Context Kernel: um *Web Service* baseado
nas dimensões de informação de contexto

Carlos Roberto Esperança de Arruda Junior

Orientadora: Profa. Dra. Maria da Graça Campos Pimentel

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

“VERSÃO REVISADA APÓS A DEFESA”

Data da Defesa:	29/04/2003
Visto do Orientador:

USP – São Carlos
Junho/2003

Este documento foi preparado com o formatador de textos $\text{\LaTeX}$. O sistema de citações de referências bibliográficas $\text{bib}\text{\LaTeX}$ utilizando o padrão *apalike*.

Dedicatória

*Dedico aos meus pais, Carlos e Marli, pelo amor, compreensão
e pela chance que me deram em poder chegar até aqui.
sempre me incentivando em todos os momentos.*

A minha irmã Liliane e minha namorada Dani, por me deixar amá-las.

Agradecimento

Inicialmente agradeço a Deus, por toda proteção e força,

À minha orientadora Graça Pimentel pelo apoio, preocupação e ensinamentos que me foram dedicados nestes anos de orientação,

À professora Renata Pontin, pelos ensinamentos e incentivos.

Aos amigos do Laboratório Intermídia que sempre me incentivaram: Ale, Andrea, Billy, Cláudia, Daniel, Laércio, Pedro, Renan, Renato e Toño,

Aos meus amigos de graduação: Adriano, Daniel, Emilio, Gustavo, Luís, Marcelo, Maroldi, Sullivan e Waldomiro,

Aos meus amigos de São Carlos e de Ourinhos, pelo companheirismo e amizade demonstrados,

E a todos que contribuíram direta ou indiretamente para a realização deste trabalho, ou que simplesmente torceram pelo meu sucesso.

Resumo

No início dos anos 90, o pesquisador Mark Weiser vislumbrou uma forma de computação baseada na integração transparente entre tecnologia e atividades humanas a qual denominou computação ubíqua. Idealizada como uma mudança no paradigma de interação entre usuários e computadores, a computação ubíqua tem como temas de pesquisa interfaces naturais, aplicações cientes de contexto e aplicações de captura e acesso de atividades humanas.

Desenvolvidos para apoiar serviços dedicados de captura, de armazenamento e de processamento de informações de contexto, esforços como *Context Fabric*, *EventHeap* e *Gaia* não exploram a plataforma de *Web Services* como alternativa para o tratamento da heterogeneidade entre aplicações; além disso, as dimensões de contexto propostas na literatura não são apoiadas diretamente. Este trabalho teve como propósito investigar e formalizar um conjunto de operações para armazenamento e recuperação de informações de contexto com base nas dimensões propostas na literatura: *who*, *when*, *where*, *what*, *why* e *how*. Como resultado, desenvolveu-se um núcleo de operações, denominado *Context Kernel*, implementado como um *Web Service*. Neste trabalho são apresentadas a modelagem do *Context Kernel*, as especificações dos serviços em WSDL, a linguagem padrão para descrição de *Web Services*, e das APIs implementadas para acesso ao *Context Kernel* por aplicações Java e PHP.

Abstract

In the 90s, the researcher Mark Weiser envisioned a form of computation based on the transparent integration between technology and human's activities which called ubiquitous computing. Idealized as a change in the paradigm of interaction between users and computers, the ubiquitous computing has as research themes natural interfaces, context awareness applications and capture and access applications.

Recent work including Context Fabric, EventHeap and Gaia support the manipulation of context information by means of mechanisms that do not exploit the Web Services platform; moreover, the context dimensions proposed in the literature are not supported directly. Aiming at providing support to the development of context awareness applications, this work has investigated a set of operations for the storage and retrieval of context information based on the dimensions proposed in literature: *who*, *when*, *where*, *what*, *why* and *how*. As result, the *Context Kernel* has been modelled and implemented as operations formalized as a *Web Service*: the *Context Kernel* is presented by means of the specification of the services described in WSDL, the standard language for the description of Web Services, and the APIs allowing access the Web Service by Java and PHP applications.

Sumário

1	Introdução	1
2	Computação Ubíqua	5
2.1	Interfaces Naturais	6
2.2	Computação Ciente de Contexto	8
2.3	Captura e Acesso de Atividades Humanas	9
2.4	Conceitos e Terminologia Relacionados	10
2.5	Computação no Cotidiano	13
2.6	Considerações Finais	15
3	Computação Ciente de Contexto	17
3.1	Definição de Informação de Contexto	18
3.2	Definição de Computação Ciente de Contexto	20
3.2.1	Categorias de Consciência de Contexto	22
3.3	Desenvolvimento de Aplicações Cientes de Contexto	24
3.4	Trabalhos Relacionados	27
3.4.1	Aplicações Cientes de Contexto	28
3.4.2	<i>Toolkit</i> e Infra-estrutura Cientes de Contexto	29
3.4.3	<i>Middleware</i> Cientes de Contexto	30
3.5	Considerações Finais	31
4	Web Services	33
4.1	Definição	34
4.2	Arquitetura	35
4.3	Linguagem XML e Definição XML Schema	36
4.3.1	XML (Extensible Markup Language)	37

4.3.2 XML Schema	37
4.4 Protocolo de Comunicação SOAP	38
4.5 Linguagem de Descrição WSDL	39
4.6 Considerações Finais	42
5 O Context Kernel	45
5.1 Modelo de Armazenamento do Context Kernel	46
5.2 Arquitetura do Context Kernel	49
5.3 Serviços Disponibilizados pelo Context Kernel	50
5.3.1 Registro	50
5.3.2 Armazenamento	52
5.3.3 Consulta	53
5.3.4 Recuperação	55
5.4 Aplicações Heterogêneas Integradas com o Context Kernel	57
5.4.1 Coteia	57
5.4.2 iClass	58
5.5 Cenários de Integração com o Context Kernel	59
5.5.1 Cenário 1: Coteia armazena e iClass consulta regras	59
5.5.2 Cenário 2: iClass armazena e Coteia consulta regras	62
5.6 Limitações do Context Kernel	64
5.7 Considerações Finais	66
6 Conclusões	67
6.1 Contribuições	68
6.2 Trabalhos Futuros	68
6.3 Considerações Finais	70
A Apêndice: Especificação WSDL	71
Referências	85

Lista de Figuras

4.1	Arquitetura Básica de <i>Web Services</i> (Adaptada de (W3C, 2002b)) . .	36
4.2	Elementos da Linguagem WSDL (Adaptada de (W3C, 2002b))	40
5.1	Modelo de Classes do Context Kernel	48
5.2	Arquitetura do Context Kernel	49

Lista de Abreviaturas

API *Application Programming Interface*

DTD *Document Type Definition*

GaTech *Georgia Institute of Technology*

HTTP *Hypertext Transfer Protocol*

ICMC *Instituto de Ciências Matemáticas e de Computação*

InCA *Infrastructure for Capture and Access*

IP *Internet protocol*

SERVE *Infrastructure for Store, Extend, Retrieve and Visualize Evolutionary Information*

SOAP *Simple Object Access Protocol*

SQL *Structured Query Language*

SSL *Secure Sockets Layer*

TLS *Transport Layer Security*

UML *Unified Modeling Language*

URI *Uniform Resource Identifiers*

XML *Extensible Markup Language*

W3C *World Wide Web Consortium*

WSDL *Web Services Description Language*

Introdução

A computação está cada vez mais difundida nos diversos ambientes, seja de trabalho, acadêmico ou doméstico. Considerado a principal referência na área de computação ubíqua, Weiser (1991) vislumbrou uma forma de computação ininterrupta e invisivelmente incorporada às mais diversas atividades diárias, que poderia propiciar benefícios aos usuários sem que esses sequer notem sua presença. Weiser denominou computação ubíqua essa forma transparente de integração entre tecnologia e atividades humanas.

Uma das classes principais da computação ubíqua é a computação ciente de contexto que tem como propósito facilitar a interação do usuário com o computador. Através da personalização de serviços, a interação entre seres humanos é beneficiada pela riqueza da linguagem humana, com sua capacidade de interpretação e de entendimento implícito de situações diárias. Entretanto, a interação usuário-computador necessita de informações explícitas para facilitar a comunicação: as informações de contexto. Por exemplo, um sistema que organize visitas para uma determinada localidade pode personalizar essas visitas de acordo com informações de contexto, como interesse do visitante, localização do visitante, tempo disponível para o passeio, restrições financeiras, capacidade de deslocamento do visitante e condições climáticas do ambiente. Independente de suas origens, domínios e características específicas, aplicações que utilizam informações de contexto são chamadas aplicações cientes de contexto.

As aplicações de computação ubíqua devem ter acesso as informações de

contexto para poderem se adaptar a diferentes tarefas, usuários e ambientes (Banavar and Bernstein, 2002). Segundo Abowd & Mynatt (2000), um dos desafios da computação ubíqua é tornar suas aplicações cientes de contexto. Para isso, sistemas recentes, como o Context Fabric (Hong and Landay, 2001), o EventHeap (Johanson and Fox, 2002) e o Gaia (Hess et al., 2002) têm sido desenvolvidos com o objetivo de fornecer suporte para a construção de serviços dedicados a captura, ao armazenamento e ao processamento de informações de contexto.

Um dos **problemas** encontrados pelos desenvolvedores desses sistemas tem sido como lidar com diferentes níveis de heterogeneidade existentes entre as aplicações. Outro **problema** é a diversidade de tratamento dada a informações de contexto, uma vez que as dimensões de contexto propostas na literatura não são apoiadas diretamente. O **objetivo** do trabalho aqui apresentado foi investigar um serviço que suporte operações transparentes de armazenamento, extensão e recuperação de informação de contexto para aplicações cientes de contexto, independente de suas diferenças de tecnologia de rede, sistema operacional ou linguagem de programação. A formalização desse serviço baseou-se nas dimensões semânticas definidas por Abowd & Mynatt (2000) e Truong et al. (2001): identificação (*who*), localização espacial (*where*), localização temporal (*when*), modo de captura e de acesso (*how*), atividade (*what*) e motivação (*why*).

Como **resultado** do trabalho, foi formalizado e implementado um conjunto de operações para o gerenciamento de informações de contexto com base nas dimensões de contexto propostas na literatura. Assim, este trabalho apresenta a especificação formal dos serviços disponibilizados pelo Context Kernel e uma biblioteca de funções implementadas para acesso ao Context Kernel por aplicações Java e PHP.

O trabalho aqui apresentado foi desenvolvido no contexto do Projeto InCA-SERVE¹(Pimentel et al., 2001a) do Programa de Cooperação Internacional entre ProTeM-CC/CNPq/Brasil e NSF/EUA. O InCA-SERVE é uma parceria entre o grupo do Dr. Gregory Abowd, do Georgia Institute of Technology (GaTech), e pesquisadores do Laboratório Intermídia do Instituto de Ciências Matemáticas e de Computação (ICMC) da USP de São Carlos.

O Projeto InCA-SERVE prevê a construção das infra-estruturas InCA e SERVE. A infra-estrutura InCA (*Infrastructure for Capture and Access*) provê serviços para captura e acesso de informações associadas a experiências diárias para vários domínios. Além de uma estrutura para aplicações que dê suporte

¹<http://coweb.icmc.usp.br/incaserve/>

à captura de atividades realizadas ao vivo, a InCA deve suportar um modelo de informação extensível para a captura de experiências que podem ser utilizadas para integrar as atividades que ocorrem fora da sessão de captura.

A infra-estrutura SERVE (*Infrastructure for Store, Extend, Retrieve and Visualize Evolutionary Information*) está sendo definida baseando-se em um conjunto de aplicações Web independentes e complementares, integradas através da definição de APIs (*Application Programming Interface*). Essas aplicações Web devem suportar serviços de armazenamento, extensão, recuperação e visualização das informações hipermídia evolucionárias capturadas pela infra-estrutura InCA. As infra-estruturas InCA e SERVE estão sendo utilizadas na construção de aplicações que suportam atividades de ensino e de reuniões nos ambientes do ICMC-USP e do GaTech.

Esta dissertação está organizada da seguinte forma: no Capítulo 2 é apresentada uma visão geral da área de computação ubíqua, discutindo suas principais áreas de pesquisa. O Capítulo 3 discute os conceitos apresentados na literatura em relação as informações de contexto e à computação ciente de contexto. No Capítulo 4 é apresentada uma introdução ao conceito de *Web Services* e a seus mecanismos para comunicação, descrição e formalização de serviços. O Capítulo 5 apresenta o projeto, a implementação e a integração do Context Kernel. No Capítulo 6 são discutidas as contribuições do trabalho e os trabalhos futuros.

Computação Ubíqua

A principal referência à computação ubíqua data de 1991, quando o pesquisador Mark Weiser (1991) introduziu o tema e visualizou pessoas e ambientes acrescidos de recursos computacionais que forneceriam informações e serviços quando e onde essas pessoas assim os desejassem. Na visão de Weiser, a tecnologia deve estar integrada ao ambiente e auxiliar os usuários em suas atividades cotidianas. Inspirado pelos cientistas sociais, filósofos e antropólogos, Weiser (1994) previu uma mudança radical no paradigma computacional ao definir que as pessoas utilizam melhor aquilo que está efetivamente invisível. Nesse contexto, os computadores são embutidos¹ de forma transparente no ambiente do usuário.

O impacto dos computadores embutidos no ambiente pode ser análogo a outras duas tecnologias que se tornaram ubíquas. A primeira é a tecnologia da escrita, cujo produto pode ser encontrado em todos os lugares, desde estampa de roupas até em quadros de aviso. A segunda é a tecnologia da eletricidade, cujo produto caminha invisível entre as paredes das casas e dos escritórios. As tecnologias da escrita e da eletricidade tornaram-se tão comuns que seu impacto em nossa vida cotidiana foi praticamente esquecido (Weiser, 1991). Ainda, existem computadores embutidos nos mais diversos ambientes e aparelhos, como interruptores de luzes, termostatos, fogões e geladeiras. A integração dessas tecnologias com as atividades humanas determinam a computação ubíqua.

¹Para fins deste trabalho, o termo *embedded* será tratado como embutido.

A visão de Weiser descrevia a proliferação de dispositivos em escalas variadas: pequenos e pessoais (*inch-scale*), médio porte (*foot-scale*) e grandes e de uso coletivo (*yard-scale*) (Weiser, 1991). A proposta de Weiser previa que a interação física entre usuário e computador estaria mais distante dos dispositivos tradicionais, como teclado e *mouse*, e mais próxima do paradigma de interação do usuário com o mundo físico, no qual os seres humanos falam, gesticulam e escrevem para se comunicar com outros seres humanos.

Segundo Weiser (1991), seria preciso desenvolver dispositivos heterogêneos para oferecer diferentes formas de interação. De fato, ocorreu a proliferação desses dispositivos que visam facilitar a comunicação, como (*Personal Digital Assistant*), *tablets* digitais², *laptops* e lousas eletrônicas.

Outro aspecto vislumbrado por Weiser (1991) era que novas aplicações deveriam ser desenvolvidas para explorar o uso desses novos dispositivos. Com isso, a computação ubíqua proposta estava além de ser apenas infra-estrutura, mas um novo paradigma de interação inspirado pelos acessos constantes a informações e capacidades computacionais. Nos últimos anos, o desenvolvimento dessas aplicações tem focado em três aspectos de interação na área de computação ubíqua: interfaces naturais, aplicações cientes de contexto, e captura e acesso de atividades humanas (Abowd and Mynatt, 2000).

Este capítulo tem como objetivo contextualizar a área de computação ubíqua e apresentar seus três principais aspectos de interação. Isso é feito nas Seções 2.1, 2.2 e 2.3. Na Seção 2.4 são apresentados os principais termos associados à computação ubíqua. Na Seção 2.5 é apresentada uma nova forma de computação ubíqua: a computação no cotidiano. Na Seção 2.6 discute-se nas considerações finais a necessidade da computação ciente de contexto na interação usuário-computador.

2.1 Interfaces Naturais

Os desafios da computação ubíqua não envolvem somente a proliferação de dispositivos e redes computacionais pelo ambiente, mas também fornecimento de uma interface natural para seus usuários. Interfaces naturais têm por propósito facilitar a capacidade de comunicação entre usuários e computadores fornecendo suporte às formas comuns de expressão humana e utilizando as ações implícitas ou explícitas que ocorrem durante essa comunicação. Hu-

²Dispositivos computacionais portáteis semelhantes a uma prancheta com telas sensíveis ao toque.

manos falam, gesticulam e escrevem para se comunicarem uns com os outros e tudo isso poderia ser usado, implícita ou explicitamente, como entrada para sistemas de computação ubíqua. Pesquisas na área de interfaces naturais têm focado nas técnicas de fala e escrita, mas interfaces para essas formas de interação ainda não lidam eficientemente com os erros que naturalmente ocorrem, além de serem extremamente difíceis de serem desenvolvidas (Abowd and Mynatt, 2000).

O objetivo do projeto de interfaces naturais tem sido remover a interface física como uma barreira entre o usuário e a tarefa a ser realizada por meio do computador. O *mouse*, por exemplo, não é transparente para o usuário, violando a visão não-intrusiva da computação ubíqua. A área que envolve técnicas de interação transparente é muito rica em pesquisas, incluindo técnicas de reconhecimento de escrita e de gestos, interação com canetas, técnicas de fala e percepção computacional, interação com sensores e manipulação de artefatos eletrônicos. A comunidade científica da área de interação usuário-computador vem pesquisando há anos o desenvolvimento de interfaces mais amigáveis aos usuários. Entre os trabalhos realizados, destacam-se técnicas de interação baseadas em voz (Witbrock and Hauptmann, 1998), em escrita (Schilit et al., 1998), manipulação de dispositivos eletrônicos (Ishii and Ullmer, 1997) e utilização de sensores em dispositivos computacionais para sua manipulação física (Harrison et al., 1998).

Interfaces que suportam interação mais próxima do paradigma de comunicação humana como fala, escrita ou gestos estão começando a substituir ou complementar os elementos do paradigma de interfaces gráficas tradicionais. Essas novas interfaces facilitam o aprendizado e a usabilidade dos sistemas sem modificar drasticamente a estrutura das tarefas humanas que são executadas. Interação baseada em canetas, por exemplo, que passou por uma pesquisa frustrada devido a resultados pouco expressivos nos anos 90, agora ressurgiu como foco de intensa pesquisa (Abowd et al., 2002).

Abowd & Mynatt (2000) fazem uma divisão dos pontos centrais para o desenvolvimento das interfaces naturais efetivas: tipos de dados naturais de primeira classe e tratamento de erros em interações baseadas em reconhecimento.

