem como principal função gerenciar os serviços existentes no ambiente e onde eles podem ser executados. Adicionalmente à descoberta de serviços individuais, ele pode avaliar configurações alternativas para selecionar aquela que melhor se adapta às preferências do usuário ou à capacidade do dispositivo.
- *Task Manager*: coordena a migração da informação relacionada a tarefa do usuário quando esse muda de ambiente. Além disso, monitora a configuração do ambiente para identificar alterações nos provedores de serviços e gerencia a execução de tarefas baseando-se em requisitos, tais como privacidade ou atividade do usuário.

2.5 Captura e Acesso de Experiências ao Vivo

Em função da inadequação de seu cérebro para registrar tudo que interessa ao seu redor, o ser humano tenta preservar memórias utilizando anotações, gravações de áudio e vídeo. Nos últimos anos, sobretudo em função da difusão da multimídia nos ambientes computacionais, o homem passou a utilizar o computador como forma de capturar e armazenar memórias, seja através de documentos tradicionais, hiperdocumentos ou arquivos de áudio e vídeo.

Uma vez provada a eficiência da utilização de recursos computacionais para o registro de memórias, aplicações surgiram no sentido de tornar mais eficiente a captura e o posterior acesso à informação capturada. Tais aplicações permitiram o registro simultâneo de diversos fluxos de informação, preservando as inter-relações existentes entre cada um deles, o que possibilita uma série de facilidades de acesso ao conteúdo capturado.

Pesquisas relacionadas ao tema captura e acesso datam de 1985, antes, portanto, do surgimento do conceito de computação ubíqua ou pervasiva. Nesse ano, (Schmandt and Arons 1985) trabalharam na captura de áudio proveniente de chamadas telefônicas, provendo posterior acesso aos dados capturados através de facilidades que incluíam a geração de resumos das conversas capturadas.

Sistemas um pouco mais recentes, tais como o Tivoli (Minneman et al. 1995; Moran et al. 1997a; Moran et al. 1997; Moran and van Melle 2000), desenvolvido no Xerox PARC e utilizado para resumir os resultados de uma reunião capturada, passaram a utilizar dispositivos de computação ubíqua, tais como lousas eletrônicas, chamadas no projeto de *Liveboards* (Elrod et al. 1992). No sistema em questão, artefatos produzidos na lousa eletrônica durante uma reunião são marcados através de *timestamps*. Essa informação temporal é associada aos fluxos de áudio e vídeo para prover um acesso mais eficiente ao conteúdo capturado. Integração similar entre *ink* e áudio/vídeo ocorre no Projeto *eClass* (Abowd et al. 1998; Abowd et al. 1998; Abowd 1999), o qual se diferencia dos demais por prover um rico processo de pós-produção de material capturado. Nesse processo, áudio, vídeo e *ink* são integrados em páginas *Web* associadas a um espaço colaborativo conhecido como *CoWeb*, permitindo a contínua extensão dos documentos gerados (Pimentel et al. 2001). Além da extensão da informação capturada, o Projeto *eClass* também possibilita a pré-produção de objetos, tais como *slides*, que podem ser incorporados ao processo de captura das atividades ao vivo. Dessa forma, o *eClass* cria um novo paradigma de captura, prevendo um conjunto de ativi-

dades executadas antes, durante e após a captura de experiências ao vivo. A Seção 2.5.1 descreve em detalhes o Projeto *eClass*.

Alguns outros trabalhos também focam questões relacionadas a pré e pós-produção de material proveniente de atividades de captura. Barger et al. (1998) e White et al. (1998) relatam o desenvolvimento de um sistema de captura que oferece suporte para a realização de anotações referentes a material capturado; entretanto, não oferece nenhum suporte para incorporação de atividades realizadas antes da sessão de captura. O sistema DOLPHIN (Streitz et al. 1994), aplicação de *groupware* projetada para oferecer suporte a reuniões remotas e presenciais, permite que usuários manipulem e estendam material previamente preparado, possibilitando inclusive a gravação de informação manuscrita.

A seguir, são apresentados alguns exemplos de aplicações de captura desenvolvidos para domínios de salas de aula e reuniões.

2.5.1 *eClass*

Com a introdução de novas técnicas de ensino em sala de aula, tais como apresentações multimídia e apresentações de conteúdos de páginas *Web*, entre outras, estudantes passaram a receber uma carga de informação cada vez maior. Para tentar absorver todo esse conteúdo, o estudante é forçado a fazer diversas anotações, perdendo a oportunidade de discutir e participar mais ativamente da aula. Liberar o estudante dessas anotações tediosas foi o principal objetivo que motivou o desenvolvimento do projeto *eClass*.

O projeto *eClass* surgiu em 1996, tendo recebido na época o nome de Classroom 2000, nome que consta na maioria das publicações relacionadas ao projeto (Abowd et al. 1998; Abowd et al. 1998; Abowd 1999).

O *eClass* implementa um ambiente de computação ubíqua que permite a captura de material multimídia em uma sala de aula convencional, e a disponibilização e acesso desse material através da *Web*. Para a captura de material utiliza-se uma sala de aula equipada com câmeras de vídeo, microfones, projetores e uma lousa eletrônica. Lousa eletrônica é uma tela sensível ao toque que, conectada a um computador, permite armazenar as interações do usuário com a tela. Esses equipamentos de captura são adaptados na sala de forma não intrusiva, conforme ilustra a Figura 2.3(a).

Uma infra-estrutura de software dá o devido suporte à lousa eletrônica e aos demais equipamentos de captura usados na sala, sendo também respon-

sável pela integração e sincronização dos diversos fluxos de informação capturados. Após o encerramento da aula, as informações capturadas pelo *eClass* são armazenadas em uma base de dados e a partir dessas informações, *scripts* geram hiperdocumentos que podem ser visitados na Web.

Figura 2.3: O sistema *eClass*. (a) Sala de aula instrumentada. (b) Hiperdocumento gerado automaticamente pelo *eClass*.

A Figura 2.3(b) apresenta uma amostra dos documentos multimídia automaticamente gerados após uma sessão de captura. A interface utilizada divide a tela em quadros (*frames*) para facilitar a visualização da informação. O quadro da direita apresenta uma aula como uma seqüência de *slides*, mostrando como eles foram apresentados ou gerados em sala. O instrutor tem a possibilidade de apresentar *slides* previamente preparados, ou gerá-los durante a aula utilizando a lousa eletrônica. O instrutor pode ainda escrever sobre *slides* previamente preparados, enriquecendo a informação existente. O sistema provê diferentes interfaces que permitem acesso aos *slides* de uma aula juntamente com o áudio ou vídeo associado. Usuários com conexões rápidas podem utilizar o conjunto de interfaces que apresentam vídeo, enquanto que, para usuários com conexões lentas, é provido um conjunto de interfaces que utiliza unicamente áudio como mídia contínua associada.

Cada aula é gravada como um único fluxo de áudio ou vídeo, e a interface de acesso disponibiliza diferentes formas para indexação dos eventos importantes ocorridos no decorrer da sessão, de forma a possibilitar acesso randômico ao conteúdo capturado. Por exemplo, acima de cada *slide* mostrado na Figura 2.3(b) existe uma hiperligação para a parte do vídeo associada à sua apresentação, e as anotações feitas pelo instrutor na lousa eletrônica também servem como indexadores para os fluxos de mídia contínua.

2.5.2 StuPad (Student NotePad)

Como já comentado, o principal objetivo do projeto *eClass* é poupar o aluno da tarefa tediosa de anotar as informações apresentadas em sala de aula. Entretanto, uma vez que as notas de aula capturadas resultavam apenas de ações dos instrutores, alguns alunos sentiam a necessidade de fazer suas próprias anotações de forma a relacionar assuntos de uma aula com seus conhecimentos prévios. Quando faziam suas próprias anotações, os estudantes se viam forçados a integrar conteúdo digital com informação anotada em papel, tarefa não trivial.

Para solucionar este problema, foi necessário permitir que os estudantes fizessem suas anotações em formato eletrônico, para que essas pudessem ser integradas com o material gerado pelo instrutor. A solução encontrada foi a disponibilização de dispositivos de anotação conhecidos como *tablets*, os quais possuem funcionalidade semelhante a de uma lousa eletrônica e ficam interligados em rede com o equipamento do instrutor. Esses dispositivos permitem que os alunos visualizem o mesmo conteúdo apresentado na lousa eletrônica, podendo adicionar suas anotações às anotações do instrutor. O processo de pós-produção armazena as anotações realizadas por cada estudante, identificando o autor de cada uma. Assim, durante a fase de consulta ao material capturado, cada aluno tem sua visão da aula segundo sua interação com o *tablet*. A Figura 2.4 ilustra o funcionamento desse mecanismo de captura.

Figura 2.4: Stupad auxiliando atividades didáticas (Truong et al. 2001).

2.5.3 SAAMPad (Software Architecture Analysis Method Pad)

O gerenciamento de projetos de software é uma tarefa complexa que consome um tempo razoável dos projetistas. A evolução de um sistema ocorre de maneira menos complexa quando os projetistas têm uma completa compreensão das decisões de projeto (*design rationale*) que resultaram na definição de

uma determinada arquitetura (Truong et al. 2001). Dessa forma, para entender por que um sistema foi projetado de uma determinada maneira, deve-se preservar as decisões de projeto.

No sentido de documentar adequadamente tais decisões, o Software Engineering Institute definiu um método de análise da arquitetura de softwares conhecido como SAAM - *Software Architecture Analysis Method* - cuja principal função é oferecer suporte a discussões estruturadas de decisões de projeto (Kazman et al. 1994). O SAAM é um processo baseado em grupos de pessoas, onde profissionais elaboram cenários que ajudam a obter informações dos impactos provocados por mudanças em sistemas existentes. Uma sessão SAAM típica é um evento ao vivo envolvendo um grupo de 3 a 10 pessoas, entre usuários, projetistas, programadores e facilitadores que discutem a arquitetura de um sistema, a qual é apresentada em um *display* público.

Analisando as características de uma sessão SAAM, Truong et al. (2001) concluíram que a infra-estrutura utilizada no Projeto *eClass* poderia ser facilmente adaptada para capturar esse tipo de sessão. Dessa forma, foi desenvolvida uma aplicação para apresentar aspectos arquiteturais de sistemas em uma lousa eletrônica, permitindo anotações sobre os diagramas apresentados, como mostra a Figura 2.5(a). A aplicação indexa interações na lousa eletrônica com fluxos de áudio e vídeo capturados nas sessões ao vivo. Dessa forma, a discussão de cada bloco arquitetural do sistema fica ricamente documentada, podendo ser revista para auxiliar decisões de projeto (Figura 2.5(b)).

(a)

(b)

Figura 2.5: SAAMPad - (a) projetistas participam de uma sessão de projeto de software; (b) tela mostra a interface de acesso de uma sessão SAAM (Truong et al. 2001).

2.5.4 DUMMBO (Dynamic, Ubiquitous, Mobile Meeting BOard)

O que motivou a implementação do sistema DUMMBO foi o fato de diversas reuniões ocorrerem de maneira não programada, quando duas ou mais pessoas se encontram casualmente. Várias idéias úteis surgem desse tipo de encontro, que ocorre muitas vezes na sala do cafezinho, ou nos corredores de uma empresa ou universidade. Entretanto, tais idéias freqüentemente se perdem dada a impossibilidade de lousas comuns armazenarem o que nelas foi escrito.

O sistema DUMMBO foi desenvolvido para preservar o resultado dessas discussões informais. Composto de uma lousa eletrônica sem projetor e de um sistema de som anexo à lousa, conforme mostra a Figura 2.6(a), o sistema é iniciado e passa a armazenar interações com a lousa toda vez que o pincel é retirado de sua posição de descanso. O DUMMBO conta ainda com sensores capazes de detectar a presença de pessoas próximas à lousa. Quando a presença de duas ou mais pessoas é detectada, a gravação do áudio é iniciada mesmo que não haja interação com a lousa.

O sistema provê uma interface de acesso via *Web*, como ilustra a Figura 2.6(b). Informações de contexto das reuniões informais (quem participou, onde e quando) são utilizadas para auxiliar na busca por conteúdo capturado. Essa interface provê uma linha de tempo mostrando os períodos de atividade do sistema, e pode aplicar filtros para localizar uma determinada reunião. Ao localizar o conteúdo desejado, a interface possibilita que as atividades realizadas na lousa eletrônica sejam exibidas de forma sincronizada com o áudio capturado.

(a)

(b)

Figura 2.6: DUMMBO - (a) interface de captura; (b) interface de acesso (Truong et al. 2001).

2.5.5 LiteMinutes

O LiteMinutes (Chiu et al. 2001) é um sistema baseado em Internet para sumarização de reuniões, permitindo a captura de anotações, *slides* e vídeo. Explorando um ambiente instrumentado com *laptops* conectados através de uma rede sem fio, os participantes da reunião podem fazer anotações em um *applet* Java acessado pela *intranet* do laboratório (Figura 2.7(b)). Uma estação de trabalho reproduz e captura imagens em um *display* situado na parede principal da sala e é gravado um vídeo da reunião em formato MPEG. Ao final da reunião as anotações são submetidas a um servidor via CGI, o servidor as processa, faz a integração temporal com as outras mídias (*slides* e vídeo), gera a página *Web* associada e envia uma versão textual da reunião por *email* para os participantes.

(a)

(b)

Figura 2.7: Sistema LiteMinutes. (a) Sala instrumentada. (b) Applet Java para anotações (Chiu et al. 2001).

2.5.6 Interactive Workspaces

O projeto Interactive Workspaces (Johanson et al. 2002), iniciado na Universidade de Stanford em 1999, investiga a integração de salas dedicadas a reuniões à aparatos computacionais que vão de grandes *displays* à pequenos dispositivos móveis. Um *Interactive Workspace* pode ser visto como um lugar com suporte efetivo ao trabalho colaborativo, onde as pessoas agregam seus recursos computacionais sobre uma rede sem fio que as possibilita compartilhar e interagir com a informação de várias maneiras.

Nesse contexto, o projeto envolve a construção de um espaço interativo denominado iRoom (interactive Room) (Johanson et al. 2003), o qual é composto por whiteboards eletrônicas instaladas nas paredes, mesas de conferência com

displays embutidos, câmeras, microfones e outros dispositivos (Figura 2.8). Alguns dos requisitos para o sistema foram identificados a partir pesquisas e experimentos com usuários, como por exemplo: suporte à movimentação de dados (incluindo aplicações) entre os múltiplos dispositivos; suporte à transferência de controle, possibilitando que um usuário assuma o controle de um dispositivo à partir de outro; coordenação dinâmica de aplicações, isto é, as tarefas e funcionalidades de uma ferramenta devem poder ser combinadas com as de outras quando apropriado; e adaptação, do ponto de vista de IHC, permitindo que uma interface seja portada para diferentes *displays* e possivelmente suportando diferentes modalidades de interação.

Figura 2.8: Exemplo de sala de reuniões equipada com o iRoom (Johanson et al. 2002).

Para satisfazer tais requisitos, foi desenvolvida uma infra-estrutura denominada iROS (*i*nteractive *R*oom *O*perating *S*ystem), a qual é composta de três sub-sistemas:

- *Data Heap*: facilita a movimentação de dados entre as aplicações. Utilizando um conjunto de atributos associados, os dados podem ser automaticamente convertidos para uma aplicação ou dispositivo específico. Por exemplo, se um dispositivo não suporta *PowerPoint*, o *Data Heap* irá extrair as imagens e convertê-las em um formato que seja aceito pelo dispositivo, como JPEG;
- *iCrafter*: provê um sistema para anúncio e invocação de serviços, juntamente com um gerador de interfaces para os mesmos. Os serviços são

similares aos disponibilizados pelo Jini¹, sendo que o gerenciador de interface do iCrafter permite que um usuário escolha um serviço e então automaticamente retorna a melhor interface para o dispositivo;

- *Event Heap*: armazena e repassa mensagens de notificação de eventos, as quais correspondem a uma coleção de campos *nome-tipo-valor*. As aplicações podem seletivamente acessar e postar eventos em um repositório central, que expiram após um determinado período de tempo.

2.5.7 Roomware

Assim como no projeto Interactive Workspaces, o projeto Roomware (Prante et al. 2004; Streitz et al. 2001), desenvolvido pelo Instituto Fraunhofer IPSI na Alemanha, tem como foco a integração de grandes *displays* e áreas de interação com o ambiente físico/arquitetural, tornando a tecnologia parte do contexto onde as pessoas processam e compartilham informação. O Roomware é baseado na noção de *Cooperative Buildings*, que consiste no uso de objetos do mundo real, tais como mobília, portas e paredes, acoplados a dispositivos computacionais para dar suporte ao trabalho cooperativo. Essa combinação dá origem à ambientes híbridos, populados por artefatos inteligentes e ubíquos, que ligam o mundo real ao virtual.

Como resultados do projeto foram desenvolvidos diversos equipamentos (Figura 2.9) como, por exemplo, o *DynaWall*, que consiste de um *display* interativo de 4.5m x 1.1m suavemente integrado à parede; o *InteracTable*, que utiliza uma tela de plasma sensível ao toque sobre uma mesa; a *CommChair*, que combina a mobilidade de uma cadeira comum à funcionalidade de um computador baseado em caneta; e as *ConnecTables*, que correspondem à versões modificadas da *CommChair* que podem ser conectadas, estendendo a área de trabalho.

Todos esses equipamentos podem ser utilizados de forma compartilhada através de uma rede sem fio, permitindo que múltiplos usuários executem tarefas em um mesmo dispositivo em paralelo, ou que esses criem separadamente diferentes visões de um objeto e depois as sincronizem. O sistema também provê uma infra-estrutura baseada em sensores que possibilita a localização de dispositivos, assegurando a mobilidade dos usuários e a integração dos componentes de acordo com sua posição dentro do ambiente.

¹Tecnologia desenvolvida pela Sun Microsystems, consiste em uma arquitetura aberta que permite a criação de serviços de rede — <http://www.sun.com/software/jini>

Figura 2.9: Equipamentos desenvolvidos no projeto Roomware: *ConnectTables*, *DynaWall*, *CommChair* e *InteracTable* (Prante et al. 2004).

2.5.8 EasyLiving

O projeto EasyLiving (Brumitt et al. 2000), conduzido pelo Microsoft Research Center, investiga o desenvolvimento de tecnologias e arquiteturas para ambientes inteligentes, também conhecidos como *smart spaces*. O objetivo desses ambientes é suportar a agregação dinâmica de diferentes tipos de dispositivos eletrônicos, tais como câmeras, microfones, sensores e atuadores, e fazê-los trabalhar conjuntamente, de modo a prover serviços que aprimorem e facilitem a execução de atividades pelo usuário.

Basicamente, a arquitetura construída no EasyLiving é constituída de um *middleware* para habilitar a comunicação entre os dispositivos, um módulo de localização geométrica e rastreamento por sensores, um sistema de câmeras para percepção do ambiente, e uma lógica de abstração e descrição de serviços.

- *Middleware*: fornece suporte à comunicação assíncrona, um mecanismo de endereçamento independente de máquina e um protocolo baseado em XML para troca de mensagens entre dispositivos. O EasyLiving adota um modelo distribuído de comunicação em que cada dispositivo funciona de forma autônoma sobre a rede, sem a necessidade de um servidor central.
- *Geometria*: provê informação sobre os relacionamentos físicos de localização entre pessoas, dispositivos, lugares e objetos. Isso permite que dispositivos de interface com usuário possam se adaptar para uma interação em particular, por exemplo, movimentando-se ou passando o controle da tarefa para outro dispositivo.
- *Percepção*: baseado em um sistema de visão computacional estéreo, é utilizado para identificar pessoas e objetos no ambiente e inferir a intenção do usuário através de sua posição (Figura 2.10). Funciona de

maneira complementar ao módulo de localização geométrica.

- **Serviços:** Assim como em outros sistemas ubíquos, o EasyLiving encapsula a diversidade de dispositivos e plataformas em serviços que separam o controle do hardware de sua interface de apresentação. Esses serviços são descritos por meio de esquemas XML, permitindo consultas sobre os comandos disponíveis e a validade de parâmetros.

Figura 2.10: EasyLiving: rastreamento de pessoas e objetos através de visão computacional. (Brumitt et al. 2000).

2.6 Terminologia e Conceitos Relacionados

A computação ubíqua, por ser uma área relativamente recente e por ainda estar em fase de definição, apresenta uma série de termos diferentes utilizados para sua representação. Embora alguns desses termos, como *pervasive computing*, *calm computing*, *invisible computing* e *wearable computing*, sejam utilizados como sinônimos de computação ubíqua por alguns pesquisadores, eles possuem definições diferentes. Apesar dessa variação, a grande maioria das definições compartilha características semelhantes àsquelas propostas por Weiser (1991).

2.6.1 Pervasive Computing

Pervasive computing, ou computação pervasiva, é um termo utilizado para caracterizar a fusão transparente de inúmeros dispositivos computacionais

nos ambientes, sobretudo de trabalho e estudo. Uma característica importante da computação pervasiva é permitir que esses inúmeros dispositivos sejam naturalmente acessíveis.

O conceito de computação pervasiva não é recente. Em 1999, Ark and Selker previam a disseminação da computação pervasiva com base na análise de quatro fatores: (i) a computação já se encontrava presente na maioria dos ambientes, (ii) dispositivos de comunicação, tais como celulares, tornavam-se cada vez mais populares, (iii) usuários apresentavam uma crescente mobilidade e (iv) a comunicação entre indivíduos, entre indivíduos e computadores e entre os próprios computadores aumentava significativamente.

De fato, o que se observa é uma tendência irreversível de incorporação de tecnologia no cotidiano das pessoas de maneira transparente, conforme previu Weiser (Weiser 1991)

2.6.2 Invisible Computing

Norman (1998) utiliza o termo *invisível* para descrever um atributo comum encontrado em certas ferramentas. Sem se preocupar com a complexidade das funções que essas ferramentas podem executar, a tecnologia utilizada para realizar uma operação não é aparente para o usuário da ferramenta. Tais ferramentas são ditas centradas nos usuários e não na tecnologia utilizada, a qual é considerada invisível.

2.6.3 Wearable Computing

A proposta de *wearable computing* é permitir uma simbiose entre os computadores e seus usuários (Starner 1999). Os usuários estariam livres para utilizar os computadores de uma maneira direta e natural, como se os dispositivos computacionais fossem uma extensão do corpo humano.

Wearable computing, explora os conceitos de usabilidade e ergonomia a fim suplantam o convencional *desktop* como a interface padrão para computação. De acordo com essa visão, o monitor poderia estar embutido nos óculos dos usuários e o *mouse* poderia ser substituído por luvas digitais.

Starner (1999) sugere que persistência e consistência são as duas principais características de uma interface para *wearable computing*. A interface é persistente porque está constantemente disponível e é utilizada ao mesmo tempo em que o usuário está realizando outras tarefas. Consistência significa

que a mesma interface e suas funcionalidades devam ser utilizadas em todas as situações, ainda que estejam se adaptando e modelando ao longo da interação com o usuário.

2.7 Computação no Cotidiano

A computação no cotidiano (*everyday computing*) é uma nova forma de computação ubíqua baseada nos eventos diários que ocorrem na vida dos indivíduos. Nesse tipo de computação, a disponibilidade de acesso computacional deve ser constante e o aspecto tempo é fundamental e necessita ser representado nas interfaces de acesso (Abowd and Mynatt 2000). Exemplos familiares, como comunicação com outras pessoas, precisam ser suportados pela dimensão temporal. Por exemplo, em uma reunião de projetos, saber se a última vez que a modelagem foi atualizada ocorreu há uma semana ou há um mês, pode ser fundamental para a tomada de decisão por parte dos envolvidos.

Abowd e Mynatt (2000) definem características próprias das atividades cotidianas:

- Não possuem início e fim claros: um usuário pode definir um processo de captura sem um início claro e tão pouco um final previsto. É importante que as aplicações forneçam visibilidade do estado corrente do processo, liberdade de diálogo e simplicidade no acesso;
- Possuem múltiplas atividades ocorrendo de forma concorrente: as aplicações devem permitir que os usuários possam monitorar e retomar essas atividades;
- Devem ser suportadas por um modelo associativo de informação: esse modelo, diferentemente do modelo hierárquico, permite que o acesso seja realizado a partir de diferentes pontos de vista. Um usuário pode querer acessar uma reunião capturada a partir de algum ponto específico, seja baseado em linha de tempo ou em ações. Por exemplo, ele pode querer rever a reunião a partir do momento que um dado participante saiu da mesma.
- As múltiplas formas de interação no mundo real são implícitas em termos de computação cotidiana. Dessa forma, essa área de pesquisa é relevante no estudo da computação ubíqua. Ainda, a implementação de ambientes de computação no cotidiano sugere que os temas abordados anteriormente — interfaces naturais, computação ciente de contexto e captura e acesso de experiências ao vivo — sejam fortemente coordenados.

No sentido de explorar a computação no cotidiano, pesquisadores do GA-Tech construíram a Aware Home (Kidd 1999), uma casa real, com as características comuns das residências americanas, porém dotada de câmeras, microfones e sensores, conforme ilustra a Figura 2.11. Tais dispositivos são acoplados a sistemas computacionais, a fim de permitir a identificação de pessoas, suas localizações e atividades. Através desse sensoriamento, é possível auxiliar a comunicação entre indivíduos e facilitar a execução de atividades domésticas do cotidiano.

Figura 2.11: Aware Home: (a) Fachada externa; (b) Equipamentos computacionais dão suporte aos dispositivos ubíquos; (c) Câmeras de vídeo e microfones instalados no teto de cada ambiente; (d) Imagem obtida a partir de uma das câmeras.

Diversas aplicações vêm sendo construídas a fim de explorar a infra-estrutura provida pela Aware Home. Algumas dessas aplicações são apresentadas a seguir.

2.7.1 *Digital Family Portrait*

Nos dias atuais é cada vez mais comum a separação geográfica entre membros de uma mesma família. Em muitos casos, parentes próximos vivem a

milhares de quilômetros uns dos outros. Tal característica da vida moderna dificulta a observação casual do cotidiano dos membros da família. Esse fato é extremamente relevante quando um parente idoso ou doente está distante. Entretanto, se tal pessoa estiver vivendo em uma Aware Home é possível monitorar suas atividades e informar diariamente a família a respeito de qualquer mudança de comportamento que possa indicar um problema.

Uma questão importante sobre esse monitoramento é que ele não deve ser intrusivo, ou seja, não deve invadir a privacidade da pessoa que está sendo monitorada. Outro ponto importante é fornecer informação de maneira clara e rápida. Para tanto, foi usada uma abstração baseada em porta retratos (*portraits*) (Mynatt et al. 2001), nos quais as bordas estáticas são substituídas por bordas digitais com desenhos que representam a atividade diária da pessoa, conforme ilustra a Figura 2.12(a). Um desenho pequeno representa um dia com pouca atividade, enquanto um grande representa um dia cheio de atividades. Dessa forma, é possível mensurar a atividade diária de uma pessoa sem invadir sua privacidade. Existem interfaces mais detalhadas que oferecem maiores informações, tais como informações climáticas. Isso é importante, pois se o tempo está frio e chuvoso um dia de pouca atividade pode ser considerado normal. A Figura 2.12(b) mostra a interface avançada do sistema.

Figura 2.12: *Digital Family Portrait*: (a) Porta retrato com bordas digitais; (b) Interface com informações detalhadas.

2.7.2 Cook's Collage

Ambientes instrumentados como o existente na Aware Home oferecem facilidades para o gerenciamento de tarefas que requerem a utilização intensa da memória, sobretudo a memória recente. Uma dessas atividades é o preparo

de receitas. Considerando que a cozinha é um lugar de alto fluxo de pessoas, e que a atividade de cozinhar é freqüentemente interrompida por outros eventos (chamadas telefônicas, por exemplo), dispositivos de captura podem ser de grande ajuda para retomar a execução de uma receita, após uma eventual interrupção.

O *Cook's Collage* (Tran et al. 2005) é um sistema ciente de contexto capaz de capturar informações transientes de atividades recentes e apresentá-las de forma resumida, oferecendo um mecanismo eficiente para extensão da memória humana no que diz respeito a eventos recentes. A Figura 2.13(a) mostra os quadros representativos do preparo de uma determinada receita. Tais quadros permitem que o cozinheiro se certifique de que colocou todos os ingredientes na receita. A Figura 2.13(b) mostra o processo de escolha dos quadros representativos a partir das seqüências de quadros capturados.

Figura 2.13: Cook's Collage: (a) Quadros representativos das principais atividades realizadas; (b) Escolha dos quadros representativos.

2.8 Considerações Finais

A computação ubíqua tem se constituído em um novo paradigma computacional, que difere de seus predecessores ao oferecer facilidades computacionais incorporadas aos ambientes de trabalho e estudo. As aplicações de computação ubíqua são desenvolvidas preferencialmente para dispositivos não centrados no *desktop*; tais dispositivos foram apresentados e suas funcionalidades foram discutidas no contexto de diferentes aplicações.

Dentre as aplicações de computação ubíqua, aquelas que realizam a captura de experiências humanas a fim de permitir o posterior acesso ao conteúdo capturado têm recebido significativa atenção de diferentes grupos de pesquisa ao redor do mundo. Essas aplicações exploram resultados de outras duas

importantes áreas da computação ubíqua — interfaces naturais e ciência de contexto — a fim de instrumentar ambientes de maneira pervasiva, permitindo capturar experiências humanas de forma minimamente intrusiva.

As aplicações de captura e acesso caminham no sentido de suportar as mais diferentes experiências humanas. Um domínio recentemente investigado é o doméstico: formas alternativas de comunicação remota e apoio a atividades do cotidiano têm sido alvos de pesquisa.

Entretanto, criar aplicações robustas capazes de atender diferentes domínios de experiências humanas requer suporte computacional adequado. As soluções existentes, tanto para tratamento de informações de contexto quanto para a captura de atividades humanas estão ainda direcionadas a domínios específicos, o que dificulta o reuso dessas soluções.

As aplicações de captura e acesso possuem a particularidade de explorar informações de contexto durante a captura ao vivo e posteriormente, para recuperar adequadamente conteúdo capturado. As soluções que oferecem suporte ao tratamento de informações de contexto encontram-se maciçamente voltadas à primeira fase. Observou-se, assim, uma carência de suporte ao tratamento dessas informações na fase de acesso aos dados capturados.

As dimensões de contexto, apresentadas na Seção 2.4, constituem uma forma simples, porém poderosa, de enriquecer o conteúdo capturado de modo a facilitar operações de recuperação e apresentação de experiências capturadas.

Esta tese propõe um modelo de desenvolvimento de aplicações de captura e acesso que explora as dimensões de contexto discutidas a fim de prover serviços de gerenciamento de conteúdo capazes de minimizar o esforço no desenvolvimento de aplicações de acesso.

O trabalho desenvolvido encontra-se no escopo de um projeto que visa oferecer abstrações de alto nível que facilitem as diferentes atividades de desenvolvimento associadas à construção de aplicações de captura e acesso. Esse projeto é discutido no próximo capítulo.

O Projeto InCA-SERVE

3.1 *Considerações Iniciais*

A construção de aplicações de captura e acesso constitui uma tarefa multidisciplinar que, em geral, requer conhecimento de diferentes áreas da ciência da computação, incluindo sistemas distribuídos, engenharia de software, sistemas hipermídia e bancos de dados, entre outras.

Outro aspecto importante relacionado ao desenvolvimento de aplicações de captura e acesso diz respeito à similaridade de projeto: questões associadas à comunicação entre componentes e também à funcionalidade desses componentes aparecem de maneira recorrente. Esses fatos sugerem uma abordagem infra-estrutural na construção dessas aplicações.

Entretanto, a literatura tem reportado a implementação de inúmeras aplicações de captura e acesso construídas segundo uma abordagem *ad hoc*, o que dificulta a manutenção e a extensão das aplicações implementadas. Além disso, a ausência de uma metodologia sistemática na implementação de aplicações eleva consideravelmente o tempo necessário para a disponibilização dessas aplicações (Truong and Abowd 2004).

Outra questão pouco tratada por boa parte das aplicações diz respeito ao pós-processamento do conteúdo capturado. Várias das aplicações reportadas na literatura limitam-se a apresentar o conteúdo capturado na *Web*. Imple-

mentações que facilitem o reuso de material capturado ou permita extensão desse conteúdo são incomuns.

O principal objetivo do Projeto InCA-SERVE foi prover um modelo de desenvolvimento capaz de (i) acelerar o processo de prototipação de aplicações, (ii) facilitar a manutenção e evolução dessas aplicações e (iii) oferecer serviços que auxiliem a pós-produção de conteúdo capturado, permitindo reuso e extensão desse conteúdo.

O Projeto InCA-SERVE teve sua origem a partir do projeto *eClass*, desenvolvido no Georgia Institute of Technology (GATech). Neste capítulo, a Seção 3.2 apresenta um breve histórico do Projeto InCA-SERVE. Em seguida, a Seção 3.3 apresenta o sub-projeto InCA, detalhando suas características e funcionalidades. A Seção 3.4 apresenta a xInCA, uma extensão que agrega componentes funcionais à infra-estrutura InCA. Na Seção 3.5, o projeto SERVE é apresentado e discutido de maneira detalhada. Finalmente, a Seção 3.6 faz as considerações finais do capítulo.

3.2 Do *eClass* ao InCA-SERVE

O projeto *eClass* (Abowd 1999) investigou o uso de computação ubíqua para a captura de atividades educacionais tendo sido utilizado na captura de mais de 1200 aulas de cursos de graduação e pós-graduação da GATech entre 1996 e 2001. Nesse período, diversas ferramentas e serviços foram integrados ao ambiente *eClass* a fim de estender suas funcionalidades. Entre eles, vale destacar a CoWeb (*Collaborative Web Site*), a qual provê um espaço extensível, persistente e colaborativo para troca de informações. A integração entre a CoWeb e o *eClass* ocorreu através de um serviço de criação manual de ligações bidirecionais entre informações capturadas pelo ambiente *eClass* e páginas criadas na ferramenta CoWeb.

Abowd et al. (1999) avaliaram a eficácia da integração dos ambientes *eClass* e CoWeb mensurando, no contexto de um curso, o número de páginas CoWeb criadas e o número de ligações entre *slides* capturados e páginas criadas. No curso monitorado, 303 páginas foram criadas e 66 ligações foram definidas entre slides capturados e páginas CoWeb. Questionários aplicados no final do curso demonstraram que 48,6 % dos alunos perceberam um melhor entendimento do material capturado quando este apresentava páginas CoWeb associadas.

Esse experimento deixou claro que extensões ao material capturado per-

mitem ampliar sua utilização e sua funcionalidade. Outras observações sugeriam a necessidade de novas extensões ao projeto *eClass*. O volume de dados capturados, por exemplo, tornava-se cada vez maior, trazendo dificuldade na busca por um determinado assunto. Os mecanismos de busca e visualização de resultados implementados mostravam-se pouco eficientes para lidar com um repositório sempre crescente de informações.

Parte dessa ineficiência estava relacionada com a forma segundo a qual as informações eram armazenadas. O modelo de armazenamento inicialmente utilizado não aproveitava algumas informações contextuais disponíveis no momento da captura, e não era capaz de expressar a referência intrínseca existente entre sessões relacionadas.