Tipos de Dados Naturais de Primeira Classe

Para o desenvolvimento de aplicações com interfaces naturais é necessário manipular outras formas de entrada de dados tão facilmente quanto aquelas provenientes dos dispositivos convencionais, como teclado ou *mouse*. Os dados

provenientes das interfaces naturais, como áudio, vídeo e informações sensoriais precisam se tornar tipos de dados de primeira classe no desenvolvimento de sistemas interativos. Da mesma forma que as ferramentas de desenvolvimento de interfaces devem fornecer um conjunto básico para suportar as operações tradicionais, como manipulação de texto, e as primitivas de entrada por meio de teclado e *mouse*, essas ferramentas devem fornecer suporte básico à manipulação de linguagem natural e outras formas de entrada de dados, como áudio, vídeo ou *eletronic ink* (interação suportada por caneta).

Tratamento de Erros em Interações Baseadas em Reconhecimento

O uso de interfaces naturais em tarefas baseadas em reconhecimento pode acarretar novos erros relacionados à entrada de dados. Tais erros estão ligados, principalmente, ao reconhecimento de linguagem natural e de escrita. O tratamento de erros associado à entrada de dados em ambientes computacionais não é um problema recente. O uso de interfaces naturais acentua erros pré-existentes e ocasiona novos tipos de erros. Com isso, surgiram novas áreas de pesquisa para manipulação de erros em interfaces baseadas em reconhecimento. Abowd & Mynatt (2000) destacam três dessas novas áreas: descoberta da ocorrência de erros, redução de erros, e recuperação e correção de erros.

2.2 Computação Ciente de Contexto

Na interação usuário-computador, Dey & Abowd (1999) definem contexto como qualquer informação relevante sobre as entidades que participam dessa interação, incluindo os próprios usuário e computador. Uma entidade pode ser uma pessoa, um lugar, ou um objeto considerados relevantes para a interação entre o usuário e a aplicação. De maneira geral, contexto refere-se às situações físicas e sociais nas quais os dispositivos computacionais estão alocados.

A computação ciente de contexto tem como propósito adquirir e utilizar informações de contexto adquiridas de um dispositivo computacional, a fim de fornecer serviços que são apropriados para uma determinada pessoa, ambiente, tempo, eventos, etc. Por exemplo, pode-se determinar que um telefone celular nunca emita som durante um concerto, caso o sistema possa reconhecer a localização do telefone e o horário do concerto. Informações de contexto são úteis quando pode-se interpretá-las. Aumentando o acesso do computador as informações de contexto, pode-se enriquecer a comunicação usuário-computador. A computação ciente de contexto será discutida detalhadamente no Capítulo 3.

2.3 Captura e Acesso de Atividades Humanas

Os seres humanos despendem grande parte de seu tempo, independentemente de sua localização, registrando eventos dos quais participam e tentando recuperar as informações mais importantes que foram registradas. Como os humanos são incapazes de registrar todas as informações de seu interesse, faz-se uso de recursos como anotações e gravações de áudio e/ou vídeo. Com a difusão da multimídia nos ambientes computacionais, o ser humano passou a utilizar o computador como forma de capturar e memorizar por meio de documentos tradicionais, hiperdocumentos ou mídias de áudio e vídeo.

Com a utilização desses recursos computacionais para suportar a captura e o acesso de informações, o ser humano pode enfim focar sua atenção em atividades que desempenha com maior eficiência, como abstração, sumarização e interpretação. Uma vez estabelecida a eficiência da utilização desses recursos para o registro de memórias, surgiram aplicações para tornar mais eficiente o processo de captura e o posterior acesso às informações capturadas. Essas aplicações permitiram o registro simultâneo de diversos fluxos de informação, preservando as inter-relações existentes entre cada um desses fluxos, o que possibilita uma série de facilidades de acesso ao conteúdo capturado.

Estudos relacionados ao tema de captura e acesso de informações surgiram antes mesmo do surgimento do conceito de computação ubíqua. Em 1984, Schmandt & Arons (1984) desenvolveram um sistema de captura de áudio das chamadas telefônicas, fornecendo posterior acesso aos dados capturados através de facilidades como geração de resumos das chamadas capturadas. Com o passar dos anos, várias aplicações foram desenvolvidas na linha de captura e acesso, principalmente relacionadas a aulas e reuniões, eventos nos quais seus participantes normalmente necessitam lembrar de detalhes abordados. Entretanto, é difícil para o ser humano capturar todos os detalhes abordados pelo palestrante nesses eventos.

Com o intuito de solucionar esse problema, iniciaram-se pesquisas para investigar os problemas associados à captura de experiências ao vivo e o fornecimento de acesso à informação capturada. Os sistemas projetados para essa finalidade geralmente capturam informações de áudio, vídeo e escrita em uma lousa eletrônica, por meio de um ambiente próprio com recursos computacionais e de multimídia. Entre os trabalhos pioneiros de captura de reuniões, destacam-se: Coral (Minneman et al., 1995), projetado inicialmente para auxiliar na produção de documentos de avaliação de projetos; DOLPHIN (Streitz et al., 1994),

suportava reuniões face-a-face e distribuídas; e Tivoli (Pedersen et al., 1993), utilizado para resumir resultados de uma reunião capturada e utilizava dispositivos de computação ubíqua, tais como lousas eletrônicas.

No contexto educacional, o Cornell Lecture Browser (Mukhopadhyay and Smith, 1999) visa não interferir no estilo de aula do instrutor e permite que ele utilize recursos como quadro-negro e lousa branca. Nesse mesmo contexto, o projeto eClass (Abowd, 1999a) deu ênfase à transparência dos processos de captura e autoria dos hiperdocumentos multimídia que são disponibilizados através da Web (Pimentel et al., 2001c, 2000). Ainda no escopo do projeto eClass, Brotherton et al. (1998) apresentam o uso de informação de contexto na tarefa de inicialização de sessões de captura de aulas.

Outros trabalhos focaram na pré e pós-produção de material proveniente de atividades de captura. Barger et al. (1999) e White et al. (2000) relatam o desenvolvimento de um sistema de captura que oferece suporte para a realização de anotações referentes a material capturado.

2.4 Conceitos e Terminologia Relacionados

A computação ubíqua, por ser uma área relativamente recente, 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, possuem definições diferentes. Apesar dessa variação, a grande maioria das definições compartilha a definição de computação ubíqua proposta por Weiser (1991). Segundo Weiser (1991), computação ubíqua refere-se a integração realizada de forma transparente entre tecnologia e atividades humanas.

Pervasive Computing

Pervasive computing é um termo utilizado para caracterizar uma tendência que está surgindo atualmente: numerosos dispositivos computacionalmente invisíveis e naturalmente acessíveis.

A fase atual da *pervasive computing*, na qual computadores estão embutidos em muitos dispositivos, pode ser analisada de várias maneiras. Ark & Selker (1999) definem quatro aspectos principais que facilitam sua propagação: a computação está difundida por todo o ambiente; as ferramentas estão se tornando mais vez mais acessíveis; os usuários apresentam cada vez mais mobili-

dade; e a comunicação está se expandindo, seja entre indivíduos, entre indivíduos e computadores ou entre computadores.

A idéia da *pervasive computing* é tornar o computador invisível e o ambiente repleto de dispositivos computacionais, como sensores e computadores portáteis. A *pervasive computing* é extremamente relacionada com esses dispositivos que compõe seu ambiente. O aumento desses dispositivos pode ser alcançado por meio da redução do tamanho dos dispositivos ou por meio do desenvolvimento de serviços que possibilitem que esses dispositivos acessem seus serviços através da rede.

A evolução computacional tem desenvolvido dispositivos cada vez menores e, com a possibilidade de acesso remoto aos serviços, tem transformado essa forma de computação em uma atividade que pode ser deslocada para acompanhar as atividades humanas. Essa diferenciação de mobilidade é que determina os conceitos de *pervasive computing* e computação ubíqua. Segundo Lyytinen & Yoo (2002), a computação ubíqua é altamente embarcada e altamente móvel, enquanto a *pervasive computing* é altamente embarcada e com baixa mobilidade.

Calm Computing

Garlan et al. (2002) definem que o recurso mais precioso em um computador não são seus processadores, memória, disco ou rede. Para ele, o recurso computacional mais precioso é a atenção do usuário. A idéia de *calm computing* (ou *calm technology*) é empregar os conceitos de centro e periferia de atenção e pode deslocar elementos de uma região para outra (Weiser and Brown, 1996). Periferia é o nome dado a tudo aquilo que não nos preocupamos de forma explícita. Por exemplo, ao dirigirmos um carro nossa atenção está concentrada na estrada, no rádio, nos passageiros, e não no barulho do motor. Entretanto, um barulho diferente e inusitado que ocorra no motor pode ser percebido imediatamente, demonstrando que estávamos sintonizados com o barulho na periferia e poderíamos rapidamente atendê-lo.

Estar na periferia não significa não ser importante. O que está na periferia em um dado momento pode estar em um próximo momento no centro de nossa atenção e ser extremamente importante. Na *calm computing*, facilmente pode ocorrer uma mudança da periferia para o centro e vice-versa. Segundo Weiser & Brown (1996), essa mudança define o conceito de tranquilizante (*calm*) por duas razões:

1. ao colocarmos um elemento na periferia, estamos aptos a coordenar muito

mais elementos do que estaríamos se todos estivessem no centro de nossa atenção. Os elementos na periferia são coordenados pela parte do cérebro dedicada ao processamento periférico. Dessa forma, a periferia é gerenciada sem sobrecarga;

2. ao passarmos elementos da periferia para o centro, adquirimos o controle sobre esses elementos. De forma periférica nós podemos nos conscientizar que alguma coisa pode não estar ocorrendo de forma correta. Ao movermos esse elemento da periferia para o centro, nós estaremos efetivando o tratamento do eventual problema.

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, que é considerada invisível. Por exemplo, os editores de texto atuais fornecem centenas de funções de processamento, cuja complexidade é transparente aos usuários.

Wearable Computing

A proposta de *wearable computing* é que os computadores e seus usuários não seriam mais conceitos separados, mas uma simbiose (Starner et al., 1998). Os usuários estariam livres para utilizar os computadores de uma maneira direta e natural, de modo a se tornarem uma extensão real dos seres humanos.

No final dos anos 90, desenvolveram-se aplicações que antes eram impraticáveis. A *wearable computing* com sua utilidade, acessibilidade e ergonomia poderia suplantiar o convencional *desktop* como a interface padrão para computação. Por exemplo, o monitor poderia estar embutido nos óculos dos usuários e o *mouse* poderia ser substituído por luvas digitais.

Com esse novo conceito, pode-se descrever um conjunto grande de dispositivos e conceitos. Starner et al. (1998) sugerem 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 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 se moldando ao longo da interação com o usuário.

Rhodes (1997) define características próprias da *wearable computing*:

- pode ser utilizada por usuários móveis;
- deve ser manipulada com a utilização mínima das mãos, sendo preferencialmente coordenada através da fala ou de dispositivos especializados;
- deve utilizar sensores para armazenar informações sobre o ambiente do usuário;
- deve produzir a todo momento serviços úteis a seus usuários.

A *wearable computing* e a computação ubíqua apresentam formas distintas de expandir as capacidades humanas através de meios computacionais. Enquanto a computação ubíqua expande o ambiente físico do usuário através de redes computacionais e sensores, a *wearable computing* fornece uma plataforma móvel visando interagir diretamente com o corpo do usuário.

2.5 Computação no Cotidiano

Segundo Abowd (1999b), uma pesquisa na área de computação ubíqua deve seguir determinados critérios:

- as pesquisas devem ser baseadas nas aplicações. Weiser (1993) já previa que as aplicações seriam o ponto central da computação ubíqua;
- os sistemas desenvolvidos devem seguir implicitamente alguma noção de escala, que neste caso pode ser o número ou o tipo dos dispositivos, o espaço físico da rede distribuída, o tempo de uso da aplicação ou o número de pessoas utilizando o sistema;
- o sistema desenvolvido deve se relacionar ao uso real e cotidiano;
- a utilização do sistema desenvolvido deve ser avaliada para que se possa determinar seu impacto em relação aos usuários.

Os dois últimos critérios são os mais importantes, uma vez que juntos eles determinam um desafio presente nas aplicações de computação ubíqua: a computação no cotidiano (*everyday computing*), segundo a qual se faz necessário

prover disponibilidade de serviço ininterrupta vinte e quatro horas por dia, sete dias por semana, visando auxiliar computacionalmente as atividades informais e não-estruturadas comuns no dia-a-dia das pessoas.

A computação no cotidiano (*everyday computing*) é uma nova forma de computação ubíqua baseada nos eventos que ocorrem em nossas vidas. 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 cotidianos, 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 & 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 ou final previsto inicialmente;
- possuem múltiplas atividades ocorrendo de forma concorrente: as aplicações que são desenvolvidas para expandir as tarefas cotidianas dos usuários devem permitir que os mesmos possam monitorar e retomar suas 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 em 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 de 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 atividades humanas – sejam fortemente coordenados.

2.6 Considerações Finais

Este capítulo apresentou o tema computação ubíqua, com suas três sub-áreas principais: interfaces naturais, computação ciente de contexto e captura e acesso de atividades humanas. Como o trabalho proposto é centrado em consciência de contexto, o capítulo não vislumbrou detalhar especificamente cada uma das sub-áreas, mas contextualizar de forma geral a área de computação ubíqua e introduzir a sub-área de computação ciente de contexto, detalhada no próximo capítulo.

Na comunicação entre humanos, as informações que são trocadas, implícita ou explicitamente, incrementam a conversação e o entendimento. O trabalho descrito nesta dissertação teve como propósito desenvolver um conjunto de operações que permitisse que aplicações Web pudessem ser acrescidas dessas informações relevantes, consideradas informações de contexto. Possibilitando-se o acesso as informações de contexto, a comunicação entre usuários e aplicações, bem como a comunicação entre aplicações poderão ser enriquecidas.

Computação Ciente de Contexto

Na interação entre humanos, boa parte das informações é trocada sem comunicação explícita. Gestos, expressões faciais, interação com outros humanos ou objetos, informações compartilhadas podem ser utilizados para auxiliar o entendimento da comunicação. Na interação usuário-computador, a informação de contexto é raramente compartilhada entre os participantes. Nesse tipo de interação, Dey & Abowd (1999) definem contexto como qualquer informação relevante sobre as entidades que participam da interação entre usuário e computador, incluindo os próprios usuário e computador.

As aplicações poderiam requisitar que seus usuários exprimissem de forma explícita todas as informações relevantes em uma dada situação, mas o objetivo da computação ciente de contexto é facilitar a interação dos usuários com o computador. Obrigá-los a inserir muitas informações é uma tarefa difícil e cansativa para os mesmos. Dey & Abowd (1999) sugerem que o desenvolvimento de aplicações cientes de contexto deve adquirir informações de contexto de modo automatizado, disponibilizando-as em um ambiente computacional em tempo de execução, e delegando aos desenvolvedores a tarefa de decidir se a informação é realmente relevante e como lidar com ela.

Com a utilização de informações de contexto, as aplicações ganhariam benefícios na realização de suas tarefas. Por exemplo, estando ciente do horário atual, aplicações como agendas ou calendários poderiam enviar avisos a seus usuários; estando ciente da identificação de seus usuários, aplicações poderiam

personalizar sua aparência. Ainda, diferentes tipos de informações de contexto podem ser utilizados em conjunto. Por exemplo, aplicações cientes de tempo e de identidade podem expandir seus serviços de histórico e de busca de dados.

Este capítulo objetiva detalhar o conceito de computação ciente de contexto bem como discutir as dimensões provedoras de informações de contexto, com base nos trabalhos de Dey (2000) e Dey & Abowd (1999). Isso é feito nas Seções 3.1 e 3.2. Na Seção 3.3 discute-se os requisitos de um *framework*¹ conceitual que suporta o desenvolvimento, a construção e a execução de aplicações cientes de contexto. Na Seção 3.4 discute-se trabalhos relacionados ao Context Kernel e na Seção 3.5 discute-se as características mais importantes da computação ciente de contexto e sua relação com o presente trabalho.

3.1 Definição de Informação de Contexto

De acordo com Dey (2000) e Dey & Abowd (1999), alguns autores definem informação de contexto baseada em exemplos. Entre esse autores, Schilit & Theimer (1994) referem-se a contexto como localização, identificação das pessoas e objetos próximos ao usuário e mudanças que ocorrem com esses objetos. Brown et al. (1997) referem-se a contexto como localização, identificação das pessoas próximas ao usuário, hora do dia, estação do ano, temperatura, entre outros. Ryan et al. (1997) referem-se a contexto como localização, ambiente, identificação e tempo. Salber et al. (1998) definem contexto como informações sobre pessoas ou dispositivos que podem ser utilizados para transformar o modo como um sistema fornece seus serviços. Como exemplo dessas informações, Salber et al. (1998) definem dados emocionais, históricos, de localização, de intenção e de foco do usuário.

Outros autores, ainda de acordo com Dey (2000) e Dey & Abowd (1999), definem informação de contexto a partir de sinônimos, referindo-se a contexto como ambiente ou situação. Entre esses autores, Brown (1996b) define que contexto são os elementos reconhecidos pelos dispositivos computacionais dentro do ambiente do usuário. Franklin & Flaschbart (1998) definem contexto como a situação do usuário. Rodden et al. (1998) definem contexto como os elementos de configuração das aplicações.

Dey (2000) e Dey & Abowd (1999) discutem em seus trabalhos que a definição de informação de contexto baseada em exemplos, assim como a utilização de

¹Segundo Johnson (1992), um *framework* é uma estrutura reusável de um programa ou de parte de um programa que pode ser expressa como um conjunto de classes.

sinônimos é muito difícil de se aplicar na prática. Conseqüentemente, eles sugerem que as definições de Schilit et al. (1997), Pascoe (1998), Dey et al. (1997) e Dey & Abowd (1999) são as que melhor se adaptam às informações de contexto. Schilit et al. (1994) definem três aspectos importantes na definição de contexto: onde o usuário está, com quem o usuário está e quais os recursos próximos a ele. Pascoe (1998) define contexto com o conjunto dos estados físico e conceitual de uma determinada entidade. Dey et al. (1997) definem contexto como estado físico, emocional e social do usuário.

Com base nessas definições, Dey & Abowd (1999) definem informação de contexto como qualquer informação que pode ser usada para caracterizar a situação de uma entidade, sendo que uma entidade pode ser uma pessoa, lugar ou objeto que são considerados relevantes para a interação entre o usuário e a aplicação. Segundo eles, se uma informação puder ser usada para caracterizar a situação de um participante em uma interação, então essa informação é considerada contexto.