Tais limitações, importantes no contexto de captura de aulas presenciais, somaram-se a outras quando os pesquisadores do projeto decidiram estender o uso da infra-estrutura de captura para atividades genéricas. Considerando o barateamento e a difusão de equipamentos de captura, avaliou-se que o conhecimento acumulado com o desenvolvimento do sistema *eClass* podia ser utilizado para construção de infra-estruturas genéricas, sobre as quais aplicações de captura e acesso de diferentes atividades humanas pudessem ser construídas. Segundo essa visão, o *eClass* passaria a ser uma dessas aplicações.

No sentido de implementar tais infra-estruturas de captura e acesso genéricas e, além disso, fornecer facilidades para o armazenamento, recuperação e extensão de informação proveniente de captura, decidiu-se desenvolver o projeto InCA-SERVE. A InCA — *Infrastructure for Capture and Access* — é a parte do projeto responsável pela implementação da infra-estrutura de captura e acesso. A SERVE — *infrastructure for Store, Extend, Retrieve and Visualize Evolutionary information* — cuida da parte de armazenamento, extensão, recuperação e visualização de informação evolucionária. De uma forma simplificada, pode-se dizer que a InCA gera informações resultantes de atividades de captura e a SERVE armazena e estende tais informações a fim de otimizar a recuperação da informação armazenada e permitir sua visualização.

O InCA-SERVE foi resultado de uma parceria entre o GATech, sob supervisão do Dr. Gregory Abowd, e o ICMC-USP, onde foi supervisionado pela orientadora deste trabalho, sendo apoiado no Brasil pelo CNPq e pela FAPESP e nos EUA pelo *National Science Foundation* (NSF). As infra-estruturas InCA e SERVE encontram-se detalhadamente descritas nas seções a seguir.

3.3 InCA

A proposta do projeto InCA é prover uma infra-estrutura que simplifique o processo de desenvolvimento de aplicações de captura e acesso, abstraindo a natureza distribuída dos sistemas de computação ubíqua. A arquitetura da InCA tem como objetivo separar aspectos pertinentes à captura, ao armazenamento, à conversão, à integração e ao acesso à informação, provendo módulos de *software* para o apoio a cada um desses aspectos. Cada módulo de *software* consiste de uma classe para dispositivos e aplicações que implementam uma determinada interface. Assim, para desenvolver a parte de uma aplicação responsável pela captura de atividades, é necessário fazê-la implementar a interface de captura (*capturer*), e possuir uma instância do módulo de captura (*capture module*) (Truong and Abowd 2004).

A InCA é resultado de um estudo que permitiu identificar e classificar os aspectos funcionais presentes em um amplo conjunto de aplicações, muitas das quais desenvolvidas pelo grupo de pesquisa da Georgia Tech (Brotherton and Abowd 1998; Brotherton et al. 1998; Abowd 1999; Richter et al. 2001).

Existem duas abstrações chave presentes na infra-estrutura InCA: a *abstração de comunicação* e a *abstração de informação*. A primeira tem a função de deixar transparente às aplicações os detalhes de comunicação (*sockets*, *streams*, etc.) entre diferentes módulos. No sentido de implementar essa abstração a InCA provê um componente denominado *Registry*, que controla de forma automática a presença de módulos de captura e acesso. Quando um determinado componente é registrado, o fluxo entre ele e os demais componentes já registrados é automaticamente habilitado. A Figura 3.1 (a) ilustra uma configuração básica para uma aplicação de captura e acesso operando em modo *stand-alone*. Configurações mais complexas são também suportadas graças à capacidade dos componentes de registro de compartilharem informações entre si, como mostra a Figura 3.1 (b).

A *abstração de informação* tem como função desvincular o desenvolvimento de uma aplicação do tipo de informação que esta deverá suportar. Isso porque quando aplicações são construídas para suportar um tipo específico de informação, a evolução dessas aplicações pode ficar comprometida, uma vez que novos tipos de informação podem não ser suportados.

A InCA implementa abstração de informação através do tratamento de dados como BLOBs — *Binary Large Objects*. A informação é organizada como objetos de dados com uma lista de pares de atributos nome-valor que descrevem informação sobre os dados. Tal representação uniforme imposta à informação

Figura 3.1: Possíveis configurações da InCA: (a) Todos os componentes residindo na mesma máquina. (b) Componentes distribuídos em várias máquinas.

capturada abstrai da infra-estrutura a especificidade da informação, ficando a cargo da aplicação a manipulação e o uso da informação capturada (Truong and Abowd 2002).

3.4 xInCA: Estendendo a Infra-estrutura InCA

A função principal da infra-estrutura InCA é permitir que desenvolvedores possam se concentrar em questões relevantes ao processo de captura e acesso, não tendo que se preocupar com detalhes de comunicação e representação da informação capturada.

Apesar de acelerar consideravelmente o desenvolvimento de aplicações, as abstrações providas pela InCA não contemplam a recorrência de funcionalidades presente em muitas das aplicações de captura e acesso, conforme ilustra a Tabela 3.4 (Cattelan et al. 2003).

A lousa eletrônica (*whiteboard*) é uma das funcionalidades mais utilizadas nas aplicações de captura e acesso, seguida pela captura de áudio e vídeo, pela troca de mensagens textuais (*chat*) e pela captura de histórico de navegação hipertexto (*Weblog*).

A xInCA¹ — *Extended Infrastructure for Capture and Access Applications* (Cattelan et al. 2003; Baldochi et al. 2004) — disponibiliza componentes de soft-

¹Desenvolvida no contexto de um trabalho de mestrado concluído em 2004

Tabela 3.1: Funcionalidades presentes em aplicações de captura e acesso

Sistema	Áudio	Texto	Vídeo	Weblog	Whiteboard
DOLPHIN (Streitz et al. 1994)	X	X	X	X	X
DUMMBO (Brotherton et al. 1999)	X	–	–	–	X
Dynamite (Wilcox et al. 1997)	X	X	–	–	–
eClass (Abowd 1999)	X	–	X	X	X
Flatland (Mynatt et al. 1999)	–	–	–	–	X
LiteMinutes (Chiu et al. 2001)	–	X	X	–	X
TeamSpace (Richter et al. 2001)	X	X	X	–	X
Tivoli (Pedersen et al. 1993)	–	–	–	–	X

ware que implementam essas funcionalidades mais comuns. As abstrações providas pela xInCA possibilitam que uma aplicação de captura e acesso possa ser construída por meio de uma combinação dos componentes desejados.

3.4.1 Componentes xInCA

A xInCA é composta por um conjunto de componentes de software reutilizáveis que implementam as funcionalidades recorrentes existentes na maioria das aplicações de captura e acesso. Entende-se por componente um pacote de artefatos de software que possa ser utilizado independentemente como uma unidade e que possa ser composto com outros componentes para se construir algo maior. Os componentes xInCA encontram-se descritos a seguir.

Whiteboard

O componente de *whiteboard*, ou lousa eletrônica, provê um espaço de edição no qual o usuário pode escrever ou desenhar utilizando um dispositivo de entrada, a qual pode ser uma caneta eletrônica. As lousas eletrônicas estão presentes em uma grande variedade de aplicações, cobrindo domínios que vão do escritório à sala de aula. No contexto educacional, a *whiteboard* substitui a lousa convencional oferecendo facilidades que incluem a exibição de imagens e gráficos, a navegação entre quadros, a remoção de quadros e de conteúdo previamente escrito — tudo isso com a possibilidade de gravar as ações do usuário e permitir a posterior reprodução dessas ações. A Figura 3.2 ilustra o componente de *whiteboard* em uso no contexto do sistema *iClass* (Baldochi et al. 2003).

Figura 3.2: Componente de *whiteboard* em uso.

Captura de Texto

O componente para captura de texto realiza a troca síncrona de mensagens textuais entre aplicações. Este componente pode ser usado em uma variedade de cenários. Em ambientes de ensino, permite que alunos encaminhem suas dúvidas para o professor ou para o monitor durante a aula.

A troca de mensagens textuais entre pessoas, conhecida como *chat*, é uma aplicação que pode ser implementada utilizando o componente de texto com cerca de 15 linhas de código. Para criar um cliente de *chat* basta instanciar o componente de texto e tratar algumas exceções (*exceptions*).

Captura de Áudio

Em função da importância da fala como meio de expressão, a captura de áudio está presente em boa parte das aplicações reportadas na literatura. A xInCA provê um componente que utiliza as facilidades providas pela API JMF ² (*Java Media Framework*) para permitir a gravação de áudio a partir de qualquer dispositivo e em qualquer formato suportado por essa API.

Em função de uma limitação da InCA, que suporta apenas a transmissão de pacotes de dados (*DataObjects*), não é possível realizar *streaming* de áudio. Para resolver essa limitação, o componente de áudio é implementado utilizando o objeto *DataSink* provido pela API JMF.

O *DataSink* armazena o fluxo de áudio capturado em um arquivo local. Ao final da sessão de captura, o componente de áudio transmite o conteúdo capturado através da InCA para as partes interessadas.

²<http://java.sun.com/jmf>

Captura de Vídeo

Apesar de não ser essencial para muitas das aplicações, o vídeo associado a um evento capturado permite reconstruir o evento com maior riqueza de detalhes. Assim, para ambientes instrumentados com câmeras de vídeo, a xInCA disponibiliza um componente de captura que, assim como o componente de áudio, também utiliza as facilidades da API JMF.

Da mesma forma que o componente de áudio, o componente de vídeo também faz uso de um objeto *DataSink* para armazenamento local. Ao final da sessão, o conteúdo armazenado localmente é transmitido pela InCA para as aplicações/usuários envolvidos.

O componente de captura de vídeo suporta os formatos AVI, QuickTime, MPEG e RTP.

Captura de Histórico de Navegação

Muitos ambientes nos quais ocorrem sessões de captura possuem acesso à *Web*. Em certos domínios, as atividades realizadas na *Web* são relevantes e devem ser registradas como parte integrante da experiência ao vivo. No contexto de atividades educacionais, por exemplo, é relevante armazenar as URLs dos *sites* visitados pelo professor, já que esses *sites* contêm informação relacionada ao conteúdo da aula.

A xInCA provê um componente de software para capturar as URLs visitadas durante a sessão de captura. Esse componente instancia um *proxy* na máquina local que monitora as requisições HTTP feitas pelo navegador Web, armazenando as URLs visitadas. As URLs são armazenadas juntamente com *timestamps* que indicam em que momento da sessão cada página foi visitada. Essa informação pode ser usada, por exemplo, para relacionar as páginas visitadas aos *slides* apresentados pelo professor.

3.4.2 Funcionamento dos Componentes

Componentes xInCA de mesma funcionalidade (instâncias de um mesmo objeto) comunicam-se por meio de um identificador de sessão. Uma sessão representa um período de interação entre componentes e possui um identificador único. Componentes xInCA que compartilham um mesmo identificador são capazes de intercambiar informações, mesmo fazendo parte de aplicações distintas.

O intercâmbio de informações entre componentes xInCA depende da abstração de comunicação provida pela InCA. Assim, componentes que devem unicamente produzir informação instanciam módulos InCA de captura. Componentes que devem somente consumir informações produzidas por outras aplicações, instanciam módulos InCA de acesso. Caso o componente deva produzir e consumir informação, este deve instanciar tanto módulos InCA de captura quanto de acesso. Para habilitar a comunicação, é necessário registrar os módulos InCA de cada componente sob um mesmo número de sessão em um *Registry*.

O mecanismo de registro funciona no esquema *publish/subscribe*. Módulos de captura são registrados como publicadores e módulos de acesso como receptores da informação capturada. A Figura 3.3 ilustra o intercâmbio de informação para componentes registrados em duas sessões distintas.

Figura 3.3: Fluxo de comunicação entre componentes xInCA em duas sessões distintas (Baldochi et al. 2003).

Na Figura 3.3, App #1 é uma aplicação de captura e acesso contendo um componente de *chat* e outro de *whiteboard*, App #2 é uma aplicação que instancia um componente de *chat* e App #3 é uma aplicação contendo um componente de *whiteboard* somente para acesso. Os componentes de *chat* das aplicações 1 e 2 estão registrados sob o mesmo identificador de sessão e, conseqüentemente, possuem um canal ativo que possibilita a troca de informação entre eles. O mesmo acontece com os componentes de *whiteboard* das aplicações 1 e 3. Entretanto, o fluxo de comunicação entre a aplicação 1 e 3 é unidirecional, uma vez que a aplicação 1 apenas captura informação e a aplicação 3 somente acessa a informação capturada pela aplicação 1.

A informação capturada por um componente é disponibilizada para a aplicação a que pertence através de métodos de *callback* definidos em um con-

junto de interfaces de programação. Cada componente possui sua própria interface de programação, que deve ser implementada pela aplicação a fim de possibilitar o uso do componente.

3.5 SERVE

Utilizando as abstrações de comunicação e recorrências de funcionalidades oferecidas pela InCA e pela xInCA é possível construir aplicações de captura e acesso capazes de produzir um rico conteúdo multimídia, juntamente com metadados que permitem reconstruir a experiência capturada.

A reconstrução dessa experiência deve ser feita com diferentes níveis de detalhe, a fim de se adequar às necessidades de cada usuário e também aos dispositivos utilizados para visualização desse conteúdo. Dessa forma, a construção de aplicações de acesso não constitui tarefa trivial e pode se beneficiar de ferramentas de suporte que facilitem sua construção.

Outra questão relevante refere-se à estruturação do conteúdo capturado, o qual deve colaborar para a construção dessas aplicações. Um modelo de dados subjacente capaz de ser facilmente transformado em um modelo de apresentação constitui um requisito importante no sentido de facilitar a construção de aplicações de acesso.

A apresentação de conteúdo capturado pode ainda se beneficiar dos metadados e informações de contexto produzidos no processo de captura. Essas informações permitem facilitar a exploração do conteúdo capturado, maximizando a experiência de acesso.

Outro ponto relevante associado ao conteúdo capturado é o fato desse conteúdo ser um candidato natural para o reuso. Isso é comum nas experiências capturadas em sala de aula, as quais quase sempre produzem um número expressivo de *slides* anotados. O esboço de uma arquitetura feito na aula de Engenharia de Software pode ser importante na aula de Hipermídia. Dessa forma, ao invés de refazer do início todo o esboço, o natural é que o *slide* capturado na aula de Engenharia de Software seja reutilizado na aula de Hipermídia.

Além do reuso, o conteúdo capturado deve também poder ser estendido, por meio de anotações ou hiperligações. Isso permite enriquecer a informação capturada, aumentando sua efetividade, sobretudo quando as atividades capturadas são colaborativas, como ocorre no domínio de reuniões e de sala de aula.

No sentido de atender esse conjunto de requisitos associado ao gerenciamento de informação capturada foi desenvolvido o sub-projeto SERVE — *infrastructure for Store, Extend, Retrieve and Visualize Evolutionary information*.

A palavra evolucionária que aparece no nome dessa infra-estrutura refere-se ao fato de que as informações capturadas crescem de maneira a preservar as referências intrínsecas entre as sessões capturadas. Tais referências podem ser preservadas através do estabelecimento explícito de hiperligações entre as sessões capturadas ou podem ser expressas de maneira implícita, por meio da identificação de informações de contexto associadas ao conteúdo capturado.

No núcleo da infra-estrutura SERVE está o StRES, serviço responsável por prover facilidades de armazenamento, recuperação e extensão de conteúdo capturado. Com base nas facilidades de armazenamento e recuperação providas pelo StRES, iniciativas que suportam o reuso, a apresentação e a extensão de conteúdo capturado foram desenvolvidos e integrados à infra-estrutura. Os módulos que compõem a infra-estrutura SERVE são discutidos nas seções a seguir.

3.5.1 StRES

O StRES — *Storing, Retrieving and Extending Service* — é um serviço que oferece suporte à estruturação e ao gerenciamento da informação produzida por aplicações de captura e acesso. Associado aos componentes xInCA, o StRES provê um modelo de desenvolvimento de aplicações que minimiza a intervenção do desenvolvedor nas questões referentes ao armazenamento, recuperação, apresentação e extensão de conteúdo capturado.

O StRES é parte serviço e parte infra-estrutura. Sua porção infra-estrutural refere-se à integração com os componentes xInCA de captura e acesso. Essa integração se dá através da definição de um modelo de representação de informação comum que envolve os componentes de captura e acesso, a informação de contexto da aplicação e, por fim, a informação representativa de uma sessão capturada.

A abordagem utilizada nessa integração consiste na definição de um esquema XML para cada componente xInCA, outro esquema para modelar as informações de contexto da aplicação, e também um esquema de sessão, responsável por agrupar todos os demais. Utilizando um mecanismo de *data binding*, os esquemas são compilados gerando classes responsáveis por encapsular as informações produzidas e intercambiadas pelas aplicações.

O StRES provê uma entidade infra-estrutural importante denominada *Transducer*. Utilizando as facilidades de comunicação providas pela InCA, o *Transducer* recebe os objetos produzidos pelos componentes xInCA de captura em tempo real. Sua função é agrupar esses objetos em uma estrutura de dados chamada objeto de sessão. A partir desse objeto, gera-se, ao final da sessão, um documento XML representativo do conteúdo capturado.

Utilizando as informações de contexto da aplicação, o StRES realiza o armazenamento automático do documento produzido, e provê facilidades para que ele seja visualizado na *Web* segundos após sua geração.

Além de facilidades de apresentação, o StRES provê um serviço de consulta baseado em informações de contexto e facilidades para extensão de conteúdo capturado.

O StRES integra diversos resultados de pesquisa obtidos no contexto desta tese de doutorado. O Capítulo 5 é todo dedicado à discussão das particularidades funcionais e arquiteturais do StRES.

3.5.2 *Autore*

No sentido de contemplar as necessidades de reuso comum nas aplicações voltadas para o ambiente de sala de aula, desenvolveu-se o Autore(Sante et al. 2004), cuja função é permitir o reuso de material nas sessões de captura.

Abowd et al. (1996) estruturam o problema de captura e acesso em 4 fases: pré-produção, gravação ao vivo, pós-produção e acesso. O Autore cuida da primeira dessas fases, uma vez que as três últimas são suportadas pela infraestrutura provida.

A fase de pré-produção está relacionada à preparação da sessão ao vivo, e requer a produção de qualquer artefato que venha a ser utilizado durante a experiência. No caso do Autore, essa preparação refere-se ao conteúdo das aulas, representado, em geral, na forma de um conjunto de *slides*.

O Autore provê a funcionalidade de preparação utilizando um conjunto de formulários HTML que permitem associar informações de contexto à aula, tais como título e curso, além de possibilitar vincular um conjunto de *slides* a aula. Esse conjunto de *slides* pode ser completamente novo ou então pode ser reaproveitado a partir de aulas já preparadas, conforme ilustra a Figura 3.4.

O reuso de conteúdo pré-existente se dá de duas formas: é possível aproveitar os *slides* carregados na preparação de outras aulas, ou *slides* captura-

Figura 3.4: Formulário do Autore para reuso de conteúdo.

dos, que são aqueles que já passaram pelo processo de captura e, em geral, possuem anotações, chamadas de *strokes*.

Nota-se portanto, que um conjunto de *slides* pode ser carregado uma única vez e reutilizado em diversas aulas. É possível também fazer o uso incremental de um conjunto de *slides*, revisitando *slides* e associando novas informações àquelas já existentes.

3.5.3 LinkDigger

Explorando técnicas de LSI — *Latent Semantic Indexing* — Camacho-Guerrero et al. (2004) desenvolveram o LinkDigger³, sistema capaz de identificar relações envolvendo conteúdo textual entre diferentes hipertextos.

Entre as funcionalidades importantes do LinkDigger destaca-se sua capacidade de analisar o conteúdo de repositórios *Web* e estabelecer eventuais relações entre os documentos desses repositórios, as quais são expressas por meio de hiperligações.

Dessa forma o LinkDigger é capaz de descobrir similaridades entre diferentes documentos e permitir a navegação entre aqueles que apresentam similaridades. Essa funcionalidade é bastante útil na exploração de conteúdo capturado, uma vez que permite descobrir, por exemplo, assuntos relacionados

³Trabalho desenvolvido no contexto de um projeto de mestrado e outro de doutorado, ambos relacionados ao Projeto InCA-SERVE

em aulas de disciplinas diferentes.

Experimentos utilizando o *LinkDigger* em um repositório contendo 1833 sessões capturadas, referentes a aulas presenciais ministradas entre 1995 e 2001 no Georgia Institute of Technology⁴, demonstraram a efetividade do serviço de ligações ao criar 822 hiperligações entre os documentos do repositório (Macedo, Baldochi, Camacho-Guerrero, Cattelan, and Pimentel 2005).

3.6 Considerações Finais

O Projeto InCA-SERVE representou um esforço no sentido de tratar os diversos problemas associados à ausência de suporte adequado ao desenvolvimento de aplicações de captura e acesso. Essa ausência tem resultado em soluções *ad hoc*, que são caracterizadas pela complexidade de implementação e pela dificuldade de evolução e reuso (Truong and Abowd 2004).

Fornecendo abstrações de comunicação, funcionalidades e gerenciamento de conteúdo capturado, os esforços de investigação feitos no contexto do projeto resultaram na disponibilização de uma infra-estrutura capaz de elevar o grau de abstração necessário à construção de aplicações de captura e acesso. Questões de reuso e evolução são tratadas na própria infra-estrutura, através de uma arquitetura extensível baseada em componentes de software.

Os resultados obtidos no contexto dessa tese provêm funcionalidades importantes associadas à automatização do processo de geração e armazenamento de documentos que representam sessões capturadas. Facilidades para armazenamento, recuperação e apresentação de conteúdo capturado somam-se a essas funcionalidades provendo o serviço StRES.

As abstrações providas conjuntamente pela InCA, pela xInCA e pelo StRES compõem a infra-estrutura InCA-SERVE, cuja utilização permite minimizar o esforço de programação envolvido na construção de aplicações de captura e acesso.

⁴Experimento realizado durante estágio sanduíche dos 3 primeiros autores do trabalho.

Modelagem de Informação de Contexto para Aplicações de Captura e Acesso

4.1 *Considerações Iniciais*

Conforme já salientado no Capítulo 2, as três grandes áreas nas quais se subdivide a Computação Ubíqua não são estanques. Uma aplicação de captura e acesso apresenta, em geral, interfaces naturais e processa informação de contexto. A representação, o armazenamento e o processamento de informação de contexto são problemas em aberto e têm sido objeto de investigação por diferentes grupos de pesquisa ao redor do mundo (Coutaz et al. 2005).

Aplicações de captura e acesso possuem a particularidade de explorar informações de contexto em três momentos distintos: antes da sessão ao vivo, quando o material de apoio é preparado — slides, por exemplo — e associado à sessão, durante a captura e, posteriormente, quando se deseja recuperar e apresentar conteúdo capturado (Pimentel et al. 2001). O pós-processamento de dados resultantes de atividades de captura está fortemente associado a informações de contexto específicas de cada aplicação. Por exemplo, a apresentação de um conjunto de aulas deverá ocorrer, preferencialmente, no contexto de uma disciplina. Por outro lado, um conjunto de reuniões capturadas deve-

rá ocorrer, de preferência, no contexto de um projeto. Assim, fica evidente a necessidade de *capturar* as informações de contexto do domínio da aplicação e utilizá-las nas operações de acesso.

No sentido de possibilitar a utilização dessas informações a fim de facilitar operações de recuperação, apresentação e extensão dos dados capturados, investigou-se um mecanismo capaz de extrair as informações de contexto a partir da modelagem UML das aplicações. Essa abordagem visa permitir a modelagem de informações de contexto utilizando uma linguagem padrão, largamente conhecida pelos desenvolvedores de aplicações.

Conforme discutido na Seção 3.5.1, as aplicações de captura e acesso desenvolvidas com o suporte da infra-estrutura InCA-SERVE são modeladas por meio de esquemas XML. Um desses esquemas tem a função de modelar as informações de contexto da aplicação com base em quatro das cinco dimensões de contexto discutidas na Seção 2.4. Assim, cada documento descreve o *que* está sendo capturado (*what*), *quem* são as pessoas envolvidas na sessão de captura (*who*), *onde* (*where*) e *quando* (*when*) a sessão ocorreu. A dimensão *porque* (*why*) não é considerada, uma vez que refere-se à uma informação subjetiva que não pode ser diretamente obtida da modelagem UML das aplicações.

De maneira geral, as informações associadas às dimensões de contexto de uma dada aplicação de captura e acesso constituem uma aplicação à parte, independente da aplicação de captura e acesso propriamente dita. Um bom exemplo desse fato ocorre com as aplicações de captura voltadas para o ambiente de sala de aula. As informações de contexto processadas por essas aplicações, as quais envolvem nomes de professores e alunos para a dimensão *quem*, nomes de cursos para a dimensão *o que* (aula de Hipermídia, por exemplo), identificação de salas para a dimensão *onde* e de semestres para a dimensão *quando*, são mantidas em sistemas acadêmicos. De maneira semelhante, aplicações de captura de reuniões de trabalho utilizam informações de contexto mantidas em sistemas corporativos.

Nos casos em que um sistema legado não se encontra disponível, é necessário implementar uma aplicação para construir e gerenciar o repositório de informações associadas às dimensões de contexto utilizadas pelas aplicações de captura e acesso. O Autore, apresentado na Seção 3.5.2, é um exemplo de aplicação que gerencia esse tipo de base de dados.

Dessa forma, observou-se a oportunidade de explorar a modelagem UML desses sistemas na definição de uma metodologia capaz de produzir o esquema de contexto da aplicação de captura e acesso de maneira semi-auto-

mática. Essa metodologia, denominada *Contextractor*, explora os mecanismos de extensão da linguagem XML a fim de permitir a associação de dimensões de contexto às entidades da modelagem UML, gerando um modelo estendido do sistema legado. A representação XMI desse modelo é então utilizada em um conjunto de transformações XSLT que produzem como resultado o esquema XML responsável por modelar as informações de contexto da aplicação de captura e acesso.

Além de permitir a geração do esquema XML, a metodologia proposta também define uma linguagem XML para a consulta dos documentos resultantes. Com base nos esquemas produzidos, implementou-se um mecanismo capaz de processar consultas contextuais elaboradas na linguagem definida, o que demonstra a importância das informações de contexto no pós-processamento da informação capturada.

O *Contextractor* explora ferramentas publicamente disponíveis, tais como processadores XSLT, para apoiar os procedimentos que compõem a metodologia proposta. Outra ferramenta importante utilizada no suporte à metodologia foi a ArgoUML, a qual recebeu novos elementos de interface que permitem a associação de dimensões de contexto a entidades presentes no modelo da aplicação.

Este capítulo apresenta a metodologia *Contextractor*, detalhando seus passos por meio de um exemplo que envolve a modelagem de informações de contexto para uma aplicação voltada para a captura de atividades em sala de aula. Inicialmente, a Seção 4.2 discute a obtenção de informações de contexto a partir de sistemas legados. A seguir, a Seção 4.3 apresenta o mecanismo que permite o mapeamento de informações de contexto a partir da modelagem UML desses sistemas. A Seção 4.4 aborda as transformações XSLT que permitem a geração dos esquemas XML que modelam as informações de contexto obtidas a partir do mecanismo de mapeamento. A Seção 4.5 apresenta a linguagem CQL — *Context Query Language* — definida no processo de transformação XSLT. A Seção 4.6 apresenta trabalhos relacionados à modelagem de informações de contexto. A Seção 4.7 discute os problemas e limitações da metodologia proposta. Finalmente, a Seção 4.8 apresenta as considerações finais do capítulo.

4.2 Informação de Contexto e Sistemas Legados

A construção de aplicações de captura e acesso ao longo dos últimos três anos permitiu observar que uma parte significativa das informações de con-

texto relevantes ao processamento de conteúdo capturado encontram-se em sistemas legados. Dessa forma, existe a oportunidade de explorar a modelagem desses sistemas na definição das informações de contexto utilizadas por aplicações de captura e acesso.

O *Contextractor* explora essa oportunidade sem, no entanto, se prender unicamente a essa abordagem, o que impediria seu uso na construção de aplicações que não contam com o apoio desses sistemas.

A solução proposta para contemplar os dois casos consiste em permitir a criação de um modelo de armazenamento subjacente a partir da modelagem UML da aplicação. Dessa forma, se o sistema legado existe, seu modelo UML é utilizado. Caso contrário, o *Contextractor* permite definir um modelo de armazenamento para o modelo UML definido a fim de atender as necessidades da aplicação de captura e acesso.

Por questões de simplificação, o modelo explorado na obtenção das informações de contexto será chamado modelo legado, independentemente da real existência de um sistema legado. A Seção 4.4.2 discute como proceder caso seja necessário criar esse sistema.

A Figura 4.1 mostra a modelagem UML de um sistema acadêmico já estendido de forma a permitir a utilização do *Contextractor*. Duas classes, ditas externas, foram adicionadas ao modelo. A função dessas classes é qualificar as sessões capturadas de acordo com o modelo legado, o que permite definir a dimensão contextual *o que (what)* de maneira automática.

A metodologia *Contextractor* define que as classes externas devem compor um relacionamento de generalização no qual a classe pai deve ser chamada *Sessão* e a classe filha deve ser uma especialização de *Sessão*, podendo ser chamada *Aula*, *Reunião*, *Seminário*, *Festa de Aniversário*, enfim, qualquer denominação capaz de refletir o que será capturado pela aplicação. A classe filha, por sua vez, deve se relacionar com alguma classe do modelo legado por meio de um relacionamento de composição. No caso do exemplo mostrado na Figura 4.1, é possível observar que a classe externa *Aula* participa de um relacionamento de composição com a classe interna *Curso*. Por meio desse relacionamento será possível definir a dimensão *what* como uma aula de um determinado curso.

Figura 4.1: Diagrama de classes para uma aplicação típica de captura de aulas presenciais.

4.3 Mapeamento das Informações de Contexto

A definição do modelo estendido é o primeiro passo no sentido de permitir a extração das informações de contexto a partir do modelo legado. O passo seguinte consiste em prover um mecanismo que permita associar dimensões de contexto às entidades presentes no modelo UML estendido. Além disso, é necessário também qualificar explicitamente as classes externas do modelo.

Para alcançar esses objetivos, o *Context extractor* utiliza os mecanismos de extensão da UML, que incluem o uso de estereótipos (*stereotypes*), pares atributo-valor (*tagged values*), restrições (*constraints*) e comentários (*comments*). O metamodelo UML provê tais mecanismos de extensão a fim de permitir a modelagem de aplicações para domínios específicos, como é o caso das aplicações de captura e acesso.

Para a qualificação das classes externas, optou-se por utilizar estereótipos, uma vez que esses ficam visíveis no diagrama de classes. Na Figura 4.1 é possível notar o uso do estereótipo «*External*» para as classes *Sessão* e *Aula*. Por outro lado, a associação das dimensões contextuais aos atributos foi feita por meio dos chamados *tagged-values*.

A criação de um *tagged-value* envolve selecionar um atributo de uma determinada classe, criar a *tag* para esse atributo e associar um valor a essa *tag*. Por exemplo, é possível selecionar o atributo *sala* da classe *Curso* e definir a

tag context-dimension com o valor *where*. O procedimento discutido, apesar de ser suficiente para a associação de informação de contexto ao modelo UML, impõe ao desenvolvedor da aplicação a necessidade de conhecer o mecanismo de extensão UML. Mais ainda, esse mecanismo é implementado de diferentes maneiras nas várias ferramentas de modelagem UML disponíveis.

Sendo assim, visando facilitar o procedimento de associação de informações de contexto, optou-se por estender a ferramenta de modelagem ArgoUML (Tigris 2003), de código aberto, inserindo no *menu popup* das classes um *check-box* com o estereótipo "External". Outra extensão feita no ArgoUML foi a inclusão de um conjunto de *radio buttons* contento as dimensões de contexto. Dessa forma, ao selecionar um atributo, o desenvolvedor pode associá-lo a uma dimensão de contexto com um simples clique de *mouse*. A Figura 4.2 ilustra essa extensão.

Figura 4.2: Extensão feita na interface gráfica da aplicação ArgoUML.

A utilização do conjunto de *radio buttons* associados às dimensões de contexto cria automaticamente um *tagged-value* para o atributo selecionado com o nome *context-dimension* e o valor do botão selecionado. No caso da Figura 4.2, a seleção do botão *where* causa a criação do *tagged value* citado com o valor *where* para o atributo *sala*.

As extensões foram implementadas de tal forma que o estereótipo *External*

ou o *tagged-value context-dimension* só aparecem no documento XMI resultante se forem explicitamente utilizados durante o processo de modelagem. Dessa forma, as extensões implementadas não impactam a modelagem de aplicações convencionais, que não utilizam informação de contexto.

4.3.1 Geração do Documento XMI

A linguagem XMI — XML Metadata Interchange (OMG 2003) — foi definida com o objetivo de permitir o intercâmbio de diagrama de classes UML entre diferentes ferramentas. Dessa forma, a linguagem XMI determina a estrutura de um documento XML capaz de representar um modelo UML.

Apesar da versão mais atual da linguagem XMI ser a 2.0, diversas ferramentas de modelagem suportam apenas a versão anterior (1.2). É esse o caso do ArgoUML. Por esse motivo, o resultado da modelagem de uma aplicação de captura e acesso é um documento XMI no formato 1.2.

O documento XMI descreve todas as particularidades do modelo, incluindo as extensões feitas por meio de estereótipos e *tagged values*. Com base no documento XMI exportado pelo ArgoUML é realizada uma sequência de transformações XSLT que culmina na definição de um conjunto de esquemas XML usados para modelar as informações de contexto de aplicações de captura e acesso, bem como para formalizar uma linguagem de consulta que permite recuperar conteúdo capturado com base nas dimensões de contexto modeladas.

4.4 Visão Geral do Processo de Transformação

A visão geral do processo de transformação que resulta na geração dos esquemas que possibilitam o processamento da informação capturada com base em suas informações de contexto é mostrada na Figura 4.3.

Figura 4.3: Sequência de transformações XSLT.

Na Figura 4.3, documentos maiores representam conteúdo XML e os menores são as folhas de estilo XSLT responsáveis pelas transformações.

O processo de transformação se inicia com a geração do modelo conceitual da aplicação (*CM*), o qual é obtido filtrando-se o documento XMI exportado pelo ArgoUML. Essa filtragem tem como objetivo eliminar a verbosidade do documento XMI, expondo o modelo conceitual de maneira mais simplificada, o que facilita as transformações posteriores.

O próximo passo do processo de transformação consiste em processar o documento *CM* a fim de facilitar o mapeamento do modelo conceitual para um modelo de armazenamento. Nessa fase são processados os relacionamentos entre as classes conceituais, gerando o modelo conceitual modificado, representado na Figura 4.3 pelo documento *CM-MOD*.

A geração do modelo conceitual modificado é feita em três passos, utilizando folhas de estilo distintas. A primeira processa os relacionamentos de generalização, gerando a inclusão de atributos identificadores nas classes filhas, quando necessário. A segunda faz ajustes nas dimensões de contexto dos atributos presentes nas classes envolvidas nos relacionamentos de generalização. Já a terceira processa os demais relacionamentos, gerando novos atributos de acordo com as multiplicidades envolvidas.

A partir do modelo conceitual modificado é possível construir os esquemas XML responsáveis pela modelagem das informações de contexto das aplicações de captura. O documento *CONTEXT-INFO.XSD* modela os dados de contexto que estarão presentes no documento de sessão (resultante da captura de uma sessão ao vivo). Já os documentos *CONTEXT-QUERY.XSD* e *VALID-QUERY.XSD* servem de suporte para o mecanismo que processa consultas contextuais.

As etapas do processo de transformação são apresentadas de maneira detalhada nas Seções 4.4.1 a 4.4.3.

4.4.1 Geração do Modelo Conceitual

Para se obter o modelo conceitual da aplicação, utiliza-se uma folha de estilo denominada *CM.XSL*, a qual filtra a verbosidade do documento XMI expondo as classes do modelo conceitual com seus respectivos atributos, bem como os relacionamentos existentes. O trecho de código a seguir ilustra o documento resultante dessa filtragem (*CM.XML*) para uma aplicação de captura de aulas presenciais.


```

<conceptual-model>
  <conceptual-class name="Sessao" external="true">
    <attribute name="id" type="integer" context-dimension="none" />
    <attribute name="data" type="date" context-dimension="when" />
  </conceptual-class>
  <conceptual-class name="Aula" external="true">
    <attribute name="titulo" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Aluno" external="false">
    <attribute name="id" type="integer" context-dimension="who" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Instructor" external="false">
    <attribute name="titulacao" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Curso" external="false">
    <attribute name="id" type="string" context-dimension="none" />
    <attribute name="nome" type="string" context-dimension="none" />
    <attribute name="semestre" type="string" context-dimension="when" />
    <attribute name="sala" type="string" context-dimension="where" />
  </conceptual-class>
  <conceptual-class name="Funcionario" external="false">
    <attribute name="id" type="integer" context-dimension="who" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <relationship name="lociona" type="association">
    <from conceptual-class="Curso" multiplicity="*" />
    <to conceptual-class="Instructor" multiplicity="1" />
  </relationship>
  <relationship name="tem" type="composition">
    <from conceptual-class="Curso" multiplicity="1" />
    <to conceptual-class="Aula" multiplicity="*" />
  </relationship>
  <relationship name="matricula" type="association">
    <from conceptual-class="Curso" multiplicity="*" />
    <to conceptual-class="Aluno" multiplicity="*" />
  </relationship>
  <relationship name="sessao-aula" type="generalization">
    <child conceptual-class="Aula" />
    <parent conceptual-class="Sessao" />
  </relationship>
  <relationship name="funcionario-instructor" type="generalization">
    <child conceptual-class="Instructor" />
    <parent conceptual-class="Funcionario" />
  </relationship>
</conceptual-model>

```

Observando o documento gerado, nota-se que o estereótipo *External* foi mapeado para um atributo de mesmo nome do elemento *conceptual-class*. Já o *tagged-value context-dimension* tornou-se um atributo do elemento *attribute*.

O próximo passo na direção de gerar os esquemas XML com as informações de contexto envolve a análise dos relacionamentos, a fim de realizar modificações no modelo para que este possa ser diretamente mapeado para um modelo de armazenamento.

4.4.2 Geração do Modelo Conceitual Modificado

Para que o modelo conceitual possa ser diretamente mapeado para um modelo de dados subjacente é necessário realizar um conjunto de ajustes, os quais produzem o modelo conceitual modificado. A geração desse modelo ocorre em três passos, conforme ilustra a Figura 4.4.

Figura 4.4: Geração do modelo conceitual modificado.

A primeira transformação trata os relacionamentos de generalização, fazendo com que as classes filho recebam os atributos identificadores das classes pai. Esse procedimento permite, por exemplo, mapear cada classe de uma generalização em uma tabela do modelo relacional. A etapa inicial da modificação do modelo conceitual é feita processando-se o documento (CM) por meio da folha de estilo CM-MOD1, resultando em um documento temporário denominado (CM-MOD-TEMP1), parcialmente exibido a seguir¹.

```

<conceptual-model>
...
<conceptual class name="Instrutor" external="false">
  <attribute name="titulacao" type="string" context-dimension="none" />
  <attribute name="idFuncionario" context-dimension="none" />
</conceptual-class>
...
<conceptual class name="Funcionario" external="false">
  <attribute name="id" type="integer" context-dimension="who" />
  <attribute name="nome" type="string" context-dimension="none" />
</conceptual-class>
...
</conceptual-model>

```

Para o exemplo em questão, a transformação acarreta mudanças apenas na classe *Instrutor*, a qual recebe o atributo identificador da classe pai. O outro relacionamento de generalização, que envolve as classes *Sessao* e *Aula*, foi desconsiderado porque ambas as classes envolvidas são externas. Para

¹Os documentos referentes a essa aplicação exemplo encontram-se exibidos na íntegra no Apêndice A

classes externas, o relacionamento de generalização fica explícito nos documentos XML em função do aninhamento dos elementos.

Observando o fragmento XML apresentado, nota-se que o atributo *idFuncionario* não herdou a dimensão de contexto *who* do atributo *id* da classe pai. Entretanto, ao analisar o diagrama UML mostrado na Figura 4.1, fica claro que a classe *Instrutor* deve ter um atributo com a dimensão *who*.

Esse fato traz à tona uma questão importante referente à representação das informações de contexto nos diagramas de classe UML: certas dimensões podem ser associadas a classes, enquanto outras podem somente ser associadas a atributos. Por exemplo, a classe *Aluno* pode ser associada à dimensão *who*, enquanto que apenas o atributo *semestre* da classe *Curso* pode ser associado à dimensão *when*.

No sentido de uniformizar o procedimento de associação de dimensões de contexto a elementos do diagrama de classes UML, optou-se por associar o contexto ao atributo identificador das instâncias de uma classe sempre que a informação de contexto estiver associada à classe. Dessa forma, o procedimento de associação de informações de contexto a elementos do diagrama UML segue um único padrão.

Essa decisão de projeto traz um inconveniente: classes filhas de relacionamentos de generalização não possuem atributos identificadores de suas instâncias. Por isso, a dimensão de contexto deve ser inicialmente atribuída ao identificador da classe pai e, posteriormente, movida para a classe filha durante o processo de geração do modelo conceitual modificado.

O segundo passo no processo de geração do modelo conceitual modificado serve justamente para fazer esse ajuste. Operando sobre o documento *CM-MOD-TEMP1*, a segunda folha de estilo, denominada *CM-MOD2*, move o qualificador de contexto do atributo identificador da classe pai para o atributo qualificador da classe filha. Em todo o processo de transformação, esse é a única etapa que requer a passagem de um parâmetro, o qual deve identificar a classe pai a partir da qual deseja-se mover o qualificador de contexto. Isso é necessário porque nem sempre será necessário mover o qualificador. No caso em que não se deseja mover o qualificador, nenhum parâmetro é passado e o documento resultante, *CM-MOD-TEMP2*, fica idêntico a *CM-MOD-TEMP1*.

Outra observação que deve ser feita no que se refere ao procedimento proposto é que o processo atua sobre um relacionamento de generalização por vez. Dessa forma, o procedimento deve ser repetido com parâmetros diferentes tantas vezes quantas forem necessárias até que todas as relações de

generalização sejam tratadas.

Finalmente, resta tratar os demais relacionamentos, o que é feito no terceiro e último passo, no qual o documento *CM-MOD-TEMP2* é processado pela folha de estilo *CM-MOD3*. A multiplicidade dos relacionamentos de associação e composição influencia o mapeamento das classes conceituais em estruturas de armazenamento, tais como tabelas do modelo relacional. O processamento de relacionamentos com multiplicidade *muitos para 1* causa a inclusão de atributos em certas classes, enquanto relacionamentos *muitos para muitos* resultam na criação de relações contendo os atributos identificadores das classes envolvidas.

O código a seguir ilustra trechos do modelo conceitual modificado (*CM-MOD.XML*), obtido após a terceira transformação. Nele é possível observar que a classe conceitual *Curso* recebeu o atributo *idFuncionario* em função da multiplicidade *muitos para 1* do relacionamento *leciona*. Por outro lado, os atributos *idCurso* e *idAluno* foram acrescentados ao relacionamento *matricula*, já que este apresenta uma multiplicidade *muitos para muitos* envolvendo as classes *Curso* e *Aluno*. Tal relacionamento deverá ser mapeado para uma relação (tabela) contendo os atributos especificados.

```
<conceptual-model>
  <conceptual-class name="Curso" external="false">
    <attribute name="codigo" type="string" context-dimension="none" />
    <attribute name="nome" type="string" context-dimension="none" />
    <attribute name="semestre" type="string" context-dimension="when" />
    <attribute name="sala" type="string" context-dimension="where" />
    <attribute name="idFuncionario" type="" context-dimension="none" />
  </conceptual-class>
  ...
  <relationship name="matricula" type="association">
    <from conceptual-class="Curso" multiplicity="*" />
    <to conceptual-class="Aluno" multiplicity="*" />
    <attribute name="idCurso" type="string" context-dimension="none" />
    <attribute name="idAluno" type="integer" context-dimension="none" />
  </relationship>
  ...
</conceptual-model>
```

O documento *CM-MOD* permite mapear o modelo conceitual da aplicação para um modelo de armazenamento subjacente. Esse procedimento é importante para os casos em que não é possível extrair as informações de um sistema legado. Nesse caso, o documento *CM-MOD* é utilizado pelo sistema *wView*, apresentado na Seção 4.6.1, o qual define *scripts* responsáveis por criar o espaço de informação para a aplicação modelada.

O documento *CM-MOD* provê as informações necessárias à geração dos esquemas XML que modelam as informações de contexto dos documentos de

sessão e formalizam uma linguagem de consulta capaz de recuperar documentos de captura com base nas dimensões de contexto modeladas.

4.4.3 Geração dos Esquemas XML

A etapa final do processo de transformação XSLT envolve a geração de três esquemas XML, conforme ilustra a Figura 4.5. O primeiro, denominado *CONTEXT-INFO.XSD*, modela as informações de contexto que deverão aparecer no documento de sessão. Os outros dois, *CONTEXT-QUERY.XSD* e *FLWOR-QUERY.XSD*, são utilizados no processo de validação de documentos contendo consultas baseadas nas dimensões de contexto modeladas. Tais esquemas têm um papel importante no mecanismo de processamento de consultas contextuais apresentado na Seção 4.5.

Figura 4.5: Geração dos esquemas XML.

As informações adicionadas ao modelo UML por meio das extensões discutidas na Seção 4.3 regem as transformações que produzem os esquemas XML. Uma característica importante associada às extensões propostas refere-se à definição das classes como externas e internas. Observando a Figura 4.2 é possível notar que os atributos de uma classe podem possuir uma entre três dimensões: *who*, *where* e *when*. Nota-se, portanto, que não é possível qualificar um atributo com a dimensão *what*.

Isso ocorre porque a dimensão *what* pode ser diretamente obtida do modelo UML. Uma característica inerente às aplicações de captura e acesso asso-

ciadas a sistemas legados é o fato dessas aplicações possuírem um relacionamento de composição unindo as classes externas e internas do modelo. Este relacionamento é o ponto de junção do modelo, já que se for rompido, dois diagramas de classes distintos são formados. São exatamente as duas classes do ponto de junção que formam a dimensão *what*. Para o exemplo das aulas presenciais, nota-se, observando o diagrama de classes da aplicação, que as classes que definem a dimensão *what* são *Aula* e *Curso*. De fato, o evento que se pretende capturar é uma aula, a qual está associada a um curso que a caracteriza.

Dessa forma, o ponto de partida para a geração do documento *CONTEXT-INFO.XSD* é a definição do elemento *what*. A folha de estilo responsável por esse processamento procura, inicialmente, por um relacionamento de composição no modelo conceitual modificado que envolva uma classe externa e outra interna. Tendo encontrado o relacionamento que satisfaz essa condição — só existe um — o elemento *what* é definido como um tipo complexo contendo a classe conceitual externa, juntamente com seus atributos, aninhando a classe conceitual interna que, por, sua vez, contém o atributo identificador das instâncias da classe.

Já para os demais elementos do esquema, o processamento é orientado pelo qualificador *context-dimension* presente em cada atributo de cada classe conceitual. Assim, para cada dimensão de contexto, define-se o elemento correspondente à referida dimensão como um tipo complexo contendo elementos correspondentes às classes conceituais que possuem um atributo igual à dimensão processada.

Tomando a dimensão *where* como exemplo, observa-se que o único atributo qualificado com essa dimensão, no caso do exemplo em questão, é *sala*, presente na classe conceitual *Curso*. Dessa forma, o elemento *where* é modelado como um tipo complexo contendo o elemento *Curso*, que, por sua vez, possui o atributo *sala*, definido como uma cadeia de caracteres (*string*). O XML resultante do processo de transformação descrito é mostrado parcialmente a seguir.

```
<xsd:schema ...>
  <xsd:complexType name="contextType">
    <xsd:sequence>
      <xsd:element name="what" type="whatType" />
      <xsd:element name="who" type="whoType" />
      <xsd:element name="where" type="whereType" />
      <xsd:element name="when" type="whenType" />
    </xsd:sequence>
  </xsd:complexType>
```

```

<xsd:complexType name="whatType">
  <xsd:sequence>
    <xsd:element name="Aula">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="Curso">
            <xsd:complexType>
              <xsd:attribute name="id" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="titulo" type="xsd:string" use="required" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
...
<xsd:complexType name="whereType">
  <xsd:sequence>
    <xsd:element name="Curso">
      <xsd:complexType>
        <xsd:attribute name="sala" type="xsd:string" use="required" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
...
</xsd:schema>

```

Uma característica importante do processo de transformação que gera o documento *CONTEXT-INFO.XSD* é a consideração da multiplicidade das relações envolvidas.

Analisando a Figura 4.1 nota-se que vários alunos podem assistir a uma aula, enquanto apenas um instrutor ministra uma aula. Na geração do esquema XML tal fato deve ser considerado. A forma utilizada para ajustar o número de ocorrências de cada elemento em função da multiplicidade das relações leva em consideração os relacionamentos entre a classe interna do ponto de junção e as demais classes internas. Quando esses relacionamentos apresentam multiplicidades *muitos para muitos* ou *muitos para um*, o elemento associado à classe que se relaciona com a classe do ponto de junção deve ser modelado com *maxOccurs="unbounded"*, significando que múltiplas ocorrências do elemento poderão aparecer nos documentos de instância. O trecho de código a seguir mostra a definição do elemento *who*, no qual a multiplicidade da relação *matricula* resultou na definição do elemento *Aluno* com um número de ocorrências indefinido.

```

...
<xsd:complexType name="whoType">
  <xsd:sequence>
    <xsd:element name="Instrutor">
      <xsd:complexType>
        <xsd:attribute name="idFuncionario" type="xsd:" use="required" />
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Aluno" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="id" type="xsd:integer" use="required" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
...
</xsd:schema>

```

O esquema XML produzido nesse processo de transformação é capaz de validar documentos como os mostrados no trecho de código a seguir. Os documentos de sessão produzidos pela aplicação de captura conterão um fragmento de código XML com as características exibidas. Vale notar que os elementos e atributos que definem cada dimensão de contexto foram todos obtidos da modelagem UML da aplicação.

```

<context-info>
  <what>
    <Aula titulo="Ponteiros">
      <Curso id="SCE-101" />
    </Aula>
  </what>
  <who>
    <Instrutor idFuncionario="10" />
    <Aluno id="33" />
    <Aluno id="12" />
    <Aluno id="57" />
    <Aluno id="26" />
  </who>
  <where>
    <Curso sala="1-002"/>
  </where>
  <when>
    <Sessao data="2004-11-20"/>
    <Curso semestre="2o_2004" />
  </when>
</context-info>

```

A observação desse código evidencia diversas características sobre a sessão capturada: trata-se de uma *aula* cujo título é "*Ponteiros*", ministrada no Curso "*SCE-101*", pelo Instrutor que possui o código funcional *10*. A aula foi assistida pelos alunos cujos números de identificação são *33*, *12*, *57* e *26*. A aula ocorreu no *segundo semestre de 2004*, mais precisamente no dia *20 de novembro*, na sala *1-002*.

Observando a Figura 4.5 nota-se que dois outros XSDs são produzidos nessa etapa do processo de transformação. Ambos são usados no mecanismo de processamento de consultas baseadas nas dimensões de contexto.

O esquema *CONTEXT-QUERY* define uma linguagem de consulta, denominada CQL — *Context Query Language*. Essa linguagem determina o formato da consulta, ou seja, sua estrutura e a tipação de dados esperada. Sua função é validar consultas, identificando aquelas que estão aptas a serem submetidas para processamento. Ao elaborar uma consulta, o desenvolvedor pode optar entre qualquer combinação de dimensões de contexto: pode usar apenas uma dimensão, ou qualquer combinação que envolva duas ou mais dimensões. Além disso, na especificação de uma dimensão o desenvolvedor não fica preso apenas ao atributo de classe associado àquela dimensão. Por exemplo, na elaboração de uma consulta envolvendo a dimensão *who*, o usuário pode especificar o nome de um aluno e não somente seu atributo *id*.

Outra questão relevante associada à estruturação das expressões de consulta diz respeito aos conectores utilizados. O esquema XML deve permitir a especificação de dois ou mais elementos para cada dimensão de contexto a fim de permitir que o conector *OU* seja implementado. Dessa forma, uma expressão como a mostrada a seguir deve ser interpretada como: "obtenha as sessões capturadas nas salas 1-002 ou 1-005".

```
<where>
  <Curso sala="1-002"/>
  <Curso sala="1-005"/>
</where>
```

A expressão anterior deve utilizar o conector *OU* porque uma sessão ocorre em um único lugar, da mesma forma que possui um único instrutor — tais informações foram modeladas no diagrama UML. Por outro lado, quando se considera a entidade *Aluno*, é possível buscar por sessões nas quais *Pedro E/OU Antônio* participaram. Novamente, essa informação é obtida do diagrama UML, mais precisamente da multiplicidade da relação que envolve a classe *Aluno* com a classe interna do ponto de junção (*Curso*).

Assim como a multiplicidade das relações envolvendo a classe interna do ponto de junção e as demais classes internas influenciou a geração do documento *CONTEXT-INFO.XSD*, essa mesma questão determina os tipos de conectores passíveis de utilização. Classes envolvidas em relacionamento *muitos para muitos* ou *muitos para um* com a classe interna do ponto de junção são modeladas com o atributo opcional *conector*, o qual pode conter o valor *AND* ou então o valor *OR*, sendo esse último o valor padrão. Se o atributo *conector*

for omitido, entende-se que o conector *OR* deve ser utilizado. Dessa forma, para recuperar as aulas nas quais os alunos *Pedro* E *Antônio* participaram deve-se utilizar o fragmento de código mostrado a seguir.

```
<who>
  <Aluno nome="Pedro" conector="AND"/>
  <Aluno nome="Antônio" conector="AND"/>
</who>
```

Um último atributo adicionado a cada elemento mapeado a partir do modelo conceitual é o modificador *NOT*, incorporando, assim, a operação de negação à linguagem CQL.

É possível resumir as diferenças entre os esquemas *CONTEXT-INFO* e *CONTEXT-QUERY* da seguinte forma: para uma determinada dimensão, o segundo esquema permite, além do atributo associado àquela dimensão, todos os demais atributos da mesma classe que não estejam associados a nenhuma dimensão de contexto. Além disso, o segundo esquema permite a repetição de elementos, que são tratados com o conector *OU* por *default*, a não ser que o atributo opcional *conector* determine o contrário. Finalmente, o esquema *CONTEXT-QUERY* define todos os elementos com *minOccurs="0"*, possibilitando que as consultas sejam definidas com qualquer combinação de elementos. Tais regras flexibilizam a formação das expressões de consulta, resultando em uma linguagem com alto grau de expressividade.

Falta, por fim, discutir a geração do esquema denominado *FLWOR-QUERY*. Como o próprio nome evidencia, esse esquema determina se uma expressão de consulta contextual está pronta para ser transformada em uma consulta *XQuery* (Boag et al. 2004). *FLWOR* é como se denomina expressões de consulta na linguagem *XQuery*. Como será apresentado na Seção 4.5.2, o processador de consultas contextuais tem como função transformar uma expressão em conformidade com o esquema *CONTEXT-QUERY* em outra expressão que esteja em conformidade com o esquema *FLWOR-QUERY*. Expressões em conformidade com o segundo esquema podem ser diretamente mapeadas em expressões *FLWOR* e submetidas ao repositório de documentos XML.

O documento *FLWOR-QUERY* é praticamente idêntico ao primeiro esquema, *CONTEXT-INFO*, com duas exceções: a primeira é que todos os elementos apresentam *minOccurs="0"*, possibilitando que consultas com quaisquer combinação de dimensões contextuais sejam consideradas válidas. A outra diferença diz respeito ao atributo *conector*, que, assim como no documento *CONTEXT-QUERY*, nesse caso também aparece junto aos elementos que apresentem multiplicidades *muitos para muitos* ou *muitos para um* em relacionamentos que envolvem a classe interna do ponto de junção.

A geração do esquema *FLWOR-QUERY* conclui o procedimento de transformações que tem por objetivo produzir os esquemas que modelam as informações de contexto produzidas e manipuladas pela aplicação de captura e acesso. O documento *CONTEXT-INFO* é utilizado no mecanismo de *data binding*, descrito no Capítulo 5. Já os outros dois últimos esquemas gerados são utilizados pelo mecanismo que processa consultas elaboradas com base nas quatro dimensões de contexto. Tal mecanismo é assunto da próxima seção.

4.5 Processamento de Consultas CQL

A linguagem CQL constitui um esforço no sentido de prover um mecanismo capaz de recuperar conteúdo com base nas dimensões de contexto comumente exploradas pelas aplicações de captura e acesso.

Apesar de existirem diferentes versões da linguagem para diferentes aplicações, a estrutura de um documento CQL é fixa — variam apenas os nomes dos elementos de marcação. Dessa forma, foi possível elaborar uma aplicação capaz de processar consultas CQL com base nos esquemas XML produzidos no processo de transformação.

Consultas CQL retornam referências a documentos XML cujas informações de contexto satisfazem às condições impostas nas expressões de consulta. Dessa forma, o processamento de uma consulta CQL sempre termina com a submissão de uma consulta ao repositório de documentos XML, o que significa que a consulta CQL deve ser mapeada para uma consulta XQuery.

Entretanto, nem sempre esse mapeamento é direto. Dependendo dos dados presentes na expressão de consulta, pode ser necessário submeter consultas inicialmente ao sistema legado a fim de preparar a expressão CQL para que ela possa ser mapeada para XQuery.

Esta seção descreve o mecanismo que integra o sistema legado ao repositório de documentos XML, possibilitando a resolução de consultas elaboradas na linguagem CQL. Dentre as informações necessárias ao funcionamento desse mecanismo, a organização do repositório de documentos XML é uma das mais relevantes.

4.5.1 Organização do Repositório de Documentos XML

O repositório de documentos resultantes de atividades de captura deve ser organizado de modo a facilitar o gerenciamento dos dados, sobretudo no que

se refere à apresentação das sessões. Possíveis organizações para o domínio de aulas presenciais consistem em estruturar as sessões capturadas (documentos XML) por semestre, e dentro de cada semestre, por disciplina. Pode-se também optar por organizar inicialmente por disciplina e, posteriormente por semestre.

Para o contexto de reuniões pode-se organizar as sessões capturadas por projeto e, dentro de cada projeto, criar coleções para os meses do ano, dentro das quais ficam armazenados os documentos de sessão.

Independentemente do domínio, a estrutura do repositório de documentos forma um caminho (*path*) composto por atributos obtidos das classes conceituais da aplicação. Assim, para o exemplo em uso, uma organização desejável para o repositório envolve as classes conceituais *Curso* e *Sessao*. Os atributos *semestre* e *nome* da classe *Curso* devem ser usados para nomear coleções do repositório, e o atributo *id* de *Sessao* deve ser usado para nomear o documento XML resultante de cada atividade de captura. Além disso, é necessário nomear a coleção raiz, de tal forma que um mesmo banco de dados possa conter diferentes repositórios, cada um representado por sua coleção raiz.

A fim de permitir a descrição da estrutura do repositório definiu-se uma linguagem XML semelhante àquela utilizada para o mapeamento das informações de contexto. Tal linguagem especifica um nome para a coleção raiz e, em seguida, um aninhamento de atributos e suas classes conceituais que definem o conjunto de coleções que irão formar o caminho até o documento XML no repositório. O último atributo do aninhamento refere-se ao identificador da sessão e será utilizado para nomear seu documento associado. O trecho de código a seguir ilustra uma instância do documento *REPOSITORY.XML* para a aplicação de aulas presenciais.

```
<repository>
  <root-collection name="!class">
    <conceptual-class name="Curso">
      <attribute name="semestre"/>
    </conceptual-class>
    <conceptual-class name="Curso">
      <attribute name="nome"/>
    </conceptual-class>
    <conceptual-class name="Sessao">
      <attribute name="id"/>
    </conceptual-class>
  </root-collection>
</repository>
```

Além de definir a estruturação do repositório de documentos XML, o documento *REPOSITORY.XML* é utilizado na elaboração de consultas ao repositório

e no mecanismo de apresentação de conteúdo capturado, discutido no Capítulo 5.

4.5.2 O processador CQL

O mecanismo que permite a resolução de consultas formuladas na linguagem CQL é denominado processador CQL. Sua função é permitir que uma expressão CQL possa ser transformada em uma expressão *FLWOR* e submetida ao repositório de documentos XML.

O processamento de uma consulta CQL é feita por meio de um mecanismo de validação baseado nos esquemas *CONTEXT-QUERY* e *FLWOR-QUERY*. O primeiro é utilizado para validar a expressão de consulta. Já o segundo tem a função de verificar se a consulta CQL pode ser diretamente mapeada para uma expressão *FLWOR*. A expressão mostrada a seguir é passível de mapeamento direto.

```
<context-info>
  <what>
    <Aula>
      <Curso id "10"/>
    </Aula>
  </what>
  <who>
    <Instructor idinstructor="12"/>
  </who>
</context-info>
```

Entretanto, se o atributo *id* do elemento *Aula* for substituído por *nome*, a expressão será válida de acordo com *CONTEXT-QUERY*, mas não estará em conformidade com o esquema *FLWOR-QUERY*, o que a impossibilita de ser diretamente mapeada em uma expressão *FLWOR*. Nesse caso, é necessário processar a consulta, transformando-a numa consulta válida pelo segundo esquema.

O programa que faz o processamento da expressão contextual funciona de maneira recursiva: a expressão de consulta é processada repetidamente até que seja válida. Assim, a cada vez que ocorre um erro de validação, o erro é tratado e uma nova expressão é gerada, a qual é submetida ao mesmo procedimento.

No caso do exemplo utilizado, considerando que o atributo *nome* apareça junto ao elemento *Curso* com o valor "Estruturas de Dados", o processo de validação terá sucesso com o primeiro esquema (*CONTEXT-QUERY*), mas falhará com o segundo. O programa trata a falha obtendo a informação ausente.

Em outras palavras, tomando como base o esquema *CONTEXT-FLWOR*, o programa elabora uma consulta SQL que obtém o valor do atributo esperado a partir do atributo que causou o erro de validação. Assim, a consulta SQL automaticamente elaborada a partir dos documentos em processamento seria a seguinte:

```
SELECT id from Curso WHERE name="Estruturas de Dados";
```

Com o resultado obtido é montada uma expressão de consulta com o atributo *id* substituindo o atributo *nome*. Tal expressão é então novamente submetida ao processo de validação, sendo, então, considerada válida.

A partir daí elabora-se a montagem da expressão *FLWOR* a ser submetida ao repositório de documentos. Na elaboração dessa expressão entra em cena o documento *REPOSITORY.XML*, discutido na Seção 4.5.1. Além de definir a organização interna do repositório, esse documento serve para otimizar consultas, uma vez que limita o escopo das consultas a coleções específicas, nas quais o documento procurado deve ser encontrado.

A expressão de caminho para a consulta contém atributos que podem ou não estar presentes na expressão CQL. No caso do exemplo, são necessários o *nome* e o *semestre* do curso. Ambos não fazem parte da expressão CQL. Entretanto, podem ser obtidos a partir do atributo *id* do elemento *Curso*. Se esse atributo não estivesse disponível, a consulta deveria ser submetida à coleção raiz do repositório, alcançando todos os documentos armazenados.

A partir da definição do caminho elabora-se uma expressão *FLWOR* concatenando as informações contidas na consulta CQL. Para o exemplo em questão, a expressão gerada seria a seguinte:

```
FOR $t in col(/iclass/2_2004/EstruturasDeDados)//Sessao WHERE  
$t//Curso[@id=10] AND $t//Course[@idInstructor=12] RETURN @nome
```

O atributo *nome* presente no elemento *Sessao* representa uma referência ao documento no contexto do repositório. Isso porque *nome* contém o caminho até o documento juntamente com o nome do próprio documento. Dessa forma, a expressão *FLWOR* gerada produz um conjunto de referências que permitem, num segundo passo, recuperar todos os documentos que atendem à consulta elaborada.

4.6 Trabalhos Relacionados

O processo de transformação desenvolvido foi inspirado no *wView*, sistema que automatiza a geração de aplicações Web por meio de sucessivas transformações XSLT. No sentido de explicitar as semelhanças e diferenças nos dois processos de transformação, esta seção traz uma descrição detalhada da abordagem *wView*. Em seguida, são apresentados e discutidos os trabalhos que enfocam a modelagem de informações de contexto encontrados na literatura.

4.6.1 *wView*

A abordagem *wView* (Andrade et al. 2004b; Andrade et al. 2004a) tem como objetivo automatizar a geração de aplicações Web. Para tanto, define um procedimento que inicia com a modelagem UML da aplicação e termina com a produção de uma aplicação Web interativa.

O desenvolvimento do *wView* foi norteado pelo princípio básico de que conteúdo, navegação e apresentação são conceitos independentes nas aplicações hipermídia, e, portanto, devem ser tratados independentemente durante o projeto das aplicações. Provendo especificações declarativas separadas para cada um desses aspectos, *wView* permite controlar de maneira explícita o formato dos nós de uma aplicação Web (estrutura e conteúdo), a navegação entre esses nós e, finalmente, a apresentação da informação associada a cada nó.

O processo de transformação *wView* necessita de apenas uma especificação: a modelagem UML da aplicação, exportada via XMI. O documento XMI serve de entrada para a *pipeline* de transformações XSLT, as quais produzem um conjunto de documentos que compõem a aplicação Web resultante.

A Figura 4.6 mostra uma visão geral do processo de transformação *wView*. A primeira transformação filtra o documento XMI explicitando o modelo conceitual da aplicação (CM). A seguir, o modelo navegacional (NM) é produzido a partir do modelo conceitual. Finalmente, o modelo de apresentação (PM) é gerado a partir do processamento do modelo navegacional da aplicação.

Conforme ilustra a Figura 4.6, o processamento de cada modelo, além de produzir o modelo subsequente, gera também documentos responsáveis por prover a funcionalidade da aplicação Web resultante. Os *scripts* produzidos após a geração do modelo conceitual servem para criar o espaço de informação, podendo ser *scripts* SQL, por exemplo. Já os documentos de conteúdo (*Content*) orientam a definição de *scripts* XSP, responsáveis por obter o

Figura 4.6: Visão geral da abordagem wView (Andrade et al. 2004a).

conteúdo de um nó sob demanda. Os documentos de estilo (*Style*) determinam como conteúdo de cada nó deverá ser exibido. Finalmente, o *Sitemap* é o documento que define uma aplicação Web suportada pelo *Framework Cocoon* (Apache 2005).

É importante notar que o *Sitemap* é gerado de maneira independente do *pipeline* de transformação, a partir de um documento de configuração (*Config*) provido pelo *wView*. Esse documento permite a configuração de informações gerais sobre a aplicação, tais como seu nome, a tecnologia utilizada, as fontes de dados e os formatos de apresentação para os nós, bem como o primeiro nó da navegação.

Na geração de uma aplicação Web, *wView* pode utilizar um total de 37 documentos, incluindo esquemas XML, documentos XML declarativos e folhas de estilo XSLT. Desse total, 24 são folhas de estilo, das quais 17 são estáticas e 7 são geradas durante o processo de transformação; 5 são documentos declarativos, providos pelo desenvolvedor da aplicação; e 8 são documentos resultantes do processo de transformação (Andrade et al. 2004a).

Diferentemente da abordagem utilizada na geração dos esquemas que modelam as informações de contexto, o *wView* processa a interação do desenvolvedor da aplicação utilizando folhas de estilo geradas dinamicamente. Dos cinco documentos declarativos que fazem parte do *wView*, um refere-se ao documento XMI, sendo obrigatório para a geração da aplicação. Os outros quatro, opcionais, provêm mecanismos que possibilitam a intervenção do desenvolvedor no processo de geração da aplicação, permitindo extensões e customizações no procedimento padrão.

O processo de transformação descrito neste capítulo explora resultados obtidos na parte inicial da abordagem *wView*, particularmente nas transfor-

mações que produzem o modelo conceitual modificado da aplicação. Mais que estender o processo de geração do modelo conceitual modificado do *wView*, o trabalho realizado cuidou de complementá-lo, tratando os relacionamentos de generalização que ficaram de fora da implementação original (Andrade et al. 2004b).

A Figura 4.7 detalha o processo de geração do modelo conceitual modificado no *wView*. A transformação inicial, que gera o modelo conceitual básico a partir do documento XMI da aplicação é semelhante ao discutido na Seção 4.4.1 — o que muda é que na abordagem apresentada inicialmente é necessário tratar as extensões UML inseridas no modelo a fim de permitir a representação das dimensões de contexto e a caracterização das classes externas.

Figura 4.7: Geração do modelo conceitual no *wView* (Andrade 2004).

A segunda transformação mostrada na Figura 4.7 serve para permitir a customização do modelo conceitual. Por meio de um documento declarativo (*CM-INSTRUCTIONS*), o desenvolvedor da aplicação pode omitir ou acrescentar classes conceituais, atributos e/ou relações. Utilizando a abordagem padrão, o *wView* provê uma folha de estilo estática que processa o documento declarativo e produz uma nova folha de estilo, denominada *CM-MOD.XSL*, a qual é usada para processar o modelo conceitual, produzindo o modelo conceitual modificado.

Além de cuidar da inserção e remoção de classes, atributos e relações, a folha de estilo *CM-MOD.XSL* adiciona identificadores para as classes conceituais e mapeia as relações de composição e associação no modelo conceitual modificado. Tais procedimentos visam facilitar a geração dos scripts que de-

finem o espaço de informação a ser navegado. Tais scripts são gerados a partir de folhas de estilo que processam o modelo conceitual modificado.

Vale notar que o processo de transformação descrito não trata os relacionamentos de generalização, impedindo o uso de modelos UML que contenham relações de herança. Para solucionar essa questão, implementou-se o procedimento descrito na Seção 4.4.2, o qual pode ser utilizado no *wView* com pequenos ajustes.

Outra diferença significativa observada nas duas abordagens diz respeito à forma de interação do desenvolvedor no processo de transformação. Enquanto o *wView* realiza a geração de folhas de estilo dinâmicas a partir de documentos declarativos contendo as especificações de interação, o *Contextractor* utiliza-se de parâmetros que determinam as mudanças no procedimento de transformação padrão.