Com o entendimento real de contexto e os diferentes modos com que ele pode ser utilizado, os desenvolvedores das aplicações podem mais facilmente determinar quais comportamentos ou características eles desejam que suas aplicações suportem e quais tipos de informação de contexto serão necessários para se alcançar tais comportamentos.

Categorias de Informação de Contexto

Com a categorização de tipos de contexto, os desenvolvedores de aplicações poderão escolher os tipos de informação de contexto que serão úteis para suas aplicações. Baseados nos estudos realizados por Ryan et al. (1997) e Schilit et al. (1994), Dey & Abowd (1999) propõem uma nova categorização baseada nos tipos de contexto que são, na prática, mais importantes. De maneira similar à proposta de Ryan et al. (1997), Dey & Abowd (1999) categorizam contexto como localização, identidade, atividade e tempo. A categoria ambiente, proposta por Ryan et al. (1997), é substituída pela categoria atividade, que corresponde ao que está ocorrendo em uma dada situação. Essas categorias correspondem aos tipos primários de informação de contexto para categorizar a situação de uma dada entidade. Segundo Dey & Abowd (1999), essa categorização atua como um indicador para outras fontes de informações de contexto. Em uma certa situação, na qual se conhece a identidade de uma pessoa, pode-se adquirir muita informação relacionada, como número de telefone, endereço, email, entre outras. Em uma outra situação, na qual se conhece a localização de uma entidade, pode-se de-

terminar quais objetos ou pessoas estão ao redor da entidade e quais atividades estão ocorrendo perto da identidade. Nessas situações, um tipo primário de informação de contexto (identidade, localização) pode ser utilizado para derivar tipos secundários de informação de contexto (endereço, número de telefone) para uma mesma entidade ou tipos primários de informação de contexto para outras entidades relacionadas (pessoas ou objetos na mesma localização).

3.2 Definição de Computação Ciente de Contexto

A computação ciente de contexto foi primeiramente definida por Schilit & Theimer (1994) como estudo de aplicações que se adaptam de acordo com a localização do usuário, grupo de pessoas, objetos próximos ao usuário e as mudanças ocorridas com esses objetos ao longo do tempo. Schilit & Theimer (1994) definem computação ciente de contexto como a habilidade de uma aplicação descobrir e reagir as mudanças que ocorrem no ambiente na qual ela está localizada.

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 (Schilit et al., 1993), foram os pioneiros no desenvolvimento de aplicações cientes de contexto baseado em localização. Esses sistemas, assim como outras tecnologias computacionais cientes de contexto, serão descritos e relacionados na Seção 3.4.

Segundo Dey & Abowd (1999), a partir de sua definição inicial, vários termos foram propostos como sinônimos para computação ciente de contexto, por exemplo, computação adaptativa (Brown, 1996a), reativa (Cooperstock et al., 1995), dirigida a respostas (Elrod et al., 1993) e dirigida ao ambiente (Fickas et al., 1997).

Assim como Schilit & Theimer (1994), outros pesquisadores definem consciência de contexto em seus trabalhos. Hull et al. (1997) definem consciência de contexto como a habilidade dos dispositivos computacionais de detectar a situação atual do usuário, interpretar o dados provenientes dos sensores e fornecer uma interface de consulta ao usuário. Ryan et al. (1997) referem-se a consciência de contexto como a habilidade de perceber e agir de acordo com informações do usuário, como dados do ambiente, localização, identificação ou aspectos temporais. Dey et al. (1997) definem que consciência de contexto modifica o paradigma de interação entre usuários e dispositivos computacionais. Segundo

Dey et al. (1997), o paradigma de interação no qual usuários procuram funcionalidades nas aplicações é modificado para o paradigma no qual as aplicações procuram pelos usuários nos momentos considerados relevantes.

Dey & Abowd (1999) propõem uma definição genérica para consciência de contexto baseada nos trabalhos de Ryan et al. (1997) e Brown (1998), que definem aplicações cientes de contexto como aplicações que dinamicamente modificam ou adaptam seus comportamentos baseadas nas informações de contexto da aplicação ou do usuário. Dey & Abowd (1999) definem um sistema como ciente de contexto caso utilize informação de contexto para fornecer informações ou serviços relevantes aos usuários, sendo que a relevância irá depender das tarefas dos usuários. Por exemplo, um guia turístico ciente de contexto deve utilizar dados de localização e de preferências do usuário na disponibilização das informações.

Entretanto, existem outras informações de contexto além de localização e de identificação de pessoas e de objetos. A maioria dos sistemas cientes de contexto ainda não incorpora muitas das informações disponíveis no ambiente, como noções de tempo, histórico e dados de outras pessoas, além dos dados do próprio usuário. Segundo Abowd & Mynatt (2000), embora ainda não exista uma definição completa de contexto, é possível descrever como obter informações de contexto com base em cinco dimensões, as quais são chamadas de "cinco Ws":

- *Who* (quem): Sistemas atuais focam suas interações na identidade de um usuário em particular desconsiderando informações relativas a outros usuários presentes no ambiente. Entretanto, os seres humanos realizam suas atividades e recordam-se de fatos passados baseados na presença de outras pessoas. Deve-se, portanto, prover informações de contexto de todas as pessoas envolvidas em uma dada atividade assistida por computador;
- *Where* (onde): A noção de localização é a mais utilizada pelos sistemas cientes de contexto. Pesquisas na área de computação ubíqua exploram a utilização dessa informação contextual associada a outras dimensões, principalmente com a dimensão temporal (*when*) e com a dimensão de identificação (*who*), no sentido de prover novas funcionalidades as aplicações. Ainda, nessa dimensão é explorada a mobilidade dos usuários, os quais poderão estar utilizando dispositivos portáteis e interagir com ambientes suportados por computador, seguindo a tendência de computação ubíqua proposta por Weiser (1991). Em ambientes móveis, o contexto do usuário, tanto sua localização como as pessoas e os objetos que estão ao seu re-

dor são dinâmicos, possibilitando numerosas situações diferentes para o usuário;

- *When* (quando): O contexto temporal tem sido utilizado para indexar um registro capturado ou para informar por quanto tempo um usuário esteve em um determinado local;
- *What* (o quê): Tem como função perceber e interpretar o quê o 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 quê um usuário está fazendo em um determinado momento pode ser uma tarefa complexa;
- *Why* (por quê): Mais complexo que perceber o quê o usuário está fazendo é entender o porquê de sua ação. Obter informações de contexto que possam caracterizar o estado de uma pessoa é talvez o mais complexo desafio da computação ciente de contexto.

Devido à complexidade para a obtenção de informações relativas à dimensão *Why*, geralmente utiliza-se a combinação das quatro primeiras dimensões na definição das informações que irão compor essa importante dimensão. Geralmente, associa-se a dimensão *What* com uma sexta dimensão (*How*) para inferir informações sobre *Why*. Para domínios de aplicação específicos, como captura e acesso, a dimensão *How* é importante e pode definir como os dados foram capturados e como eles podem ser acessados (Truong et al., 2001).

3.2.1 Categorias de Consciência de Contexto

Na literatura existem três propostas de categorização das características de aplicações cientes de contexto. A proposta mais recente, realizada por Dey & Abowd (1999), é uma combinação das taxonomias anteriores definidas por Schilit et al. (1994) e Pascoe (1998), e considera as principais diferenças existentes entre elas. Segundo Dey & Abowd (1999), Schilit et al. (1994) baseavam as tarefas executadas pelas aplicações nas informações de contexto existentes e dividia as tarefas em duas diferentes dimensões: a) para obtenção de informação ou para execução de comandos, e b) executadas manual ou automaticamente. Ainda segundo Dey & Abowd (1999), Pascoe (1998) concentrou-se na identificação das quatro características centrais para consciência de contexto:

- Percepção contextual: capacidade de detectar informações de contexto e apresentá-las para o usuário;
- Adaptação contextual: capacidade de executar ou modificar um serviço automaticamente, baseado nas informações de contexto existentes;
- Descoberta de recursos de contexto: permite que aplicações cientes de contexto localizem e explorem recursos e serviços que são relevantes para o usuário;
- Expansão de informação de contexto: permite associar dados com o contexto do usuário, ou seja, permite a inserção e posterior recuperação de informações em um dado contexto do usuário.

De maneira similar à proposta de Pascoe (1998), Dey & Abowd (1999) propõem uma categorização que objetiva generalizar as taxonomias a fim de adequá-las às aplicações cientes de contexto existentes. As três tarefas que uma aplicação ciente de contexto pode suportar são:

- apresentação de informações e serviços para um usuário;
- execução automática de um serviço para um usuário;
- identificação da informação de contexto para posterior recuperação.

Dey & Abowd (1999) discutem em seu trabalho a relação das tarefas propostas com as definições anteriores. A tarefa de apresentação é uma combinação das definições de seleção aproximada e de comando contextual, ambas propostas por Schilit et al. (1994), e da definição de apresentação de informação de contexto para o usuário proposta por Pascoe (1998). A tarefa de execução automática está associada com a definição de ações centradas em informação de contexto proposta por Schilit et al. (1994) e com a definição de adaptação contextual proposta por Pascoe (1998). A terceira tarefa, a identificação da informação de contexto, está associada com a definição de incremento da informação de contexto proposta por Pascoe (1998).

A categorização de tarefas de consciência de contexto fornece dois importantes benefícios. O primeiro é o auxílio na especificação dos tipos de aplicações para as quais as arquiteturas devem fornecer suporte. O segundo benefício é a apresentação dos tipos de características que devem ser consideradas no desenvolvimento de aplicações cientes de contexto.

3.3 Desenvolvimento de Aplicações Cientes de Contexto

Segundo Dey (2000), ao se identificar, implementar e suportar as abstrações e os serviços corretos para se lidar com informações de contexto, pode-se construir um *framework* para facilitar o processo de desenvolvimento das aplicações cientes de contexto. O conjunto de características que esse *framework* deve possuir será derivado principalmente das características comuns encontradas entre aplicações cientes de contexto. Visando simplicidade, Dey (2000) focou seu *framework* em desenvolvedores de aplicações que não irão modificar ou incrementar a arquitetura, mas apenas utilizar os componentes arquiteturais existentes. Para esses desenvolvedores, Dey (2000) lista as características necessárias para o desenvolvimento de um *framework* básico: especificação de informações de contexto; separação de aquisição e utilização de informações de contexto; interpretação das informações de contexto; comunicação distribuída e transparente; disponibilidade contínua dos componentes de captura de informações de contexto; armazenamento de informações de contexto; e descoberta de recursos. Ao considerar essas características, os desenvolvedores de aplicações poderão se beneficiar ao construir aplicações cientes de contexto.

Especificação de Informações de Contexto

Um *framework* centrado em informações de contexto deve suportar um mecanismo que permita que desenvolvedores especifiquem quais os tipos de informações de contexto que uma dada aplicação irá requisitar. Segundo Dey (2000), o mecanismo de especificação e, em particular, a linguagem de especificação utilizada devem permitir que os desenvolvedores expressem as informações de contexto de acordo com certas dimensões: fragmentos simples ou múltiplos fragmentos, informações de contexto relacionadas ou não-relacionadas, informações de contexto filtradas ou não-filtradas, informações de contexto interpretadas ou não-interpretadas.

O mecanismo de especificação deve notificar a aplicação quando a informação de contexto especificada está disponível e deve permitir que a aplicação decida se deve notificar o usuário final ou se existe a necessidade de modificar sua especificação das informações de contexto. O mecanismo deve suportar múltiplas especificações originárias de múltiplas aplicações que requisitaram a mesma informação de contexto ou diferentes conjuntos de informações de contexto.

Separação de Aquisição e de Utilização de Informações de Contexto

Uma das principais razões de não se utilizar informações de contexto com mais frequência é que não existe uma maneira padrão de se adquirir e manipular essas informações. De maneira geral, as informações de contexto são manipuladas de maneira improvisada, ou seja, os desenvolvedores de aplicação optam pelas técnicas mais fáceis de implementação, abdicando da generalização e do reuso.

Para a aquisição de informações de contexto, as aplicações geralmente utilizam gerenciamento de eventos, seja através do uso de mecanismos de consulta ou através de mecanismos de notificação. Não é necessário que ambos sejam implementados, mas por razões de flexibilidade, é interessante que as aplicações possam disponibilizar ambos os mecanismos (Rodden et al., 1998). Uma vez que as informações de contexto tenham sido adquiridas, a aplicação deve determinar, entre outras coisas, se as informações de contexto foram modificadas e se as modificações ocorridas são úteis para ela.

O importante dessa característica é separar o processo de aquisição do processo de utilização de informações de contexto. Com isso, aplicações podem utilizar as informações de contexto adquiridas sem se preocupar com os detalhes de como essas informações foram adquiridas.

Interpretação das Informações de Contexto

Para a interpretação das informações de contexto existe a necessidade de entender os mecanismos de notificação e consulta para permitir que as aplicações possam recuperar informações de contexto. Além disso, existem múltiplas camadas envolvidas no processo de aquisição de informações de contexto e, sob a perspectiva dos desenvolvedores de aplicação, essas camadas devem ser transparentes. Para suportar tal transparência, as informações de contexto devem ser, na maior parte das vezes, interpretadas antes de serem utilizadas pelas aplicações (Dey, 2000). Nesse caso, a interpretação pode ser, por exemplo, a conversão de um texto selecionado pelo usuário para um nome válido ou para o endereço de email correspondente.

O mecanismo de interpretação, combinado com o mecanismo de especificação, permite que aplicações possam requisitar as informações de contexto desejadas e adquiri-las em sua forma interpretada quando necessário.

Comunicação Distribuída e Transparente

A entrada convencional de dados dos usuários é geralmente proveniente de dispositivos como teclado e mouse. Esses dispositivos são conectados diretamente ao computador que os utiliza. Na manipulação de informações de contexto, os sensores utilizados para capturar essas informações não estão, em sua maioria, alocados no mesmo computador que está executando a aplicação responsável pela captura do contexto. Os sensores geralmente são distribuídos fisicamente e não são conectados a um único computador (Dey, 2000).

A comunicação distribuída deve ser transparente para os sensores e para as aplicações. Com isso, facilita-se o processo de desenvolvimento tanto dos sensores quanto das aplicações, liberando o desenvolvedor da obrigação de construir um *framework* de comunicação. Sem a transparência, o desenvolvedor deveria especificar e implementar um protocolo de comunicação e um esquema de codificação e decodificação para a transmissão das informações de contexto.

Disponibilidade Contínua dos Componentes de Captura de Informações de Contexto

Na grande maioria das aplicações tradicionais, os componentes das interfaces gráficas, como botões e menus, são instanciados, controlados e utilizados por uma única aplicação. Ao contrário, as aplicações cientes de contexto não devem instanciar componentes individuais, mas devem estar aptas a acessarem os componentes existentes. Ainda, múltiplas aplicações podem requisitar o mesmo fragmento das informações de contexto. Com isso, é necessário que os componentes que capturam informação de contexto devam ser executados independentemente das aplicações que os executam (Dey, 2000).

Uma vez que os componentes são executados independentemente das aplicações, é necessário que eles sejam persistentes, disponíveis todo o tempo. Partindo da premissa de que não se conhece quando aplicações irão requisitar certas informações de contexto, os componentes devem estar executando continuamente para permitir que aplicações os contatem quando necessário.

Armazenamento de Informações de Contexto

A necessidade de manter informações históricas é um requisito ligado à necessidade de disponibilidade contínua dos componentes de captura de informações de contexto. Um histórico de contexto pode ser utilizado para estabelecer tendências e prever valores de contextos futuros. Sem o armazenamento de in-

formações de contexto, esse tipo de análise não é possível de ser realizado (Dey, 2000).

Os componentes devem estar aptos a capturar informações de contexto mesmo quando nenhuma aplicação estiver interessada nas informações de contexto disponíveis em um dado momento. Essas informações de contexto capturadas podem ser utilizadas por uma aplicação que necessite do histórico de um dado contexto. Por exemplo, uma aplicação pode necessitar do histórico de localização de um usuário, a fim de prever sua localização futura. Por razões como essa, o *framework* deve suportar o armazenamento de informações de contexto.

Descoberta de Recursos

Ao se comunicar com um dispositivo de captura de informações de contexto, seja um sensor ou um filtro de informações, uma aplicação deve saber quais tipos de informações o dispositivo pode fornecer, qual sua localização e qual o modo de se comunicar com ele. No momento que uma determinada aplicação é iniciada, deve-se especificar o tipo de informação de contexto requisitado (Dey, 2000). Com isso, o mecanismo de descoberta de recursos responsabiliza-se por encontrar componentes adequados e fornecer mecanismos de acesso para eles.

O mecanismo de descoberta de recursos não é útil somente em tempo de execução. Uma aplicação pode utilizá-lo para determinar se a informação de contexto requisitada está disponível no ambiente. Ainda, o mecanismo pode ser combinado com o mecanismo de especificação e com os componentes de captura para determinar quais situações podem ser capturadas e se uma dada requisição de informação de contexto pode ser realizada pela infra-estrutura em execução. Caso a infra-estrutura não suporte a informação de contexto necessária, o desenvolvedor saberá quais os componentes que precisarão ser adicionados.

3.4 Trabalhos Relacionados

Grande parte das aplicações cientes de contexto desenvolveram seus próprios mecanismos de captura, armazenamento e processamento de informações de contexto. Entretanto, existem aplicações que utilizam serviços especializados, seja na forma de *toolkits*, infra-estruturas ou *middleware*. Para o entendimento desta seção faz-se necessária uma definição prévia dos conceitos empregados.

Os *toolkits* são desenvolvidos com base em *frameworks* e oferecem um conjunto amplo de componentes reusáveis para funcionalidades comuns. Infra-

estrutura é um conjunto de tecnologias bem definidas, confiáveis e publicamente acessíveis que atuam como base para outros sistemas (Hong and Landay, 2001). *Middleware* é uma interface entre aplicações e serviços de baixo nível oferecidos sobre uma rede distribuída.

Middleware fornece uma camada de abstração que permite que desenvolvedores de aplicações possam acessar serviços distribuídos sem a preocupação com detalhes tais como protocolos de comunicação, gerenciamento de dados, balanceamento de carga e capacidade dos computadores na rede de trabalharem simultaneamente (Yellin, 2001).