A abordagem definida pelo *Contextractor* permite utilizar um número menor de transformações para produzir os documentos resultantes. Isso é relevante, uma vez que a escrita dos documentos de transformação XSLT constitui uma tarefa não trivial.

Andrade et al. (2004a) discutem os problemas encontrados no uso de XSLT no desenvolvimento do *wView*, os quais também foram percebidos no desenvolvimento do *Contextractor*. Tais problemas decorrem do fato de que a XSLT é uma linguagem declarativa, e não imperativa. O único comando que permite controle imperativo sobre a ordem de execução é o *for-each*. Entretanto, a iteração dos elementos em um comando *for-each* é inteiramente determinado pelo documento de entrada. Apesar de ser possível utilizar expressões XPath para selecionar subconjuntos desses elementos, não é possível determinar esses subconjuntos a partir de características do documento de saída, ou a partir de um outro documento declarativo.

Outra questão diz respeito às variáveis, que não funcionam como variáveis, já que não podem ter seu conteúdo alterado a partir do momento que recebem um valor inicial. Essas limitações da linguagem XSLT dificultam a realização de tarefas que seriam triviais em linguagens de programação procedurais ou orientadas a objetos.

O resultado dessas limitações é o aumento do número de transformações necessárias para se obter o resultado esperado. O uso de parâmetros, conforme descrito na Seção 4.4.2 é uma alternativa para o problema, mas nem sempre pode ser aplicado a todos os casos.

4.6.2 Modelagem de Contexto na Literatura

A modelagem de informação de contexto pode ser feita a partir de diferentes abordagens. Gu et al. (2004) classificam a modelagem de contexto em três categorias: abordagens orientadas a aplicações, a modelos e a ontologias. Um estudo mais completo feito por Strang and Linnhoff-Popien (2004) divide as abordagens de modelagem em seis tipos: baseadas em pares atributo-valor, baseadas em *profiles*, gráficas, orientadas a objetos e baseadas em ontologias.

Independentemente da abordagem utilizada, os modelos produzidos são, em geral, parte integrante de *frameworks*, *toolkits* ou infra-estruturas que têm por objetivo facilitar o desenvolvimento de aplicações sensíveis a contexto. Nesses ambientes, a modelagem de contexto define o formato de representação de entidades, tais como pessoas e lugares, e um conjunto de operações sobre essas entidades, permitindo que as aplicações utilizem operações padrão para armazenamento e recuperação de informação de contexto.

No domínio de aplicações de captura e acesso, o uso de informações de contexto pode ocorrer durante as fases de captura, acesso, ou ambas. Em ambientes instrumentados para a captura de atividades, tais como salas de aula e de reunião, a obtenção das informações de contexto é facilitada: procedimentos de autenticação nas aplicações que executam nos dispositivos de captura permitem associar a identidade das pessoas envolvidas. Questões de localização são também simplificadas, uma vez que a mobilidade dos usuários fica restrita aos ambientes instrumentados. Nesse contexto, o uso de sensores não enriquece substancialmente a funcionalidade das aplicações.

Dessa forma, a modelagem de informações de contexto para essas aplicações têm como objetivo principal facilitar a recuperação de conteúdo capturado com base em conceitos bem definidos: identificação de pessoas envolvidas, identificação de atividades capturadas, tempo e localização dos eventos. Em outras palavras, a modelagem das informações de contexto visa primariamente suportar aplicações que fazem acesso a conteúdo previamente capturado.

Apesar das particularidades das aplicações de captura e acesso influenciarem o processo de modelagem de contexto, é possível encontrar similaridades entre as abordagens de modelagem citadas na literatura e a abordagem utilizada neste trabalho.

De acordo com Strang and Linnhoff-Popien (2004), o *Context extractor* é classificado como um modelo gráfico, uma vez que explora os diagramas de classe providos pela linguagem UML, propondo uma forma de extensão desses dia-

gramas.

Entre as iniciativas que provêem facilidades gráficas para a modelagem de informações de contexto, destaca-se o trabalho liderado por Karen Henriksen (Henricksen et al. 2002; Henriksen et al. 2003; Henriksen and Indulska 2004), o qual estende o modelo gráfico ORM — Object-Role Modeling (Halpin 1995).

A modelagem ORM representa tipos de objeto como elipses e relações sobre um ou mais tipos de objeto como fatos, desenhados como seqüências de retângulos representando papéis, sendo cada papel interligado aos seus tipos de objeto associados. A extensão realizada por Henriksen et al. permitiu categorizar os fatos em estáticos e dinâmicos, de acordo com sua persistência e origem. Outra extensão importante foi a definição de um tipo de fato denominado temporal, o qual, como o próprio nome sugere, modela os fatos com base no tempo. Por fim, uma última extensão introduzida no ORM foram as dependências factuais, que representam possíveis relações entre fatos.

O conjunto de extensões implementadas por Henriksen et al. resultou na definição da linguagem CML — Context Modeling Language — a qual permite a especificação formal dos requisitos de contexto para aplicações cientes de contexto (Henricksen and Indulska 2004). Por meio de um mecanismo de transformação, é possível mapear um modelo CML para um sistema de gerenciamento de contexto suportado por um banco de dados relacional. A tradução do modelo CML para o modelo relacional estende o procedimento de mapeamento do modelo ORM para o relacional, acrescentando o tratamento necessário para as extensões feitas pela linguagem CML ao ORM.

O mecanismo de mapeamento adotado na abordagem de Henriksen possui similaridades com o *Contexttractor*. Ambos estendem ferramentas de modelagem pré-existentis provendo funcionalidades capazes de representar informação contextual. Definem também mecanismos similares de mapeamento entre os modelos produzidos com as extensões fornecidas e modelos de armazenamento subjacente.

4.7 *Dificuldades Encontradas e Limitações da Metodologia*

O desenvolvimento do *Contexttractor* envolveu adaptações nas interfaces gráficas da ferramenta ArgoUML, de modo a suportar o uso dos mecanismos de extensão da linguagem UML de maneira transparente, a definição de sete

diferentes folhas de estilo XSLT, e a implementação do processador de consultas contextuais.

Os desafios encontrados no desenvolvimento do *Contextractor* estiveram associados tanto a questões de implementação de suas ferramentas de apoio, como também ao desenvolvimento das folhas de estilo que compõem a pipeline de transformação XSLT.

A extensão do ArgoUML não consistiu simplesmente na alteração das interfaces gráficas da ferramenta. A interação com os novos elementos de interface providos resultam na alteração do modelo e, conseqüentemente, do documento XMI que o representa. Dessa forma, foi necessário estudar o código do ArgoUML, o qual é aberto, a fim de compreender como o mecanismo de extensão UML havia sido implementado. Com base nesse estudo, foi então possível programar as novas interfaces gráficas para que elas refletissem no modelo as interações dos usuários.

A dificuldade associada à programação dessas interfaces deu-se em função da necessidade de estender um código de terceiros não documentado. Entretanto, o maior desafio ficou por conta da definição das folhas de estilo que compõem a *pipeline* de transformações definida pelo *Contextractor*.

Parte significativa desses desafios estão relacionados ao uso da linguagem XSLT, cujas limitações foram discutidas na Seção 4.6.1. O resultado dessas limitações são folhas de estilo complexas, algumas com cerca de duzentas linhas de código, o que dificulta o entendimento e a manutenção desse código.

Apesar da complexidade envolvida, foi possível gerar com sucesso o esquema XML que modela as informações de contexto de cada aplicação e o par de esquemas responsáveis por suportar a consulta contextual de conteúdo capturado.

Entretanto, algumas limitações da metodologia são perceptíveis. A primeira delas envolve a definição de atributos capazes de representar de maneira inequívoca as instâncias de cada classe. O *Contextractor* utiliza um atributo fixo denominado *id* como um identificador das instâncias de uma classe. Na ausência desse atributo, o primeiro atributo definido para a classe é considerado o atributo identificador da mesma.

A definição de um atributo identificador para a classe é relevante, uma vez que esse atributo é candidato natural a ser transformado em chave primária, caso o modelo conceitual da aplicação deva ser mapeado para o modelo relacional. Dessa forma, não é possível representar adequadamente classes que usam dois ou mais atributos para identificar suas instâncias. Isso traz uma

clara limitação no uso de sistemas legados — aqueles que possuem classes com as características mencionadas não são adequadamente suportados pela metodologia.

Um outro problema constatado envolve o uso dos conectores *AND* e *OR*. Na linguagem SQL, por exemplo, esses conectores podem ter um nível de aninhamento indeterminado. Entretanto, esquemas XML não suportam esse tipo de aninhamento, uma vez que os aninhamentos devem ser declarados de maneira explícita. Assim, por depender de um par de esquemas XML, a linguagem CQL possui a limitação de não suportar o uso conjunto dos conectores *AND* e *OR* para um mesmo conjunto de elementos. Uma possível solução para esta limitação é discutida na Seção 7.2, que trata de trabalhos futuros.

4.8 Considerações Finais

A modelagem de informação de contexto para aplicações de computação ubíqua tem sido objeto de investigação por inúmeros grupos de pesquisa ao redor do mundo (Strang and Linnhoff-Popien 2004). Entre as diferentes formas de modelagem propostas, aquelas que utilizam uma abordagem gráfica apresentam-se como uma alternativa interessante para o usuário, uma vez que abstraem a complexidade do modelo utilizando representações gráficas simples, muitas vezes já conhecidas pelos desenvolvedores.

As aplicações de captura e acesso possuem a particularidade de explorar informações de contexto antes, durante e depois da captura de sessões ao vivo. Uma forma simples, porém poderosa, de facilitar a recuperação de informação capturada consiste em explorar as dimensões de contexto *who*, *what*, *when* e *where* discutidas no trabalho de Abowd and Mynatt (2000).

Com base na experiência adquirida ao longo de três anos modelando e implementando aplicações de captura e acesso, observou-se que as informações associadas às dimensões contextuais podem ser obtidas diretamente dos diagramas UML de sistemas legados, tais como sistemas acadêmicos e corporativos.

Assim, explorando a familiaridade dos desenvolvedores de aplicações com as ferramentas de modelagem UML, desenvolveu-se o *Contextractor*, abordagem gráfica que permite extrair informações de contexto a partir do diagrama UML estendido obtido a partir da modelagem desses sistemas (Baldochi, Cattelan, and Pimentel 2005).

O intuito do *Contextractor* é produzir de maneira automática um esquema

XML no qual os principais elementos são as dimensões contextuais, as quais são definidas a partir das classes e atributos presentes no modelo UML estendido da aplicação. Esse esquema é posteriormente utilizado para modelar as informações que caracterizam sessões de captura e acesso. Além disso, a partir dele são definidos dois outros esquemas que modelam uma linguagem de consulta baseada nas dimensões contextuais modeladas denominada CQL — Context Query Language.

Tendo em mente a automatização do processo de produção desses esquemas, optou-se por prover ao desenvolvedor ferramentas que minimizem sua intervenção nesse procedimento. Para tanto, foi necessário estender a ferramenta de modelagem ArgoUML, provendo interfaces amigáveis, nas quais o desenvolvedor utiliza as extensões UML do tipo estereótipos e *tagged-values* de maneira implícita.

Em seguida desenvolveu-se um conjunto de folhas de estilo XSLT que permitem produzir os esquemas XML a partir do documento XMI gerado pela ferramenta de modelagem. Por fim, utilizando os esquemas produzidos, implementou-se um mecanismo denominado Processador CQL, o qual é capaz de processar expressões de consulta contextual, retornando referências aos documentos que atendem às restrições impostas em cada consulta.

Levando-se em conta que o conhecimento de UML é praticamente obrigatório à maioria dos desenvolvedores de aplicações computacionais, a metodologia de modelagem definida pelo *Contexttractor* requer um esforço mínimo de aprendizagem por parte do usuário. É necessário apenas se familiarizar com as duas interfaces gráficas implementadas no ArgoUML e ter conhecimentos rudimentares de transformação XSLT.

Além das vantagens trazidas ao procedimento de modelagem, a abordagem apresentada neste capítulo também facilita o desenvolvimento de aplicações de acesso a conteúdo capturado, uma vez que provê uma interface de programação capaz de processar consultas contextuais. A integração do Processador CQL ao repositório de documentos XML produzidos pelas aplicações é apresentada a seguir, no Capítulo 5.

StRES: Automatizando o Armazenamento e a Recuperação de Informação

5.1 Considerações Iniciais

O aumento da largura de banda dos *backbones* da Internet, juntamente com a crescente oferta de conexões de banda larga a preços cada vez mais atraentes e avanços nas tecnologias sem fio têm tornado reais muitas das visões de Mark Weiser. Entretanto, conectividade e *gadgets* não representam muito sem ubiquidade no acesso a dados. A disponibilização de armazenamento pervasivo a aplicações requer novas técnicas no gerenciamento de dados, uma vez que as informações necessárias para suportar aplicações ubíquas devem ser eficientemente armazenadas, consultadas e recuperadas.

Franklin (2001) discute três tópicos que impõem desafios associados ao gerenciamento de dados que precisam ser superados a fim de que se possa prover acesso pervasivo às aplicações: suporte à mobilidade, ciência de contexto e suporte à colaboração.

No que diz respeito ao suporte à mobilidade, espera-se que aplicações possam operar em ambientes dinâmicos e heterogêneos, sendo capazes de migrar de uma rede para outra de maneira automática.

Em relação à ciência de contexto, espera-se apoiar computacionalmente atividades informais do dia-a-dia, envolvendo uma variedade de domínios que vão da residência das pessoas, aos ambientes de trabalho e estudo. O suporte constante a atividades informais recebe a denominação de computação do cotidiano (Abowd and Mynatt 2000). A fim de prover tal suporte, aplicações devem ter ciência do ambiente, dos usuários e de suas atividades. Assim, o sistema de gerenciamento de dados subjacente deve prover formas de encapsular, armazenar e recuperar informações de contexto.

Quanto ao suporte à colaboração, espera-se que as aplicações que suportam atividades em grupo, presentes por exemplo nos domínios de reuniões e salas de aula, possam contar com ferramentas de gerenciamento que facilitem armazenar, recuperar e manter dados compartilhados.

O suporte à colaboração encontra-se implícito nas aplicações de captura e acesso, assim como o processamento de informações de contexto. Certas aplicações de captura podem se beneficiar de dispositivos sem fio, tais como *tablets* e *PDAs*. Dessa forma, percebe-se que os desafios associados ao gerenciamento de informação constituem um problema significativo para o desenvolvimento de aplicações de captura e acesso.

No sentido de prover soluções para as questões associadas ao suporte à colaboração e ao processamento de informações de contexto, desenvolveu-se o StRES — *Storing, Retrieving and Extending Service* — serviço associado à infraestrutura xInCA que objetiva prover mecanismos que possibilitem facilitar o gerenciamento de conteúdo produzido pelas aplicações de captura e acesso.

Integrado aos componentes xInCA, o StRES provê um modelo para desenvolvimento de aplicações que permite automatizar o processo de geração de documentos que representam sessões capturadas. Esse modelo contempla o tratamento de informações de contexto, permitindo a definição de uma linguagem de consulta que possibilita recuperar conteúdo capturado com base nas dimensões de contexto associadas a cada aplicação. O modelo implementado ainda permite direcionar o desenvolvimento de folhas de estilo capazes de exibir o conteúdo capturado em diferentes formatos de apresentação.

Uma característica importante da abordagem utilizada é o uso de documentos XML como blocos arquiteturais da infra-estrutura disponibilizada. Diferentemente de outras abordagens, nas quais documentos são utilizados somente como uma forma de apresentar o conteúdo capturado na *Web*, a infra-estrutura implementada utiliza documentos XML na geração do código das aplicações, na definição das informações intercambiadas por seus componentes, na representação de conteúdo capturado, na consulta a esse conteúdo

e até mesmo na sua apresentação.

Este capítulo apresenta o serviço StRES, enfatizando seus detalhes arquiteturais, os quais são apresentados na Seção 5.2. Em seguida, a Seção 5.3, apresenta o modelo de desenvolvimento de aplicações a partir dos documentos XML associados aos componentes da infra-estrutura e também às informações de contexto da aplicação. A Seção 5.4 apresenta os mecanismos de acesso e extensão providos pelo StRES que permitem recuperar, apresentar e estender conteúdo capturado. Finalmente, a Seção 5.5 traz as considerações finais do capítulo, discutindo os vantagens e limitações da solução implementada.

5.2 A Abordagem xInCA-StRES: Visão Arquitetural

Considerando que os dados produzidos por uma aplicação de captura e acesso estão associados aos dispositivos utilizados na captura, e que esses dispositivos encontram-se atrelados às aplicações que os gerenciam, o esperado é que os dados manipulados por essas aplicações façam parte do conteúdo capturado. Dessa forma, uma lousa eletrônica (dispositivo físico) é gerenciada pelo componente de software *whiteboard*, produzindo *slides* e *strokes*. Assim, uma aplicação que utiliza lousa eletrônica deverá produzir como informação capturada dados capazes de representar *slides* e *strokes*.

Por outro lado, considerando que as informações capturadas devem ser visualizadas primariamente na *Web*, é desejável representá-las como documentos passíveis de serem apresentados nesse ambiente. A partir dessas considerações, investigou-se alternativas para representar sessões capturadas como documentos XML.

O primeiro passo no sentido de modelar documentos capazes de representar conteúdo capturado consistiu na definição de esquemas XML para todos os componentes xInCA. Em seguida, foi definido um esquema XML para a sessão de captura, o qual agrega os esquemas dos componentes juntamente com um esquema contendo as informações de contexto da aplicação.

A implementação inicial utilizava um esquema fixo para as informações de contexto (Baldochi et al. 2002; Baldochi et al. 2003; Baldochi et al. 2004). Essa abordagem, apesar de atender as necessidades de boa parte das aplicações, não era suficientemente expressiva para permitir a modelagem de uma linguagem de consulta baseada nas dimensões de contexto. Dessa forma, optou-se por formalizar um mecanismo de modelagem das informações de contexto a partir de diagramas UML das aplicações. Esse mecan-

ismo encontra-se descrito de maneira detalhada no Capítulo 4.

O esquema de sessão é então utilizado por um mecanismo de data binding chamado JAXB — *Java Architecture for XML Binding*¹. Provida pela Sun Microsystems, JAXB faz o mapeamento entre documentos XML e objetos Java. As classes geradas por esse mapeamento cuidam de todos os detalhes de *parsing* e formatação das informações, permitindo que os desenvolvedores implementem aplicações que lêem, manipulam e criam documentos XML sem ter que lidar diretamente com elementos e atributos.

Instâncias das classes geradas são os tipos de dados primitivos manipulados e intercambiados pelos componentes xInCA. Objetos dessas classes são utilizados por uma entidade denominada *Transducer*, a qual é responsável pela geração dos documentos que representam as sessões capturadas.

A palavra inglesa *transduce* significa converter de um formato para outro. Assim, denominou-se *Transducer* a classe responsável por converter os dados produzidos pelos componentes, juntamente com as informações de contexto, em um documento XML representativo da sessão capturada.

O *Transducer* instancia componentes xInCA de acesso para cada componente de captura presente na aplicação e também objetos produzidos pelo JAXB. O *Transducer* recebe nos componentes de acesso instanciados as informações produzidas pelos componentes equivalentes de captura. Essa comunicação acontece de forma transparente, conforme explicado na Seção 3.4.2. As informações recebidas são então carregadas nos objetos produzidos pelo mecanismo de *data binding*.

A infraestrutura provida esconde esses detalhes de implementação do desenvolvedor da aplicação. De fato, tudo o que desenvolvedor deve fazer a fim de possibilitar a geração de documentos para as sessões capturadas por sua aplicação é instanciar um único objeto, chamado *StorageComponent*. Esse objeto encapsula as informações de contexto, as quais devem estar em conformidade com o esquema *CONTEXT-INFO*, produzido pela metodologia *Context extractor*. Além das informações de contexto, o *StorageComponent* contém também o identificador da sessão, usado para determinar o fluxo de informação entre os módulos de captura e acesso registrados na InCA, e uma variável que qualifica a sessão em *local* ou *distribuída*.

Essa última variável revela uma característica importante das sessões de captura: elas podem ser distribuídas, possuindo vários clientes produzindo e acessando informação em tempo real intercambiada por uma rede local,

¹<http://java.sun.com/xml/jaxb>

ou até mesmo pela Internet. Nesse caso, cada cliente representa parte de uma aplicação distribuída que deve ter seu conteúdo produzido armazenado em um único documento. Isso indica que a instanciação de um *StorageComponent* não significa a instanciação de um *Transducer*, o qual é instanciado uma única vez, quando o primeiro cliente da aplicação distribuída se torna ativo. Os demais clientes pertencentes à mesma aplicação (que compartilham o mesmo identificador de sessão) fazem um 'join' em um *Transducer* já instanciado.

A fim de permitir sessões nas quais clientes são adicionados e removidos dinamicamente foi necessário implementar um *daemon* responsável pela manutenção dos *Transducers* associados a sessões ativas. Esse *daemon*, denominado *Sessiond*, implementa duas listas: uma armazena os identificadores das sessões ativas e a outra contém os respectivos *Transducers* associados a cada identificador. Ao receber uma nova conexão, o *Sessiond* verifica se o identificador que acaba de receber faz parte da primeira lista. Em caso afirmativo, o *Transducer* associado ao identificador é recuperado e seu método *join* é executado, recebendo como parâmetro as informações de contexto do novo cliente. Caso contrário, o *daemon* instancia um novo *Transducer*, passando ao construtor do mesmo as informações de contexto do cliente. Finalizando, o novo identificador e o objeto recém instanciado são incluídos às listas mantidas pelo *daemon*.

A Figura 5.1 ilustra a arquitetura descrita usando como exemplo uma aplicação de captura e acesso contendo dois componentes xInCA: *whiteboard* e áudio. O componente de armazenamento, presente na aplicação de captura, é responsável por estabelecer uma conexão via *socket* com o *Sessiond*, o qual instancia o *Transducer*. A partir de sua instanciação, a comunicação entre o cliente e o *Transducer* é transparentemente provida pelo *Registry* da InCA.

A InCA provê um mecanismo de *callbacks* que reporta a ocorrência de eventos relacionados aos módulos nela registrados. Por meio desse mecanismo o *daemon* é capaz de determinar o fim de uma sessão e então finalizar o *Transducer* associado àquela sessão, procedimento que causa a geração do documento XML que representa o conteúdo capturado. Nesse momento ocorre a conversão que justifica o nome dado à essa classe: objetos das classes JAXB são convertidos em fragmentos XML, os quais, juntos, formam o documento de sessão.

A última ação do *Transducer* antes de ser finalizado consiste no armazenamento do documento produzido. Para tanto, ele utiliza o documento *REPOSITORY.XML*, obtido a partir das informações de contexto da aplicação, conforme

Figura 5.1: Arquitetura xInCA-StRES

discutido na Seção 4.5.1. Esse documento contém as informações referentes à coleção na qual o documento recém-gerado deve ser armazenado.

5.3 Processamento de Documentos

O modelo de desenvolvimento de aplicações implementado é centrado na transformação de esquemas XML. O processamento desses documentos pelo mecanismo de *binding* constitui um bloco arquitetural importante da infraestrutura provida. Esta seção apresenta os aspectos de modelagem dos esquemas XML desenvolvidos e mostra como novos componentes podem ser agregados à infraestrutura existente.

A versão inicial da infraestrutura foi desenvolvida usando DTDs — *Document Type Declarations* — para representar os esquemas de dados (Baldochi et al. 2003). O uso de DTDs foi imposto pela versão do JAXB disponível na época (*version 1.0ea — early access*), a qual não suportava o uso de esquemas XML. Apesar das limitações dos DTDs quando comparados aos esquemas XML, a primeira implementação da infraestrutura permitiu o desenvolvimento de diversas aplicações, dentre as quais estão o CAT e o iClass, apresentados no Capítulo 6.

O maior problema dos DTDs é o fato de possuírem um esquema de tipação de dados limitado. Para solucionar esse problema, a versão inicial do JAXB exigia, para cada DTD, um esquema auxiliar chamado *XML-Java binding schema*, cuja função era mapear a tipação de elementos e atributos para os tipos de dados da linguagem Java. Em função desse esquema auxiliar tornar mais

complicada a manutenção de um grande número de esquemas de dados, optou-se por migrar para a versão mais recente do JAXB, que suporta o uso de esquemas XML.

O esquema do componente de *whiteboard* da xInCA é apresentado a seguir. Para diversos domínios, incluindo a sala de aula e ambientes de reunião, *whiteboard* é o componente xInCA mais utilizado. Dois elementos estão presentes no primeiro nível desse esquema: *slide* e *visit*. Em função do tamanho desse esquema, a definição do elemento *slide* é mostrada a seguir e a definição do elemento *visit* é mostrada na página seguinte.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb" jxb:version="1.0">
  <xsd:complexType name="Whiteboard">
    <xsd:sequence>
      <xsd:element name="slide" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="title" type="xsd:string" minOccurs="0"/>
            <xsd:element name="body" minOccurs="0">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="sentence" type="xsd:string"
                    minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="annotation" type="xsd:string"
                    minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="url" type="xsd:string"/>
          </xsd:sequence>
          <xsd:attribute name="slide_id" type="xsd:int" use="required"/>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="visit" minOccurs="0" maxOccurs="unbounded">
        ...
      </xsd:element>

    </xsd:sequence>
    <xsd:attribute name="board_width" type="xsd:int" use="required"/>
    <xsd:attribute name="board_height" type="xsd:int" use="required"/>
  </xsd:complexType>
</xsd:schema>
```

O elemento *slide* modela a informação associada a cada *slide* produzido durante uma sessão ao vivo. O componente de *whiteboard* suporta a pré-carga de *slides* criados em PowerPoint™, por exemplo. Esses *slides* são passados ao componente de *whiteboard* como arquivos de imagem (JPG ou PNG). Assim, é possível utilizar o componente de *whiteboard* para escrever sobre *slides* previamente preparados, o que não impede a inserção de *slides* em branco em qualquer ponto da apresentação. Se o usuário preferir, pode apenas escrever sobre *slides* em branco durante toda a captura.

Quando arquivos *Power Point*TM são utilizados, é possível extrair o texto de cada *slide* e salvá-lo nos elementos *title* e *body*. Esses elementos podem ser usados posteriormente na consulta a conteúdo capturado.

O intervalo de tempo no qual um *slide* é apresentado configura uma visita, a qual é modelada pelo elemento *visit*, cujo código é apresentado a seguir. Cada *slide* pode ser visitado inúmeras vezes e, em cada visita, o instrutor pode fazer anotações (*strokes*) no *slide*. Por esse motivo, o elemento *stroke* está definido dentro do elemento *visit*. Isso possibilita identificar em qual visita um determinado *stroke* foi feito. Essa informação é usada para permitir um tipo de apresentação que faz o *play back* das interações do usuário com a lousa.

```
<xsd:element name="visit" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="stroke" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="min_x" type="xsd:int"/>
            <xsd:element name="min_y" type="xsd:int"/>
            <xsd:element name="max_x" type="xsd:int"/>
            <xsd:element name="max_y" type="xsd:int"/>
            <xsd:element name="points">
              <xsd:simpleType>
                <xsd:annotation>
                  <xsd:appinfo>
                    <jxb:javaType name="br.usp.icmc.xinca.data.Points"
                      parseMethod="br.usp.icmc.common.data.TransPoints.parsePoints"
                      printMethod="br.usp.icmc.common.data.TransPoints.printPoints"/>
                  </xsd:appinfo>
                </xsd:annotation>
                <xsd:restriction base="xsd:string"/>
              </xsd:simpleType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="stroke_id" type="xsd:int"/>
          <xsd:attribute name="creation_time" type="xsd:string" use="required"/>
          <xsd:attribute name="deletion_time" type="xsd:string"/>
          <xsd:attribute name="color" use="required">
            <xsd:annotation>
              <xsd:appinfo>
                <jxb:javaType name="java.awt.Color"
                  parseMethod="br.usp.icmc.common.data.TransColor.parseColor"
                  printMethod="br.usp.icmc.common.data.TransColor.printColor"/>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:attribute>
          <xsd:attribute name="width" type="xsd:int" use="required"/>
          <xsd:attribute name="person_id" type="xsd:string" use="required"/>
          <xsd:attribute name="timestamp" type="xsd:long" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="visit_id" type="xsd:int" use="required"/>
    <xsd:attribute name="duration" type="xsd:long" use="required"/>
  </xsd:complexType>
</xsd:element>
```



```

<xsd:attribute name="slide_id" type="xsd:int" use="required"/>
<xsd:attribute name="timestamp" type="xsd:long" use="required"/>
</xsd:complexType>
</xsd:element>

```

As informações que compõem o elemento *slide* são obtidas ao final da primeira visita a cada *slide*, quando é também gerado o arquivo de imagem que o representa. O elemento *url* contido dentro do elemento *slide* é um ponteiro para esse arquivo de imagem. Caso ocorram novas visitas a um mesmo *slide*, cada visita, juntamente com seus *strokes* associados, produz um novo elemento *visit* e um novo arquivo de imagem, o qual sobrepõe o arquivo precedente. Como resultado final, além do documento XML, um arquivo de imagem é gerado para cada *slide* capturado.

Um fato a se notar nesse esquema é a declaração do espaço de nomes *jxb*. Esse *namespace* está relacionado com o mecanismo de *data binding*. JAXB mapeia os tipos de dado XML em um conjunto pré-definido de tipos de dado e classes Java. Entretanto, pode ser necessário fazer *binding* de um elemento XML com uma classe customizada, o que não é possível utilizando o mapeamento padrão do JAXB. Para solucionar esse problema, o mecanismo de *binding* provê o elemento *jxb:javaType*, o qual permite a definição de mapeamentos customizados. Exemplos dessa customização podem ser vistos na declaração dos elementos *points* e *color*.

Esquemas semelhantes foram definidos para os demais componentes xlnCA. O código a seguir ilustra o esquema de sessão, no qual os componentes xlnCA são integrados juntamente com o esquema contendo as informações de contexto da aplicação.

```

<xsc:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="context-info.xsd"/>
  <xsd:include schemaLocation="audioml.xsd"/>
  <xsd:include schemaLocation="chatml.xsd"/>
  <xsd:include schemaLocation="videoml.xsd"/>
  <xsd:include schemaLocation="weblogml.xsd"/>
  <xsd:include schemaLocation="whiteboardml.xsd"/>
  <xsd:element name="session">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="context-info" type="ContextType"/>
        <xsd:element name="audioml" type="Audio" minOccurs="0"/>
        <xsd:element name="chatml" type="Chat" minOccurs="0"/>
        <xsd:element name="videoml" type="Video" minOccurs="0"/>
        <xsd:element name="weblogml" type="Weblog" minOccurs="0"/>
        <xsd:element name="whiteboardml" type="Whiteboard" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:attribute name="id" type="xsd:int" use="required"/>
  <xsd:attribute name="type" type="xsd:int" use="required"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>

```

```

</xsd:complexType>
</xsd:element>
</xsd:schema>

```

O elemento *session* contém três atributos importantes. O primeiro (*id*), refere-se ao identificador de sessão, valor que não se repete e é usado para indexar os documentos produzidos. O segundo (*type*), é uma constante que identifica quais componentes xInCA foram usados na sessão de captura. Finalmente, o terceiro (*name*), serve para armazenar a referência ao documento no contexto do repositório de documentos, sendo composto pelo caminho (seqüência de coleções) e o nome do arquivo XML. O atributo *name* é usado para recuperar documentos utilizando consultas *FLWOR* ou *CQL*.

O esquema de sessão foi elaborado de modo a ser flexível. Todos os seus elementos de primeiro nível, com exceção de *context-info*, possuem o atributo *minOccurs* com valor "0", o que possibilita a construção de aplicações com qualquer combinação de componentes.

O esquema com as informações de contexto deve ser gerado de acordo com o procedimento descrito no Capítulo 4. Por serem indispensáveis à recuperação do conteúdo capturado, toda sessão deve possuir o elemento *context-info*.

Tendo definido todos os esquemas necessários e, considerando que o esquema de sessão referencia todos os demais esquemas, basta submetê-lo ao compilador JAXB para gerar classes capazes de processar o conteúdo de elementos e atributos presentes nos esquemas processados. Esse procedimento é ilustrado na Figura 5.2.

Conforme ilustra a figura, as classes geradas pelo JAXB são compostas de métodos *get* e *set* que permitem obter ou ajustar o valor de variáveis correspondentes a elementos e atributos presentes nos esquemas. Além disso, toda classe gerada possui os métodos *validate*, *marshal* e *unmarshal*. O primeiro serve para validar os dados contidos nas variáveis de instância das classes. Para uma dada classe, se o método *validate* executa sem erros, significa que é possível executar o método *marshal*, o qual gera um documento XML a partir das variáveis de instância da classe na qual o método foi chamado. Se uma classe *A* referencia uma classe *B*, chamar o método *marshal* em *A* faz com que esse método também seja chamado em *B*. Dessa forma, chamar o método *marshal* na classe *Session*, causa a execução do mesmo método em todas as classes associadas a *Session*, o que produz um documento XML contendo dados referentes a todos componentes xInCA presentes na aplicação, juntamente com suas informações de contexto. O método *unmarshal* faz o papel inverso do *marshal*, permitindo que um documento XML seja representado por seus

Figura 5.2: Compilação do esquema de sessão pelo JAXB

objetos Java correspondentes.

O uso de JAXB facilita a manutenção da infra-estrutura, possibilitando que novos componentes sejam inseridos, ou que componentes existentes sejam alterados, sem afetar o funcionamento de outros componentes ou da infra-estrutura como um todo. Quando um esquema de dados é atualizado, basta recompilar o esquema de sessão e, eventualmente, alterar algumas linhas de código do *Transducer*, para que a infra-estrutura reflita a atualização realizada.

Do ponto de vista do desenvolvedor, a implementação do mecanismo de gerenciamento de dados para uma aplicação de captura e acesso segue um procedimento simples. Basta produzir o esquema contendo as informações de contexto da aplicação, conforme discutido no Capítulo 4, e, em seguida, compilar o esquema obtido, gerando classes Java que devem ser colocadas no mesmo diretório das demais classes geradas via *data binding*.

5.4 Suporte ao Acesso e à Extensão de Conteúdo Capturado

Conforme discutido na Seção 5.1, o StRES tem por objetivo prover serviços de armazenamento, recuperação e extensão de conteúdo capturado. Entretanto, antes de poder armazenar, recuperar e estender informação é necessário representar essa informação de maneira adequada, fornecendo um mecanismo capaz de mapear dados produzidos por diferentes dispositivos para o formato de representação definido.

Realizar esse mapeamento é o papel da abordagem xlnCA-StRES. Dessa forma, o StRES é parte infra-estrutura e parte serviço. O StRES, enquanto serviço, passa a existir a partir do momento em que o documento XML representativo da sessão capturada é produzido.

A partir desse momento, é necessário (i) armazenar o documento em um repositório não volátil, (ii) prover um mecanismo de recuperação de conteúdo, preferencialmente baseado em informações de contexto, (iii) facilitar a apresentação de conteúdo capturado e (iv) permitir que esse conteúdo possa ser estendido.

As quatro subseções a seguir apresentam o suporte oferecido pelo StRES para esses quatro requisitos.

5.4.1 Repositório de Documentos

Em função da infra-estrutura implementada produzir documentos XML como resultado final do processo de captura, abordagens tradicionais baseadas no modelo relacional não são adequadas para o armazenamento da informação resultante.

No sentido de permitir o armazenamento eficiente de documentos XML foram desenvolvidos os chamados bancos de dados XML nativos. Essa denominação surgiu junto com o Tamino², um dos precursores dessa tecnologia. Em função da inexistência de uma definição formal, a comunidade de desenvolvedores que participam da lista de discussão XML:DB³ define um banco XML nativo como aquele que:

- define um modelo para documentos XML (não para os dados contidos

²<http://www1.softwareag.com/corporate/products/tamino/default.asp>

³<http://archive.xmldb.org>

nos documentos), e utiliza tal modelo para armazenar e recuperar documentos;

- tem como unidade fundamental de armazenamento (lógico) um documento XML;
- não apresenta necessariamente um modelo de armazenamento físico específico, podendo ser construído sobre um banco de dados relacional, orientado a objetos ou hierárquico, ou então, usar um formato de armazenamento proprietário.

Atualmente existe uma ampla variedade de bancos de dados XML nativos, tanto de código aberto quanto de código proprietário. Mesmo grandes fabricantes de bancos de dados relacionais, tais como Oracle e Microsoft, têm atualizado seus produtos a fim de prover facilidades para o armazenamento nativo de documentos XML.

Dentre os diversos bancos de dados XML nativos de código aberto, um em especial, denominado eXist⁴, se destaca por prover um conjunto de funcionalidades que facilitam a recuperação e a apresentação dos documentos XML resultantes das sessões de captura. Tais funcionalidades incluem o suporte à linguagem de consulta XQuery e à linguagem de atualização de documentos XUpdate, indispensáveis para a recuperação e extensão de conteúdo capturado; a disponibilização de uma API de acesso baseada no protocolo XML-RPC, a qual permite que aplicações escritas em diferentes linguagens possam acessar o banco; e a integração ao *Cocoon, framework* para desenvolvimento de aplicações Web que oferece suporte à transformações XSLT sob demanda.

No sentido de tirar proveito das funcionalidades providas pelo eXist, optou-se por construir o repositório de documentos utilizando esse banco de dados. O eXist, assim como outros bancos XML nativos, organiza os documentos em uma hierarquia de coleções, similar ao processo de armazenamento de arquivos em um sistema de arquivos convencional.

Conforme discutido na Seção 4.5.1, a organização dos documentos no StRES é determinada pelo documento *REPOSITORY.XML*, gerado durante o processo de modelagem das informações de contexto de cada aplicação. A criação da estrutura de coleções que contém os documentos produzidos por uma dada aplicação é feita pelo *Transducer*, após a geração dos documentos de captura.

Por exemplo, para a aplicação de captura de aulas o documento *REPOSITORY.XML* indica que a coleção raiz deve se chamar *iClass*, dentro dela deve

⁴<http://exist-db.org>

haver coleções com os semestres nos quais hajam aulas capturadas. Dentro da coleção referente a cada semestre, deve haver uma coleção para cada curso que teve aulas capturadas naquele semestre. Finalmente, dentro da coleção de cada curso deve haver os documentos XML que representam suas aulas capturadas.

Assim, quando um documento XML que representa uma aula do Curso de Estruturas de Dados, referente ao segundo semestre de 2004 é gerado, o *Transducer* acessa o banco e verifica se existe o seguinte caminho no banco de dados:

```
/iclass/2o_2004/EstruturasDeDados
```

Caso alguma das coleções não exista, ela é criada para que o documento possa ser armazenado adequadamente.

5.4.2 Recuperação de Conteúdo Capturado

A escolha do eXist para suportar o repositório de documentos de captura decorreu, sobretudo, da flexibilidade de acesso provida por esse banco. Além de implementar a API padrão para bancos de dados XML, denominada XML:DB⁵, o eXist ainda provê uma API baseada no protocolo XML-RPC, outra, denominada *REST-Style Web API*, baseada no protocolo HTTP e uma terceira baseada no protocolo SOAP.

Esse conjunto de interfaces de programação permite que o desenvolvedor tenha uma boa flexibilidade para implementar aplicações que manipulam o conteúdo armazenado. Os métodos incluídos nas diferentes APIs permitem operar sobre documentos XML e coleções, provendo operações de inclusão, exclusão e recuperação de recursos.

Além disso, o banco provê, em suas interfaces de acesso, facilidades para submissão de consultas na linguagem XQuery. A linguagem de consulta XQuery tem, para um banco XML, a mesma importância que a linguagem SQL para um banco relacional.

Apesar de serem suficientes para a construção de uma grande variedade de aplicações, as facilidades providas não exploram a semântica dos documentos armazenados. Os diversos repositórios mantidos no banco tem em comum informações de contexto, modeladas de acordo com o procedimento descrito no Capítulo 4.

⁵<http://xmldb.org>

No sentido de centralizar os serviços de acesso em uma única interface, optou-se por estender a API XML-RPC com o método *executeContextQuery*, responsável por submeter uma consulta contextual ao repositório de documentos. A assinatura desse método é mostrada a seguir.