3.4.1 Aplicações Cientes de Contexto

Nesta seção são detalhadas duas aplicações cientes de contexto que implementam seus serviços de captura, armazenamento e gerenciamento de informações de contexto. A aplicação Active Badge (Want et al., 1992) foi uma das primeiras aplicações a utilizarem informações de contexto para expandir suas funcionalidades, e a aplicação Campus Aware (Burrell et al., 2002) é uma aplicação baseada em dados espaciais para auxílio a turistas.

A Active Badge foi uma solução proposta para minimizar o problema de determinar a localização de um indivíduo no ambiente. Para isso, desenvolveu-se um sistema de monitoramento que combina detecção de posicionamento com comunicação. Esse sistema foi baseado em sensores ópticos que emitiam sinais a cada fração de segundo. A aplicação inicial utilizada para esse sistema foi de auxílio as telefonistas, e com o utilização de sensores alocados nos escritórios e nas áreas comuns de um edifício, o sistema fornecia uma listagem com o nome dos usuários e os números dos telefones que estavam localizados nos lugares mais próximos a cada um desses usuários. Uma vez que um usuário se movimentasse pelo edifício, o sistema atualizava a listagem para garantir que o sistema sempre registrasse o telefone mais próximo a esse usuário. O Active Badge foi um sistema extremamente preciso, mas que não evoluiu em razão de ser muito oneroso e com objetivos de longo prazo.

A aplicação Campus Aware, também baseada em dados espaciais, foi desenvolvida para atuar como um guia turístico no espaço físico da Universidade de Cornell, nos Estados Unidos. Essa aplicação permite que o estudantes da universidade expressem seus conhecimentos e façam anotações sobre a universidade e as atividades que ocorrem nesse ambiente. O objetivo da aplicação é auxiliar os futuros estudantes, os quais podem anotar questões e ler comentários

realizados previamente.

O sistema é baseado em dispositivos móveis e o armazenamento é realizado em um repositório central com três bases de dados responsáveis pelas informações de contexto tais como anotações dos usuários e características físicas da universidade.

3.4.2 Toolkit e Infra-estrutura Cientes de Contexto

Nesta seção são descritos um *toolkit* e uma infra-estrutura que objetivam auxiliar o desenvolvimento de aplicações cientes de contexto. O Context Toolkit (Salber et al., 1999) é uma implementação do *framework* conceitual descrito na Seção 3.3 e a Context Fabric (Hong and Landay, 2001) é uma infra-estrutura com características de armazenamento semelhantes as do Context Kernel. O Context Toolkit concentra-se preferencialmente na aquisição das informações de contexto, enquanto o Context Fabric concentra-se na modelagem das informações de contexto.

O Context Toolkit contém uma combinação de características e abstrações para auxiliar no desenvolvimento de aplicações cientes de contexto. O Context Toolkit pode fornecer padronização e generalização as aplicações, possibilitando o reuso de métodos de desenvolvimento na construção de novas aplicações. No Context Toolkit, os componentes compartilham um mecanismo de comunicação comum (XML sobre HTTP) que permite a transparência no processo de aquisição de contexto. Com isso, as aplicações não necessitam saber se os componentes responsáveis pelo contexto estão sendo executados local ou remotamente.

O Context Toolkit consiste de três abstrações principais: componentes de interação (*widgets*), agregadores e interpretadores. Assim como os *widgets* das interfaces gráficas convencionais fazem a interação entre o usuário e uma aplicação, os *widgets* de contexto fazem a interação entre o usuário e o ambiente. Eles encapsulam informações de contexto, como localização ou identificação, e fornecem uma interface uniforme e transparente para os componentes ou aplicações que utilizam informação de contexto. Os interpretadores de contexto são utilizados para abstrair informações de contexto de baixo nível em informações de alto nível. Por exemplo, dados de identificação, localização e tempo podem ser utilizados para abstrair que uma dada aula esteja ocorrendo.

Os agregadores de contexto são classificados como *meta-widgets*, uma vez que eles possuem todas as capacidades dos *widgets*, além da capacidade de agregar informações de contexto das entidades do mundo real como, por exem-

plo, de usuários ou de lugares. Os agregadores atuam como uma ligação entre as aplicações e os *widgets* básicos, fornecendo ainda mais transparência para o processo.

A infra-estrutura Context Fabric foi desenvolvida com foco em uma modelagem de armazenamento de informações de contexto dinâmicas e distribuídas. Essa infra-estrutura permite separação de aquisição, processamento e utilização dessas informações, liberando as aplicações desse gerenciamento. Assim como o Context Toolkit, essa infra-estrutura é responsável por manter a integridade dos dados armazenados e facilitar o desenvolvimento das aplicações e dos dispositivos de captura.

A Context Fabric consiste de um modelo lógico e de um modelo físico para armazenamento de informações. O modelo lógico é responsável por armazenar entidades, como pessoas e lugares. O modelo físico é responsável por gerenciar as informações de contexto armazenadas.

3.4.3 *Middleware Clientes de Contexto*

Nesta seção são descritos dois *middleware* que focam no armazenamento de informações de contexto. O EventHeap (Johanson and Fox, 2002) foi desenvolvido para um ambiente específico, no qual aplicações, por exemplo, podem armazenar suas informações de contexto e serem avisadas quando um evento externo acontecer. O Gaia (Hess et al., 2002) possui um serviço de armazenamento de informações de contexto baseado em arquivos e componentes específicos para consulta e inferência dessas informações armazenadas.

O *middleware* EventHeap foi desenvolvido para ser utilizado em espaços de trabalhos interativos. Esses espaços são constituídos de dispositivos computacionais conectados no qual os usuários se reúnem para realizar atividades colaborativas como revisões, reuniões, conferências ou palestras. O EventHeap permite que agentes (aplicações, dispositivos, serviços) possam se registrar e enviar dados ao servidor na forma de tuplas constituídas como seqüências de campos flexíveis e bem definidos. Ainda, esse *middleware* permite que agentes no lado do cliente possam consultar esse servidor e serem avisados quando outros agentes registrados submeterem novos dados.

O *middleware* Gaia foi desenvolvido para fornecer suporte a ambientes constituídos de usuários, dispositivos, tarefas e informações de contexto associadas a cada um dos componentes anteriores. Esses ambientes são considerados problemáticos, uma vez que necessitam de reconfiguração a cada mudança de um de

seus componentes. O Gaia é responsável por fornecer armazenamento, gerenciamento e distribuição das informações de contexto, sendo que o armazenamento é realizado por um serviço de gerenciamento de dados que organiza os dados em servidores de arquivos distribuídos.

O serviço de contexto do Gaia permite que aplicações possam se registrar e realizar consultas aos dados previamente armazenados. O Gaia possui também um serviço que captura informações de contexto no ambiente através de sensores. Ainda, o serviço de contexto permite a inferência de informações de contexto com base nas informações capturadas.

3.5 Considerações Finais

A computação ciente de contexto consiste na integração das atividades humanas e dos serviços computacionais no sentido de expandir e facilitar tais atividades. Humanos naturalmente fornecem contexto na sua forma própria de comunicação, seja por sinais de frustração, confusão, alegria ou outros quaisquer. Computadores não possuem a capacidade de sentir, perceber e utilizar esses sinais. Por isso, se fazem necessárias as informações de contexto. A computação ciente de contexto consiste na exploração do relacionamento entre atividades humanas e serviços computacionais no sentido de expandir e facilitar tais atividades.

O uso de informações de contexto é importante para o desenvolvimento de sistemas computacionais. Além de permitir o desenvolvimento de novos tipos de sistemas com base na natureza dessa informações, permite também que os sistemas existentes possam expandir suas funcionalidades com a utilização de informações de contexto. Aplicações interativas, por exemplo, podem ser expandidas com o uso de informações de contexto. Nesse contexto, uma aplicação pode ser modelada de acordo com uma dada situação do usuário, aumentando os benefícios para o mesmo.

Uma outra forma de explorar informação contextual está nas técnicas de reconhecimento de objetos. Usando tais técnicas é possível reconhecer áreas em vídeo e imagens de forma a entender o contexto de uma cena ou de uma fotografia. Isso possibilita enriquecer apresentações com informações de contexto e realizar busca baseada em conteúdo sob 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).

Informações de contexto como localização espacial e aspectos temporais são facilmente detectados, enquanto outras informações, como atividade e motivação do usuário, são extremamente difíceis para os computadores inferirem. Os seres humanos, por sua vez, possuem capacidade para detectar, entender e inferir sobre atividade e motivação de outros seres humanos. O projeto descrito nesta dissertação faz uso dessa capacidade humana de interpretar informações de contexto.

Web Services

*A*s vantagens da Web incluem a disseminação de informações e o desenvolvimento de serviços. A essência de *Web Services* é que esses serviços possam ser utilizados por aplicações monolíticas com o objetivo de torná-las mais completas. Por exemplo, uma aplicação financeira pode expandir sua funcionalidade ao utilizar um serviço de autenticação digital.

Web Services são padrões e protocolos que definem e formalizam a comunicação entre sistemas de software em uma rede computacional, seja utilizando uma rede interna ou protocolo Internet. Os *Web Services* não foram definidos para serem utilizados exclusivamente na Internet, apesar de muitos leigos considerarem os conceitos de Web e Internet como sinônimos. Internet é uma infraestrutura de rede que conecta computadores globalmente através de uma diversidade de protocolos. Web é um dos meios de acessar informações, sendo que a Internet pode disseminar suas informações através de outros meios como, por exemplo, correio eletrônico, grupos de mensagens, mensagens instantâneas ou transferência de arquivos.

Apesar de existirem tecnologias que permitem a invocação remota de serviços distribuídos, os *Web Services* apresentam algumas vantagens sobre as tecnologias relacionadas (Stal, 2002). Segundo Stal (2002), tecnologias como CORBA (Object Management Group, 2001) e Java RMI (Microsystems, 2002b), por exemplo, são complexas e necessitam de alto poder de comunicação ao custo de baixa escalabilidade e baixa capacidade de operarem em conjunto. *Web Ser-*

vices fornecem, entre outras vantagens, interoperabilidade, flexibilidade e simplicidade a baixo custo de processamento, ao utilizar mecanismos como o protocolo HTTP (*Hypertext Transfer Protocol*) (Fielding et al., 1999) para transporte e a linguagem XML para organização e representação de suas informações (Kleijnen and Raju, 2003).

Web Services fornecem uma solução simples para resolver o problema de heterogeneidade causada por diferenças entre tecnologias de rede, dispositivos, sistemas operacionais, linguagens de programação, mecanismos de comunicação e estruturação dos dados. Os *Web Services* permitem que sistemas de software possam interoperar por meio de padrões Web (Martin et al., 2003). Este capítulo tem como objetivo definir o conceito de *Web Services* e apresentar as principais tecnologias relacionadas.

A definição de *Web Services* é feita na Seção 4.1. Na Seção 4.2 é apresentada a arquitetura básica de *Web Services*, utilizada pelo Context Kernel. Na Seção 4.3 são apresentadas as linguagens XML e a especificação XML Schema, que definem o formato de representação de dados nas mensagens trocadas entre os sistemas de software. Na Seção 4.4 é apresentado o protocolo de comunicação utilizado pelos *Web Services* e na Seção 4.5 é apresentada sua linguagem de descrição. Na Seção 4.6 discute-se a essência e a evolução dos *Web Services*.

4.1 Definição

Segundo a definição do Consórcio World Wide Web (W3C), *Web Services* são sistemas de software identificados por uma URI e definidos por um documento de descrição baseado na linguagem XML. Essa descrição deve permitir que outros sistemas possam identificar o sistema descrito e interagir com o mesmo segundo sua definição, utilizando mensagens baseadas em XML conduzidas por protocolos Internet (W3C, 2002a). O trabalho apresentado nesta dissertação é baseado nas especificações propostas pelo W3C para a plataforma *Web Services*: a linguagem de representação XML, a linguagem de descrição WSDL e o protocolo de empacotamento SOAP.

Os *Web Services* apresentam uma série de características próprias, como:

- disponibilidade: os serviços fornecidos devem estar disponíveis aos sistemas de software que desejem utilizá-los;
- descrição: devem possuir uma descrição possível de ser entendida pelos computadores e utilizada para identificar a localização dos serviços e o

modo como acessá-los;

- interoperabilidade: definem um padrão comum que permite que diferentes sistemas atuem em conjunto. Por exemplo, uma aplicação implementada em C++ pode acessar serviços escritos em Java;
- baseado em padrões: baseia-se em padrões já estabelecidos, como as linguagens XML e WSDL e os protocolos SOAP e HTTP;
- propósito geral: podem prover infra-estrutura para múltiplos propósitos, podendo ser especializada para cada tipo de sistema de software.

Além dessas características, os *Web Services* devem publicar a descrição dos serviços em um repositório no qual sistemas de software possam identificá-los. No Context Kernel, principal contribuição deste trabalho, a descrição do serviço não está publicada em um repositório público por ainda estar em fase de testes.

4.2 Arquitetura

A arquitetura básica de *Web Services* é composta de mecanismos capazes de enviar e receber mensagens, descrevê-los e publicar essas descrições. Essa arquitetura define a interação entre agentes de software como troca de mensagens entre aqueles que requisitam serviços e aqueles que fornecem serviços. Os fornecedores de serviços são responsáveis por publicar a descrição de um ou mais serviços. Os agentes que requisitam os serviços devem ser aptos a encontrarem e identificarem os serviços.

Essa estrutura básica é definida utilizando-se a linguagem XML para classificação e estruturação das mensagens, e o protocolo HTTP para transporte. A arquitetura pode ser expandida com extensões da especificação XML, como a linguagem de definição XML Schema (W3C, 2002b), e com extensões do protocolo de transporte HTTP, como os protocolos TCP (Postel, 1981), UDP (Postel, 1980), BEEP (Rose, 2001), JMS (Microsystems, 2002a) ou sistemas proprietários. Em um cenário típico, um fornecedor de serviços define a descrição de seus serviços e a publica para aqueles que irão requisitá-los. Esses agentes poderão recuperar essa descrição e utilizá-la para identificar os aspectos de interface do *Web Service*. Após a descrição ser analisada, os serviços podem ser requisitados.

A descrição fornece o formato das mensagens de envio e resposta, protocolos utilizados e a localização de cada serviço disponibilizado. Os detalhes de implementação são transparentes na descrição, permitindo que o agente requisitante

possa utilizar o serviço independente da plataforma de hardware e da linguagem de programação nas quais o serviço foi implementado.

A arquitetura definida prevê que uma mensagem XML seja enviada pelo agente requisitante e recebida pelo *Web Service*. Essa mensagem XML é normalmente gerada pelo próprio agente e deve incluir elementos opcionais de cabeçalho e um corpo com os dados a serem trocados. Os elementos de cabeçalho contêm informações como, por exemplo, aspectos de segurança, roteamento ou existência de pontos intermediários no transporte da mensagem. O corpo da mensagem, definido pelo agente requisitante, inclui o conteúdo das informações baseado na estrutura definida na descrição do *Web Service*. As mensagens são associadas com endereços únicos que identificam onde o agente deverá enviar a sua mensagem de requisição.

Figura 4.1: Arquitetura Básica de *Web Services* (Adaptada de (W3C, 2002b))

A Figura 4.1 define uma arquitetura básica de *Web Services* na qual uma aplicação qualquer deseja requisitar serviços de um *Web Service*. Nessa arquitetura utiliza-se o protocolo de empacotamento SOAP e, por exemplo, o protocolo HTTP para o transporte das mensagens. A mensagem SOAP da aplicação é inserida no corpo da mensagem de requisição, a qual é processada pelo *Web Service*. Após o processamento, uma mensagem SOAP de resposta é gerada e inserida no corpo da mensagem de resposta para ser enviada de volta à aplicação requisitante.

4.3 Linguagem XML e Definição XML Schema

Web Services são baseados na linguagem XML para que diferentes serviços possam trocar informações em um formato padrão. Como um dos objetivos é

integrar sistemas heterogêneos, é necessária a existência de um mapeamento. Com isso, os agentes requisitantes e os fornecedores de serviços podem entender os dados que são trocados. A linguagem de definição XML Schema é utilizada para definir a estrutura e a formatação das mensagens XML que são trocadas entre sistemas.

4.3.1 XML (*Extensible Markup Language*)

A linguagem XML (*Extensible Markup Language*) (W3C, 2002) é apropriada para a representação de dados, documentos e demais entidades (relatório, protocolo, script, folha de estilo, entre outros) cuja essência fundamenta-se na capacidade de agregar informações (W3C, 2002). Por conseguinte, aplicações podem associar semântica a dados e campos de documentos e viabilizar o processamento automático de documentos manipulados. A especificação XML 1.0 é recomendação W3C desde fevereiro de 1998 e a especificação XML 1.1 é recomendação W3C desde outubro de 2002.

Uma das grandes vantagens trazidas por XML é a separação explícita das dimensões de um hiperdocumento: conteúdo, estrutura, semântica, metadados e malha de hipertexto. Um documento XML apresenta, essencialmente, duas dimensões de informação: conteúdo e estrutura. Devido à capacidade de XML fornecer recursos para definição de classes de elementos, atributos e regras bem definidas, outras dimensões (apresentação, semântica, metadados e modelo de ligação de hipertexto, por exemplo) são fornecidas por padrões derivados de XML. Cada uma dessas dimensões é discutida por especificações complementares baseadas em XML, como a linguagem para transformação de documentos XML - XSL Transformations (XSLT) (W3C, 2002) - e a linguagem para referenciar partes específicas de documentos XML - XPath (W3C, 1999b).

4.3.2 XML Schema

Para a especificação de documentos XML, é importante determinar a sequência e o encadeamento de seus elementos e definir seus atributos, valores e tipos de dados. Informações como definição de sequência e encadeamento formam gramáticas para documentos XML que são direta e indiretamente associadas a esses documentos.

As aplicações que utilizam XML precisam, em geral, de um mecanismo de validação das informações que estão sendo trocadas. Validação é um processo no qual é verificado se a estrutura e o conteúdo do documento respeitam a

gramática que define a estrutura desse documento. Documentos em conformidade com uma determinada gramática são ditos válidos. Cabe a um parser XML verificar a validade de documentos. Existem dois tipos de gramáticas para XML: DTD (*Document Type Definition*) e XML Schema (W3C, 2001).

XML Schema é a especificação do W3C que permite especificar e delimitar documentos XML, sendo recomendação desde maio de 2001. Uma especificação XML Schema define os tipos dos elementos e dos atributos em um documento XML e a estrutura em que esses elementos devem aparecer. Ainda, oferece vantagens como possibilitar que sejam derivados novos tipos de dados a partir dos tipos existentes, bem como é possível utilizar herança e agrupamento na definição dos elementos.

A especificação XML Schema foi desenvolvida com o intuito de superar as limitações de gramáticas especificadas por DTDs. XML Schema implementa validação de tipos de dados que permite verificar, por exemplo, se unidades específicas de informações possuem o tipo de dado correto e se os dados informados se encontram no intervalo de valores definido. Esse tipo de validação não é possível quando a especificação é realizada com DTD. Ainda, XML Schema apresenta duas importantes vantagens em relação à gramática DTD:

- DTDs apresentam comandos e construtores que são difíceis de serem lidos, entendidos e mantidos pelos usuários finais;
- DTDs são especificados para estruturar dados XML, mas não são escritos em XML. Como XML Schema utiliza regras sintáticas definidas em XML, os mesmos processadores e validadores podem ser utilizados.

4.4 Protocolo de Comunicação SOAP

A implementação do protocolo de comunicação é a etapa mais importante no desenvolvimento de um *Web Service*. Esse protocolo de comunicação define a sintaxe, a semântica e a ordem das mensagens que serão trocadas entre aplicações e servidores. O Context Kernel utiliza o protocolo HTTP para transporte e o protocolo SOAP para a troca de mensagens, como detalhado na Seção 5.2.

SOAP (Simple Object Access Protocol) é um protocolo de comunicação de dados extensível para troca de informações em ambientes descentralizados e distribuídos (Gudgin et al., 2001). SOAP define um conjunto de regras para serialização e convenção de chamadas entre sistemas de software. Ainda, SOAP utiliza

um protocolo de transporte e define XML como a linguagem de estruturação para suas mensagens.

O protocolo SOAP define um paradigma de troca de mensagens com o qual aplicações podem criar interações complexas como combinação de requisição e resposta simples ou requisições e respostas múltiplas. SOAP apresenta duas características principais:

- interoperabilidade: ao não restringir a forma de implementação dos serviços;
- simplicidade: ao estabelecer a linguagem XML para formatar suas mensagens.

Em relação a sua estrutura, o protocolo SOAP contém o elemento *envelope* como elemento principal e dois elementos internos: *header* e *body*. O conteúdo desses elementos internos é definido pelas aplicações e não fazem parte da especificação do protocolo.

O elemento *header* é opcional e define informações como rota de transmissão e meta-informações relacionadas ao processamento da mensagem. Esse elemento permite que a mensagem seja estendida de forma específica por cada aplicação. O elemento *body* é obrigatório e define a informação que será transmitida na mensagem.

4.5 Linguagem de Descrição WSDL

Uma linguagem de descrição é utilizada para expressar informações estruturais e semânticas, e com as especificações contidas nesta linguagem, sistemas de software poderão automaticamente gerar implementações para que aplicações possam se comunicar com os *Web Services*. Nesse contexto, a linguagem WSDL, também baseada em XML, é o formato de representação utilizado.

A linguagem de descrição WSDL (*Web Services Description Language*) permite a descrição completa dos serviços oferecidos por um *Web Service* (W3C, 2003). Uma aplicação que deseje utilizar um *Web Service* depara-se com questões como: quais os serviços oferecidos pelo *Web Services*? Qual a forma de requisitá-los e quais informações são necessárias para essa chamada? Qual o formato das informações que serão enviadas como resposta? A linguagem WSDL fornece mecanismos para responder todas essas questões.

A linguagem WSDL permite que um *Web Services* publique a interface de seus serviços, ou seja, o formato das mensagens de requisição e das mensagens de

resposta. Ainda, WSDL é naturalmente extensível, uma vez que a especificação define as interfaces abstratas e as estruturas de relacionamento, permitindo que a implementação e a formatação das mensagens sejam realizadas através de extensões. O Context Kernel utiliza as extensões do protocolo SOAP e a linguagem de definição XML Schema, como detalhado no Apêndice A.

Para a descrição de um *Web Service*, a linguagem WSDL define uma estrutura baseada em XML e composta de elementos e atributos. O elemento *definitions* é o elemento principal da estrutura e pode conter os elementos *type*, *message*, *portType*, *binding* e *service*.

O elemento *definition* possui uma série de *namespaces*¹ como atributos. O atributo *targetNamespace* é obrigatório e contém a definição dos elementos e dos atributos principais da descrição. O elemento *definition* pode compreender *namespaces* auxiliares. Como detalhado no Apêndice A, o Context Kernel utiliza os *namespaces* WSDL, SOAP, XSD (Definição do XML Schema), HTTP e *namespaces* dos documentos XML Schema de cada uma de suas operações, uma vez que a linguagem WSDL fornece mecanismos para que os documentos de especificação dos tipos de dados possam ser alocados como recursos separados. Isso pode ser um recurso interessante com mensagens complexas.

Como visualizado na Figura 4.2, a descrição completa de um serviço é composta da definição da interface e da definição de implementação. Com esses dois componentes de definição é possível descrever a forma de invocar e interagir com um *Web Service*.

Figura 4.2: Elementos da Linguagem WSDL (Adaptada de (W3C, 2002b))

¹Segundo W3C (W3C, 1999a), um *namespace* é um documento unicamente identificado na Web que é utilizado para estabelecer a estrutura lógica de outro documento.

Uma definição de interface é uma definição abstrata dos serviços que pode ser reutilizada por múltiplas definições de implementação. A linguagem WSDL define que a definição de interface compreende os elementos *type*, *message*, *portType* e *binding*.

O elemento *type* é opcional e utilizado para a formatação das mensagens. Esse elemento define os tipos dos dados, uma vez que WSDL não implementa seus próprios tipos. Como detalhado no Apêndice A, o Context Kernel utiliza a definição XML Schema para a formatação de suas mensagens. Os elementos *message* definem a modelagem das mensagens que serão trocadas entre os sistemas de software, definindo as informações de entrada e saída para uma determinada operação.

Os elementos *portType* agrupam mensagens em uma determinada operação abstrata. Cada elemento representa a abstração de um agrupamento lógico de métodos em uma classe. Cada *portType* pode apresentar um conjunto de elementos *operation*, que especificam se as mensagens definidas previamente são mensagens de chamada ou mensagens de respostas. Cada elemento *operation* deve possuir pelo menos um elemento *input* ou um elemento *output* ou ambos, que definem a entrada e a saída de cada operação. O elemento *input*, caso exista, representa a mensagem de entrada e o elemento *output*, caso exista, representa a mensagem de saída. É possível combinar esses elementos de quatro formas:

- *input-output*: são necessárias mensagens de entrada e de saída, nessa ordem. Quando a mensagem de entrada é recebida, a mensagem de saída é gerada;
- *input-only*: apenas a mensagem de entrada é necessária e não é produzida nenhuma mensagem de saída;
- *output-only*: apenas a mensagem de saída é necessária. O serviço envia a mensagem, mas não espera nada como resposta;
- *output-input*: são necessárias as mensagens de saída e entrada, nessa ordem. O serviço envia a mensagem de saída e espera a mensagem de resposta.

Os elementos *binding* fazem o mapeamento entre um elemento *portType* e um protocolo de comunicação. Descrevem as mensagens de requisição e de resposta e indicam um único protocolo de transporte a ser utilizado. Cada elemento *binding* fornece a ligação entre o modelo lógico e o modelo físico da descrição ao relacionar as operações definidas pelo elemento abstrato *portType* com a descrição

física de como elas são transmitidas ao serviço. Para isso, faz-se necessária a definição do protocolo de transporte e do protocolo de comunicação. Como detalhado no Apêndice A, o Context Kernel utiliza os protocolos HTTP e SOAP.

O protocolo SOAP utiliza-se de extensões de sua definição para descrever como as mensagens serão enviadas. A extensão do protocolo SOAP utilizada no elemento *binding* possui dois atributos: *style* e *transport*. O primeiro é opcional e descreve a natureza das operações dentro do elemento *binding*. O segundo atributo define o protocolo de transporte utilizado pelo serviço nessa operação.

A Figura 4.2 explicita também a definição de implementação, que descreve como uma determinada definição de interface relaciona-se com um determinado serviço. Ainda, a definição de implementação descreve a localização do serviço a ser requisitado. A linguagem WSDL define que essa definição deve ser baseada nos elementos *service* e *port*.

Os elementos *service* agrupam um ou mais elementos de definição de endereço que determinam o endereço específico de cada uma das operações do *Web Service*. Cada elemento *service* possui pelo menos um elemento *port*, que é o responsável por definir as particularidades de um endereço no qual um dado serviço é disponibilizado.

4.6 Considerações Finais

A Web tem sido a plataforma mais rápida e amplamente adotada para o desenvolvimento de aplicações distribuídas. O motivo principal é a simplicidade de suas interações, ou seja, a facilidade de adicionar novas aplicações e servidores e a baixa complexidade ao estabelecer conexões entre aplicações e serviços. Além disso, tudo é realizado de maneira descentralizada e com considerável ganho em interoperabilidade, escalabilidade e gerenciamento. Os *Web Services* foram desenvolvidos para aproveitarem esses ganhos.

A essência de *Web Services* é fornecer uma plataforma para o desenvolvimento de sistemas de software distribuídos que possam se comunicar independente das possíveis diferenças existentes entre eles, como sistema operacional ou linguagem de programação.

A diferença entre *Web Services* e aplicações tradicionais da Web residem em dois pontos principais: os *Web Services* são baseados em padrões para a comunicação e descrição, e não restringem o transporte de suas mensagens a um único protocolo de transporte.

A comunicação das mensagens é realizada utilizando o protocolo SOAP, que

define um modelo extensível para a passagem de mensagens XML de um sistema de software para outro. O protocolo SOAP define como essas mensagens serão enviadas ao *Web Service* utilizando um determinado protocolo de transporte. Atualmente, o protocolo HTTP é amplamente utilizado pelos *Web Services* para transporte de suas mensagens. Para a descrição dos *Web Services* é utilizada a linguagem WSDL, que fornece a definição da interface e a identificação dos serviços fornecidos.

Um dos pontos fracos de *Web Services* é que a plataforma de desenvolvimento é baseada em especificações ainda em desenvolvimento, como SOAP e WSDL. Em razão disso, existe a necessidade da implementação de mecanismos que fortaleçam alguns pontos que foram pouco explorados por seus desenvolvedores, como segurança, confiabilidade na troca de mensagens e especificação de suas informações. Entretanto, seus benefícios estão tornando *Web Services* amplamente aceito pelas principais companhias de desenvolvimento de software, que estão promovendo a iniciativa de *Web Services* em seus novos produtos.

O Context Kernel

Sistemas computacionais são eficientes para reunir e gerenciar dados, enquanto seres humanos são eficientes para reconhecer situações e escolher ações apropriadas. Segundo Erickson (2002), um sistema ciente de contexto eficiente deve considerar os seres humanos como parte do sistema, liberando os computadores das ações de entendimento e interpretação, conforme sugerido por Weiser (1991; 1993). Por exemplo, uma aplicação, independente do usuário, pode detectar que um novo usuário está acessando o sistema e personalizar sua aparência. Essa adaptação é baseada em informações e regras de contexto (Banavar and Bernstein, 2002).

Nesse contexto foi desenvolvido o Context Kernel, um *Web Service* dedicado ao armazenamento e recuperação de informações de contexto, que permite às aplicações gerenciarem suas informações de contexto com base nas seis dimensões definidas na literatura: *who* (identificação), *where* (localização espacial), *when* (localização temporal), *how* (modo de captura ou de acesso), *what* (atividade) e *why* (motivação) (Abowd and Mynatt, 2000; Truong et al., 2001). A partir dessas definições, foi definida uma modelagem baseada em dimensões primitivas e dimensões derivadas (Arruda Jr et al., 2003).

As dimensões primitivas podem ser armazenadas e recuperadas independentemente de outras dimensões. Nessa categoria encontram-se as dimensões de identificação (*who*), localização espacial (*where*), localização temporal (*when*) e meios de captura ou de acesso as informações (*how*). Por exemplo, a dimensão

de localização espacial pode ser utilizada para localizar um indivíduo em um determinado ambiente e pode ser extraída diretamente das informações desse ambiente, como latitude ou longitude.

As dimensões derivadas são resultantes do relacionamento de outras dimensões. Nessa categoria encontram-se as dimensões de atividade (*what*) e motivação (*why*). Por exemplo, uma aplicação pode determinar a motivação de um usuário com base nas informações de identificação (*who*: dimensão primitiva) e atividade desse usuário (*what*: dimensão derivada).

Este capítulo apresenta as decisões de projeto realizadas durante a definição da infra-estrutura do Context Kernel, assim como os principais aspectos envolvidos em sua implementação. Na Seção 5.1 é discutida a modelagem conceitual do Context Kernel, com as principais características de seu modelo de armazenamento. Na Seção 5.2 é apresentada a arquitetura do Context Kernel e os passos necessários para a utilização de seus serviços. Na Seção 5.3 são detalhados os serviços disponibilizados pelo Context Kernel e na Seção 5.4 são apresentadas duas aplicações no contexto do Projeto InCA-SERVE que utilizam os serviços disponibilizados pelo Context Kernel nos dois cenários apresentados na Seção 5.5. Na Seção 5.6 são discutidas algumas das limitações do Context Kernel e na Seção 5.7 são discutidas vantagens de utilizar o Context Kernel para o suporte de aplicações cientes de contexto.

5.1 Modelo de Armazenamento do Context Kernel

O Context Kernel é dedicado ao gerenciamento de informações de contexto provindas das aplicações que o utilizam. Como o armazenamento e a recuperação dessas informações são as operações principais do Context Kernel, o passo inicial para o seu desenvolvimento foi a formalização dessas informações. Baseado nas definições de Abowd & Mynatt (2000) e Truong et al. (2001), foi definida uma modelagem baseada em dimensões primitivas e dimensões derivadas (Arruda Jr et al., 2003).

O Context Kernel formaliza as dimensões primitivas como triplas no formato tipo-valor-qualificador. Por exemplo, a tripla (tipo = hora; valor = 14:33:22; qualificador = time) corresponde a uma instância da dimensão *when*. O tipo identifica a categoria do valor, que é determinada pela aplicação. O qualificador é um atributo opcional utilizado para restringir o valor. A versão atual do Context Kernel disponibiliza qualificadores para a dimensão *when*, como *time*, *datetime* e *date*, os quais podem ser estendidos nas próximas versões do Context Kernel. A uti-

lização de qualificadores tem como objetivo expandir o mecanismo de consulta do Context Kernel.

As dimensões derivadas são determinadas por meio de regras que contêm pelo menos uma premissa e uma inferência. Premissas são quádruplas no formato dimensão-tipo-valor-qualificador e inferências são pares dimensão-valor. As premissas e as inferências podem conter informações de contexto de dimensões primitivas ou derivadas. As regras são inferidas pelas aplicações e armazenadas no Context Kernel para consultas. Por exemplo, uma regra pode ser baseada nas premissas (dimensão=where,tipo=latitude, valor=23) e (dimensão=where,tipo=longitude,valor=46) e determinar a inferência (dimensão=what, valor=cidade de São Paulo). O conjunto de regras é denominado esquema.

Definidas as dimensões e a formalização das informações de contexto, a etapa seguinte ao projeto foi a definição da modelagem conceitual da base de dados de suporte ao armazenamento e à recuperação dessas informações. A modelagem conceitual do Context Kernel, representada por um Modelo UML (*Unified Modeling Language*), é apresentada na Figura 5.1, o qual indica que:

- a classe *Aplicação* é uma abstração para toda aplicação que se registra no servidor. Os atributos dessa entidade compreendem os dados enviados pela aplicação e as informações extraídas pelo servidor quando da operação de registro. Os dados enviados pela aplicação são: *nome*, *descrição*, *endereço na Web (URL)*, *última versão*, *data de última versão* e *identificador de domínio (acadêmico ou comercial)*. As informações extraídas pelo servidor são: *data de registro*, *identificador do IP (Internet Protocol)* e *domínio* da máquina que requisitou o registro;
- a classe *Desenvolvedor* é uma abstração dos desenvolvedores das aplicações que se registram no Context Kernel. Caso a aplicação envie informações de *nome* e *endereço eletrônico* de seus desenvolvedores, essas informações serão armazenadas na base;
- a classe *Informações de Contexto* representa o conceito atômico das informações de contexto e pode ser generalizada em informações primitivas, derivadas e em regras de contexto;
- a classe *Primitivo* é uma abstração das informações de contexto de dimensão primitiva (*who*, *where*, *when* e *how*) e possuem como atributos: *identificador do IP* e *domínio* da máquina responsável pelo envio da mensagem, *dimensão*, *tipo*, *valor* e *qualificador*;

- a classe *Derivado* é uma abstração das informações de contexto de dimensão derivada (*what* e *why*) e possuem como atributos: *identificador do IP* e *domínio* da máquina responsável pelo envio da mensagem, *dimensão* e *valor*;
- a classe *Regra* refere-se às regras que podem ser enviadas pelas aplicações e possui como atributos: *identificador do IP*, *domínio* da máquina responsável pelo envio da mensagem e as datas de início de fim da regra.

Figura 5.1: Modelo de Classes do Context Kernel

De acordo com a modelagem apresentada na Figura 5.1, define-se os seguintes relacionamentos entre classes:

- uma aplicação pode possuir N desenvolvedores e um desenvolvedor pode ser responsável por N aplicações;
- uma informação derivada é gerada por meio de N informações primitivas e que uma mesma informação primitiva pode fazer parte de N informações derivadas;

- uma aplicação pode conter N informações primitivas e uma mesma informação primitiva pode fazer parte de N aplicações;
- uma aplicação pode conter N regras de contexto e que uma mesma regra pode fazer parte de N aplicações.

5.2 Arquitetura do Context Kernel

A arquitetura básica de um *Web Service* compreende mecanismos capazes de enviar e receber mensagens na comunicação entre os sistemas de software. Para a troca de mensagens, os *Web Services* utilizam o protocolo SOAP que define a semântica e a ordem das mensagens trocadas entre os pares. No Context Kernel, o protocolo HTTP é utilizado para o transporte das mensagens. Ainda, os *Web Services* possuem na linguagem WSDL o seu mecanismo de descrição, que fornece a definição da interface e a identificação dos serviços fornecidos.

Figura 5.2: Arquitetura do Context Kernel

A Figura 5.2 ilustra a arquitetura do Context Kernel e os passos necessários para que uma aplicação possa utilizar seus serviços. No passo 1, o Context Kernel publica a descrição de seus serviços em um documento público WSDL. No passo 2, as aplicações requisitantes recuperam as informações contidas no documento WSDL. Com a análise desse documento, a aplicação obtém a descrição dos serviços disponibilizados pelo Context Kernel, suas localizações na Web e o formato como invocá-los.