```
Vector executeContextQuery(String cqlQuery) throws Exception
```

Esse método instancia um objeto da classe *CQLProcessor*, cujo funcionamento é apresentado na Seção 4.5.2. Essa classe define o método *processQuery*, que recebe a *string* contendo a expressão CQL e produz um vetor de *strings* como resultado.

Expressões de consulta inválidas geram *exceptions* que são repassadas para a classe chamadora. Em caso de sucesso, a função *executeContextQuery* retorna um vetor contendo referências para os documentos que atendem às condições impostas na expressão de consulta. Um vetor vazio indica que, apesar da expressão de consulta ser válida, nenhum documento atende às restrições da expressão de consulta.

De posse do conjunto de referências retornado, obter o conteúdo de cada documento requer somente a chamada do método *getDocument*, o qual utiliza a referência como parâmetro para recuperar o conteúdo do documento.

5.4.3 Apresentação de Conteúdo Capturado

A função primária de uma aplicação de captura e acesso é preservar memórias de experiências ao vivo. Dessa forma, todo o esforço associado à instrumentação de ambientes e ao desenvolvimento de dispositivos de captura e seus respectivos componentes de software objetiva permitir que a experiência capturada possa ser reproduzida no futuro.

Para que uma experiência ao vivo seja adequadamente registrada, deve-se capturar e indexar o maior número de fluxos de informação possível. Essa abundância de informação permite que as aplicações de apresentação possam prover diferentes formas de visualização do conteúdo capturado.

A abordagem xlnCA-StRES produz um documento rico em metadados, o que possibilita a geração de documentos de apresentação que exploram diferentes características das mídias capturadas. É possível produzir de maneira automática apresentações centradas no uso de um dispositivo, tal como a *whiteboard*, ou então privilegiar a interação entre os dados capturados pelos

diversos dispositivos de captura, provendo apresentações que mesclam áudio e/ou vídeo, com *slides*, texto e páginas *Web*.

A automatização do processo de geração de documentos de apresentação é possível em função da estrutura bem definida dos documentos de sessão, o que possibilita que folhas de estilo possam ser construídas para processar porções específicas dos documentos resultantes.

Para aplicações centradas no uso de *whiteboard*, que são as mais comuns para a maioria dos domínios suportados, foram desenvolvidas folhas de estilo capazes de transformar os documentos de captura em documentos de apresentação em diferentes formatos, incluindo HTML, SMIL e XHTML+SMIL. A Figura 5.3 ilustra os documentos de apresentação gerados a partir da transformação dos documentos de sessão.

Figura 5.3: Documentos de apresentação nos formatos XHTML+SMIL, HTML e SMIL (Baldochi et al. 2004).

Uma outra alternativa de apresentação consiste em fazer o *playback* da sessão utilizando um applet Java. Esse tipo de apresentação é adequado nos casos em que a aplicação de captura tenha utilizado um componente de *whiteboard*. O applet repete cada interação com o componente de captura da *whiteboard* tal qual foi feita durante a sessão de captura, podendo sincronizar a

animação com o áudio ou o vídeo da sessão.

Da Geração dos Documentos de Apresentação

O processo de transformação XSLT que produz os documentos de apresentação pode ocorrer imediatamente após a geração do documento XML que representa a sessão, ou posteriormente, quando o usuário quiser visualizar o conteúdo capturado. Cada abordagem apresenta vantagens e desvantagens.

A geração dos documentos de apresentação no momento da geração do documento XML foi uma abordagem usada nas primeiras implementações da infra-estrutura, retratadas em Baldochi et al. (2002) e Baldochi et al. (2003). Nessa abordagem, os documentos de apresentação eram produzidos pelo *Transducer*, logo após a geração do documento de sessão, e ficavam armazenados no sistema de arquivos, juntamente com os arquivos de mídia produzidos durante a captura.

Nesse caso, para apresentar a informação capturada em um determinado formato, bastava recuperar o documento de apresentação no formato desejado. Essa abordagem, embora rápida, faz com que uma mesma informação seja repetida em diferentes documentos. No caso, quatro documentos continham informações repetidas.

Uma outra desvantagem diz respeito à manutenção dos documentos de apresentação. Para manter um mesmo aspecto visual dos documentos de apresentação, toda vez que as folhas de estilo sofriam alguma alteração, era necessário gerar novamente todos os documentos de apresentação, o que agregava uma complexidade considerável à manutenção dos documentos.

Uma forma de tratar os dois problemas apresentados consiste em gerar os documentos de apresentação sob demanda. Nessa abordagem, o único documento que fica armazenado é o documento de sessão. Quando uma determinada sessão de captura precisa ser apresentada em um determinado formato, seu documento XML associado é processado sob demanda pela folha de estilo apropriada, produzindo o documento de apresentação.

O ponto desfavorável dessa abordagem é a sobrecarga trazida pelo processamento em tempo de execução. Entretanto, a ferramenta utilizada para suportar a geração de documentos sob demanda é capaz de produzir tais documentos em um tempo aceitável para os padrões de aplicações *Web*.

O Framework Cocoon

O mecanismo utilizado para suportar a geração sob demanda de documentos de apresentação utilizou o suporte do *Framework Cocoon*, o qual permite a separação clara entre lógica, conteúdo e estilo em aplicações *Web* (Apache 2005). Para permitir essa separação de interesses⁶, o *Cocoon* suporta a execução de *scripts* XSP (lógica) e o processamento de documentos XML (conteúdo) por folhas de estilo XSLT (estilo).

O *Framework Cocoon* permite servir arquivos estáticos, ou produzir dinamicamente o conteúdo servido. Quando produz conteúdo dinamicamente, o *Cocoon* é capaz de executar transformações XSLT com múltiplos passos, permitindo a passagem de parâmetros em cada transformação realizada. Como resultado dessas transformações, é possível produzir conteúdo em uma grande variedade de formatos, tais como HTML, WML, PDF, TXT, XML, SMIL, VRML, SVG, PNG, JPEG, entre outros.

Para permitir o processamento de documentos em múltiplos passos, o *Cocoon* utiliza o conceito de *pipeline*. Os documentos avançam na *pipeline* por meio de eventos SAX. Cada passo da *pipeline* produz algum conteúdo, em geral codificado em XML.

A Figura 5.4 ilustra uma *pipeline* de três passos que converte um arquivo XML em um arquivo de apresentação HTML. Inicialmente, o arquivo XML é processado pelo componente *File Generator* dando início à *pipeline*. Em seguida, o documento já em memória é processado pelo componente *XSLT Transformer*, associado a uma folha de estilo que gera conteúdo em HTML. Finalmente, para que um arquivo HTML possa ser produzido, utiliza-se um componente *HTML Serializer*.

Figura 5.4: Exemplo de *pipeline* do *Cocoon*.

A definição de cada *pipeline* e a declaração dos componentes a serem uti-

⁶O termo em inglês é *separation of concerns*

lizados são feitas em um arquivo de gerenciamento denominado *sitemap*. O trecho de código a seguir ilustra a estrutura básica de um arquivo *sitemap*, no qual é possível observar a presença dos elementos exibidos na Figura 5.4.

```
<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>
```

Os documentos XML utilizados no processamento de cada passo de uma *pipeline* podem já existir ou serem gerados dinamicamente. O *Cocoon* oferece suporte a linguagem de *script* XSP (XML Server Pages). Assim como JSP suporta a inclusão de código Java em páginas HTML, XSP suporta a inclusão de código Java em documentos XML. Dessa forma, é possível gerar documentos XML dinamicamente utilizando *scripts* XSP.

A possibilidade de executar código Java na construção de documentos XML permite que as APIs providas pelo banco eXist possam ser utilizadas dentro de arquivos XSP. Dessa forma, é possível escrever *scripts* XSP capazes de obter informações mantidas no banco, o que possibilita, por exemplo, gerar um documento XML a partir de dois ou mais arquivos armazenados. Outra possibilidade é utilizar as informações de estruturação do banco (a estrutura de coleções por exemplo) para construir documentos XML.

Explorando essa funcionalidade, implementou-se um *sitemap* que define um conjunto de *pipelines* capazes de produzir, sob demanda, páginas HTML nas quais é possível navegar na estrutura de coleções de cada repositório. Como essa estrutura é definida a partir da modelagem conceitual de cada aplicação, a navegação possui semântica associada, conforme ilustra o exemplo na Figura 5.5.

O exemplo exibido refere-se a um repositório de aulas capturadas. A Figura 5.5 (a) exhibe os semestres que possuem aulas capturadas. Escolhendo um semestre, obtém-se a página mostrada na Figura 5.5 (b), a qual lista os cursos do semestre escolhido que tiveram aulas capturadas. A seleção de um curso leva à página mostrada na Figura 5.5 (c), que exhibe as aulas capturadas para a disciplina selecionada, provendo ligações para as diferentes opções de visualização providas.

Cocoon faz *caching* de páginas. Dessa forma, uma página só é efetivamente gerada quando não se encontra no *cache*. Mesmo quando a página deve ser gerada, o tempo de resposta é bastante próximo dos casos em que o documento já está pronto para ser servido.

O comportamento dessa aplicação Web é determinado pelo seu *sitemap*. Cada ação do usuário é tratada por uma *pipeline* específica. Por exemplo, ao selecionar a hiperligação correspondente a um dado semestre, é necessário criar a página que contém os cursos do semestre selecionado. Isso é feito por um *script* XSP que chama um método Java responsável por listar as coleções filhas da coleção selecionada. Em seguida, constrói-se um documento XML contendo elementos que encapsulam os nomes das coleções filhas. Esse documento é processado por uma folha de estilo, gerando um arquivo HTML que contém uma hiperligação para cada coleção contida no arquivo XML.

O mecanismo de apresentação descrito complementa a abordagem xInCA-StRES permitindo a automatização do procedimento de produção de conteúdo, a organização desse conteúdo em uma estrutura hierárquica obtida a partir do modelo conceitual da aplicação, e a visualização desse conteúdo na Web em diferentes formatos.

Além da aplicação Web automaticamente produzida, é possível também visualizar conteúdo selecionado por meio de consultas contextuais. Para tanto, foi implementada uma segunda aplicação Web, também baseada no *Framework Cocoon*, na qual é possível especificar uma consulta contextual e obter como resposta uma página idêntica à mostrada na Figura 5.5 (c). Essa aplicação é descrita a seguir.

Aplicação Web Para Consultas Contextuais

O processador CQL foi desenvolvido como uma extensão ao banco de dados eXist, podendo ser acessado por qualquer programa que implemente o protocolo XML-RPC, o qual é suportado por um grande número de linguagens de programação. Essa abordagem foi adotada a fim de flexibilizar o acesso ao conteúdo capturado.

Ainda assim, o desenvolvedor de aplicações que utiliza a infra-estrutura InCA-SERVE conta com uma aplicação Web que permite a submissão de consultas contextuais ao repositório de documentos. O formulário para submissões de consultas⁷, é composto de uma caixa de texto na qual as consultas devem ser digitadas. Ao lado da caixa de texto é exibido o esquema XML

⁷Disponível em: <http://iclass.icmc.usp.br/cqlprocessor>

que formaliza cada consulta — esse esquema resulta do processo de transformação definido pelo *Contextractor*.

Essa aplicação foi implementada com o suporte do *Framework Cocoon*. A *pipeline* utilizada cuida do processamento de cada consulta produzindo uma página na qual o usuário é capaz de visualizar os documentos que atendem às restrições impostas na expressão de consulta.

O processamento do conteúdo digitado na caixa de texto é feito por uma página XSP, a qual embute código Java que utiliza as classes e métodos providos pelo processador CQL. O resultado da consulta produz um fragmento XML composto pelas referências aos documentos que atendem às solicitações especificadas. Esse código XML é então processado na *pipeline* por uma folha de estilo que gera uma página equivalente à exibida na Figura 5.5 (c).

5.4.4 Extensão de Conteúdo Capturado

Existem duas formas de extensão do conteúdo capturado mantido pelo StRES. Uma delas envolve alterar os documentos XML inserindo conteúdo adicional em partes específicas dos documentos. A anotação de *slides* capturados, por exemplo, envolve a alteração do elemento `<slide>` presente no documento de sessão. A outra consiste no estabelecimento de hiperligações entre documentos. Essa forma de extensão permite explorar relacionamentos que não podem ser expressos somente utilizando informações de contexto.

Anotação de documentos XML

A necessidade de anotar conteúdo capturado está ligado às características de cada aplicação. Dessa forma, o suporte a anotações encontra-se na forma de operações genéricas que permitem adicionar anotações textuais aos documentos como um todo, ou a partes específicas do documento — no momento, apenas os *slides* podem receber anotações.

Assim como a consulta contextual, as operações de extensão foram implementadas como extensões da API XML-RPC que expõe o repositório de documentos. A assinatura dos dois métodos que executam operações de extensão são as seguintes:

```
boolean updateDocument(String docName, String annotation)
```

```
boolean updateSlide(String docName, String slideId, String annotation)
```

Esses métodos inserem o elemento *<annotation>* com o conteúdo contido na *string* passada como parâmetro. No caso do primeiro método, esse elemento é inserido como filho do elemento *<session>*. No caso do segundo, como filho do elemento *<slide>*.

Extensão por meio de hiperligações

O repositório de documentos capturados por aplicações tais como o iClass, apresentado na Seção 6.5, contém um volume expressivo de informação, a qual apresenta interrelacionamentos não perceptíveis no modelo conceitual da aplicação e que, portanto, não podem ser expressos unicamente pelas informações de contexto modeladas.

No sentido de explorar esses interrelacionamentos foi desenvolvida a infraestrutura *CALiSP — infrastructure for Capturing, Accessing, Linking, Storing and Presenting information* (Macedo, Baldochi, Camacho-Guerrero, Cattelan, and Pimentel 2005). Essa infra-estrutura reúne as funcionalidades presentes na infra-estrutura InCA-SERVE juntamente com o *LinkDigger*, serviço de ligações descrito na Seção 3.5.3.

Usando as funcionalidades de ligação do *LinkDigger*, o qual recebe a denominação de *Link Builder* na *CALiSP*, é possível criar uma base de hiperligações que associa documentos em função da similaridade de seus conteúdos textuais. A base criada constitui uma forma de estender cada documento relacionado, permitindo navegar entre documentos semanticamente associados.

5.5 Considerações Finais

Apesar do grande número de iniciativas de pesquisa em curso, o suporte ao desenvolvimento de aplicações de computação ubíqua encontra-se ainda em sua infância. Os resultados até o momento obtidos formam ilhas isoladas capazes de servir aplicações específicas (Streitz and Nixon 2005).

A infra-estrutura InCA-SERVE busca aproximar essas ilhas, suportando o desenvolvimento de aplicações de captura e acesso dos mais diferentes domínios. Para tanto, cuida de diferentes requisitos associados à implementação de aplicações. Um dos principais requisitos tratados refere-se ao gerenciamento de informação produzida e manipulada pelas aplicações, tema central desta tese.

Para ser efetivo, o suporte ao gerenciamento da informação deve tratar

desde questões de modelagem e estruturação dos dados até o acesso e extensão dos mesmos, passando por questões associadas ao seu armazenamento. Esse cenário indica a necessidade de suporte infra-estrutural para a produção da informação e suporte de serviços para facilitar acesso, apresentação e extensão do conteúdo produzido.

O StRES, apresentado em detalhes nesta seção, oferece o suporte infra-estrutural e de serviços de maneira a atender os requisitos discutidos. Sua utilização permite minimizar e, em certos casos, até suprimir, os esforços de implementação necessários para armazenar, recuperar e apresentar conteúdo capturado. Além disso, oferece facilidades para consulta baseada em informações de contexto e operações que permitem estender o conteúdo capturado.

No sentido de prover esse conjunto de facilidades às aplicações, o StRES explora ferramentas de suporte ao processamento de informação codificada em XML, tais como mecanismos de *data binding*, processadores XSLT e bancos de dados XML, em sua maioria, ferramentas de código aberto. Essas ferramentas possibilitaram acelerar o processo de desenvolvimento das facilidades providas, uma vez que permitiram o aumento do grau de abstração na construção da própria infra-estrutura e no conjunto de serviços disponibilizado.

Obviamente a solução adotada também apresenta desvantagens. A maior delas está associada a questões de eficiência na recuperação e na apresentação do conteúdo capturado. Bancos de dados XML nativos utilizam, de maneira geral, modelos e algoritmos proprietários de armazenamento os quais não apresentam o grau de otimização já alcançado em bancos relacionais.

Outro aspecto também associado ao banco de dados refere-se à linguagem de consulta XQuery, utilizada pelo processador CQL para recuperar conteúdo no banco XML. Por se tratar de uma tecnologia nova, sua implementação é ainda experimental no banco de dados utilizado.

Por fim, o uso do *Framework Cocoon* como suporte à aplicação Web que apresenta conteúdo capturado pode também trazer problemas de eficiência, já que as páginas Web são geradas sob demanda em um processo de transformação XSLT.

Apesar de constituírem problemas em potencial, as aplicações construídas até o momento não apresentaram limitações em função dessas decisões de projeto. Por outro lado, esses problemas tendem a ser minimizados em função da ampla adoção da linguagem XML, o que tem motivado a investigação de maneiras mais eficientes para armazenamento e consulta.

O Capítulo 6, a seguir, detalha os passos envolvidos na construção de apli-

cações com o uso da infra-estrutura InCA-SERVE e apresenta um amplo conjunto de aplicações já desenvolvidas com o suporte infra-estrutural provido. Com base nas aplicações apresentadas, é feito, ao final do Capítulo 6, uma comparação entre a infra-estrutura desenvolvida e outros trabalhos reportados na literatura.

Construção de Aplicações

6.1 *Considerações Iniciais*

A infra-estrutura InCA-SERVE tem como objetivo principal prover mecanismos capazes de elevar o grau de abstração na construção de aplicações de captura e acesso. Para tanto, oferece um modelo para o desenvolvimento dessas aplicações suportado por uma infra-estrutura e um conjunto de serviços, os quais foram descritos nos Capítulos 3, 4 e 5.

O desenvolvimento de aplicações usando a infra-estrutura InCA-SERVE pressupõe sua correta instalação e configuração. Por depender de um conjunto de ferramentas publicamente disponíveis e de código aberto que precisam ser adequadamente configuradas, foi necessário desenvolver um procedimento de instalação e configuração a fim de permitir a distribuição da infra-estrutura.

Dessa forma, este capítulo descreve inicialmente esse procedimento de configuração na Sessão 6.2. Em seguida, apresenta na Seção 6.3 um estudo de caso que detalha de que forma as diversas APIs, serviços e metodologias providos pela infra-estrutura InCA-SERVE são usados na construção de aplicações.

Nas Seções 6.4 a 6.7 são apresentadas algumas das várias aplicações desenvolvidas com suporte infra-estrutural ao longo dos últimos três anos. Entre elas, o *iClass* (Baldochi et al. 2003) tem papel de destaque, uma vez que seu

uso contínuo desde 2002 tem gerado uma retroalimentação importante para o refinamento da infra-estrutura e para o aperfeiçoamento dos mecanismos de gerenciamento da informação capturada.

Trabalhos mais recentes têm investigado o uso de uma variação do *iClass*, denominada *wiClass*, na captura de atividades fora do ambiente instrumentado da sala de aula (Pimentel, Baldochi, and Cattelan 2005). O *wiClass* é capaz de operar em dispositivos móveis com conectividade reduzida, intermitente ou até mesmo desconectados. Explorando o uso do *wiClass* em *tablet PCs*, docentes das mais diferentes áreas — arquitetura, química, biologia, farmácia — têm utilizado o *wiClass* na captura de aulas de laboratório e em atividades de campo. O uso do *wiClass* fora do ambiente de sala de aula é relatado nas Seções 6.7.1 e 6.7.2.

A infra-estrutura InCA-SERVE é uma entre diversas iniciativas que objetivam auxiliar o desenvolvimento de aplicações de computação ubíqua por meio de algum tipo de suporte infra-estrutural. Dessa forma, a Seção 6.8 apresenta as similaridades e diferenças da abordagem utilizada em relação a outras reportadas na literatura. Finalmente, a Seção 6.9 traz as considerações finais do capítulo.

6.2 Instalação e configuração da infra-estrutura InCA-SERVE

A construção da infra-estrutura InCA-SERVE envolveu esforços de investigação e desenvolvimento em diferentes áreas da computação, incluindo sistemas distribuídos, sistemas hipermídia, engenharia de software e banco de dados. Uma parte significativa dos resultados obtidos exploram ferramentas publicamente disponíveis, tais como mecanismos de *data binding*, *parsers* e bancos de dados XML.

Dessa forma, o funcionamento da infra-estrutura InCA-SERVE está associado à disponibilidade desse conjunto de softwares. No sentido de facilitar a instalação e a configuração da infra-estrutura e de seus softwares de apoio foi definido um procedimento cujos passos são:

1. Geração do esquema de contexto da aplicação;
2. Compilação do esquema de contexto e atualização da infra-estrutura;
3. Instalação e configuração dos softwares de apoio.

A geração do esquema de contexto conta com o apoio do *Contextractor*, discutido no Capítulo 4. O esquema resultante deve ser processado por um mecanismo de *data binding* produzindo classes que constituem blocos arquiteturais da infra-estrutura InCA-SERVE e, portanto, devem ter seu código integrado ao arquivo de distribuição da infra-estrutura. Esse procedimento não é trivial, uma vez que envolve a atualização de um arquivo *jar*.

A fim de permitir que o passo 2 do procedimento de atualização seja feito de maneira automática, foi desenvolvido o *Context Compiler*, software responsável por compilar o esquema de contexto e fazer as atualizações necessárias no arquivo de distribuição da infra-estrutura.

Além do *Context Compiler*, uma outra ferramenta foi também disponibilizada no sentido de permitir a instalação e a configuração dos softwares de apoio (passo 3). O papel dessa ferramenta, denominada DBExtender, é atualizar a versão de instalação do banco de dados XML eXist de forma que proveja os serviços de consulta contextual, extensão e apresentação de conteúdo capturados discutidos na Seção 5.4.

Uma descrição detalhada do procedimento de instalação e configuração da infra-estrutura InCA-SERVE encontra-se disponível no Apêndice B.

6.3 Remote Meeting: Um Estudo de Caso

Esta seção descreve a implementação do *Remote Meeting*, na qual são exploradas todas as facilidades de modelagem, infra-estrutura e serviços no desenvolvimento de uma aplicação de suporte a reuniões distribuídas, voltadas para a discussão de projetos no contexto de uma empresa de desenvolvimento de *software*.

A função do *Remote Meeting* é permitir a discussão de projetos de *software* entre usuários remotos. Não faz parte dos requisitos atendidos por essa versão prover facilidades para o *design rationale* de sistemas, nem tão pouco suportar todo o processo de modelagem de uma aplicação de maneira distribuída.

Casos de uso do *Remote Meeting* envolvem a discussão entre dois desenvolvedores de um diagrama de classes UML já pronto, com a finalidade de tornar claro algum aspecto da modelagem. Além de fazer *upload* de diagramas, o *Remote Meeting* também permite subir trechos de código para discussão. Dessa forma, a ferramenta auxilia o processo de desenvolvimento, permitindo que o programador tire dúvidas com o analista sem ter que interromper a atividade de desenvolvimento. As discussões ficam gravadas, a fim

de registrar as decisões de projeto tomadas.

O *Remote Meeting* pode também ser utilizado no desenvolvimento de projetos de *software* livre, nos quais os desenvolvedores trabalham em locais distantes, muitas vezes em países diferentes. Nesses casos, a discussão on-line de questões de projeto pode complementar o e-mail, otimizando o processo de desenvolvimento de *software*.

O *Remote Meeting* possui duas funcionalidades: *whiteboard* e *chat*. Utilizando o componente de *whiteboard* o usuário pode fazer *upload* de figuras e escrever sobre as mesmas. O componente de *chat* permite a troca de mensagens textuais, que ficam associadas às figuras correntemente exibidas.

O conteúdo produzido pelo *Remote Meeting* fica automaticamente acessível através da aplicação de apresentação provida pelo StRES, a qual permite também a consulta contextual do conteúdo capturado.

As seções a seguir detalham os passos no desenvolvimento do *Remote Meeting*, iniciando com a modelagem das informações de contexto, seguindo com a instalação e configuração da infra-estrutura e seus *softwares* de apoio, passando pela implementação do *front-end* da aplicação na forma de um *applet* Java e finalizando com o desenvolvimento de aplicações de acesso customizadas.

6.3.1 Modelagem da Informação de Contexto

Este estudo de caso adota o pressuposto que a aplicação de captura será utilizada no suporte a atividades de desenvolvimento de *software* em uma empresa que mantém dados referentes a funcionários, departamentos e projetos em uma base de dados relacional, a modelagem das informações de contexto da aplicação pode ser obtida de acordo com o procedimento descrito no Capítulo 4. A Figura 6.1 ilustra o diagrama de classes, a partir do qual serão extraídas as informações de contexto do *Remote Meeting*.

O diagrama mostrado na Figura 6.1 apresenta a seguinte associação entre atributos e dimensões contextuais:

- Dimensão *who*: atributo *id* da classe *Funcionario*;
- Dimensão *where*: atributo *id* da classe *Departamento*;
- Dimensão *when*: Atributo *date* da classe *Sessao*;

A dimensão *what* é definida a partir da relação de composição *tem*, a qual

Figura 6.1: Diagrama de classes para uma aplicação de reuniões informais distribuídas.

une as classes interna e externa do modelo, *Projeto* e *Reuniao*, respectivamente, conforme discutido na Seção 4.4.3.

Partindo do documento XMI produzido pelo ArgoUML e seguindo a sequência de transformações estabelecida pelo *Context extractor*, obtém-se os 3 esquemas que permitem processar as informações de contexto nas fases de captura e acesso da aplicação. É necessário também definir o documento *REPOSITORY.XML*, de forma a possibilitar a estruturação do repositório de documentos. Nesse exemplo, a estruturação utilizada consiste em agrupar os documentos de sessão por projetos.

O esquema gerado deve ser utilizado no procedimento de atualização da infraestrutura, descrito na Seção 6.2. A partir dessa atualização, o *front-end* da aplicação pode ser construído.

6.3.2 Implementação do Front-end da Aplicação

A xInCA estende o modelo de desenvolvimento de aplicações da InCA, no qual as aplicações devem instanciar módulos de captura e/ou acesso e implementar as interfaces de programação desses módulos. Ao implementar tais

interfaces, as aplicações são obrigadas a declarar os métodos da mesma, que são do tipo *callback* — por meio desses métodos a aplicação recebe notificações da infra-estrutura.

A principal diferença entre os *callbacks* da InCA e da xInCA é que os primeiros transmitem objetos genéricos (*DataObjects*), enquanto os últimos manipulam objetos produzidos pelo processo de *data binding* provido pelo JAXB.

Para o desenvolvimento da aplicação *Remote Meeting*, o passo inicial consiste na definição de uma classe Java que implemente as interfaces de programação dos componentes de chat e de *whiteboard*.