Baseado nessas informações, a aplicação está apta a enviar requisições HTTP utilizando o protocolo SOAP para o empacotamento das mensagens. O Context Kernel foi implementado para aceitar os métodos POST e GET. O método POST é utilizado para requisitar recursos identificados por uma URI (*Uniform Resource Identifiers*) e é utilizado principalmente para anotar recursos existentes, enviar mensagens para grupos de notícias ou para lista de correio eletrônico, e submeter dados por meio de formulários. O método GET realiza a recuperação das informações produzidas pelos recursos identificados por uma URI.

No passo 3, a aplicação requisita os serviços disponibilizados pelo Context Kernel. No passo 4, o Context Kernel comunica-se com sua base de dados e armazena/recupera as informações que são enviadas/requisitadas pelas aplicações e analisadas pelo Context Kernel. Dependendo do resultado dessa análise e do sucesso no armazenamento ou recuperação da base de dados, o Context Kernel envia uma mensagem de retorno apropriada (passo 5).

5.3 Serviços Disponibilizados pelo Context Kernel

O Context Kernel formaliza um conjunto de operações baseados na linguagem XML associados às dimensões de contexto primitivas e derivadas. Essas operações compõem seus serviços de gerenciamento de informações de contexto e podem ser categorizadas como registro, armazenamento, consulta e recuperação.

O registro aplica-se exclusivamente à análise e ao armazenamento das informações inerentes às aplicações. As operações de armazenamento compreendem o armazenamento de informações primitivas e de regras de contexto. O Context Kernel é exclusivamente incremental, ou seja, não é possível a remoção de dados armazenados em sua base. Essa decisão de projeto foi tomada tendo em vista as aplicações de computação ubíqua. A recuperação compreende as regras de contexto e as informações armazenadas no registro das aplicações. Mais importante que as operações de armazenamento e de recuperação, a de consulta de informações de contexto é a operação principal do Context Kernel. As operações de registro, armazenamento, consulta e recuperação são detalhadas a seguir.

5.3.1 Registro

O registro de informações é o passo inicial que as aplicações devem seguir para a utilização dos serviços disponibilizados pelo Context Kernel. Uma apli-

cação que deseje utilizar o serviço do Context Kernel deve enviar as informações necessárias a esse método, como o nome da aplicação, a descrição, o endereço na Web, o número e a data de sua última versão, o domínio (comercial ou acadêmico) e o nome e o endereço eletrônico de seus desenvolvedores.

O método *InfoApp* irá processar essas informações e, caso os dados sejam válidos, retornará dois identificadores para a aplicação requisitante: um privado e um público. O identificador privado é utilizado para armazenar informações de contexto. O identificador público é utilizado para a consulta e recuperação de dados armazenados, sem permissão de armazenamento; o objetivo é permitir o intercâmbio entre as aplicações. O fragmento XML a seguir descreve as informações da aplicação WebNote¹ (Pimentel et al., 2001b) que devem ser enviados ao método *InfoApp* para seu registro.

```
<application>
  <name>WebNote</name>
  <description>A hypermedia tool for collaborative annotations</description>
  <url>http://coweb.icmc.usp.br/webnote</url>
  <lastRelease>2001-08-26</lastRelease>
  <version>1.0</version>
  <status>academic</status>
  <developers>
    <author email="cizeki@icmc.usp.br">Claudia Akemi Izeki</author>
    <author email="mgp@icmc.usp.br">Graca Pimentel</author>
  </developers>
</application>
```

O método *InfoApp* é responsável pela análise e pela validação dessas informações. Caso a mensagem de envio esteja em conformidade com a definição XML Schema definida no documento de descrição WSDL, o método deve enviar uma mensagem XML de retorno. A mensagem deve fornecer um identificador público de 20 caracteres e um identificador privado de 30 caracteres. Caso a mensagem não seja válida ou a operação de armazenamento na base apresente algum problema, o retorno será uma mensagem de erro. A seguir é ilustrado o fragmento XML retornado pelo método *InfoApp* para uma mensagem de registro bem sucedida.

```
<return>
  <publicID>d6GvPDiOp4ukw2Lbvb9ovecyqh</publicID>
  <privateID>YoNRfmxix4LnQtYLevvEXB6ui79crM</privateID>
</return>
```

¹Aplicação desenvolvida no contexto do Projeto InCA-SERVE que permite anotações individuais e de grupo, na qual cada usuário possui seu próprio repositório de anotações na Web.

5.3.2 Armazenamento

O Context Kernel permite que as aplicações possam armazenar informações primitivas e regras de contexto de forma assíncrona. Essa liberdade possibilita que as aplicações possam armazenar informações primitivas à medida que elas são capturadas, independente das regras de contexto as quais elas pertencem. Para o armazenamento das informações primitivas, o Context Kernel fornece o método *PutData*, e para o armazenamento de regras de contexto, o método *PutRules*. Esses métodos são baseados nos conceitos de premissas, inferências, regras e esquema, todos definidos pelo Context Kernel, conforme detalhado na Seção 5.1.

Os métodos de armazenamento necessitam do identificador privado da aplicação para o mapeamento entre essas informações e a aplicação a qual essas informações pertencem. O método *PutData* requer que as aplicações enviem suas informações de contexto na forma de quádruplas dimensão-tipo-valor-qualificador, sendo que a dimensão deve ser uma das dimensões primitivas definidas pelo Context Kernel. No fragmento XML a seguir, uma aplicação detectou que "o usuário Renato estava no Laboratório Intermídia às 17 horas do dia 15 de março" e deseja armazenar essas informações por meio do método *PutData*.

```
<context>
  <privateID>CSagsqFgO7bmP4NBgwd6mUPOhyGI5e</privateID>
  <primitive>
    <whoS>
      <who type="usuario" value="Renato"/>
    </whoS>
    <whereS>
      <where type="sala" value="Laboratorio Intermidia"/>
    </whereS>
    <whenS>
      <when type="date" value="2003-03-15T17:00:00" qualifier="datetime" />
    </whenS>
  </primitive>
</context>
```

Após a análise e a validação da mensagem de requisição, o método *PutData* retorna uma mensagem apropriada para a aplicação requisitante. Caso a mensagem de requisição esteja em conformidade com a descrição XML Schema definida no documento WSDL e a operação de armazenamento tenha sucesso, o retorno será uma mensagem identificando o sucesso da operação. Caso contrário, o retorno será uma mensagem de falha.

Para o armazenamento de regras de contexto, a aplicação deve enviar ao servidor um ou mais dados primitivos que fazem parte da premissa e um ou

mais dados derivados que fazem parte da inferência da regra. Ainda, a aplicação pode definir um valor temporal para o início e o fim de validade da regra, ou seja, definir um intervalo de tempo para o qual a regra poderá ser utilizada para consultas. Caso uma regra tenha expirado seu tempo final, ela será considerada inativa e armazenada no servidor com finalidade de permitir consultas ao servidor.

No fragmento XML a seguir, uma aplicação identificada por seu identificador privado requisita o método *PutRules* para armazenar a seguinte regra: "uma pessoa cujo nome é Graça Pimentel está todas as quartas-feiras na Sala 1-004 ministrando aula de Interação Usuário-Computador". Ainda, essa regra é considerada válida somente para o primeiro semestre de 2003.

```
<context>
  <privateID>0LdWc7sw405zSGkL080EiYsuczAcSa</privateID>
  <schema>
    <rule begin="01/01/2003" end="01/07/2003">
      <premiseS>
        <premise dimension="who" type="pessoa" value="Graca Pimentel"/>
        <premise dimension="where" type="sala" value="1-004"/>
        <premise dimension="when" type="data" value="quarta-feira"/>
      </premiseS>
      <inferenceS>
        <inference dimension="what" value="Aula de Interacao Usuario-Computador"/>
      </inferenceS>
    </rule>
  </schema>
</context>
```

Assim como o método *PutData*, o método *PutRules* realiza a análise e a validação da mensagem de requisição, e retorna uma mensagem apropriada para a aplicação requisitante.

5.3.3 Consulta

O Context Kernel fornece mecanismos de armazenamento e recuperação de informações de contexto. Porém, seu principal objetivo é permitir que aplicações possam consultar as regras de contexto armazenadas em sua base. Com isso, uma aplicação pode armazenar suas regras de contexto e disponibilizar seu identificador público, permitindo assim que outras aplicações possam consultar essas regras armazenadas previamente. O Context Kernel fornece o método *GetAny* para as operações de consulta e define as informações que as aplicações devem enviar ao método.

O método *GetAny* realiza consultas na base de dados por meio da linguagem de consulta estruturada SQL (*Structured Query Language*). Ao requisitar esse

método, as aplicações podem realizar consultas com base nas dimensões, tipos e valores das informações de contexto armazenadas. Para que esse método seja válido é necessário que a aplicação requisitante envie o identificador público da aplicação e uma especificação XML de consulta baseado nos operadores booleanos AND e OR. Ainda, pode-se requisitar o número máximo de respostas retornadas (atributo *value* do elemento *last*) e quais as dimensões de resposta.

No fragmento XML a seguir, uma aplicação identificada por seu identificador público requer saber "o que foram as duas últimas atividades que uma pessoa chamada Alessandra fez no Recinto de Palestras no dia 4 de março de 2003".

```
<context>
  <publicID></publicID>
  <last value="2"/>
  <premiseS>
    <boolean type="AND">
      <premise dimension="who" type="pessoa" value="Alessandra"/>
      <premise dimension="where" type="sala" value="Recinto de Palestras"/>
      <premise dimension="when" type="data" value="2003-03-04" qualifier="date"/>
    </boolean>
  </premiseS>
  <inferenceS>
    <inference dimension="what"/>
  </inferenceS>
</context>
```

O processo de consulta é realizado internamente pelo método *PutRules*, seguindo as seguintes etapas de processamento:

1. Análise dos elementos *premise* e recuperação de seus identificadores na base de dados;
2. Montagem de uma estrutura de consulta com base nos identificadores recuperados e no operador booleano enviado pela aplicação;
3. Recuperação dos identificadores das regras retornadas pela consulta;
4. Análise das regras recuperadas e obtenção dos valores compreendidos pela dimensão requisitada.

Após a análise e a validação da mensagem de requisição, o método *PutRules* retorna uma mensagem apropriada para a aplicação requisitante. Caso a mensagem de requisição esteja em conformidade com a descrição XML Schema definida no documento WSDL e a operação de armazenamento tenha sucesso, o retorno será composto dos valores obtidos pela consulta ou uma mensagem identificando que nada foi encontrado. Caso ocorra algum erro, o retorno será

uma mensagem de falha. O fragmento XML a seguir identifica um possível retorno para a consulta anterior.

```
<return>
  <value>Saiu da Sala</value>
  <value>Ministrando Palestra sobre Information Retrieval</value>
</return>
```

5.3.4 Recuperação

O Context Kernel fornece três métodos para a recuperação das informações armazenadas em sua base. O método *GetRules* permite a recuperação das regras de contexto armazenadas. Os métodos *StatusApp* e *StatusApps* permitem a recuperação das informações armazenadas no registro das aplicações.

O método *GetRules* permite que uma aplicação possa recuperar as regras de contexto armazenadas previamente por meio do identificador público da aplicação. A própria aplicação ou quaisquer outras que possuam o identificador público podem recuperar as regras de contexto que foram armazenadas. O fragmento XML a seguir corresponde a uma mensagem de retorno para a aplicação eMeet² (Eleutério and Pimentel, 2001).

```
<schema>
  <rule>
    <premiseS>
      <premise dimension="who" type="professor" value="Renata Pontin" qualifier=""/>
      <premise dimension="who" type="aluno" value="Lisandra" qualifier=""/>
      <premise dimension="who" type="aluno" value="Daniel" qualifier=""/>
      <premise dimension="where" type="sala" value="1-004" qualifier=""/>
      <premise dimension="when" type="horario" value="14:00:00" qualifier="time"/>
    </premiseS>
    <inferenceS>
      <inference dimension="what" value="Reuniao de Engenharia de Software"/>
    </inferenceS>
  </rule>
  <rule>
    <premise dimension="who" type="professor" value="Graca Pimentel" qualifier=""/>
    <premise dimension="who" type="professor" value="Renata Pontin" qualifier=""/>
    <premise dimension="who" type="professor" value="Edson Moreira" qualifier=""/>
    <premise dimension="where" type="sala" value="1-004" qualifier=""/>
    <premise dimension="when" type="horario" value="16:00:00" qualifier="time"/>
  </premiseS>
  <inferenceS>
    <inference dimension="what" value="Reuniao do Grupo de Hipermidia"/>
  </inferenceS>
</rule>
</schema>
```

²Aplicação desenvolvida no contexto do Projeto InCA-SERVE para suporte a reuniões distribuídas em grupo

Além de recuperar as regras de contexto, é possível que uma aplicação possa recuperar as informações enviadas ao servidor na operação de registro. Por meio do identificador público da aplicação, o método *StatusApp* implementa a recuperação dessas informações. Uma aplicação qualquer, mesmo que não registrada no servidor, pode recuperar essas informações, que serão retornadas como no fragmento XML ilustrado a seguir.

```
<applications>
  <application>
    <name>CoTeia</name>
    <description>A tool for collaborative web page authoring</description>
    <url>http://coweb.icmc.usp.br/</url>
    <lastRelease>2001-01-06</lastRelease>
    <version>1.0</version>
    <status>academic</status>
    <publicID>991R0K1ARiPaGJzazbYhvwV40</publicID>
    <toCK>2003-01-03T14:36:20</toCK>
    <developers>
      <author email="credajun@icmc.usp.br">Carlos Arruda Junior</author>
    </developers>
  </application>
  <application>
    <name>iClass</name>
    <description>A capture and access application</description>
    <url>http://catuaba.icmc.usp.br/eclass/</url>
    <lastRelease>2002-08-07</lastRelease>
    <version>1.0</version>
    <status>academic</status>
    <publicID>z1q8UYXEjRzySsqJuWdUInkFY</publicID>
    <toCK>2003-01-04T14:36:20</toCK>
    <developers>
      <author email="cattelan@icmc.usp.br">Renan Cattelan</author>
    </developers>
  </application>
</applications>
```

Os métodos descritos anteriormente, a menos do método de registro *InfoApp*, necessitam dos identificadores público ou privado. Com base nessa necessidade, o método *StatusApps* fornece a descrição de todas as aplicações cadastradas no servidor e é possível a obtenção do identificador público de cada uma delas. Por motivo de segurança das informações armazenadas, os identificadores privados não são disponibilizados. Por exemplo, caso uma aplicação necessite da URL da ferramenta WebNote, pode-se requisitar o método *StatusApps* e enviar a chave pública dessa ferramenta. O fragmento XML a seguir ilustra o retorno dessa mensagem.

```
<application>
  <name>WebNote</name>
  <description>A hypermedia tool for collaborative annotations</description>
```

```

<url>http://coweb.icmc.usp.br/webnote</url>
<lastRelease>2001-08-26</lastRelease>
<version>1.0</version>
<status>academic</status>
<publicID>d6GvPDiOp4ukw2Lbvb9oveyqh</publicID>
<toCK>2003-01-05T13:33:11</toCK>
<developers>
  <author email="cizeki@icmc.usp.br">Claudia Akemi Izeki</author>
  <author email="mgp@icmc.usp.br">Graca Pimentel</author>
</developers>
</application>

```

O conteúdo dos elementos *toCK*, definidos para os métodos de recuperação, define o valor temporal no qual a aplicação foi registrada no Context Kernel.

5.4 Aplicações Heterogêneas Integradas com o Context Kernel

Nesta seção são descritas duas aplicações desenvolvidas no Projeto InCAsERVE em parceria entre o grupo do Dr. Gregory Abowd, do Georgia Institute of Technology (GaTech), e pesquisadores do Laboratório Intermedia do Instituto de Ciências Matemáticas e de Computação (ICMC) da USP de São Carlos. Neste contexto são descritas as aplicações CoTeia (Arruda Jr and Pimentel, 2001) e iClass (Cattelan et al., 2003), e é discutido como essas aplicações heterogêneas podem fazer uso dos serviços fornecidos pelo Context Kernel.

5.4.1 Coteia

CoTeia³ é uma ferramenta hipermídia colaborativa assíncrona para a edição de páginas Web, possuindo como características principais: (a) simplicidade de uso, ao permitir que seus usuários editem os documentos na própria ferramenta de navegação e não impor conhecimentos prévios em linguagens de marcação, (b) ausência de estrutura pré-definida das páginas, e (c) existência de funcionalidades tais como anotações, *chat* e mapa de repositório que facilitam a colaboração de informações entre os usuários.

A ferramenta foi desenvolvida com base na linguagem PHP para a implementação, especificações XML para estruturação e apresentação dos documentos e banco de dados MySQL para armazenamento de suas informações. Utilizada no ICMC desde o início do ano de 2001, a ferramenta CoTeia vem demonstrando

³<http://coweb.icmc.usp.br/>

várias utilidades das quais destacam-se: escrita colaborativa de artigos, agenda de reuniões de grupos de pesquisa, repositório de documentos, *homepage* de projetos e disciplinas, e acompanhamento das atividades de disciplinas de graduação e pós-graduação do curso de Ciências de Computação (Arruda Jr et al., 2002). Nesses três anos de utilização, a ferramenta CoTeia tem sido utilizada em cerca de uma centena de cursos de graduação e pós-graduação.

A ferramenta CoTeia pode ser estendida utilizando informações de contexto tais como identificação do usuário (nome, login, senha, email, preferências e permissões), identificação dos hiperdocumentos gerados (identificador numérico, endereço local no servidor), localização espacial (meio de acesso e endereço da máquina de acesso), localização temporal (login e logout) e atividade (criação/edição de páginas e transferências de arquivos).

5.4.2 iClass

A ferramenta iClass⁴ implementa um ambiente de computação ubíqua que permite a captura de material multimídia em uma sala de aula convencional, e a disponibilização e o acesso desse material através da Web. A versão utilizada no ICMC desde o segundo semestre do ano de 2002 foi implementada com base na plataforma Java, padrões XML para estruturação e apresentação de informações e banco de dados Xindice para armazenamento de documentos XML.

Para a captura e a disponibilização do material capturado, a ferramenta utiliza uma infra-estrutura adaptada na sala de aula de forma não intrusiva. A ferramenta iClass utiliza equipamentos de captura que incluem lousa eletrônica, projetor e microfone. A lousa eletrônica é uma tela sensível ao toque que permite armazenar as interações do usuário com a tela. Ainda, utiliza-se módulos de software para captura, sincronização dos diversos fluxos de informação capturados, armazenamento e geração automática de hiperdocumentos.