```
public class RemoteMeeting extends JApplet implements XBoardListener,  
XChatListener, StorageListener
```

O próximo passo é fazer a instanciação dos dois componentes de *software*. Isso pode ser feito em qualquer método da classe *RemoteMeeting*, até mesmo em seu construtor. A assinatura do principal método construtor do componente *XChat* é a seguinte:

```
public XChat(int capSessionID, int accSessionID,  
XChatListener xChatListener, String incadHost,  
int incadPort, String userName, boolean captureOn,  
boolean accessOn)
```

O entendimento dessa assinatura é tudo que o desenvolvedor necessita para utilizar os componentes providos. É importante notar que esse conjunto de parâmetros reforça os detalhes arquiteturais da infra-estrutura discutidos anteriormente. Por exemplo, os parâmetros *capSessionID* e *accSessionID* revelam que um mesmo componente pode ser usado em sessões distintas — pode capturar dados para uma dada sessão e acessar (receber) dados de uma outra sessão. Essa característica dos componentes xInCA é discutida na Seção 3.4.2.

Outro parâmetro importante é o *xChatListener*, o qual é usado para identificar a classe que deverá receber os *callbacks* produzidos pela infra-estrutura. Assim como ocorre com a maioria das aplicações, o *Remote Meeting* passa o valor *this* como parâmetro, indicando que a própria classe *RemoteMeeting* deverá receber os *callbacks*.

O parâmetro *userName* é usado para identificar o usuário que estará utilizando o componente. Essa informação é, normalmente, obtida a partir da

modelagem das informações de contexto da aplicação. No caso do componente de *chat*, o *userName* é utilizado para identificar os autores de cada frase recebida durante a conversação.

Os parâmetros *captureOn* e *accessOn* servem para habilitar e desabilitar as opções de captura e acesso. Um componente pode ser usado para capturar informação, acessar a informação capturada por outros componentes, ou ambos.

Finalmente, os parâmetros *incadHost* e *incadPort* servem para identificar o *Registry* que deverá ser utilizado. É importante notar que diferentes *Registries* podem estar operando simultaneamente, até mesmo para permitir uma distribuição da carga do servidor. Por isso os construtores dos componentes *xInCA* permitem que o desenvolvedor escolha o *Registry* ao qual deseja se conectar.

Detalhes infra-estruturais, tais como a escolha do *Registry*, podem ser abstraídos em ambientes nos quais haja apenas um *daemon* da InCA. Caso o desenvolvedor opte por um controle minimalista sobre o ambiente de execução, é possível utilizar um construtor com apenas quatro parâmetros. Nesse caso, é necessário informar apenas o *userName*, o *listener (this)*, e os parâmetros que habilitam/desabilitam a captura e o acesso. As demais informações são obtidas de diferentes fontes: *incadHost* e *incadPort* podem ser lidos do arquivo *config.xml*, enquanto *capSessionID* e *accSessionID* são gerados automaticamente pela própria infra-estrutura.

O construtor do componente de *whiteboard* possui dois parâmetros a mais que o componente de *chat*. O primeiro informa as dimensões em *pixels* das imagens que serão geradas para cada *slide* produzido durante a captura. O segundo serve para alimentar a *whiteboard* com um vetor de imagens, que são apresentadas como *slides* pré-produzidos, sobre os quais é possível anotar.

As funcionalidades de navegação entre *slides*, criação, remoção e duplicação de *slides*, ajuste das cores e espessura dos traços da caneta são providas por uma classe denominada *XBoardToolbar*. A decisão de implementar uma barra de ferramentas independente do componente *XBoard* visa permitir que esse componente opere com diferentes ferramentas de acordo com as necessidades de cada aplicação. Operações de customização do componente são facilitadas, já que não impactam o funcionamento da infra-estrutura, uma vez que a funcionalidade de captura e acesso fica isolada do processo de customização por estar na classe *XBoard*.

A aplicação Caderno On-Line, apresentada na Seção 6.6, é um exemplo

que utiliza uma customização do componente de *whiteboard* na qual botões que permitem o desenho de figuras geométricas foram adicionados à barra de ferramentas.

A abordagem utilizada na componente de *whiteboard* ressalta a preocupação em prover flexibilidade ao processo de desenvolvimento (Cattelan, Baldochi, Pimentel, and Truong 2003). Explorando as facilidades providas pelo pacote *javax.swing*, o desenvolvedor tem a possibilidade de criar diferentes opções de interface. É possível, por exemplo, posicionar a barra de ferramentas acima, abaixo ou em uma das laterais da XBoard. Pode-se decidir em posicionar os componentes de captura (*whiteboard* e *chat*) um acima do outro, ou lado a lado. Uma opção interessante, que provê flexibilidade ao usuário, consiste em produzir janelas disjuntas para cada componente. Nessa opção, o *applet* cria telas do tipo *pop-up* nas quais os componentes de captura são executados. A Figura 6.2 ilustra essa abordagem.

Figura 6.2: *Remote meeting* em ação: componentes executando em janelas independentes.

O aplicação mostrada na Figura 6.2 é basicamente formada pela instanciação das três classes apresentadas (*XBoard*, *XChat* e *XBoardToolbar*) embutidas em componentes de interface. Isso é suficiente para fazer do *RemoteMeeting* uma aplicação de captura e acesso. Se duas ou mais instâncias dessa aplicação forem ativadas com um mesmo identificador de sessão

(mesmo valor para os parâmetros *capSessionID* e *accSessionID* e todas referenciarem o mesmo *Registry*, a comunicação entre seus componentes será automaticamente habilitada, permitindo a troca de *strokes* e o intercâmbio de mensagens entre os *peers* da aplicação.

Nota-se, entretanto, que a informação produzida pela aplicação é perdida, já que as facilidades de armazenamento providas pelo StRES não estão sendo utilizadas pela aplicação. O procedimento necessário para permitir o armazenamento automático das informações capturadas pelo *Remote Meeting* é apresentado a seguir.

Armazenando e recuperando conteúdo capturado

O procedimento necessário a fim de permitir que a aplicação *Remote Meeting* seja capaz de armazenar o conteúdo capturado por seus componentes envolve a instancição de um objeto da classe *StorageComponent*. A assinatura do construtor dessa classe é a seguinte:

```
public StorageComponent(PeerContext peerContext)
```

A classe *StorageComponent* cumpre apenas um papel: comunicar-se com o *Sessiond*, *daemon* responsável por controlar a instancição de objetos *Transducer*. Durante essa comunicação, a qual ocorre por meio de um *socket*, o *StorageComponent* passa o objeto *PeerContext* para o *daemon*. O *PeerContext* determina o comportamento do *Sessiond*, podendo fazer com que um novo *Transducer* seja instanciado, ou implicando na junção de um *peer* a um *Transducer* existente.

PeerContext é a única classe do pacote *common.data* não produzida pelo processo de *binding XML*. Sua função é encapsular objetos do tipo *ContextType*, juntamente com informações adicionais que caracterizam uma sessão de captura, as quais incluem o identificador de sessão (*sessionId*), uma constante que identifica os componentes de captura utilizados pela aplicação (*sessionType*) e uma string identificando o caminho (*path*) até a coleção na qual o documento XML produzido na sessão de captura deve ser armazenado. A semântica desse caminho é dada pelo documento *REPOSITORY.XML* produzido no procedimento de modelagem da aplicação.

A Classe *ContextType*, assim como todas as classes produzidas no processo de *binding XML*, é composta por um conjunto de métodos *get* e *set* por meio dos quais os objetos das classes *WhatType*, *WhoType*, *WhereType* e *WhenType*

se interligam a *ContextType*. Da mesma forma, cada uma das classes interligadas possuem métodos *get* e *set*, os quais são usados para ajustar (*set*) e obter (*get*) o conteúdo de cada dimensão contextual. Assim, toda aplicação de captura deve obter e ajustar as informações de contexto utilizando esses métodos.

No caso do *Remote Meeting*, as informações de contexto são obtidas a partir de uma interface *Web* acessível a partir de um procedimento de autenticação permitido apenas para desenvolvedores (participantes de algum projeto). Após a autenticação, o usuário visualiza a interface mostrada na Figura 6.3, na qual deve escolher entre participar de uma reunião iniciada por outro usuário, ou iniciar uma nova reunião. Se escolher a primeira opção, o *combo box* à esquerda seleciona as sessões nas quais o usuário tem permissão de participar. Basta, então, escolher uma dessas sessões e clicar no botão *Join meeting* para se juntar à reunião escolhida.

Figura 6.3: *Remote meeting*: formulário de configuração

Caso a escolha seja por iniciar uma sessão, o usuário deverá preencher um título para a reunião e escolher o projeto ao qual a nova reunião se refere. As escolhas feitas nessa interface permitem a definição das informações de contexto que farão parte do objeto *ContextType*. Após a obtenção das infor-

mações, o usuário ativa o botão *Start New Meeting*. Essa ação ativa o *applet* de captura, passando as informações coletadas como parâmetros.

Independentemente da opção escolhida pelo usuário (iniciar nova sessão ou participar em sessão já existente), as mesmas informações de contexto são encapsuladas no objeto *contextType* e passadas para o *Sessiond* pelo componente de armazenamento. O que determina a instanciação de um novo *Transducer* ou a utilização de um *Transducer* existente é o identificador de sessão. Caso o novo *peer* informe um identificador de sessão já utilizado por outro *peer*, a junção é automática. Para um novo identificador, um novo objeto *Transducer* é instanciado. A Figura 6.4 mostra os detalhes arquiteturais da aplicação *Remote Meeting* com dois *peers* ativos.

Figura 6.4: *Remote meeting*: visão arquitetural

Supondo que a *Aplicação 1* da Figura 6.4 foi a primeira a passar para o *Sessiond* o identificador de sessão utilizado, sua ação resultou na instanciación do *Transducer*. Ao passar o mesmo identificador de sessão para o *Sessiond*, a *Aplicação 2* faz um *join* no *Transducer* anteriormente instanciado. Obviamente quem determina a junção é o *Sessiond*, pois mantém uma lista dos *Transducers* ativos e seus respectivos identificadores de sessão associados. A comunicação entre a *Aplicação 1* e a *Aplicação 2* é transparentemente provida pelo *Registry* da InCA, que cuida também de enviar as informações produzidas por cada *peer* para o *Transducer*.

O *Remote Meeting*, assim como todas as aplicações desenvolvidas utilizando a metodologia descrita, dispõe da aplicação *Cocoon* que permite a navegação pelo banco de dados e a apresentação do conteúdo capturado. Cabe ao desenvolvedor somente a preparação das folhas de estilo utilizadas na geração dos diferentes formatos de visualização.

6.4 CAT

O modelo de desenvolvimento de aplicações provido pela infra-estrutura implementada exige conhecimento da linguagem de programação Java. No sentido de permitir que usuários não programadores sejam capazes de especificar e utilizar uma aplicação de captura e acesso desenvolveu-se o CAT — Capture and Access Toolkit (Baldochi et al. 2002).

O CAT foi o primeiro esforço no sentido de demonstrar a efetividade da infra-estrutura InCA-SERVE na construção de aplicações de captura e acesso. Explorando a arquitetura modular provida pela xInCA, esse *toolkit* permite a "montagem" de aplicações através do arranjo de componentes em interfaces Web.

O CAT é basicamente composto de interfaces Web que permitem a customização de aplicações de captura e acesso. Utilizando o formulário mostrado na Figura 6.5, um usuário pode especificar quais funcionalidades deseja ter em sua aplicação. O usuário deve ainda fornecer as informações de contexto, que envolvem o título da sessão, a identificação dos participantes, o tipo (local ou distribuída) e o controle de acesso (pública ou privada).

Após configurar a aplicação, o usuário recebe um *applet* (*front-end*) com as funcionalidades requeridas. No lado do servidor, um *Transducer* é instanciado com componentes de acesso equivalentes aos componentes de captura presentes no *front-end*. É importante notar que a configuração de uma apli-

Figura 6.5: Interface para configuração de aplicações (Baldochi et al. 2002).

cação no CAT está atrelada à definição de uma sessão, uma vez que a ação de configuração de uma aplicação sempre resulta na geração de uma sessão de captura.

As informações referentes a usuários (*Participants*) e lugares (*Place*) são mantidas em documentos XML gerenciados pelo StRES. O repositório de documentos produzido pelo CAT é organizado de maneira temporal. A coleção de documentos raiz (CAT) possui coleções que representam anos, dentro das quais existem coleções que representam meses. Dessa forma, as sessões capturadas em um dado mês ficam armazenadas dentro de uma mesma coleção, o que facilita os mecanismos de busca que atuam sobre o restante da informação contextual presente nos documentos. Essa abordagem é semelhante àquela utilizada no sistema DUMMBO, apresentado na Seção 2.5.4.

Assim como ocorre com a aplicação *Remote Meeting*, aplicações configuradas com o CAT permitem que usuários juntem-se a sessões em andamento. Fazendo *join* em uma sessão existente, o usuário herda as características dessa sessão, recebendo um *front-end* idêntico ao dos demais participantes. Existem restrições no procedimento de junção: um novo usuário só pode fazer junção caso a sessão tenha sido configurada como pública por seu dono (*owner* — aquele que originou a sessão). No caso de uma sessão privada, o novo usuário deve fazer parte da lista definida pelo proprietários da sessão. Também é o criador da sessão que determina o seu fim: quando ele realiza fi-

naliza sua aplicação a sessão como um todo é finalizada, gerando o documento XML que a representa.

6.5 iClass

A implementação do CAT demonstrou a efetividade da infra-estrutura InCA-SERVE no suporte à construção de aplicações de captura e acesso. Entretanto, uma avaliação mais significativa dos benefícios trazidos por essa abordagem requer mais que simplesmente analisar as facilidades providas na construção de aplicações. É necessário acompanhar a evolução de aplicações em um período de tempo longo, no qual alterações de menor ou maior porte são necessárias a fim de adequar as aplicações a novas necessidades que aparecem no decorrer do tempo.

Por três anos a infra-estrutura InCA-SERVE tem suportado o desenvolvimento do sistema *iClass* (Cattelan et al. 2002; Baldochi et al. 2003; Baldochi et al. 2004), aplicação de captura de aulas presenciais utilizado no ICMC-USP. Desde que entrou em funcionamento, o *iClass* foi utilizado na captura de mais de 300 aulas de disciplinas de graduação e pós-graduação. Durante esse tempo, a infra-estrutura implementada tem se mostrado robusta o suficiente para suportar os ajustes que se fizeram necessários na aplicação ao longo do tempo.

O *iClass* segue as fases sugeridas por Abowd et al. (1996) para estruturar o problema de captura e acesso. Conforme ilustra a Figura 6.6, são quatro as fases em que se divide uma aplicação: pré-produção, gravação ao vivo, pós-produção e acesso. Nota-se, observando a figura, que os blocos arquiteturais providos pela xInCA e pelo StRES são usados no suporte a cada uma dessas fases, as quais são detalhadamente descritas a seguir.

6.5.1 Pré-produção

A fase de pré-produção está associada à configuração da aplicação para que ela possa ser corretamente iniciada. De maneira geral, nessa fase as aplicações fazem a obtenção das informações de contexto que qualificam a sessão. Como essas informações são específicas do domínio da aplicação, desenvolveu-se um mecanismo capaz de associar dimensões de contexto às entidades do modelo conceitual da aplicação, conforme apresentado no Capítulo 4.

Figura 6.6: A Arquitetura do sistema *iClass* (Baldochi et al. 2003).

O documento *CONTEXT-INFO.XSD*, gerado a partir do mecanismo implementado, determina quais informações de contexto devem ser obtidas na fase de pré-produção. No caso do *iClass*, essas informações incluem o título da aula e o identificador do curso ao qual essa aula pertence.

A Figura 6.7 ilustra o formulário Web¹ construído para obter as informações de contexto do sistema *iClass*. Vale notar que esse formulário não referencia pessoas ou lugares. Isso ocorre porque as informações referentes ao instrutor e aos alunos podem ser obtidas a partir do curso, que é selecionado de um *combo box*, o qual, por sua vez, é gerado a partir de informações mantidas em um banco de dados acadêmico.

As informações de contexto obtidas são utilizadas na ativação do *applet* de captura, o qual instancia os componentes *xINCA* de captura e/ou acesso, bem como o *StorageComponent*, responsável por ativar no servidor as classes encarregadas de realizar o armazenamento das informações produzidas. No momento da ativação desse *applet* inicia-se a fase de gravação ao vivo.

¹O conjunto de formulários que permitem preparar uma sessão de captura foram desenvolvidos no contexto do trabalho de mestrado do aluno Daniel G. Sante (Sante 2003; Sante et al. 2004).

The screenshot shows a web browser window with the iClass application. The navigation bar at the top contains the following links: **AutorE class**, **Preparation**, **Reference**, **Expansion**, **iClass**, and **Exit**. The main content area is titled **Preparation Module** and contains a **New Lecture** form. The form has the following fields and values:

- Course:** Multimedia (SI 4275)
- Section:** Podcasts de video
- To Semester:** 2004
- Topic:** (empty text box)
- Abstract:** (empty text box)
- Classroom and Site:** Classroom and Site

At the bottom of the form, there are two buttons: **BACK** and **CREATE**.

Figura 6.7: iClass: formulário para criação de aulas.

6.5.2 Gravação ao Vivo

A gravação ao vivo envolve a comunicação entre o *applet* de captura e o *Sessiond*, iniciando um objeto *Transducer* no servidor responsável por receber os dados produzidos pelos componentes de captura.

Diferentemente do *Remote Meeting*, a versão básica do *iClass* é composta por apenas um *peer* de captura — o *front-end* que executa na máquina conectada à lousa eletrônica na sala de aula. Assim, todo procedimento de configuração resulta na definição de uma nova sessão de captura.

O *iClass* sempre utiliza o componente de *whiteboard*. Opcionalmente, pode capturar áudio e vídeo, que são mantidos em arquivos na máquina local e transferidos para o servidor ao final da sessão. É possível ainda utilizar o componente de *weblog* para capturar as URLs das páginas visitadas pelo professor durante a aula.

A fase de gravação ao vivo termina quando o *applet* de captura é finalizado. Essa ação inicia a fase de pós-processamento.

6.5.3 Pós-produção

O *Sessiond* implementa uma interface da InCA que faz com que ele receba *callbacks* que informam a finalização dos módulos de captura. Quando todos os módulos são finalizados, o *Sessiond* infere que o cliente não está mais ativo e chama o procedimento que gera o documento XML representativo da sessão capturada. Um exemplo de documento de sessão é mostrado na Figura 6.8.

É importante notar que o documento produzido possui mais metadados do que dados. A mídia audiovisual capturada fica armazenada no sistema de arquivos do servidor, em diretórios acessíveis pela *Web*. O documento de sessão, entretanto, contém as URLs desses arquivos multimídia juntamente com seus metadados associados. Por exemplo, para uma imagem que representa um *slide*, o documento armazena os *timestamps* de cada um de seus *strokes*, assim como os instantes em que a imagem foi visualizada pelos usuários.

Como etapa final desse processo, o documento gerado é armazenado no repositório de documentos, seguindo a organização determinada pelo documento *REPOSITORY.XML*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<session id="46" type="2" date="2002.10.24 AD at 12:11:00 BRST">
+ <context_intros>
  <chatml />
  <whiteboardml board_width="640" board_height="480">
    <slide slide_id="0">
    <slide slide_id="1">
    <slide slide_id="6">
    <slide slide_id="2">
    <slide slide_id="3">
    <slide slide_id="4">
    <slide slide_id="5">
      <visit visit_id="0" duration="5038" slide_id="0" />
      <visit visit_id="1" duration="1372" slide_id="1" />
      <visit visit_id="2" duration="891" slide_id="2" />
      <visit visit_id="3" duration="2070" slide_id="3" />
      <visit visit_id="4" duration="2423" slide_id="4" />
      <visit visit_id="5" duration="8466" slide_id="5" />
      <visit visit_id="6" duration="77758" slide_id="3" />
    <visit visit_id="7" duration="201694" slide_id="1">
    <visit visit_id="8" duration="400999" slide_id="1">
    <visit visit_id="9" duration="208863" slide_id="6">
    <visit visit_id="10" duration="36408" slide_id="1">
    <visit visit_id="11" duration="62987" slide_id="6">
    <visit visit_id="12" duration="64527" slide_id="1">
  </whiteboardml>
  <weblogml />
</session>
```

Figura 6.8: Documento XML produzido pelo sistema *iClass*

6.5.4 Acesso

O procedimento de acesso aos dados capturados pode ocorrer por meio da aplicação *Web* apresentada na Seção 5.4.3, ou, de maneira programática, utilizando a API provida pelo StRES.

Nos dois casos é possível elaborar consultas contextuais para recuperar as sessões desejadas. A diferença é que, na primeira opção, o acesso está associado à apresentação do conteúdo capturado. Já na segunda, a resposta consiste de uma lista de referências aos documentos que atendem às restrições especificadas na expressão de consulta.

A forma mais utilizada para acesso ao conteúdo capturado pelo *iClass* envolve a aplicação *Web* suportada pelo *Cocoon*. Entretanto, optou-se por customizar a aplicação padrão provida pelo StRES com o objetivo de gerar interfaces com funcionalidades mais avançadas. O resultado dessa customização é mostrado na Figura 6.9.

Entender a estrutura do arquivo de *sitemap* e ter algum conhecimento da linguagem de transformação XSLT é o suficiente para realizar essa customização. No caso particular da aplicação mostrada na Figura 6.9, foram elaboradas novas *stylesheets*² para processar os documentos que contêm as informações referentes aos semestres e às disciplinas de cada semestre. No restante, os arquivos utilizados são os mesmos produzidos para a versão padrão. Nota-se, portanto, que uma customização simples pode melhorar substancialmente o aspecto visual e navegacional da aplicação.

6.5.5 iClass vs. eClass: Uma Análise Comparativa

O *iClass* é equivalente em funcionalidade ao sistema *eClass*, apresentado na Seção 2.5.1. Apesar da similaridade, o desenvolvimento das duas aplicações é bastante diferente quando se avalia o esforço de implementação para torná-las funcionais. A Tabela 6.3 apresenta o número de linhas de código não comentadas (NCSS — *Non Commented Source Statements*) presentes em cada aplicação.

²Desenvolvidas no contexto do trabalho de mestrado da aluna Andrea R. Andrade (Andrade 2004).

Tabela 6.1: *eClass* vs. *iClass*: comparação do código fonte (Cattelan, Baldochi, Pimentel, and Truong 2003)

Systems	Classes	Functions	NCSS	Javadocs	Packages
eClass	56	532	11256	115	2
iClass	8	70	986	50	1

A diferença na quantidade de linhas de código utilizadas nas duas implementações evidencia as vantagens da abordagem infra-estrutural quando comparada à abordagem *ad hoc* utilizada no desenvolvimento do sistema *eClass*.

Figura 6.9: Aplicação Web para apresentação de aulas capturadas pelo *iClass*

6.6 Caderno On-Line

O Caderno On-Line³ foi concebido com o objetivo de substituir o caderno de papel por um caderno digital disponível na Web. Para tanto, estende as funcionalidades providas pelo *iClass*, proporcionando ao aluno a possibilidade de anotar sobre os *slides* produzidos pelo professor durante a aula.

O ambiente idealizado para o Caderno On-Line envolve uma sala instrumentada com uma lousa eletrônica, na qual o professor ministra a aula utilizando *slides* pré-preparados, e cada aluno dispõe de *tablets* interconectados por uma rede sem fio.

Entretanto, é possível utilizar o Caderno On-Line em um laboratório de informática usual, desde que instrumentado com uma lousa eletrônica, conforme ilustra a Figura 6.10. Nesse caso, o aluno pode lançar mão de uma ferramenta para anotação textual, que associa o texto digitado ao *slide* correntemente apresentado. Há ainda a possibilidade de produzir *strokes* utilizando o *mouse*, apesar dessa opção ser de pouca praticidade.

Figura 6.10: Uso do Caderno On-Line em laboratório de informática.

O Caderno On-Line foi desenvolvido como um conjunto de extensões ao *iClass*. Além da ferramenta para anotação textual, outra extensão implementada é o *xCurioso*, um *software* que permite mostrar os 4 últimos *slides* pro-

³Projeto desenvolvido nas Faculdades COC de Ribeirão Preto.

duzidos pelo professor projetados em uma tela de grandes dimensões, conforme ilustra a Figura 6.10. Essa extensão foi desenvolvida com o intuito de permitir uma melhor dinâmica para as aulas presenciais, permitindo que o professor referencie *slides* anteriores sem a necessidade de sair do *slide* corrente.

Uma terceira extensão refere-se à continuidade de uma aula (Santos, Lobato, and Baldochi 2004). A implementação original do *iClass* associa uma aula a uma sessão de captura. Assim, se uma aula é interrompida, ela só poderá ser retomada por meio de uma nova sessão, o que gera dois ou mais arquivos de sessão para uma mesma aula. Nesse caso, entende-se por aula um conjunto de *slides* que correspondem a uma dada matéria e não a um determinado período de tempo no qual o professor e os alunos estão em sala de aula.

No Caderno On-Line, implementou-se um mecanismo capaz de restaurar uma sessão interrompida, permitindo que ela continue a partir da interrupção e produza, ao final da aula, um único arquivo de sessão, o que facilita o gerenciamento da informação produzida.

A última extensão relevante implementada diz respeito à recuperação da informação produzida pelos alunos durante uma aula (Silva, Lobato, and Baldochi 2004). Como todos os alunos de uma determinada aula participam da mesma sessão, os *strokes* produzidos por eles ficam armazenados no documento de sessão produzido. Diferentemente do *iClass*, no Caderno On-Line, o envio de *strokes* é unidirecional: só os *strokes* do professor são enviados aos alunos. Os *strokes* produzidos pelos alunos são enviados apenas ao *Transducer*, que os armazena.

Dessa forma, cada aluno visualiza em seu *tablet* ou microcomputador os *strokes* produzidos pelo professor, em tempo real, e pode realizar seus próprios *strokes*, bem como usar a ferramenta de digitação para associar suas anotações a cada um dos *slides*.

A fim de permitir a recuperação do conteúdo produzido individualmente por cada aluno, foi implementado um mecanismo que filtra o documento de sessão original, produzindo um documento temporário no qual constam apenas os *strokes* produzidos pelo instrutor e pelo aluno, juntamente com as eventuais anotações textuais que ele tenha digitado. Este documento é utilizado juntamente com os *slides* originais apresentados na aula (sem anotações) na produção de imagens que contém os referidos *strokes*. O procedimento de recuperação produz uma página HTML que exibe os *slides* com os *strokes* e com as anotações textuais abaixo de cada *slide*, quando elas existirem.

6.7 wiClass

O uso do *iClass* como ferramenta de apoio ao ensino presencial foi inicialmente explorado pelos professores do Laboratório Intermídia, nas dependências do ICMC-USP. Com o passar do tempo, outros professores do instituto e fora dele passaram a utilizar o *iClass*.

Alguns desses professores sugeriram o uso do *iClass* fora do ambiente de sala de aula, suportando atividades experimentais em laboratório e também atividades de campo. O suporte a tais atividades envolve, sobretudo, as facilidades de anotação do *iClass* e a possibilidade de assistir o *playback* dessas anotações. Em particular, o que despertou o interesse desses professores foi a possibilidade de assistir passo a passo a solução de um problema pelos alunos de uma turma e perceber as diferentes abordagens adotadas em cada solução.

O uso do *iClass* nesse contexto evidentemente não envolve a utilização de dispositivos de captura de grande porte, tais como as lousas eletrônicas presentes nas salas de aula instrumentadas. Essas atividades são suportadas por dispositivos bem menores, preferencialmente portáteis, tais como tablets.

Para permitir o uso da aplicação de captura nesses dispositivos, foi desenvolvida uma versão *standalone* do *iClass*, denominada *wiClass* — *wireless iClass* (Pimentel, Baldochi, and Cattelan 2005). O *wiClass* é uma aplicação que embute parte das facilidades infra-estruturais providas pela abordagem xInCA-StRES, permitindo a captura de *strokes* feitos no componente de *whiteboard* e a eventual captura de áudio e vídeo.

Diferentemente do que ocorre com o *iClass*, o processamento do *wiClass* é centralizado: o processo de captura e até mesmo a geração do documento XML que representa a sessão capturada ocorre em uma única máquina, no caso, um *tablet PC*. A conectividade wireless provida por esses dispositivos é utilizada para transferir o conteúdo capturado para servidores de conteúdo, e, em certos casos, para permitir a aquisição de informações de contexto utilizadas na caracterização das sessões de captura.

Na versão atual, esse tipo de aplicação não suporta o compartilhamento de espaços colaborativos, tais como ocorre com a *whiteboard* da aplicação *Remote Meeting*. Apenas a interação local com os componentes xInCA é passível de captura.

As duas sessões a seguir apresentam casos de uso do *wiClass* envolvendo a captura de atividades experimentais em laboratório e a captura de atividades que ocorrem em ambientes abertos, fora dos limites das instituições de ensino.

6.7.1 Chemistry Lab e-Workbook

O objetivo desse estudo foi avaliar a adequação do *wiClass* no suporte a aulas experimentais de laboratório de um curso de química da USP. Realizado com a colaboração dos pesquisadores Paula Botero e João Farah, o estudo consiste em utilizar *tablets* como ferramenta de suporte a aulas de laboratório. No início da aula, cada aluno recebe um tablet com um experimento especificado e espaço para esboçar a solução adotada na realização do experimento, conforme ilustra a Figura 6.11.

Figura 6.11: *wiClass* suportando aulas de laboratório de química (Pimentel, Baldochi, and Cattelan 2005)

No caso desse experimento, a *whiteboard* é o principal componente de captura utilizado. É possível também capturar áudio utilizando um microfone acoplado ao *tablet*. Durante o experimento o aluno tem liberdade para realizar as anotações que quiser, podendo, inclusive, criar *slides* em branco caso o espaço fornecido nos *slides* pré-preparados seja insuficiente. Ao final do experimento, a informação capturada é transferida por meio de uma conexão *wireless* para um servidor de conteúdo.

Segundo os pesquisadores envolvidos, a grande vantagem trazida pela utilização do *wiClass* nas aulas experimentais está na possibilidade de rever, passo a passo, a abordagem de solução adotada por cada aluno na solução dos experimentos propostos. Observando a Figura 6.11 é possível notar que os alunos utilizam abordagens de solução completamente distintas para um mesmo experimento. Dessa forma, a análise posterior das anotações, utilizando as facilidades de *playback* providas pela infra-estrutura InCA-SERVE permite entender o esforço cognitivo de cada aluno durante a sessão de laboratório.

Essa análise permite também entender de que forma a intervenção do professor influencia a solução de cada aluno. Por exemplo, o professor pode resolver e disponibilizar uma parte do experimento em um determinado momento da aula. É importante dimensionar o quanto essa intervenção auxiliou

o aluno que está com dificuldade na elaboração do trabalho. Por meio dos *timestamps* gerados no processo de captura, é possível observar eventuais mudanças de abordagem a partir da interação do instrutor.

O entendimento do processo de aprendizagem constitui o principal benefício do uso do *wiClass* nesse experimento, conforme relatam os pesquisadores envolvidos. A captura dos experimentos tem permitido observar formas inovadoras e originais na solução dos problemas, as quais estão sendo usadas para refinar o processo de aprendizagem.

6.7.2 Mobile Pinhal Digital

Desde o ano de 2003 um grupo multidisciplinar de pesquisadores de diferentes áreas, incluindo Arquitetura, História, Cinema e Química, reúnem-se no mês de setembro na fazenda Conde do Pinhal com o objetivo de coletar e organizar dados associados à história da região de São Carlos.

A Fazenda Conde do Pinhal preserva edificações, fotos e documentos que retratam o apogeu da cultura cafeeira no estado de São Paulo. Os pesquisadores envolvidos nesse projeto têm por objetivo explorar esse vasto material histórico com o cuidado de preservá-lo para as futuras gerações. Dessa forma, não é permitido retirar fotos, documentos e objetos das dependências da fazenda.

A forma encontrada para estudar os aspectos arquitetônicos e históricos dos recursos da Fazenda do Pinhal foi a utilização da informática, mais particularmente de dispositivos computacionais móveis. Para tanto, instalou-se uma rede wireless conectando um conjunto de *tablets* nos quais o *wiClass* é executado.

Nesse contexto, o *wiClass* tem permitido a execução de algumas tarefas interessantes. Para os arquitetos, cuja principal preocupação consiste em documentar os aspectos arquitetônicos das edificações, o *wiClass* fornece o ferramental para o desenho livre de formas formas arquitetônicas, conforme ilustrado na Figura 6.12. Outra funcionalidade do *iClass* utilizada pelos arquitetos é a importação de plantas baixas das construções, permitindo a correção de medidas e o ajuste de informações incorretas que constavam nas plantas originais.

Um aspecto importante desse evento está no fato de que seus participantes são, em sua maioria, da área de humanidades. Assim, foi possível avaliar o uso do *wiClass* por pessoas que não utilizam a computação nas suas

Figura 6.12: *wiClass* suportando atividades de campo (Pimentel, Baldochi, and Cattelan 2005)

atividades cotidianas. Ainda assim o resultado foi considerado um sucesso, fazendo com que alguns desses pesquisadores dissessem frases como: "mudei minha concepção a respeito dos computadores".

A utilização do *wiClass* nesse evento ocorreu pela primeira vez em 2004, tendo resultado na produção de 1.5 GB de informação digital. Esse resultado é expressivo, sobretudo quando se considera que na edição anterior a maior parte da informação foi coletada utilizando papel e caneta. Espera-se que o material já coletado em formato digital, juntamente com outros que venham a ser coletados em edições futuras do encontro, possam resultar em produções multimídia e outros artefatos que auxiliem nas atividades de ensino e pesquisa.

6.8 Trabalhos Relacionados

Mark Weiser, considerado por muitos o pai da computação ubíqua, resumiu em uma frase a essência da pesquisa nessa área: "*Applications are of course the whole point of ubiquitous computing*" (Weiser 1991). Dessa forma, o suporte ao desenvolvimento de aplicações de computação ubíqua tem sido o principal foco de investigação de inúmeros grupos de pesquisa espalhados ao redor do globo.

De maneira geral, esse suporte aparece na forma de APIs, *frameworks*, *toolkits* e infra-estruturas de *software* que auxiliam o desenvolvimento de aplicações normalmente voltadas para ambientes instrumentados, nos quais a ciência de contexto é fortemente explorada. Com base em dispositivos passivos (sensores) e ativos (atuadores), essas aplicações são capazes de inferir a presença humana e ajustar o ambiente para atender as necessidades de cada usuário. O suporte infra-estrutural a esse tipo de aplicação é reportado em projetos tais como o EasyLeaving (Brumitt et al. 2000), o CoolTown (Kindberg

2002), o Aura (Sousa and Garlan 2002) e o Gaia (Roman et al. 2002).

A infra-estrutura InCA-SERVE está também voltada ao desenvolvimento de aplicações suportadas por ambientes instrumentados. Entretanto, essa instrumentação se dá principalmente na forma de dispositivos de captura e acesso, tais como câmeras de vídeo, microfones, *tablets* e lousas eletrônicas. Dessa forma, identificar usuários e inferir suas ações está fora do escopo do trabalho desenvolvido nesta tese.

A literatura também reporta projetos desenvolvidos para suportar aplicações voltadas para domínios específicos. O iRoom (Johanson et al. 2002) exemplo desse tipo de suporte, envolve a instrumentação de ambientes corporativos (escritórios, salas de reunião, entre outros) com dispositivos que incluem lousas eletrônicas da dimensão de paredes, projetores, microfones, alto falantes, PDAs e laptops. O objetivo principal desse projeto é auxiliar atividades colaborativas provendo facilidades de interação com o espaço físico instrumentado.

Semelhante ao iRoom, o projeto Roomware (Prante et al. 2004; Streitz et al. 2001) também investiga a instrumentação de ambientes de trabalho compartilhados. O RoomWare introduz o conceito de *Cooperative Buildings*, que consiste no uso de objetos do mundo real, tais como mobília, portas e paredes, acoplados a dispositivos computacionais para dar suporte ao trabalho cooperativo.

Diversas funcionalidades presentes no iRoom e também no Roomware são encontradas nas aplicações desenvolvidas com a infra-estrutura InCA-SERVE. Isso se deve, sobretudo, à similaridade dos dispositivos de captura utilizados nos dois projetos, e também aos objetivos comuns de suportar a comunicação distribuída e a captura de interações humanas. Uma diferença evidente entre o InCA-SERVE e esses dois projetos está no escopo: enquanto o iRoom e o *Roomware* estão voltado primariamente para ambientes instrumentados de escritório, o trabalho reportado nesta tese visa suportar diferentes domínios, incluindo salas de aula, laboratórios experimentais e até atividades de campo.

A pesquisa em computação ubíqua abre espaço para cenários visionários, que lembram livros e filmes de ficção científica. Inspirados por esses cenários, alguns projetos apresentam objetivos bastante ambiciosos, como é o caso do Projeto Oxygen (Dertouzos 1999), atualmente em desenvolvimento no MIT — *Massachusetts Institute of Technology*.

O Oxygen tem por objetivo suportar "*atividades humanas variadas e altamente dinâmicas*". Para tanto, é dividido em subprojetos que tratam de

desafios de pesquisa específicos na área de computação ubíqua. Um desses subprojetos, denominado Aire⁴, investiga o desenvolvimento de sistemas de *software* para computação pervasiva centrada na interação humana. O suporte a esse tipo de interação é provido por um sistema de captura de reuniões denominado *eFacilitator*, o qual provê facilidades para organização, anotação e intercâmbio de documentos em *tablets* e lousas eletrônicas. O *eFacilitator* permite ainda a captura de áudio, strokes e *slides*, possibilitando o *playback* do conteúdo capturado e provendo mecanismos de busca que permitem recuperar porções específicas de uma reunião.

o *eFacilitator* provê muitas similaridades funcionais com aplicações tais como o *iClass* e o *Remote Meeting*. Entretanto, trata-se de um sistema e não de uma infra-estrutura. Em outras palavras, é uma solução *ad hoc* para um domínio específico. A infra-estrutura InCA-SERVE, por outro lado, permite a construção de aplicações similares ao *eFacilitator*, assim como aplicações para diversos outros domínios.

A sala de aula é outro domínio que apresenta projetos bastante interessantes. Dois deles, que por coincidência possuem o mesmo nome — *Smart Classroom* — apresentam suporte infra-estrutural para captura de atividades em sala de aula.

O trabalho desenvolvido por Yau et al. (2003) visa suportar atividades colaborativas entre estudantes. Tendo como base um *middleware* denominado RCSM — *Reconfigurable Context-Sensitive Middleware* (Yau et al. 2002) — o qual permite a formação de redes *ad hoc* compostas por PDAs, o *Smart Classroom* oferece suporte consistente a atividades didáticas realizadas em grupo. Os PDAs utilizados nesse sistema possuem ciência de contexto em relação à situação corrente em que se encontram: localização, mobilidade e proximidade de pontos de luz e som podem ser identificados. Essas informações coletadas do ambiente são usadas para facilitar o processo de comunicação entre os atores envolvidos: estudantes, monitores e professores.

É possível perceber semelhanças entre o trabalho de (Yau et al.) e as aplicações *Caderno On-Line* e *wiClass*. No caso do *Caderno On-Line*, as semelhanças referem-se ao intercâmbio de informação produzida nos *tablets* e na lousa eletrônica. É importante notar que essa interação ocorre de maneira pré-definida e estática no *Caderno On-Line*, enquanto que o *Smart Classroom* permite a formação de grupos *ad hoc*. Já o *wiClass* se assemelha ao *Smart Classroom* por permitir a captura de atividades utilizando *tablets*. Entretanto, por operar em modo *stand alone*, o *wiClass* não permite o intercâmbio de

⁴<http://aire.csail.mit.edu>

informação em tempo real entre diferentes *tablets*.

O outro sistema também denominado *SmartClassroom* (Shi et al. 2003) é um exemplo de aplicação de captura e acesso construída com suporte infra-estrutural. Entretanto, o foco desse trabalho é a transmissão ao vivo de aulas para audiências remotas. Usando mecanismos de autenticação, o *SmartClassroom* permite comunicação bidirecional, apresentando funcionalidade semelhante aos sistemas de vídeo conferência, o que permite não só transmitir a aula para audiências remotas, como também suportar a interação entre alunos remotos e o instrutor.

6.9 Considerações Finais

O desenvolvimento de aplicações de computação ubíqua requer suporte de diferentes áreas da computação. Engenharia de *software*, sistemas distribuídos e bancos de dados são algumas das áreas que colaboram para compor o cenário multidisciplinar no qual se insere esse tipo de aplicação.

Dessa forma, suportar o desenvolvimento de aplicações de computação ubíqua requer esforços de pesquisa nessas diferentes áreas, objetivando prover abstrações de alto nível que facilitem a tarefa do desenvolvedor de aplicações. A infra-estrutura InCA-SERVE agrupa tais abstrações em componentes de *software* que minimizam o esforço de programação na construção de um tipo particular de aplicação, denominada aplicação de captura e acesso.

A utilização das abstrações de alto nível disponibilizadas na infra-estrutura InCA-SERVE exigem do desenvolvedor conhecimento do modelo de programação subjacente, assim como requerem o conhecimento do procedimento de instalação e configuração do ambiente necessário para a execução das aplicações.

Dessa forma, este capítulo procurou descrever, por meio de um exemplo concreto, os procedimentos necessários para o desenvolvimento de uma aplicação de captura e acesso. A descrição desse procedimento ajuda a entender a forma como as abstrações de comunicação, funcionalidade e armazenamento providas pela infra-estrutura encontram-se modeladas em classes e métodos da linguagem Java.

Em seguida, foram apresentadas as diversas aplicações já construídas explorando a infra-estrutura InCA-SERVE. Essas aplicações, apresentadas de maneira cronológica, demonstram a evolução do trabalho nos últimos três anos. O CAT foi o primeiro protótipo implementado com o suporte infra-

estrutural desenvolvido. Sua função foi, sobretudo, demonstrar a facilidade e a flexibilidade do modelo de programação baseado em componentes funcionais, o qual permite a definição de aplicações de captura e acesso por meio do agrupamento dos componentes que provêem a funcionalidade desejada.

Após o CAT, os esforços de validação do trabalho foram focados no desenvolvimento de aplicações voltadas para o ambiente de sala de aula. A fim de suportar a captura de aulas presenciais, foi desenvolvido o *iClass*, aplicação que se encontra em uso desde 2002, tendo sido utilizada na captura de mais de 300 aulas de cursos de graduação e pós-graduação do ICMC-USP. Mais importante que o conteúdo capturado, tem sido a retroalimentação dos inúmeros usuários desse sistema.

O sucesso do *iClass* inspirou o desenvolvimento de aplicações que estendem suas funcionalidades básicas, como é o caso do Caderno On-Line, extensão desenvolvida nas Faculdades COC de Ribeirão Preto por meio de uma parceria formal com o ICMC-USP. O Caderno On-Line explora a interação dos alunos no material gerado pelos professores, possibilitando que ao final da aula cada aluno tenha sua versão anotada do conteúdo ministrado.

Outra extensão relevante refere-se ao *wiClass*, versão *stand alone* do *iClass* capaz de executar em dispositivos móveis, tais como *tablet PCs*. As facilidades de captura aliadas à mobilidade permitem explorar a captura de novas atividades, em ambientes pouco convencionais, tais como laboratórios experimentais de química, museus e zoológicos.

O surgimento constante de novas aplicações e uso dessas aplicações por um grupo crescente de pessoas, muitas das quais sem *background* em computação demonstram a efetividade da infra-estrutura InCA-SERVE no suporte ao desenvolvimento de aplicações de captura e acesso.

Conclusão

*Brasileiro troca de celular a cada 19 meses*¹. Essa notícia, que a princípio parece irrelevante, evidencia um fato importante no contexto da computação ubíqua: existe uma demanda crescente por dispositivos ubíquos: novos modelos de celulares, PDAs e *tablets* surgem a todo momento, agregando continuamente novas funcionalidades àquelas já existentes. Esses *gadgets* inspiram diversos cenários de uso, capazes de tornar as atividades de trabalho muito mais produtivas e a vida cotidiana cada vez mais cômoda.

Entretanto, a realização desses cenários visionários depende, sobretudo, da capacidade de dotar programadores *normais* com ferramentas que auxiliem a implementação de aplicações de computação ubíqua.

O ferramental disponível ao desenvolvedor de aplicações de computação ubíqua, no entanto, está longe de prover as facilidades de desenvolvimento necessárias. O que se constata nos dias atuais é que o desenvolvedor de aplicações ubíquas precisa conhecer diversas áreas da computação e ser capaz de adaptar diferentes ferramentas de desenvolvimento a fim de construir aplicações. Não é por acaso que a quase totalidade das aplicações existentes foram desenvolvidas no contexto de projetos de pesquisa, seja no ambiente acadêmico ou corporativo (Davies and Gellersen 2002).

Diante desse cenário, Johanson (2002) é enfático ao afirmar que as necessidades dos programadores de aplicações ubíquas estão longe de estar devi-

¹<http://www.estadao.com.br/tecnologia/telecom/2005/mar/28/71.htm>

damente atendidas. Mensurar a gravidade dessa afirmação é, de certa forma, simples. Basta confrontá-la com a frase célebre de Mark Weiser: "*Applications are, of course, the whole point of ubiquitous computing*" (Weiser 1991).

Assim, fica claro que as inúmeras vantagens trazidas pela computação ubíqua só serão possíveis a medida que os programadores possam contar com ferramentas que auxiliem mais efetivamente o desenvolvimento de aplicações. Em outras palavras, é possível afirmar que o avanço da computação ubíqua depende da elevação do grau de abstração necessário para a construção de aplicações.

No sentido de prover tais abstrações para um tipo especial de aplicação, voltada para a captura de experiências ao vivo e o posterior acesso ao conteúdo capturado, foi desenvolvido o Projeto InCA-SERVE. Com o intuito de suportar o desenvolvimento de aplicações para diversos domínios, o projeto investiga possíveis formas de elevar o grau de abstração na construção de aplicações.

Esta tese de doutorado constitui um esforço no sentido de elevar o grau de abstração associado ao gerenciamento da informação produzida por aplicações de captura e acesso. Explorando resultados precursores que trataram questões de comunicação e recorrência de funcionalidades, este trabalho disponibiliza um modelo de desenvolvimento de aplicações que permite automatizar o processo de estruturação, armazenamento e apresentação de conteúdo capturado, contribuindo, de maneira significativa, para a elevação do grau de abstração necessário para a construção de aplicações de captura e acesso.

7.1 Resultados

O efetivo suporte à estruturação e ao gerenciamento dos dados manipulados pelas aplicações de captura e acesso passa, obrigatoriamente, pela modelagem consistente das informações de contexto específicas de cada aplicação.

No sentido de suportar adequadamente essa modelagem, desenvolveu-se o *Contextractor*, metodologia que permite associar informações presentes no modelo conceitual das aplicações às dimensões de contexto comumente utilizadas para referenciar atividades, pessoas, lugares e intervalos de tempo. Como resultado, o *Contextractor* produz esquemas usados como blocos arquiteturais na definição de uma infra-estrutura baseada em componentes de software que oferece abstrações de alto nível para o desenvolvimento de aplicações de captura e acesso.

O tratamento consistente das informações de contexto na camada infra-

estrutural permite disponibilizar serviços de acesso a conteúdo capturado que minimizam o esforço de desenvolvimento associado à consulta, à recuperação e à apresentação de conteúdo capturado. Esses serviços, associados à metodologia de modelagem de informações de contexto e ao modelo de desenvolvimento de aplicações suportado constituem os resultados dessa tese.

É importante destacar também os esforços de implementação que permitiram (i) estender a ferramenta ArgoUML, provendo facilidades gráficas para a modelagem de informação de contexto; (ii) disponibilizar aplicações Web baseadas no *Framework Cocoon*, as quais permitem a navegação no repositório de documentos; (iii) apresentar conteúdo capturado em diferentes formatos, explorando transformações XSLT sob demanda e (iv) prover facilidades de instalação e configuração da infra-estrutura disponibilizada.

O conjunto de abstrações implementadas na infra-estrutura InCA-SERVE permitiu o desenvolvimento de diversas aplicações, as quais foram apresentadas no Capítulo 6. As aplicações desenvolvidas, ao mesmo tempo que fornecem subsídios para a validação de modelos e serviços, representam, também, um resultado por si próprias.

O *iClass* é um exemplo desse fato. Sua implementação possibilitou a captura de mais de 300 aulas que podem ser acessadas via Web. O sucesso de sua implementação motivou outros projetos, tais como o *Caderno On-Line* e as versões para captura em dispositivos móveis, na qual é chamado de *wiClass*.

Os resultados obtidos nessa tese fornecem subsídios para o desenvolvimento de diversos trabalhos futuros, os quais são discutidos a seguir.

7.2 Trabalhos Futuros

A partir dos resultados obtidos, é possível identificar oportunidades de trabalhos futuros em duas diferentes áreas envolvidas nesta tese: o suporte à construção de aplicações de captura e acesso, e o suporte ao gerenciamento de conteúdo capturado.

7.2.1 Suporte à Construção de Aplicações

Apesar de aumentar significativamente o grau de abstração necessário à construção de aplicações de captura e acesso, o modelo de desenvolvimento provido requer conhecimento da linguagem de programação Java. Mesmo sendo uma ferramenta amplamente usada, impor seu uso restringe o alcance

do suporte infra-estrutural desenvolvido.

A preocupação em estender o acesso das facilidades desenvolvidas para usuários não programadores não é recente. Desde o início do trabalho, procurou-se formas alternativas de permitir a construção de aplicações. Prova disso é o CAT, apresentado no Capítulo 6, o qual permite a especificação das características de uma aplicação em um formulário Web e a disponibilização de um *applet* contendo a aplicação especificada.

O CAT foi uma prova da versatilidade do modelo de desenvolvimento baseado em componentes. Entretanto, no CAT não ocorre a efetiva construção de aplicações. Os *applets* disponibilizados já estão prontos no servidor, de tal forma que o procedimento de especificação serve apenas para selecionar o *applet* que atende as necessidades do usuário.

Outro problema associado ao CAT é sua dificuldade de extensão. A inclusão de novos componentes de captura requer a reimplementação de uma porção significativa do *software*, desde páginas Web até *applets* que devam incorporar o novo componente.

Uma possível forma de permitir a elevação do grau de abstração na construção de aplicações de forma a suportar de maneira mais flexível a evolução da infra-estrutura subjacente consiste no uso de uma linguagem declarativa baseada em XML para a especificação de aplicações.

Evidentemente, a aplicação especificada precisa ser mapeada para uma aplicação Java, sem a qual o uso da infra-estrutura fica impossibilitado. Essa transformação pode ser feita por meio de folhas de estilo XSLT, utilizando um procedimento similar ao adotado na metodologia *Context extractor*.

O esquema XML parcialmente exibido a seguir ilustra a estrutura básica de um documento de especificação de aplicação de captura e acesso.

```
<xsd:element name="application">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="deployAddr" type="xsd:string"/>
      <xsd:element name="multiuser" type="xsd:boolean"/>
      <xsd:element name="sessionType" type="tSession"/>
      <xsd:element name="users" type="tUsers" minOccurs="0"/>
      <xsd:element name="component" type="tComponent" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="className" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>
```

O processamento de um documento em conformidade com esse esquema envolve a geração de duas classes Java e de uma página HTML. As classes ger-

adas produzem um *applet* e uma aplicação de captura e acesso. A página Web serve para referenciar o *applet*, o qual abre uma janela contendo a aplicação. Essa abordagem garante a automação do processo de criação e disponibilização da aplicação, de tal forma a permitir que usuários leigos sejam capazes de criar aplicações — o único requisito é o entendimento da linguagem XML para a especificação da aplicação.

Dessa forma, além da execução das transformações XSLT, é necessário compilar as classes geradas e disponibilizá-las em diretórios acessíveis por algum servidor Web. Para realizar esse conjunto de tarefas de maneira seqüencial, é necessário utilizar algum *software* capaz de executar cada tarefa, utilizando cada resultado parcial como entrada para a tarefa subsequente. A ferramenta Ant da Apache² oferece facilidades para a execução dessas tarefas.

As informações *className* e *deployAddr* modeladas no esquema servem para nomear as classes Java geradas para a aplicação e definir o endereço Web em que a mesma será disponibilizada. Os elementos booleanos *multiuser* e *sessionType* estão relacionados com o controle de execução da aplicação, especificando se a mesma é multiusuário e se as sessões que define são públicas ou privadas. Aplicações privadas têm uma lista de usuários associada, representada no esquema pelo elemento *users*. Finalmente, o elemento *component* tem a função de permitir a escolha dos componentes desejados.

O tipo *tComponent* é uma enumeração que contém os nomes de todas as classes de componentes definidos pela xInCA. A escolha de uma enumeração para conter os componentes não ocorre por acaso: ela visa permitir a inclusão de novos componentes. Para entender as questões de extensibilidade, é necessário compreender o processo de transformação que visa produzir as classes da aplicação.

No processo de geração de código, duas folhas de estilo são utilizadas — ambas encontram-se em processo de elaboração. A primeira tem a função de produzir o *applet* que instancia a aplicação propriamente dita. A segunda, cuida de criar a classe principal da aplicação, na qual são instanciados os componentes de captura.

A segunda folha de estilo agrega uma complexidade considerável, uma vez que cuida de criar uma aplicação Java completa, contendo declarações das classes importadas, declaração das interfaces que devem ser implementadas, criação de um construtor para a classe, instanciação dos objetos de captura e, finalmente, declaração de métodos de *callback* das interfaces declaradas.

²<http://ant.apache.org>

Essa folha de estilo está sendo implementada utilizando *templates* XSLT que realizam tarefas específicas. Por exemplo, para criar a declaração de importação de classes, foram definidos dois *templates*. O primeiro, chamado *makeDefaultImports*, cuida de gerar a importação de classes ou pacotes que estão presentes em todas as aplicações geradas. O outro template, denominado *makeImportsForComponents*, cuida de importar as classes necessárias pelos componentes definidos no documento de especificação.

Seguindo essa linha, *templates* específicos para a geração de variáveis, para a definição do nome da classe juntamente com as interfaces que deve implementar e para as demais tarefas necessárias a fim de criar a aplicação estão sendo definidos.

Uma complementação importante desse trabalho consiste em permitir que extensões na infra-estrutura sejam transparentemente integradas à linguagem de especificação de aplicações. A implementação de um procedimento capaz de realizar essa integração é possível em função da modularização da infra-estrutura InCA-SERVE. Graças à essa modularização, os componentes xInCA encontram-se todos agrupados em um único pacote, que contém dois arquivos associados a cada componente: um deles é classe que define esse componente e o outro corresponde à sua interface associada.

Essa estruturação permite criar um procedimento que verifica as classes contidas no referido pacote, alimentando a enumeração que forma o tipo complexo *tComponent* anteriormente discutido. Dessa forma, cada novo componente criado pode ser automaticamente incluído nessa enumeração. Evidentemente, esse não é o único procedimento necessário: alterações nos *templates* da folha de estilo deverão ser feitos de modo a permitir a instanciação do novo componente, bem como a importação das suas classes associadas e a definição dos métodos exigidos por sua interface de programação.

Os resultados parciais já obtidos envolvendo versões preliminares das folhas de estilo mostram a viabilidade dessa abordagem. Entretanto, há um longo caminho até que aplicações contendo diversos componentes possam ser construídas. Há também todo um esforço de implementação a ser desenvolvido na construção do mecanismo que reflete no esquema XML e nas folhas de estilo as extensões realizadas na infra-estrutura.

7.2.2 Suporte ao Gerenciamento de Conteúdo

A abundância, a natureza multimídia e a riqueza de metadados e informações de contexto associadas ao conteúdo capturado inspiram diferentes

formas de exploração no sentido de maximizar o uso dos dados capturados. Esta sessão investiga novas possibilidades de extensão e consulta de conteúdo capturado, as quais caracterizam trabalhos futuros importantes.

Extensão

A extensão da informação capturada por meio de anotações e hiperligações constitui uma forma de enriquecer substancialmente a informação capturada, conforme demonstra a discussão apresentada na Seção 3.2.

O mecanismo implementado no StRES para permitir extensão oferece funcionalidade limitada, uma vez que possibilita apenas a inclusão de comentários textuais associados à sessão como um todo, ou a um *slide*.

Uma forma mais eficiente de implementar extensão envolve a utilização de uma base de ligações, também chamada *linkbase*. Utilizando uma *linkbase*, é possível estender os documentos XML que representam sessões sem ter que alterar esses documentos. Para tanto, basta colocar o comentário em outro arquivo XML, e criar na base de links uma ligação entre o arquivo comentado e o arquivo de comentário. Isso pode ser feito utilizando a linguagem XLink (DeRose et al. 2001), linguagem padrão do W3C para definição de ligações entre documentos XML.

Utilizando uma aplicação *Cocoon* e folhas de estilo apropriadas é possível visualizar o conteúdo comentado sob demanda. Para tanto, antes de exibir o conteúdo associado a um documento, é necessário verificar se não existe alguma hiperligação que o associa a um documento de extensão. Caso haja, o documento de extensão deve ser recuperado e seu conteúdo associado ao conteúdo do documento a ser exibido.

Experimentações realizadas com *slides* associados a comentários mantidos em arquivos externos mostraram o funcionamento dessa abordagem.

Assim, um trabalho futuro relevante consiste na criação de um mecanismo capaz de permitir a extensão de conteúdo a partir da definição de bases de ligações. A funcionalidade desse mecanismo passa por esforços de implementação capazes de permitir a inserção de comentários durante a visualização dos objetos de sessão e a posterior visualização desses comentários sob demanda, conforme já descrito.

Vale salientar, entretanto, que esse trabalho futuro não envolve apenas esforços de programação. A criação e manutenção da base de ligações e a utilização da linguagem XLink envolvem esforços de projeto e investigação

também relevantes.

Consulta

A linguagem CQL representa um primeiro esforço no sentido de prover facilidades de consulta contextual para aplicações de captura e acesso. Entretanto, formatar consultas como documentos XML é, no mínimo, antiprodutivo: o usuário tem a obrigação de escrever um documento bem formado, em conformidade com o esquema de contexto.

Dessa forma, a simplificação do processo de elaboração de consultas é um trabalho futuro importante, o qual deve colaborar para uma melhor usabilidade do mecanismo de consulta contextual.

Essa simplificação pode ocorrer através da implementação de uma interface gráfica que disponibilize um esquema de *Query By Example* (QBE), no qual o usuário precisa apenas completar lacunas para elaborar a consulta.

Vale notar que, também neste caso, o trabalho envolve mais que simples implementação. A solução desenvolvida deve ser genérica o suficiente para atender qualquer aplicação modelada. Isso significa que a solução deve se basear no esquema XML da linguagem de consulta para construir os elementos de interface. Dessa forma, para permitir a consulta de conteúdo gerado por diferentes aplicações, o serviço de consulta deve ser capaz de ajustar os elementos de interface com base nas informações do usuário.

Possível Solução para Limitação da Linguagem CQL

Conforme discutido na Seção 4.7, a linguagem CQL apresenta uma limitação que impossibilita o uso conjunto dos conectores *AND* e *OR* atuando sobre um mesmo conjunto de elementos. Essa limitação decorre da solução proposta para processar as consultas CQL, a qual é baseada na validação de documentos XML.

No sentido de permitir o aninhamento indeterminado de conectores suportado pela maioria das linguagens de consulta — SQL e XQuery, por exemplo — é necessário construir esquemas com valores mínimos e máximos de ocorrência de elementos ajustados para 0 e *unbounded*, respectivamente. Isso permitiria validar expressões com qualquer combinação de aninhamento para os conectores, mas também permitiria a validação de expressões inválidas. Um exemplo de expressão corretamente validada é mostrado a seguir.


```

...
<who>
  <conector name="OR">
    <conector>
      <conector name="AND">
        <Aluno id="33" />
        <Aluno id="12" />
      </conector>
      <conector name="AND">
        <Aluno id="57" />
        <Aluno id="26" />
      </conector>
    </conector>
  </conector>
</who>
...

```

Esse trecho de código XML especifica que devem ser recuperados os documentos referentes às sessões em que os alunos 33 *E* 12 participaram, *OU* aquelas nas quais estiveram presentes os alunos 57 *E* 26.

Entretanto, a expressão seguinte também seria considerada válida, apesar de não possuir nenhum sentido, já que possui um elemento vazio.

```

...
<who>
  <conector name="OR">
    <conector>
      <conector name="AND">
        <Aluno id="33" />
        <Aluno id="12" />
      </conector>
      <conector name="AND" />
    </conector>
  </conector>
</who>
...

```

A solução para o problema consiste em alterar o processador CQL, de modo a permitir a detecção desse tipo de erro sem a utilização do esquema *CONTEXT-QUERY*. Para tanto, é necessário implementar um duplo esquema de validação: após a validação usual, que utiliza um *parser* validador XML, é necessário verificar especificamente a validação dos elementos *conector*. As regras para validação desse elemento devem ser embutidas diretamente no código.

Uma vez que a estrutura do documento sofrerá modificações, será necessário atualizar o programa que produz as expressões *FLWOR*, de maneira que os elementos *conector* sejam corretamente mapeados para essas expressões.

7.3 Considerações Finais

Esta tese investigou problemas associados ao gerenciamento da informação produzida por aplicações de captura e acesso. As investigações realizadas apontaram a necessidade de prover abstrações infra-estruturais capazes de facilitar a representação e o intercâmbio da informação capturada, além de serviços capazes de facilitar o acesso a esse conteúdo devidamente integrado e disponibilizado na *Web*.

De maneira a atender a essas necessidades, foram desenvolvidos um modelo para implementação de aplicações baseado em componentes de software; um serviço capaz de oferecer facilidades de armazenamento e apresentação de conteúdo capturado de maneira transparente; e uma linguagem para consulta de conteúdo capturado baseada nas informações de contexto da aplicação.

O suporte infra-estrutural e de serviços disponibilizado possibilitou a construção de aplicações capazes de capturar experiências humanas nos mais diferentes ambientes: salas de aula, laboratórios de química e eventos de campo são alguns exemplos. Algumas dessas aplicações, tais como o *iClass*, vêm sendo utilizadas continuamente por mais três anos, o que demonstra a robustez do suporte infra-estrutural provido.

O *iClass* é um exemplo de aplicação que evolui de protótipo para uma aplicação real, com características próximas de um produto de software. A transformação de protótipos em aplicações robustas consiste em uma importante forma de validação do trabalho desenvolvido.

Referências

- Abowd, G. (1999). Classroom 2000: An experiment with the instrumentation of a living educational environment. *IBM Systems Journal* 38(4), 508–530.
- Abowd, G., C. Atekeson, J. Brotherton, T. Enqvist, P. Gully, and J. Lemon (1998). Investigating the capture, integration and access problem of ubiquitous computing in educational setting. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 440–447.
- Abowd, G., J. Brotherton, and J. Bhalodia (1998). Classroom 2000: A system for capturing and accessing multimedia classroom experiences. In *Proceedings of the Conference on Human Factors in Computing Systems*, pp. 20–21. CHI 98 conference summary on Human factors in computing systems.
- Abowd, G. and E. Mynatt (2000). Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction* 7(1), 29–58.
- Abowd, G., M. Pimentel, B. Kerimbaev, Y. Ishiguro, and M. Guzdial (1999). Anchoring discussions in lecture: An approach to collaboratively extending classroom digital media. In *Proceedings of the Computer Support for Collaborative Learning Conference*, pp. 11–19.
- Abowd, G. D., C. G. Atkeson, A. Feinstein, C. Hmelo, R. Kooper, S. Long, N. Sawhney, and M. Tani (1996). Teaching and learning as multimedia authoring: The classroom 2000 project. In *Proceedings of the ACM Multimedia Conference*, pp. 187–198.
- Abowd, G. D., C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton (1997). Cyberguide: a mobile context-aware tour guide. *Wirel. Netw.* 3(5), 421–433.

- Abowd, G. D., M. E. D., and R. T. (2002). The human experience. *IEEE Pervasive Computing* 1(1), 48–57.
- Andrade, A. R. (2004). Geração automática de aplicações web para aplicações de captura e acesso. Master's thesis, Universidade de São Paulo.
- Andrade, A. R., E. Munson, and M. G. C. Pimentel (2004a). A document based approach to the generation of web applications. In *Proceedings of the ACM Symposium on Document Engineering 2004*, pp. 45–47.
- Andrade, A. R., E. Munson, and M. G. C. Pimentel (2004b). Engineering web applications with xml and xslt. In *WebMedia & LAWeb 2004 Joint Conference*, pp. 86–93. IEEE CS Press.
- Apache (2005). The Apache Software Foundation: Apache Cocoon. URL:<http://cocoon.apache.org>.
- Ark, W. S. and T. Selker (1999). A look at human interaction with pervasive computers. *IBM Systems Journal* 38(4), 540–507. Special issue on Pervasive Computing.
- Baldochi, L. A., A. R. Andrade, R. G. Cattelan, and M. G. C. Pimentel (2004). Architecture and components for capture and access applications. In *WebMedia & LAWeb 2004 Joint Conference*, pp. 150–157. IEEE CS Press.
- Baldochi, L. A., R. G. Cattelan, and M. G. Pimentel (2003). Building a middleware infrastructure for capture and access applications. In *Proceedings of the XXX SEMISH - Integrated Seminar on Software and Hardware*, pp. 299–313.
- Baldochi, L. A., R. G. Cattelan, and M. G. C. Pimentel (2005). Modeling contextual information for capture and access applications. Submitted to the ACM Symposium on Document Engineering - DocEng 2005.
- Baldochi, L. A., R. G. Cattelan, M. G. C. Pimentel, and K. N. Truong (2002). Automatic generation of capture and access applications. In *Proceedings of the 8th Brazilian Symposium on Multimedia and Hypermedia Systems*, pp. 100–115.
- Barger, D., A. Gupta, E. Sanochi, and J. Grudin (1998). Annotations for streaming video on the web: System design and usage studies. Technical report MSR-TR-98-60, Microsoft Research Center.
- Boag, S., D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon (2004, October). Xquery 1.0: An xml query language. W3C Working Draft – <http://www.w3.org/TR/xquery/>.
- Brotherton, J. and G. Abowd (1998). Rooms take note: Room takes notes! In *Papers from the AAAI Spring Symposium*, pp. 23–30.

- Brotherton, J., G. Abowd, and K. Truong (1999). Supporting capture and access interfaces for informal and opportunistic meetings. Technical report GIT-GVU-99-06, GVU Center, Georgia Institute of Technology.
- Brotherton, J., J. Bhalodia, and G. Abowd (1998). Automated capture, integration, and visualization of multiple media streams. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, pp. 54–63.
- Brumitt, B., B. Meyers, J. Krumm, A. Kern, and S. Shafer (2000). Easyliving: Technologies for intelligent environments. In *Second International Symposium on Handheld and Ubiquitous Computing (HUC)*, pp. 12–29.
- Cahill, V., A. Fox, T. Kindberg, and B. Noble (2004). Guest editors' introduction: Building and evaluating ubiquitous system software. *Pervasive Computing* 3(3), 20–21.
- Camacho-Guerrero, J. A., A. A. Macedo, and M. da Graça Campos Pimentel (2004). A look at some issues during textual linking of homogeneous web repositories. In *DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering*, New York, NY, USA, pp. 74–83. ACM Press.
- Cattelan, R. G., A. R. Andrade, C. F. R. Penedo, and 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.
- Cattelan, R. G., L. A. Baldochi, and M. G. C. Pimentel (2003). Processing and storage middleware support for capture and access applications. In *Companion Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference*, pp. 315.
- Cattelan, R. G., L. A. Baldochi, M. G. C. Pimentel, and K. N. Truong (2003). Experiences on building capture and access applications. In *Proceedings of the 9th Brazilian Symposium on Multimedia and Hypermedia Systems*, pp. 112–127.
- Chiu, P., J. Boreczky, A. Girgensohn, and D. Kimber (2001). Liteminutes: an internet-based system for multimedia meeting minutes. In *Proceedings of the Tenth International World Wide Web Conference*, pp. 140–149.
- Coutaz, J., J. L. Crowley, S. Dobson, and D. Garlan (2005). Context is key. *Commun. ACM* 48(3), 49–53.
- Davies, N. and H.-W. Gellersen (2002). Beyond prototypes: challenges in deploying ubiquitous systems. *Pervasive Computing* 1(1), 26–35.
- DeRose, S., E. Maler, and D. Orchard (2001). XML linking language (XLink) version 1.0. W3C Recommendation - Available in:

<http://www.w3.org/TR/xlink>.

- Dertouzos, M. (1999, August). The future of computing. *Scientific American*.
- Dey, A. K. and G. D. Abowd (2000). The context toolkit: Aiding the development of context-aware applications. In *Workshop on Software Engineering for Wearable and Pervasive Computing*.
- Elrod, S., R. Bruce, R. Gold, D. Goldberg, F. Halasz, W. Janssen, D. Lee, K. McCall, E. Pedersen, K. Pier, J. Tang, and B. Welch (1992). Liveboard: a large interactive display supporting group meetings, presentations, and remote collaborations. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 599–607.
- Franklin, M. J. (2001). Challenges in ubiquitous data management. In R. Wilhelm (Ed.), *Informatics - 10 Years Back. 10 Years Ahead.*, Volume 2000 of *Lecture Notes in Computer Science*, pp. 24–33. Springer.
- Garlan, D., D. Siewiorek, A. Smailagic, and P. Steenkiste (2002). Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing* 1(2), 22–31.
- Gu, T., X. H. Wang, H. K. Pung, and D. Q. Zhang (2004). An ontology-based context model in intelligent environments. In *Communication Networks and Distributed Systems Modeling and Simulation Conference*.
- Halpin, T. A. (1995). *Conceptual Schema and Relational Database Design* (2nd ed.). Prentice Hall Australia.
- Harrison, B., K. Fishkin, A. Gujar, C. Mochon, and R. Want (1998). Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 17–24.
- Hauptmann, A. (1995). Speech recognition in the informedia digital video library: Uses and limitations. In *Proceedings of the IEEE International Conference on Tools with AI*, pp. 288.
- Henricksen, K. and J. Indulska (2004). A software engineering framework for context-aware pervasive computing. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004)*, pp. 77–86. IEEE Computer Society.
- Henricksen, K., J. Indulska, and A. Rakotonirainy (2002). Modeling context information in pervasive computing systems. In *Proceedings of 1st International Conference on Pervasive Computing (LNCS 2414)*, pp. 167–180.
- Henricksen, K., J. Indulska, and A. Rakotonirainy (2003). Generating context management infrastructure from high-level context models. In *In-*

dustrial Track Proceedings of the 4th International Conference on Mobile Data Management (MDM2003), pp. 1–6.

Hess, C. K., M. Román, and R. H. Campbell (2002). Building applications for ubiquitous computing environments. In *Proceedings of the First International Conference on Pervasive Computing*, Volume 2414 of *Lecture Notes in Computer Science*, Zurich, pp. 16–29. Springer-Verlag.

Hong, J. I. (2002). The context fabric: an infrastructure for context-aware computing. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, New York, NY, USA, pp. 554–555. ACM Press.

Hong, J. I. and J. A. Landay (2001). An infrastructure approach to context-aware computing. *Human-Computer Interaction Journal* 16(2-4), 287–303.

Ishii, H. and B. Ullmer (1997). Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 234–241.

Johanson, B., A. Fox, and T. Winograd (2002). The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing* 1(2), 67–74.

Johanson, B., T. Winograd, and A. Fox (2003). Interactive Workspaces. *IEEE Computer* 36(4), 99–103.

Johanson, B. E. (2002). *Application Coordination Infrastructure for Ubiquitous Computing Rooms*. Ph. D. thesis, Department of Electrical Engineering – Stanford University.

Kazman, R., L. Bass, and M. Abowd, G. and Webb (1994). Saam: A method for analyzing the properties of software architectures. In *Proceedings of International Conference on Software Engineering*, pp. 81–90.

Kidd, C. D. e. a. (1999). The aware home: A living laboratory for ubiquitous computing research. In *Proceedings of the Second International Workshop on Cooperative Buildings - CoBuild'99*.

Kindberg, T. (2002). Implementing physical hyperlinks using ubiquitous identifier resolution. In *Proceedings of the Eleventh International World Wide Web Conference*, pp. 191–199.

Liogkas, N., B. MacIntyre, E. Mynatt, Y. Smaragdakis, E. Tilevich, and S. Volda (2004). Automatic partitioning for prototyping ubiquitous computing applications. *Pervasive Computing* 3(3), 40–47.