Em relação as informações de contexto, a ferramenta iClass pode fornecer informações primitivas e derivadas, como identificação do usuário (nome, login, senha), identificação das aulas capturadas (título, professor), localização espacial (sala de aula) e localização temporal (data de captura, período de uma disciplina, tempo de visita de acesso).

⁴<http://catuaba.icmc.usp.br/iclass/>

5.5 Cenários de Integração com o Context Kernel

O Context Kernel implementado como um *Web Service* não restringe as características de implementação das aplicações que desejam requisitar seus serviços. Com isso, é possível que qualquer aplicação Web possa utilizar os serviços fornecidos pelo Context Kernel, independente de heterogeneidade em termos de linguagem de programação ou sistema operacional.

Nesta monografia são ilustrados dois cenários de utilização dos serviços fornecidos pelo Context Kernel. Na Seção 5.5.1, a aplicação CoTeia (Arruda Jr and Pimentel, 2001), implementada em PHP, armazena suas regras de contexto e a aplicação iClass (Cattelan et al., 2003), implementada em Java, realiza a consulta dessas informações. Na Seção 5.5.2, a aplicação iClass armazena suas regras e a aplicação CoTeia realiza a consulta.

As aplicações Web implementadas em PHP ou Java podem utilizar a biblioteca de funções disponibilizadas pelo Context Kernel, simplificando ainda mais o processo de armazenamento e recuperação das informações de contexto.

As aplicações CoTeia e iClass possuem informações de contexto primitivas e derivadas. Além disso, possuem regras de contexto que podem ser armazenadas no Context Kernel. Por exemplo, a aplicação CoTeia pode construir uma regra de contexto baseada nas premissas de identificação do usuário (*who*) e localização temporal (*when*), e inferir a atividade do usuário (*what*). Esse processo de inferência e construção de regras é dependente das aplicações.

5.5.1 Cenário 1: Coteia armazena e iClass consulta regras

O registro da aplicação é passo inicial para a utilização dos serviços disponibilizados pelo Context Kernel e, como descrito na Seção 5.3.1, o método *InfoApp* é responsável pelo registro das aplicações. A seguir é apresentada uma mensagem de requisição para o registro da aplicação CoTeia.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
  soap-env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <soap-env:Header/>
  <soap-env:Body>
    <ck:InfoAppRequest xmlns:ck="http://mexcal.intermidia.icmc.usp.br/ck/xsd/InfoApp.xsd">
      <application>
        <name>CoTeia</name>
        <description>Ferramenta colaborativa assincrona para edicao de paginas Web</description>
        <url>http://coweb.icmc.usp.br/</url>
        <lastRelease>2001-01-06</lastRelease>
        <version>1.0</version>
      </application>
    </ck:InfoAppRequest>
  </soap-env:Body>
</soap-env:Envelope>
```

```

<status>academic</status>
<developers>
  <author email="credajun@icmc.usp.br">Carlos Arruda Junior</author>
  <author email="mcp@icmc.usp.br">Graca Pimentel</author>
</developers>
</application>
</ck:InfoAppRequest>
</soap-env:Body>
</soap-env:Envelope>

```

A aplicação CoTeia, entretanto, pode utilizar a biblioteca de funções fornecidas pelo Context Kernel para simplificar o processo de registro. Com a utilização dessas funções, a montagem da mensagem torna-se transparente à aplicação. A seguir é ilustrado um fragmento de código PHP utilizado pela aplicação CoTeia e baseado na biblioteca de funções.

```

<?PHP

//biblioteca de funcoes
include_once("functionCK.inc");

//constroi a mensagem SOAP e retorna a mensagem de resposta
$retornoInfoApp = infoApp($name,$desc,$url,$ultRelease,$version,$status,$developers);

//obtem identificador publico
$publicID = regPubID($retornoInfoApp);

//obtem identificador privado
$privID = regPrivID($retornoInfoApp);

?>

```

Por meio da função *regPrivId*, a aplicação CoTeia pode recuperar o identificador privado gerado pelo Context Kernel e armazenar suas regras de contexto. Como descrito na Seção 5.3.2, o método *PutRules* é responsável pelo armazenamento das regras de contexto. A seguir é visualizada uma mensagem de requisição para o armazenamento de regras de contexto da aplicação CoTeia, a qual indica que "o usuário credajun está editando o documento 15.3.3".

```

<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
  soap-env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <soap-env:Header/>
  <soap-env:Body>
    <ck:PutRulesRequest xmlns:ck="http://coweb.icmc.usp.br/ck/xsd/PutRulesRequest.xsd">
      <privateID>0LdWc7sw405zSSkLO80EiYsucZAcSa</privateID>
      <context>
        <schema>
          <rule>
            <premiseS>
              <premise dimension="who" type="usuario" value="credajun" qualifier=""/>

```



```

        <premise dimension="who" type="documento" value="15.3.3" qualifier="" />
    </premiseS>
    <inferenceS>
        <inference dimension="what" value="edição" />
    </inferenceS>
    </rule>
</schema>
</context>
</ck:PutRulesRequest>
</soap-env:Body>
</soap-env:Envelope>

```

Uma vez que existam regras armazenadas, é possível que se realize consultas a essas regras. Por exemplo, para que a aplicação iClass possa consultar as regras da aplicação CoTeia, é necessário que se obtenha o identificador público da aplicação CoTeia, que pode ser recuperado por meio do método *StatusApps*. Como descrito na Seção 5.3.4, o método *StatusApps* retorna a descrição e o identificador público de todas as aplicações registradas no servidor. A seguir é ilustrado o fragmento de código Java utilizado pela aplicação iClass para a obtenção do identificador público da aplicação CoTeia.

```

import context.StatusApps;

//instancia da classe StatusApps
StatusApps statusApps = new StatusApps();

/* requisita o servico StatusApps e retorna a mensagem de resposta */
SOAPMessage selem = statusApps.call();

/* recupera o identificador publico da aplicacao CoTeia */
String publicID = statusApps.getPubID(selem, "CoTeia");

```

Com a obtenção do identificador público, é possível que a aplicação iClass consulte as regras armazenadas pela aplicação CoTeia. Como descrito na Seção 5.3.3, o método *GetAny* é responsável por realizar consultas na base de dados e de acordo com a mensagem XML a seguir, a aplicação iClass deseja saber "a última atividade do usuário credajun".

```

<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
  soap-env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <soap-env:Header/>
  <soap-env:Body>
    <ck:GetAnyRequest xmlns:ck="http://mexcal.intermidia.icmc.usp.br/ck/xsd/GetAnyRequest.xsd">
      <publicID>CSagsqFg07hmp4NBgwd6mUP0hyGI5e</publicID>
      <lastValue="1"/>
      <context>
        <premiseS>
          <boolean type="AND">
            <premise dimension="who" type="usuario" value="credajun" qualifier="" />

```

```

        </boolean>
    </premiseS>
    <inferenceS>
        <inference dimension="what"/>
    </inferenceS>
</context>
</ck:GetAnyRequest>
</soap-env:Body>
</soap-env:Envelope>

```

A seguir é ilustrado o fragmento de código Java utilizado pela aplicação iClass para a consulta ao Context Kernel. Neste exemplo, o conteúdo da variável *valor* é "edição".

```

import context.GetAny;

//instancia da classe StatusApps
GetAny valorGA = new GetAny();

/* requisita o servico GetAny e retorna a mensagem de resposta */
SOAPMessage selem = valorGA.call(GAnyObj);

/* recupera o valor retornado */
String valor = valorGA.getValue(selem);

```

5.5.2 Cenário 2: iClass armazena e Coteia consulta regras

Para que a aplicação iClass possa armazenar suas regras no Context Kernel, é necessário o registro da aplicação. Por meio da biblioteca de métodos Java fornecida pelo Context Kernel, a aplicação iClass pode simplificar o processo de registro. A seguir é ilustrado um fragmento de código Java utilizado para o registro da aplicação iClass.

```

import context.InfoApp;

//instancia da classe InfoApp
InfoApp registro = new InfoApp();

/* requisita o servico InfoApp e retorna a mensagem de resposta */
SOAPMessage selem = registro.call(IAppObj);

/* recupera o identificador público */
String pubID = registro.getPubID(selem);

/* recupera o identificador privado */
String privID = registro.getPrivID(selem);

```

Neste cenário, a aplicação iClass deseja armazenar que "o professor Carlos ministra aula de Estrutura de Dados na sala 1-002 todas as quartas-feiras". O fragmento de código Java apresentado a seguir corresponde a uma requisição ao método *PutRules* com esse conteúdo.

```

import context.InfoApp;

//instancia da classe InfoApp
PutRules ptRules = new PutRules();

/* informacoes primitivas */
String premises[][] = new String[2][4];
premises[0][0] = "who";
premises[0][1] = "professor";
premises[0][2] = "Carlos";
premises[0][3] = "";

premises[1][0] = "where";
premises[1][1] = "sala";
premises[1][2] = "1-003";
premises[1][3] = "";

premises[2][0] = "when";
premises[2][1] = "data";
premises[2][2] = "quarta-feira";
premises[2][3] = "";

/* informacao derivada */
String inferences[][] = new String[1][2];
inferences[0][0] = "what";
inferences[0][1] = "Ministra aula de estrutura de dados";

/* envia essas informações ao servidor */
SOAPMessage selem = ptRules.call(privID,premises,inferences);

/* recebe a confirmação de sucesso ou erro de envio */
String status = ptRules.getStatus(selem);

```

Uma vez armazenadas as regras de contexto da aplicação iClass, a aplicação CoTeia pode consultar a base de dados. Para a consulta utiliza-se o método *GetAny*, descrito na Seção 5.3.3. O fragmento de código PHP ilustrado a seguir corresponde à requisição ao método *StatusApp* para a obtenção do identificador público da aplicação iClass.

```

<?PHP

//biblioteca de funcoes
include_once("functionCK.inc");

// requisita o método StatusApps
$ret = statusApps();

// recupera o identificador publico retornado pelo metodo StatusApps
$pubID = getPublicID($ret, "iClass");

?>

```

O fragmento de código PHP a seguir corresponde à requisição ao método *GetAny*, ao qual a aplicação Coteia precisa consultar "o que o professor Car-

los faz as quartas-feiras?". O conteúdo da variável *valor* é "Ministra aula de estrutura de dados" e responde à consulta realizada pela aplicação CoTeia.

```
<?PHP

//biblioteca de funcoes
include_once("functionCK.inc");

//premissas
$premises[0][0] = "who";
$premises[0][1] = "professor";
$premises[0][2] = "Carlos";
$premises[0][3] = "";

$premises[1][0] = "when";
$premises[1][1] = "data";
$premises[1][2] = "quarta-feira";
$premises[1][3] = "";

//independente de localizacao
$premises[2][0] = "where";
$premises[2][1] = "";
$premises[2][2] = "";
$premises[2][3] = "";

//dimensao de retorno
$inferences[0] = "what";

// consulta o Context Kernel para obter a dimensao WHAT
$ret = getAny($publicID, "1", "AND", $premises, $inferences);

// obtem o conteudo da dimensao WHAT
$valor = getValue($ret);

?>
```

5.6 Limitações do Context Kernel

A versão do Context Kernel descrita nesta dissertação está preparada para integrar as aplicações desenvolvidas no contexto do Projeto InCA-SERVE. Entretanto, por se tratar da primeira versão, apresenta limitações que incluem:

- ausência de um mecanismo de processamento automático para o documento de descrição do Context Kernel. Uma nova aplicação que deseje requisitar os serviços fornecidos pelo Context Kernel deve recuperar e interpretar a descrição, ou seja, deve fazer a análise do documento para construir mensagens válidas para a requisição dos serviços;

- a plataforma de *Web Services* propõe que as descrições sejam mantidas em um repositório público dedicado à publicação de *Web Services*. Como o Context Kernel se encontra em estágio de testes e integração com aplicações internas ao Projeto InCA-SERVE, a publicação de sua descrição foi propositalmente postergada;
- ausência de uma análise estatística quanto ao desempenho e escalabilidade do Context Kernel. A consulta ao servidor, por exemplo, pode ser realizada por um processo em execução na aplicação requisitante e deve apresentar um tempo de resposta eficiente. Essas medidas devem ser analisadas e, caso não sejam satisfatórias, devem conduzir à otimização do servidor;
- ausência de um mecanismo de segurança integrado ao Context Kernel. Ainda que seus serviços não sejam confidenciais, é interesse utilizar mecanismos de criptografia ou autenticação para expandir a segurança às aplicações que requisitam esses serviços;
- o Context Kernel fornece uma biblioteca de funções para as aplicações implementadas nas linguagens PHP e Java, que compreendem as aplicações internas ao Projeto InCA-SERVE. Aplicações implementadas em outras linguagens deverão desenvolver suas próprias operações de chamadas e interpretação das mensagens de resposta. Ainda que exista tal limitação, os serviços disponibilizados pelo Context Kernel são independentes de implementação;
- o mecanismo de consulta do Context Kernel é baseado na combinação exata de palavras. Por exemplo, uma aplicação pode armazenar uma instância da dimensão *where* na qual o tipo é "sala de aula". Como consultas por "salas de aula", "sala" ou "room" não se aplicam a essa instância, o Context Kernel deve desenvolver um mecanismo de busca semântica para minimizar essa restrição. Ainda, o mecanismo de consulta é baseado nos operadores booleanos AND ou OR. Para que aplicações possam realizar consultas mais flexíveis e complexas, é necessário expandir o mecanismo e formalizar novos operadores.

Essas limitações são tema de trabalho futuro no contexto do Projeto InCA-SERVE.

5.7 Considerações Finais

Segundo Abowd & Mynnat (2000), um dos principais desafios da computação ubíqua é tornar as aplicações cientes de contexto. Imbuído desse objetivo, o Context Kernel formaliza um conjunto de operações para armazenamento e recuperação de informações de contexto com base nas dimensões de contexto propostas na literatura. Esse núcleo de operações, responsável pela manipulação dessas informações de contexto, deve permitir que aplicações de domínios distintos possam especializá-lo de acordo com seus domínios e seus diferentes elementos de contexto.

O objetivo do Context Kernel é permitir que aplicações possam tirar proveito de informações de contexto tais como personalização, navegação, histórico e consulta de informações. Implementado como um *Web Service*, o Context Kernel possibilita que quaisquer aplicações possam se relacionar independente de suas diferenças. Ainda que apresente limitações, esta primeira versão do Context Kernel está preparada para integrar as aplicações desenvolvidas no contexto do Projeto InCA-SERVE.

Conclusões

A computação ubíqua foi definida por Weiser (1991) como uma forma de computação transparente incorporada às mais diversas atividades cotidianas do ser humano. Uma das classes principais da computação ubíqua é a computação ciente de contexto, a qual tem como propósito facilitar a interação do usuário-computador ao fazer uso de informações de contexto tais como identificação (*who*), localização espacial (*where*), localização temporal (*when*), modo de captura e de acesso (*how*), atividade (*what*) e motivação (*why*), chamadas dimensões de contexto.

Um dos desafios da computação ubíqua é apoiar aplicações cientes de contexto (Abowd and Mynatt, 2000). Sistemas como o Context Fabric (Hong and Landay, 2001), o EventHeap (Johanson and Fox, 2002) e o Gaia (Hess et al., 2002), desenvolvidos para apoiar serviços dedicados de captura, de armazenamento e de processamento de informações de contexto não exploram a plataforma de *Web Services* como alternativa para o tratamento da heterogeneidade entre aplicações e nem apoiam diretamente as dimensões de contexto propostas na literatura.

O trabalho aqui apresentado teve como objetivo propor uma conjunto de operações transparentes de armazenamento, de extensão e de recuperação de informação de contexto independentemente de diferenças em termos de tecnologia de rede, de sistema operacional ou de linguagem de programação. Como resultado, foi modelado e implementado o Context Kernel que, explorando a plataforma de

Web Services para garantir a independência almejada, implementa um conjunto de operações para o gerenciamento de informações de contexto com base nas dimensões de contexto propostas na literatura (Arruda Jr et al., 2003).

6.1 Contribuições

As principais contribuições deste trabalho são:

- um modelo de armazenamento de informações de contexto baseado nas dimensões de contexto propostas por Abowd & Mynatt (2000) e Truong et al. (2001). O Context Kernel é responsável pela manipulação dessas informações de contexto e permite que aplicações de domínios distintos possam especializá-lo de acordo com seus domínios e seus diferentes elementos de contexto;
- um núcleo de operações para manipulação de armazenamento e recuperação de informações de contexto. O Context Kernel, implementado como um *Web Service*, fornece suporte para que aplicações heterogêneas possam utilizar os serviços disponibilizados e trocar informações em um paradigma de armazenamento e consulta;
- uma biblioteca de funções de acesso ao Context Kernel por programas implementados nas linguagens PHP e Java. Com isso, as aplicações implementadas em uma dessas linguagens pode utilizar essa biblioteca para facilitar a manipulação de registro, armazenamento, recuperação e consulta de informações de contexto.

6.2 Trabalhos Futuros

O Context Kernel, como modelado e implementado, compreende a primeira versão do núcleo de operações para armazenamento, extensão e recuperação de informações de contexto do Projeto InCA-SERVE. São temas de trabalho futuro a superação das principais limitações identificadas na versão atual do Context Kernel: a ausência de mecanismo automatizado para análise da especificação recebida por parte das aplicações, limitações em termos da flexibilidade das consultas suportadas e ausência de mecanismos de segurança, como detalhado a seguir. Além disso, se fazem necessárias a avaliação de desempenho do serviço e a exploração de mecanismos para a aquisição de informações semânticas.

Aquisição de informações semânticas. A Web é uma fonte de informação com potencial virtualmente ilimitado. Entretanto, este potencial é muito pouco aproveitado devido a dificuldade das máquinas para processar e recuperar as informações de forma significativa. Recentemente, pesquisadores vêm explorando o potencial de se associar significados explícitos aos conteúdos dos documentos presentes na Web, a fim de criar a Web Semântica (Berners-Lee et al., 2001). O objetivo da Web Semântica é desenvolver padrões e tecnologias para auxiliar os computadores na tarefa de fornecer significado as informações. Com isso, os computadores podem expandir seus mecanismos de descoberta de informações, integração de dados, navegação e automatização de tarefas.

Com o desenvolvimento dos *Web Services* e a evolução da Web Semântica, pode-se almejar a automatização de algumas das principais tarefas dos *Web Services*, como, por exemplo, a descrição e a composição de seus serviços (Narayanan and Mellraith, 2002). Como trabalho futuro, pretende-se expandir, no contexto de um projeto de doutorado (Bulcão Neto, 2003), o mecanismo de armazenamento e recuperação das informações de contexto com os benefícios proporcionados com a utilização de ontologias.

Expansão do método de consulta. O Context Kernel permite que uma aplicação consulte as regras de contexto armazenadas por outra aplicação. Com a evolução do mecanismo de consulta, pode-se expandir, no contexto de um projeto de mestrado (Godoy, 2003), essa troca de informações entre as diferentes aplicações. Como trabalho futuro propõe-se:

- flexibilizar o modelo formal das mensagens de consulta as informações de contexto;
- expandir o mecanismo de consulta com o objetivo de automatizar o processo que notifica as aplicações da ocorrência de uma determinada regra. Por exemplo, pode-se determinar a execução de um processo a cada ocorrência de uma determinada regra de contexto.

Utilização de mecanismos de segurança. Para o empacotamento de mensagens para a comunicação, os *Web Services* utilizam o protocolo SOAP, que desde suas primeiras versões não contempla eficientemente a questão de segurança. Como o protocolo SOAP possibilita o acesso a serviços remotos e o conteúdo de suas mensagens não é garantido, faz-se necessária a utilização de mecanismos para proteger o servidor de requisições maliciosas (Treese, 2002).

Apesar de SOAP não implementar um mecanismo próprio de segurança, existem soluções para aumentar a segurança do protocolo. A solução mais simples é

utilizar protocolos de segurança como SSL (*Secure Sockets Layer*) ou TLS (*Transport Layer Security*). Outra solução seria utilizar padrões de segurança baseados em XML, como a recomendação XML Signature (W3C, 2002c), desenvolvida pelo W3C. Esses mecanismos de segurança devem ser integrados ao Context Kernel como trabalho futuro.

6.3 Considerações Finais

A implementação do Context Kernel sob a forma de um *Web Service* permite que seus serviços sejam utilizados não somente no contexto do Projeto InCA-SERVE, mas por quaisquer aplicações que desejem armazenar e consultar informações de contexto. No Projeto InCA-SERVE, o Context Kernel possui importante função, uma vez que as diversas aplicações do projeto podem se beneficiar das vantagens fornecidas não apenas em termos do gerenciamento de informações de contexto mas também pelo intercâmbio de informações de contexto entre aplicações.

Apêndice: Especificação WSDL