Macedo, A. A., L. A. Baldochi, J. A. Camacho-Guerrero, R. G. Cattelan, and M. G. C. Pimentel (2005). Automatically linking live experiences captured

- with a ubiquitous infrastructure. *Submitted to Multimedia Tools and Applications (MTAP) Journal*, 39p.
- Minneman, S., S. Harrison, B. Janseen, G. Kurtenback, T. Moran, I. Smith, and B. Van Melle (1995). A confederation of tools for capturing and accessing collaborative activity. In *Proceedings of the Third ACM International Conference on Multimedia*, pp. 523–534.
- Moran, T. P., P. Chiu, and W. Van Melle (1997a). Pen-based interaction techniques for organizing material on an electronic whiteboard. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*.
- Moran, T. P., L. Palen, S. Harrison, P. Chiu, D. Kimber, S. Minneman, W. Van Melle, and P. Zellweger (1997). I'll get that of the audio: A case study of salvaging multimedia meeting records. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 202–209.
- Moran, T. P. and W. van Melle (2000). Tivoli: integrating structured domain objects into a freeform whiteboard environment. In *CHI '00 extended abstracts on Human factors in computer systems*, pp. 20–21. ACM Press.
- Mueller, R. and T. Ottmann (2000). The "authoring on the fly" system for automated recording and replay of (tele) presentations. *ACM/Springer Multimedia Systems Journal* 3(8), 158–176.
- Mukhopadhyay, S. and B. Smith (1999). Passive capture and structuring of lectures. In *Proceedings of the ACM Multimedia Conference*, pp. 477–487.
- Mynatt, E., J. Rowan, S. Craighill, and A. Jacobs (2001). Digital family portraits: Providing peace of mind for extended family members. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 333–3340.
- Mynatt, E. D., T. Igarashi, W. K. Edwards, and A. LaMarca (1999). Flatland: New dimensions in office whiteboards. In *Proceedings of the Conference on Human Factors in Computing Systems*, pp. 346–353.
- Norman, D. (1998). *The invisible computer*. MIT Press.
- OMG (2003). Xml Metadata Interchange. <http://www.omg.org/technology/documents/formal/xmi.htm>.
- Pedersen, E., K. McCall, T. Moran, and F. Halasz (1993). Tivoli: An electronic whiteboard for informal workgroup meetings. In *Proceedings of the ACM INTERCHI Conference on Human Factors in Computing Systems*, pp. 391–398.

- Pimentel, M., Y. Ishiguro, B. Kerimbaev, G. Abowd, and M. Guzdial (2001). Supporting long-term educational activities through dynamic web interfaces. *Interacting with Computers Journal* 13, 353–374.
- Pimentel, M. G. C., L. A. Baldochi, and R. G. Cattelan (2005). Prototyping applications to document the human experience. *Submitted to the IEEE Pervasive Computing Magazine*, 8p. Issue on Rapid Prototyping for Ubiquitous Computing.
- Prante, T., N. Streitz, and P. Tandler (2004). Roomware: Computers Disappear and Interaction Evolves. *IEEE Computer* 37(12), 47–54.
- Richter, H., G. Abowd, W. Geyer, L. Fuchs, S. Daijavad, and S. Poltrock (2001). Integrating meeting capture within a collaborative team environment. In *Proceedings of the Third International Conference on Ubiquitous Computing*, pp. 123–138.
- Roman, M., C. K. Hess, R. Cerqueira, A. Ranganathan, and K. N. Campbell (2002). Gaia: A middleware infrastructure for active spaces. *IEEE Pervasive Computing* 1(4), 74–83.
- Sante, D. (2003, Maio). Autore: suportando autoria evolucionária em ambientes de captura. Master's thesis, Instituto de Ciências Matemáticas e de Computação da USP, São Carlos – SP.
- Sante, D. G., J. A. Camacho-Guerrero, A. A. Macedo, and R. P. M. Fortes (2004). Preparing, extending and retrieving captured information. In *WebMedia & LAWeb 2004 Joint Conference*, pp. 158–160. IEEE CS Press.
- Santos, F. S., D. C. Lobato, and L. A. Baldochi (2004). Extensões de funcionalidades do projeto caderno on-line: Continuidade de uma aula. In *WebMedia & LAWeb 2004 Joint Conference*, Volume 2, pp. 391–393.
- Schilit, B., G. Golovchinsky, and M. Price (1998). Beyond paper: Supporting active reading with free form digital ink annotations. In *Proceedings of the Conference on Human Factors in Computing Systems*, pp. 249–256.
- Schmandt, C. and B. Arons (1985). Phoneslave: A graphical telecommunications interface. In *Proceedings of the Society for Information Display*, pp. 79–82.
- Shi, Y., W. Xie, G. Xu, R. Shi, E. Chen, Y. Mao, and F. Liu (2003). The smart classroom: Merging technologies for seamless teleeducation. *IEEE Pervasive* 2(2), 47–55.
- Silva, R. C. B. S., D. C. Lobato, and L. A. Baldochi (2004). Extensões de funcionalidades do projeto caderno on-line: Recuperação de aulas. In *WebMedia & LAWeb 2004 Joint Conference*, Volume 2, pp. 388–390.

- Sousa, J. and D. Garlan (2002). Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Third IEEE/IFIP Conference on Software Architecture*, pp. 29–43.
- Starner, T. (1999). *Wearable Computing and Contextual Awareness*. Ph. D. thesis, MIT Media Laboratory.
- Strang, T. and C. Linnhoff-Popien (2004). A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*.
- Streitz, N., J. Geibler, J. Haake, and J. Hol (1994). DOLPHIN: integrated meeting support across local and remote desktop environments and live-boards. In *Proceedings of the Conference on Computer Supported Cooperative Work*, pp. 345–358.
- Streitz, N. and P. Nixon (2005). Introduction. *Commun. ACM* 48(3), 32–35.
- Streitz, N. A., P. Tandler, C. Müller-Tomfelde, and S. Konomi (2001). Roomware: Towards the next generation of human-computer interaction based on an integrated design of real and virtual worlds. In J. A. Carroll (Ed.), *Human-Computer Interaction in the New Millennium*, pp. 553–578. Addison Wesley.
- Tigris (2003). Argouml: A uml design tool with cognitive support. <http://argouml.tigris.org/>.
- Tran, Q., G. Calcaterra, and E. Mynatt (2005). Cook’s collage: Memory aid display for cooking. In A. Sloane (Ed.), *Proceedings of the IFIP WG 9.3 HOIT2005 Conference*, IFIP International Federation for Information Processing.
- Truong, K. and G. Abowd (2002). Enabling the generation, preservation & use of records and memories of everyday life. Technical report GATECH GIT-GVU-02-02, GVU Center, Georgia Institute of Technology.
- Truong, K. and G. Abowd (2004). Inca: A software infrastructure to facilitate the construction and evolution of ubiquitous capture & access applications. In *Proceedings of the Second International Conference on Pervasive Computing*, pp. 140–157.
- Truong, K., G. Abowd, and J. Brotherton (2001). Who, what, when, where, how: Design issues of capture & access applications. In *Proceedings of the Third International Conference on Ubiquitous Computing*, pp. 209–224.
- Wactlar, H., T. Kanade, M. Smith, and S. Stevens (1996). Intelligent access to digital video: The informedia project. *IEEE Computer* 29(5), 46–52.

- Want, R., A. Hopper, F. V., and J. Gibbons (1992). The active badge location system. *ACM Transactions on Information Systems* 10(1), 91–102.
- Want, R., B. Schilit, N. Adams, R. Gold, K. Petersen, J. Ellis, D. Goldberg, and M. Weiser (1995). The parctab ubiquitous computing environment. Technical Report CSL-95-1, Xerox PARC.
- Weiser, M. (1991). The computer for the 21st century. *Scientific American* 265(3), 94–104. Reprinted in *IEEE Pervasive Computing* vol 1, n.1, 2002.
- Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM* 7(6), 75–84.
- Weiser, M. and J. S. Brown (1997). *The coming age of calm technology*, pp. 75–85. Copernicus.
- White, S., A. Gupta, J. Grudin, H. Chesley, G. Kimberly, and E. Sanocki (1998). A software system for education at a distance: case study results. Technical Report MSR-TR-98-61, Microsoft Research Center.
- Wilcox, L., B. Schilit, and N. Sawhney (1997). Dynamite: A dynamic organized ink and audio notebook. In *Proc. of the Conf. on Human Factors in Computing Systems*, pp. 186–193.
- Yau, S. S., S. K. S. Gupta, F. Karim, S. Ahamed, Y. Wang, and B. Wang (2003). A smart classroom for enhancing collaborative learning using pervasive computing technology. In *Proceedings of the 6th WFEO World Congress on Engineering Education and the 2nd ASEE International Colloquium on Engineering Education (ASEE 2003)*, pp. 21–30.
- Yau, S. S., F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta (2002). Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing* 1(3), 33–40.

Glossário

Caching — Armazenamento temporário de informação em memória rápida com a finalidade de otimizar operações de acesso a essa informação.

Callback — Forma de comunicação entre objetos de software que envolve o uso de interfaces de programação. Nessa forma de comunicação, existe a classe que implementa a interface e a classe que instancia essa interface. A que implementa deve declarar todos os métodos da interface, preenchendo-os com o código que deve ser executado quando o método for chamado. A classe que instancia a interface pode chamar qualquer um dos métodos, provocando a execução desse método na classe que o implementa.

Daemon — Um processo Unix que executa continuamente. Na computação cliente servidor, *daemon* é um programa servidor que executa continuamente, em geral monitorando uma porta usada para a comunicação com os programas clientes.

DataSink — Depósito de dados gerados via *streaming*.

Data binding — ver XML data binding.

FLWOR — Expressão de consulta da linguagem *XQuery* baseada nas palavras chave *For*, *Let*, *Where*, *Order by* e *Return*. Expressões FLWOR são sintaticamente e semanticamente semelhantes a expressões SQL, por isso permitem a recuperação de conteúdo de maneira mais eficiente em relação à linguagem *XPath*, sua predecessora.

Framework — Sistema ou subsistema de software que provê uma estrutura básica para uma classe específica de aplicações. Disponibilizando blocos arquiteturais, os *frameworks* provêem modelos de desenvolvimento que permitem o reuso de soluções. Os *frameworks* devem especificar de maneira clara os chamados pontos de adaptação ou extensão, de forma a permitir que fun-

cionalidades específicas das aplicações possam ser facilmente acrescentadas às suas arquiteturas.

Infra-estrutura — Conjunto de tecnologias bem estabelecidas, pervasivas e robustas que servem de apoio para o desenvolvimento de sistemas de software.

Marshal — Ação associada ao mecanismo de *data binding* que permite gerar documentos XML a partir de informações mantidas em objetos de software.

Middleware - Camada de software responsável por prover serviços e abstrações de alto nível com a finalidade de facilitar o desenvolvimento e a integração de aplicações, bem como apoiar tarefas de gerenciamento das aplicações suportadas.

Parser — Programa capaz de separar informação em pedaços menores. Um parser XML permite processar elementos e atributos de um documento.

Peer — Uma das partes de uma aplicação distribuída *peer to peer*.

Pipeline — Seqüência de tarefas nas quais o resultado produzido por uma tarefa é utilizado como entrada na tarefa subsequente.

Proxy — Software que monitora a interação de programas com a World Wide Web.

Publish/subscribe — paradigma de troca de mensagens assíncrono, também conhecido como pub/sub. Possui como entidades centrais publicadores e subscritores, os quais, respectivamente, produzem e consomem informação, a qual é intercambiada por meio de *middlewares* baseados no enfileiramento de mensagens, conhecidos como MOM — *Message Oriented Middleware*. Um MOM pode ser implementado através de uma entidade central de registro conhecida como *Registry*, a qual registra os publicadores e os subscritores. A função do *Registry* é receber mensagens e distribuí-las entre os subscritores registrados para receber essas mensagens.

Registry — Ver publish/subscribe.

Sessiond — *Session daemon*. Servidor que atende aplicações de captura e acesso que desejam salvar conteúdo capturado.

Sitemap — Documento XML que especifica o funcionamento de uma aplicação Web gerenciada pelo *Framework Cocoon*.

Socket — Abstração de software que permite a comunicação entre programas remotos utilizando o protocolo TCP/IP.

Stand-alone — Programa que executa isoladamente, sem acesso a uma rede

de computadores.

Stereotype — Estereótipo: um dos mecanismos de extensão da UML.

Stroke — Movimento de caneta ou pincel de forma a produzir um traço. No contexto de dispositivos ubíquos tais como lousas eletrônicas, *stroke* é o resultado da interação com esses dispositivos usando uma caneta especial que produz um traço virtual representado por um conjunto de pontos.

Subscribe — ver *publish/subscribe*

Tagged values — Pares atributo-valor: um dos mecanismos de extensão da UML.

Toolkit — Geralmente construído sobre um *framework*, um *toolkit* prove um conjunto de ferramentas de software que auxiliam o desenvolvimento de aplicações. Um exemplo comum da sinergia entre *frameworks* e *toolkits* pode ser observado nos sistemas que facilitam a criação de interfaces gráficas. Nesse caso, um sistema gerenciador de eventos seria um *framework* e o conjunto de botões, *checkboxes* e caixas de texto que utiliza as facilidades do gerenciador de eventos seria um *toolkit*.

Transducer — Entidade responsável por receber, em tempo real, informação capturada e produzir um documento XML capaz de representar uma sessão de captura.

Unmarshal — Ação associada ao mecanismo de *data binding* que permite produzir documentos XML a partir de conteúdo mantido em objetos de software especialmente produzidos para manipular a informação presente nesses documentos.

XML Data binding — Mapeamento bidirecional entre documentos XML e objetos especificamente criados para manipular os dados contidos nesses documentos. As implementações de *data binding* provêem ferramentas que permitem a geração automática de classes a partir de esquemas XML. Interfaces de programação disponibilizadas por essas implementações permitem representar elementos e atributos dos documentos XML utilizando objetos e variáveis das classes geradas, o que possibilita a manipulação programática do conteúdo de documentos XML. De maneira inversa, é possível produzir documentos XML a partir do conteúdo desses objetos. As implementações de *data binding* provêem abstrações de alto nível que permitem processar conteúdo XML de maneira mais natural, representando uma opção ao uso de DOM.

Whiteboard — Lousa eletrônica. Pode também referenciar o software que gerencia a lousa eletrônica.

Documentos produzidos pelo Contextractor

Documentos produzidos pelo *Contextractor* para uma aplicação de captura de aulas presenciais.

Modelo Conceitual (CM)

```
<conceptual-model>
  <conceptual-class name="Sessao" external="true">
    <attribute name="id" type="integer" context-dimension="none" />
    <attribute name="data" type="date" context-dimension="when" />
  </conceptual-class>
  <conceptual-class name="Aula" external="true">
    <attribute name="titulo" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Aluno" external="false">
    <attribute name="id" type="integer" context-dimension="who" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Instructor" external="false">
    <attribute name="titulacao" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Curso" external="false">
    <attribute name="id" type="string" context-dimension="none" />
    <attribute name="nome" type="string" context-dimension="none" />
    <attribute name="semestre" type="string" context-dimension="when" />
    <attribute name="sala" type="string" context-dimension="where" />
  </conceptual-class>
  <conceptual-class name="Funcionario" external="false">
    <attribute name="id" type="integer" context-dimension="who" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <relationship name="leciona" type="association">
    <from conceptual-class="Curso" multiplicity="*" />
    <to conceptual-class="Instructor" multiplicity="1" />
  </relationship>
  <relationship name="tem" type="composition">
    <from conceptual-class="Curso" multiplicity="1" />
    <to conceptual-class="Aula" multiplicity="*" />
  </relationship>
</conceptual-model>
```

```

<relationship name="matricula" type="association">
  <from conceptual-class="Curso" multiplicity="*" />
  <to conceptual-class="Aluno" multiplicity="*" />
</relationship>
<relationship name="sessao-aula" type="generalization">
  <child conceptual-class="Aula" />
  <parent conceptual-class="Sessao" />
</relationship>
<relationship name="funcionario-instrutor" type="generalization">
  <child conceptual-class="Instrutor" />
  <parent conceptual-class="Funcionario" />
</relationship>
</conceptual-model>

```

Modelo Conceitual Modificado Temporário 1 (CM-MOD-TEMP1)

```

<conceptual-model>
  <conceptual-class name="Sessao" external="true">
    <attribute name="id" type="integer" context-dimension="none" />
    <attribute name="data" type="date" context-dimension="when" />
  </conceptual-class>
  <conceptual-class name="Aula" external="true">
    <attribute name="titulo" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Aluno" external="false">
    <attribute name="id" type="integer" context-dimension="who" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Instrutor" external="false">
    <attribute name="titulacao" type="string" context-dimension="none" />
    <attribute name="idFuncionario" type="integer" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Curso" external="false">
    <attribute name="id" type="string" context-dimension="none" />
    <attribute name="nome" type="string" context-dimension="none" />
    <attribute name="semestre" type="string" context-dimension="when" />
    <attribute name="sala" type="string" context-dimension="where" />
  </conceptual-class>
  <conceptual-class name="Funcionario" external="false">
    <attribute name="id" type="integer" context-dimension="who" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <relationship name="leciona" type="association">
    <from conceptual-class="Curso" multiplicity="*" />
    <to conceptual-class="Instrutor" multiplicity="1" />
  </relationship>
  <relationship name="tem" type="composition">
    <from conceptual-class="Curso" multiplicity="1" />
    <to conceptual-class="Aula" multiplicity="*" />
  </relationship>
  <relationship name="matricula" type="association">
    <from conceptual-class="Curso" multiplicity="*" />
    <to conceptual-class="Aluno" multiplicity="*" />
  </relationship>
  <relationship name="sessao-aula" type="generalization">
    <child conceptual-class="Aula" />
    <parent conceptual-class="Sessao" />
  </relationship>
  <relationship name="funcionario-instrutor" type="generalization">
    <child conceptual-class="Instrutor" />
    <parent conceptual-class="Funcionario" />
  </relationship>
</conceptual-model>

```

Modelo Conceitual Modificado Temporário 2 (CM-MOD-TEMP2)

```
<conceptual-model>
  <conceptual-class name="Funcionario" external="false">
    <attribute name="id" context-dimension="none" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Instrutor" external="false">
    <attribute name="titulacao" type="string" context-dimension="none" />
    <attribute name="idFuncionario" type="integer" context-dimension="who" />
  </conceptual-class>
  <conceptual-class name="Sessao" external="true">
    <attribute name="id" type="integer" context-dimension="none" />
    <attribute name="data" type="date" context-dimension="when" />
  </conceptual-class>
  <conceptual-class name="Aula" external="true">
    <attribute name="titulo" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Aluno" external="false">
    <attribute name="id" type="integer" context-dimension="who" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Curso" external="false">
    <attribute name="id" type="string" context-dimension="none" />
    <attribute name="nome" type="string" context-dimension="none" />
    <attribute name="semestre" type="string" context-dimension="when" />
    <attribute name="sala" type="string" context-dimension="where" />
  </conceptual-class>
  <relationship name="leciona" type="association">
    <from conceptual-class="Curso" multiplicity="*" />
    <to conceptual-class="Instrutor" multiplicity="1" />
  </relationship>
  <relationship name="tem" type="composition">
    <from conceptual-class="Curso" multiplicity="1" />
    <to conceptual-class="Aula" multiplicity="*" />
  </relationship>
  <relationship name="matricula" type="association">
    <from conceptual-class="Curso" multiplicity="*" />
    <to conceptual-class="Aluno" multiplicity="*" />
  </relationship>
  <relationship name="sessao-aula" type="generalization">
    <child conceptual-class="Aula" />
    <parent conceptual-class="Sessao" />
  </relationship>
  <relationship name="funcionario-instrutor" type="generalization">
    <child conceptual-class="Instrutor" />
    <parent conceptual-class="Funcionario" />
  </relationship>
</conceptual-model>
```

Modelo Conceitual Modificado (CM-MOD)

```
<conceptual-model>
  <conceptual-class name="Funcionario" external="false">
    <attribute name="id" context-dimension="none" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Instrutor" external="false">
    <attribute name="titulacao" type="string" context-dimension="none" />
    <attribute name="idFuncionario" type="integer" context-dimension="who" />
  </conceptual-class>
  <conceptual-class name="Sessao" external="true">
    <attribute name="id" type="integer" context-dimension="none" />
    <attribute name="data" type="date" context-dimension="when" />
  </conceptual-class>
  <conceptual-class name="Aula" external="true">
    <attribute name="titulo" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Aluno" external="false">
    <attribute name="id" type="integer" context-dimension="who" />
    <attribute name="nome" type="string" context-dimension="none" />
  </conceptual-class>
  <conceptual-class name="Curso" external="false">
    <attribute name="id" type="string" context-dimension="none" />
    <attribute name="nome" type="string" context-dimension="none" />
    <attribute name="semestre" type="string" context-dimension="when" />
    <attribute name="sala" type="string" context-dimension="where" />
    <attribute name="idFuncionario" type="integer" context-dimension="none" />
  </conceptual-class>
  <relationship name="leciona" type="association">
    <from conceptual-class "Curso" multiplicity "*" />
    <to conceptual-class "Instrutor" multiplicity="1" />
  </relationship>
  <relationship name="matricula" type="association">
    <from conceptual-class "Curso" multiplicity "*" />
    <to conceptual-class "Aluno" multiplicity="*" />
    <attribute name="idCurso" type="string" context-dimension="none" />
    <attribute name="idAluno" type="integer" context-dimension="none" />
  </relationship>
  <relationship name="tem" type="composition">
    <from conceptual-class "Curso" multiplicity="1" />
    <to conceptual-class "Aula" multiplicity="*" />
  </relationship>
  <relationship name="sessao aula" type="generalization">
    <child conceptual-class="Aula" />
    <parent conceptual-class="Sessao" />
  </relationship>
  <relationship name="funcionario-instrutor" type="generalization">
    <child conceptual-class="Instrutor" />
    <parent conceptual-class="Funcionario" />
  </relationship>
</conceptual-model>
```

Esquema Contextual (CONTEXT-INFO)

```
<xsd:schema elementFormDefault="qualified" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="context-info" type="contextType" />
  <xsd:complexType name="contextType">
    <xsd:sequence>
      <xsd:element name="what" type="whatType" />
      <xsd:element name="who" type="whoType" />
      <xsd:element name="where" type="whereType" />
      <xsd:element name="when" type="whenType" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whatType">
    <xsd:sequence>
      <xsd:element name="Aula">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Curso">
              <xsd:complexType>
                <xsd:attribute name="id" type="xsd:string" use="required" />
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="Titulo" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whoType">
    <xsd:sequence>
      <xsd:element name="Instructor">
        <xsd:complexType>
          <xsd:attribute name="idFuncionario" type="xsd:integer" use="required" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Aluno" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="id" type="xsd:integer" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whereType">
    <xsd:sequence>
      <xsd:element name="Curso">
        <xsd:complexType>
          <xsd:attribute name="sala" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whenType">
    <xsd:sequence>
      <xsd:element name="Sessao">
        <xsd:complexType>
          <xsd:attribute name="data" type="xsd:date" use="required" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Curso">
        <xsd:complexType>
          <xsd:attribute name="semestre" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Esquema para Validação de Consultas CQL (CONTEXT-QUERY)

```
<xsd:schema elementFormDefault="qualified" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="context-info" type="contextType" />
  <xsd:complexType name="contextType">
    <xsd:sequence>
      <xsd:element name="what" type="whatType" minOccurs="0" />
      <xsd:element name="who" type="whoType" minOccurs="0" />
      <xsd:element name="where" type="whereType" minOccurs="0" />
      <xsd:element name="when" type="whenType" minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whatType" maxOccurs="unbounded">
    <xsd:sequence>
      <xsd:element name="Aula">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Curso" minOccurs="0">
              <xsd:complexType>
                <xsd:attribute name="id" type="xsd:string" />
                <xsd:attribute name="nome" type="xsd:string" />
                <xsd:attribute name="idFuncionario" type="xsd:integer" />
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="titulo" type="xsd:string" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whoType">
    <xsd:sequence>
      <xsd:element name="Instructor" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="idFuncionario" type="xsd:integer" />
          <xsd:attribute name="titulacao" type="xsd:string" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Aluno" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="id" type="xsd:integer" />
          <xsd:attribute name="nome" type="xsd:string" />
          <xsd:attribute name="conector" type="conectorType" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whereType">
    <xsd:sequence>
      <xsd:element name="Curso" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="sala" type="xsd:string" />
          <xsd:attribute name="id" type="xsd:string" />
          <xsd:attribute name="nome" type="xsd:string" />
          <xsd:attribute name="idFuncionario" type="xsd:integer" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whenType">
    <xsd:sequence>
      <xsd:element name="Sessao" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="data" type="xsd:date" />
          <xsd:attribute name="id" type="xsd:integer" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Curso" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="semestre" type="xsd:string" />
          <xsd:attribute name="id" type="xsd:string" />
          <xsd:attribute name="nome" type="xsd:string" />
          <xsd:attribute name="idFuncionario" type="xsd:integer" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="conectorType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="and" />
            <xsd:enumeration value="or" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

Esquema de Mapeamento CQL-FLWOR (FLWOR-QUERY)

```
<xsd:schema elementFormDefault="qualified" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="context-info" type="contextType" />
  <xsd:complexType name="contextType">
    <xsd:sequence>
      <xsd:element name="what" type="whatType" minOccurs="0" />
      <xsd:element name="who" type="whoType" minOccurs="0" />
      <xsd:element name="where" type="whereType" minOccurs="0" />
      <xsd:element name="when" type="whenType" minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whatType">
    <xsd:sequence>
      <xsd:element name="Aula" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Curso" minOccurs="0">
              <xsd:complexType>
                <xsd:attribute name="id" type="xsd:string" use="required" />
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="Titulo" type="xsd:string" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whoType">
    <xsd:sequence>
      <xsd:element name="Instructor" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="idFuncionario" type="xsd:integer" use="required" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Aluno" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="id" type="xsd:integer" use="required" />
          <xsd:attribute name="conector" type="conectorType" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whereType">
    <xsd:sequence>
      <xsd:element name="Curso" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="sala" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="whenType">
    <xsd:sequence>
      <xsd:element name="Sessao" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="data" type="xsd:date" use="required" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Curso" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="semestre" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="conectorType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="and" />
      <xsd:enumeration value="or" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```


Instalação e configuração da infra-estrutura InCA-SERVE

As facilidades de infra-estrutura e serviços são providas ao desenvolvedor de duas formas: (i) por meio de APIs Java e (ii) através de extensões ao banco de dados eXist.

As APIs Java são disponibilizadas em um arquivo *jar*, denominado *incaserve*, o qual contém os pacotes de *software* da InCA, da xInCA e parte dos pacotes do StRES. A organização desse arquivo *jar* é mostrada na Tabela B.1.

Tabela B.1: Organização dos pacotes de *software* da InCA, da xInCA e do StRES

Sistema	Caminho dos pacotes
InCA	edu.gatech.coc.inca
xInCA	br.usp.icmc.xinca
StRES	br.usp.icmc.storage
xInCA-StRES	br.usp.icmc.xinca.common.data
Config	br.usp.icmc.config

Os pacotes da InCA provêem classes que permitem instanciar os objetos que realizam a troca de informação por meio de um *Registry* que implementa um mecanismo de intercâmbio *publish/subscribe*, o qual também é disponibilizado no arquivo *jar* sob a forma de um *daemon*, o qual precisa estar ativo a fim de prover a comunicação transparente entre os *peers* da aplicação.

Os pacotes da xInCA utilizam classes contidas nos pacotes da InCA com o intuito de prover componentes aptos a intercambiar dados. Já os pacotes do StRES provêm classes responsáveis pela geração automática de documentos que encapsulam a informação capturada pelos componentes xInCA. Assim como a InCA, o StRES provê um *daemon* denominado *Sessiond*, o qual deve estar ativo para permitir a geração dos documentos de sessão.

Existe ainda um conjunto de classes que fazem parte tanto da xInCA quanto do StRES. Localizadas no pacote *common.data*, essas classes resultam do processamento via JAXB dos esquemas de dados definidos para os componentes da xInCA e também do esquema contendo as informações de contexto.

Observa-se, portanto, que o arquivo *jar* não está pronto para uso, uma vez que depende das classes que manipulam as informações de contexto da aplicação. Em implementações preliminares, as APIs eram disponibilizadas como arquivos compactados. Cabia aos desenvolvedores produzir as classes necessárias e colocá-las no devido pacote a fim de possibilitar o funcionamento da infra-estrutura. Essa abordagem, evidentemente, não é adequada, uma vez que obriga o desenvolvedor a conhecer detalhes de implementação que devem ser abstraídos.

A solução para o problema foi implementar um programa que se responsabiliza por compilar o esquema de contexto e atualizar automaticamente o pacote *common.data*. Esse programa, denominado *Context Compiler* é parte de um conjunto de *softwares* utilizados para configurar a infra-estrutura, a fim de que ela funcione de acordo com as particularidades de cada aplicação. Os programas de configuração são mantidos no pacote *config*, conforme ilustra a Tabela B.1

O *Context Compiler* utiliza o compilador JAXB para gerar as classes que cuidarão do processamento das informações de contexto da aplicação. O resultado da compilação dos esquemas é um conjunto de classes Java não compiladas. Dessa forma, o passo seguinte executado pelo *Context Compiler* consiste na compilação dessas classes, produzindo arquivos executáveis (*.class*). Os arquivos *.java* e *.class* produzidos são mantidos em um diretório criado pelo *Context Compiler*. É possível utilizar as facilidades providas pelo JAXB para gerar um *javadoc* referente às classes produzidas, o qual será útil ao desenvolvedor na implementação da aplicação.

Para qualquer esquema compilado, o pacote gerado é composto pelas seguintes classes: *ContextType*, *WhatType*, *WhoType*, *WhereType*, *WhenType* e *ObjectFactory*. Essa invariabilidade do nome das classes produzidas é essencial para possibilitar o funcionamento do mecanismo que permite a junção

posterior de classes ao arquivo *incaserve.jar*. Isso porque a classe *Transducer* referencia objetos de contexto. Assim, se as classes associadas a esses objetos não existem, não é possível compilar a classe *Transducer* e produzir o arquivo *jar* utilizado para distribuição.

Para contornar esse problema, um esquema genérico é utilizado para gerar classes *dummy*¹, as quais são posteriormente substituídas pelas classes geradas no processo de compilação do esquema efetivo da aplicação.

Após a geração e compilação das classes, o programa acessa o arquivo *incaserve.jar* copiando suas classes para um novo arquivo, no qual são substituídas as classes *dummy* pelas classes geradas no processo de compilação. A implementação ocorre dessa forma porque não é possível atualizar arquivos *jar* — é necessário gerar um novo arquivo para efetuar a atualização. O funcionamento do *Context Compiler* é ilustrado na Figura B.1.

Figura B.1: Atualização do arquivo *incaserve.jar*.

Vale notar que a utilização do *Context Compiler* depende da instalação da API do JAXB, cujos pacotes devem estar no *classpath* a fim de permitir a utilização de seu compilador.

A partir da execução do *Context Compiler* o arquivo *jar* está pronto para ser utilizado. Entretanto, antes de produzir o *front-end* da aplicação, é necessário instalar e configurar *softwares* que auxiliam no armazenamento e na apresentação do conteúdo a ser produzido.

¹Classes com funcionalidade irrelevante que serão posteriormente substituídas pelas classes reais.

Softwares de Apoio

Por contar com blocos estruturais centrados no processamento de documentos XML, a infra-estrutura InCA-SERVE se beneficiou de um conjunto de ferramentas de código aberto que suportam o processamento, o armazenamento e a apresentação desse tipo de informação.

Algumas dessas ferramentas sofreram extensões a fim de prover a funcionalidade requerida pelas aplicações de captura e acesso. Foi o que aconteceu com a ferramenta de modelagem ArgoUML e com o banco de dados XML eXist. Outras, como o *Framework Cocoon* e o Apache Web Server, são utilizadas sem extensões.

Dessa forma, o desenvolvedor deve instalar os referidos *softwares* e realizar um procedimento de atualização do banco de dados eXist. A atualização do eXist envolve a substituição do arquivo *xml-rpc-1.2-patched.jar*, a fim de que a interface XML-RPC do banco ofereça o método que permite processar expressões de consulta contextual. Para que as consultas funcionem é necessário também inserir o pacote *cqlprocessor.jar* no diretório *lib* do banco. Um último procedimento de ajuste consiste na instalação da aplicação *Cocoon* que processa os documentos de sessão mantidos no banco. O *Framework Cocoon* não precisa ser instalado, uma vez que já vem incluído na distribuição do eXist.

Assim como ocorre com o arquivo *incaserver.jar*, a atualização das extensões do banco é feita de forma automática por outro programa incluído no pacote *config*, denominado *DBExtender*. Os arquivos utilizados na atualização (pacotes *jar* e a aplicação *Cocoon*) são fornecidos na distribuição compactados em um arquivo zip.

O servidor *Web* é utilizado para permitir a recuperação das mídias associadas a cada sessão de captura. Os *slides* produzidos pela *whiteboard* são armazenados como imagens PNG ou JPG. Fluxos de áudio e vídeo capturados são também salvos como arquivos de mídia e ficam em uma estrutura de diretórios idêntica à organização interna do repositório de documentos XML. Os documentos de sessão possuem ponteiros para os arquivos de mídia, de tal forma que é possível gerar um arquivo HTML a partir do documento XML com uma simples transformação XSLT.

A Tabela B.2 traz todos os softwares de apoio necessários para o desenvolvimento de aplicações de captura e acesso, listando a forma de obtenção dos mesmos e seus procedimentos de instalação.

Tabela B.2: softwares de apoio utilizados pela infra-estrutura InCA-SERVE

software	Distribuição	Programa de Configuração
ArgoUML estendido	InCA-SERVE	Nenhum
JAXB	JWSDP 1.3 ou superior	Nenhum
Apache Web Server	Apache	Nenhum
eXist XML Database	exist-db.org	DBExtender
Apache Cocoon	exist-db.org	DBExtender

Arquivo de Configuração para Daemons

Os diversos módulos que compõem a infra-estrutura InCA-SERVE precisam conhecer a localização dos *daemons* que provêem as facilidades de comunicação e armazenamento providas. A solução encontrada para flexibilizar a instalação da infra-estrutura em diferentes máquinas foi a definição de um arquivo de configuração, especificado como um documento XML. Denominado *config.xml*, esse documento determina a URL e a porta dos serviços disponibilizados, e também informa o diretório onde devem ser salvos os arquivos de mídia produzidos nas sessões de captura, conforme ilustra o trecho de código a seguir.

```
<incaserve_config>
  <incad_host>iclass.icmc.usp.br</incad_host>
  <incad_port>6789</incad_port>
  <sessiond_host>iclass.icmc.usp.br</sessiond_host>
  <sessiond_port>4321</sessiond_port>
  <exist_host>iclass.icmc.usp.br</exist_host>
  <exist_port>8080</exist_port>
  <legacydb_host>iclass.icmc.usp.br</legacydb_host>
  <legacydb_port>3306</legacydb_port>
  <medias_directory>/var/www/html/medias</medias_directory>
  <medias_url>http://iclass.icmc.usp.br/medias</medias_url>
  <app_homepage>http://iclass.icmc.usp.br/remotemeeting</app_homepage>
  <apache_html_root>/var/www/html</apache_html_root>
</incaserve_config>
```

O arquivo de configuração apresentado está ajustado para a máquina *iclass.icmc.usp.br*. Para a configuração de um outro *host*, na maioria dos casos é necessário apenas ajustar o nome da máquina. Entretanto, alguns outros ajustes nas portas podem ser necessários. Por exemplo, o banco legado, nesse exemplo, é o MySQL. Em outro servidor pode ser utilizado um outro banco, acarretando o ajuste da porta.

O arquivo de configuração deve ser colocado no mesmo diretório do arquivo *incaserve.jar* para que possa ser corretamente acessado pelos programas da infra-estrutura InCA-SERVE.

A definição do arquivo de configuração constitui o passo final do processo de instalação e configuração da infra-estrutura InCA-SERVE. Após esse procedimento, o desenvolvedor pode iniciar o desenvolvimento de aplicações de captura e acesso.