```
<?xml version="1.0"?>
<definitions targetNamespace="http://coweb.icmc.usp.br/ck/ContextKernel.wsdl"
xmlns:tns="http://coweb.icmc.usp.br/ck/ContextKernel.wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:schema1="http://coweb.icmc.usp.br/ck/xsd/InfoAppRequest.xsd"
xmlns:schema2="http://coweb.icmc.usp.br/ck/xsd/InfoAppResponse.xsd"
xmlns:schema3="http://coweb.icmc.usp.br/ck/xsd/PutDataRequest.xsd"
xmlns:schema4="http://coweb.icmc.usp.br/ck/xsd/PutDataResponse.xsd"
xmlns:schema5="http://coweb.icmc.usp.br/ck/xsd/PutRulesRequest.xsd"
xmlns:schema6="http://coweb.icmc.usp.br/ck/xsd/PutRulesResponse.xsd"
xmlns:schema7="http://coweb.icmc.usp.br/ck/xsd/GetAnyRequest.xsd"
xmlns:schema8="http://coweb.icmc.usp.br/ck/xsd/GetAnyResponse.xsd"
xmlns:schema9="http://coweb.icmc.usp.br/ck/xsd/GetRulesRequest.xsd"
xmlns:schema10="http://coweb.icmc.usp.br/ck/xsd/GetRulesResponse.xsd"
xmlns:schema11="http://coweb.icmc.usp.br/ck/xsd/StatusAppRequest.xsd"
xmlns:schema12="http://coweb.icmc.usp.br/ck/xsd/StatusAppResponse.xsd"
xmlns:schema13="http://coweb.icmc.usp.br/ck/xsd/StatusAppsRequest.xsd"
xmlns:schema14="http://coweb.icmc.usp.br/ck/xsd/StatusAppsResponse.xsd">

<documentation>Descrição dos serviços oferecidos pelo Context Kernel</documentation>

<message name="CKinfoAppRequest">
  <part name="body" element="schema1:InfoAppRequest"/>
</message>
<message name="CKinfoAppResponse">
  <part name="body" element="schema2:InfoAppResponse"/>
</message>
<message name="CKputDataRequest">
  <part name="body" element="schema3:PutDataRequest"/>
</message>
<message name="CKputDataResponse">
```

```

        <part name="body" element="schema4:PutDataResponse"/>
    </message>
    <message name="CKputRulesRequest">
        <part name="body" element="schema5:PutRulesRequest"/>
    </message>
    <message name="CKputRulesResponse">
        <part name="body" element="schema6:PutRulesResponse"/>
    </message>
    <message name="CKgetAnyRequest">
        <part name="body" element="schema7:GetAnyRequest"/>
    </message>
    <message name="CKgetAnyResponse">
        <part name="body" element="schema8:GetAnyResponse"/>
    </message>
    <message name="CKgetRulesRequest">
        <part name="body" element="schema9:GetRulesRequest"/>
    </message>
    <message name="CKgetRulesResponse">
        <part name="body" element="schema10:GetRulesResponse"/>
    </message>
    <message name="CKstatusAppRequest">
        <part name="body" element="schema11:StatusAppRequest"/>
    </message>
    <message name="CKstatusAppResponse">
        <part name="body" element="schema12:StatusAppResponse"/>
    </message>
    <message name="CKstatusAppsRequest">
        <part name="body" element="schema13:StatusAppsRequest"/>
    </message>
    <message name="CKstatusAppsResponse">
        <part name="body" element="schema14:StatusAppsResponse"/>
    </message>

    <portType name="CKPortTypeInfoApp">
        <operation name="getInfoApp">
            <input message="tns:CKinfoAppRequest" name="infoAppInput"/>
            <output message="tns:CKinfoAppResponse" name="infoAppOutput"/>
        </operation>
    </portType>
    <portType name="CKPortTypePutData">
        <operation name="putData">
            <input message="tns:CKputDataRequest" name="putDataInput"/>
            <output message="tns:CKputDataResponse" name="putDataOutput"/>
        </operation>
    </portType>
    <portType name="CKPortTypePutRules">
        <operation name="putRules">
            <input message="tns:CKputRulesRequest" name="putRulesInput"/>
            <output message="tns:CKputRulesResponse" name="putRulesOutput"/>
        </operation>
    </portType>
    <portType name="CKPortTypeGetAny">
        <operation name="getAny">
            <input message="tns:CKgetAnyRequest" name="getAnyInput"/>
            <output message="tns:CKgetAnyResponse" name="getAnyOutput"/>
        </operation>
    </portType>

```

```

</portType>
<portType name="CKPortTypeGetRules">
  <operation name="getRules">
    <input message="tns:CKgetRulesRequest" name="getRulesInput"/>
    <output message="tns:CKgetRulesResponse" name="getRulesOutput"/>
  </operation>
</portType>
<portType name="CKPortTypeStatusApp">
  <operation name="getStatusApp">
    <input message="tns:CKstatusAppRequest" name="getStatusAppInput"/>
    <output message="tns:CKstatusAppResponse" name="getStatusAppOutput"/>
  </operation>
</portType>
<portType name="CKPortTypeStatusApps">
  <operation name="getStatusApps">
    <input message="tns:CKstatusAppsRequest" name="getStatusAppsInput"/>
    <output message="tns:CKstatusAppsResponse" name="getStatusAppsOutput"/>
  </operation>
</portType>

<binding name="CKInfoAppBinding" type="tns:CKPortTypeInfoApp">
  <soap:binding style="document" transport="http://www.w3.org/2002/12/soap/bindings/HTTP"/>
  <operation name="getInfoApp">
    <input>
      <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
    </input>
    <output>
      <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
    </output>
  </operation>
</binding>
<binding name="CKputDataBinding" type="tns:CKPortTypePutData">
  <soap:binding style="document" transport="http://www.w3.org/2002/12/soap/bindings/HTTP"/>
  <operation name="putData">
    <input>
      <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
    </input>
    <output>
      <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
    </output>
  </operation>
</binding>
<binding name="CKputRulesBinding" type="tns:CKPortTypePutRules">
  <soap:binding style="document" transport="http://www.w3.org/2002/12/soap/bindings/HTTP"/>
  <operation name="putRules">
    <input>
      <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
    </input>
    <output>
      <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
    </output>
  </operation>
</binding>
<binding name="CKgetAnyBinding" type="tns:CKPortTypeGetAny">
  <soap:binding style="document" transport="http://www.w3.org/2002/12/soap/bindings/HTTP"/>
  <operation name="getAny">

```

```

        <input>
          <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
        </input>
        <output>
          <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
        </output>
      </operation>
    </binding>
    <binding name="CKgetRulesBinding" type="tns:CKPortTypeGetRules">
      <soap:binding style="document" transport="http://www.w3.org/2002/12/soap/bindings/HTTP"/>
      <operation name="getRules">
        <input>
          <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
        </input>
        <output>
          <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
        </output>
      </operation>
    </binding>
    <binding name="CKstatusAppBinding" type="tns:CKPortTypeStatusApp">
      <soap:binding style="document" transport="http://www.w3.org/2002/12/soap/bindings/HTTP"/>
      <operation name="getStatusApp">
        <input>
          <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
        </input>
        <output>
          <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
        </output>
      </operation>
    </binding>
    <binding name="CKstatusAppsBinding" type="tns:CKPortTypeStatusApps">
      <soap:binding style="document" transport="http://www.w3.org/2002/12/soap/bindings/HTTP"/>
      <operation name="getStatusApps">
        <input>
          <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
        </input>
        <output>
          <soap:body encodingStyle="http://www.w3.org/2002/12/soap-envelope/encoding/none" use="literal"/>
        </output>
      </operation>
    </binding>
  </service name="ContextKernelService">
    <port name="CKinfoAppPort" binding="tns:CKinfoAppBinding">
      <soap:address location="http://mexcal.intermidia.icmc.usp.br:80/context/InfoApp"/>
    </port>
    <port name="CKputDataPort" binding="tns:CKputDataBinding">
      <soap:address location="http://mexcal.intermidia.icmc.usp.br:80/context/PutData"/>
    </port>
    <port name="CKputRulesPort" binding="tns:CKputRulesBinding">
      <soap:address location="http://mexcal.intermidia.icmc.usp.br:80/context/PutRules"/>
    </port>
    <port name="CKgetAnyPort" binding="tns:CKgetAnyBinding">
      <soap:address location="http://mexcal.intermidia.icmc.usp.br:80/context/GetAny"/>
    </port>
    <port name="CKgetRulesPort" binding="tns:CKgetRulesBinding">

```

```
        <soap:address location="http://mexcal.intermidia.icmc.usp.br:80/context/GetRules"/>
    </port>
    <port name="CKstatusAppPort" binding="tns:CKstatusAppBinding">
        <soap:address location="http://mexcal.intermidia.icmc.usp.br:80/context/StatusApp"/>
    </port>
    <port name="CKstatusAppsPort" binding="tns:CKstatusAppsBinding">
        <soap:address location="http://mexcal.intermidia.icmc.usp.br:80/context/StatusApps"/>
    </port>
</service>
</definitions>
```


Referências Bibliográficas

- Abowd, G. D. (1999a). Classroom 2000: an experiment with the instrumentation of a living educational environment. *IBM Systems Journal*, 38(4):508–530.
- Abowd, G. D. (1999b). Software Engineering Issues for Ubiquitous Computing. In *Proceedings of the 21st International Conference on Software Engineering*, pages 75–84. Los Angeles, USA.
- Abowd, G. D. and Mynatt, E. D. (2000). Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):29–58.
- Abowd, G. D., Mynatt, E. D., and Rodden, T. (2002). The Human Experience. *Pervasive Computing*, 1(1):48–57.
- Ark, W. S. and Selker, T. (1999). A Look at Human Interaction with Pervasive Computers. *IBM Systems Journal*, 38(4):504–507.
- Arruda Jr, C. R. E., Bulcão Neto, R., and Pimentel, M. G. C. (2003). Open context-aware storage as a Web Service. In *1st International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 03)*. To Appear.
- Arruda Jr, C. R. E. and Pimentel, M. G. C. (2001). Projeto e Implementação de um Sistema Colaborativo de Edição. *Revista Eletrônica de Iniciação Científica da Sociedade Brasileira de Computação (REIC)*, 1(2). In Portuguese.
- Arruda Jr, C. R. E., Pimentel, M. G. C., and Izeki, C. A. (2002). Coteia: Uma ferramenta colaborativa de edição baseada na web. Workshop de Ferramentas e Aplicações - SBMIDIA.

- Banavar, G. and Bernstein, A. (2002). Software infrastructure and design challenges for ubiquitous computing applications. *Communications of the ACM*, 45(12):92–96.
- Barger, D., Gupta, A., Sanocki, E., and Grudin, J. (1999). Annotations for streaming video on the Web: System design and usage studies. In *Proceedings of the World Wide Web Conference*, pages 1139–1153.
- Berners-Lee, T., H., J., and Lassila, O. (2001). The Semantic Web. On-line in World Wide Web. <http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html>.
- Brotherton, J. A. and Abowd, G. D. (1998). Rooms take note: Room takes notes! In *Spring AAAI Symposium on Intelligent Environments*, pages 23–31.
- Brown, M. (1996a). Supporting user mobility. In *IFIP World Conference on Mobile Communications*, pages 69–77.
- Brown, P. J. (1996b). The stick-e document: A framework for creating context-aware applications. In *Origination, Dissemination, and Design (EP-ODD)*, pages 1–13.
- Brown, P. J. (1998). Triggering information by context. *Personal Technologies*, 2(1):1–9.
- Brown, P. J., Bovey, J. D., and Chen, X. (1997). Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64.
- Bulcão Neto, R. (2003). Modelagem e Construção de Aplicações de Computação Ubíqua via Web. Pedido de bolsa de doutorado solicitado à FAPESP (Em julgamento). Disponível em <http://coweb.icmc.usp.br/coweb/mostra.php?ident=5.10.1>.
- Burrell, J., Gay, G. K., Kubo, K., and Farina, N. (2002). Context-Aware Computing: A Test Case. In *Proceedings of the International Conference on Ubiquitous Computing*, pages 1–15.
- Cattelan, R. G., Andrade, A. R., Rocha, C. F. P., and Pimentel, M. G. C. (2003). iClass: um sistema para captura e acesso de sessões em ambiente educacional. *Revista Eletrônica de Iniciação Científica da Sociedade Brasileira de Computação (REIC)*, 3(1).

- Cooperstock, J. R., Tanikoshi, K., Beirne, G., Narine, T., and Buxton, W. A. S. (1995). Evolution of a reactive environment. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 170–177. ACM Press/Addison-Wesley Publishing Co.
- Dey, A. K. (2000). *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology.
- Dey, A. K. and Abowd, G. D. (1999). Towards a Better Understanding of Context and Context-awareness. Gvu technical report git-gvu-99-22, College of Computing, Georgia Institute of Technology.
- Dey, A. K., Abowd, G. D., Pinkerton, M., and Wood, A. (1997). Cyberdesk: A framework for providing self-integrating ubiquitous software services. In *ACM Symposium on User Interface Software and Technology*, pages 75–76.
- Eleutério, P. M. S. and Pimentel, M. G. C. (2001). Geração Sob Demanda de Hiperdocumentos Multimídia a partir de Informação Capturada em Reuniões Distribuídas. *Anais do I Workshop de Teses e Dissertações do SBMídia*, pages 193–186.
- Elrod, S., Hall, G., Costanza, R., Dixon, M., and Rivières, J. D. (1993). Responsive office environments. *Communications of the ACM*, 36(7):84–85.
- Erickson, T. (2002). Some problems with the notion of context-aware computing. *Communications of the ACM*, 45(2):102–104.
- Fickas, S., Kortuem, G., and Segall, Z. (1997). Software organization for dynamic and adaptable wearable systems. In *International Symposium on Wearable Computers*, pages 56–64.
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). RFC 2616: Hypertext Transfer Protocol - http/1.1. <http://www.ietf.org/rfc/rfc2616.txt>.
- Franklin, D. and Flachsbar, J. (1998). All gadget and no representation makes jack a dull environment. In *Proceedings of AAAI 1998 Spring Symposium on Intelligent Environments*, pages 155–160. AAAI TR SS-98-02.
- Garlan, D., Siewiorek, D., Smailagic, A., and Steenkiste, P. (2002). Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31.

- Godoy, R. P. (2003). Context Daemon: um serviço de notificação de informações de contexto. Monografia de Qualificação de Mestrado submetida à CPG/ICMC/USP. (45 páginas). Disponível em <http://coweb.icmc.usp.br/coweb/mostra.php?ident=5.10>.
- Gudgin, M., Hadley, M., Moreau, J. J., and Nielsen, H. F. (2001). Simple Object Access Protocol (SOAP) version 1.2 - W3C Working Draft. On-line in World Wide Web. Available online in <http://www.w3.org/TR/soap12/>.
- Harrison, B. L., Fishkin, K. P., Gujar, A., Mochon, C., and Want, R. (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*, pages 17–24. ACM Press/Addison-Wesley Publishing Co.
- Hess, C. K., Román, M., and Campbell, R. H. (2002). Building Applications for Ubiquitous Computing Environments. In *Proceedings of the International Conference on Pervasive Computing*, pages 16–29.
- Hong, J. I. and Landay, J. A. (2001). An Infrastructure Approach to Context-Aware Computing. *Human-Computer Interaction*, 16(2-3).
- Hull, R., Neaves, P., , and Bedford-Roberts, J. (1997). Towards situated computing. In *First International Symposium on Wearable Computers*, pages 146–153.
- Ishii, H. and Ullmer, B. (1997). Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 234–241.
- Johanson, B. and Fox, A. (2002). The Event Heap: An Coordination Infrastructure for Interactive Workspaces. In *Proceedings of the IEEE Workshop Mobile Computer Systems and Applications (WMCSA 2002)*, pages 83–93.
- Johnson, R. E. (1992). Documenting frameworks using patterns. In *Proceedings on Object-oriented programming systems, languages, and applications*, pages 63–76. ACM Press.
- Kleijnen, S. and Raju, S. (2003). An Open Web Services Architecture. *Queue*, 1(1):38–46.
- Lyytinen, K. and Yoo, Y. (2002). Introduction. Special Issue: Issues and challenges in ubiquitous computing. *Communications of the ACM*, 45(12):62–65.

- Martin, J., Arsanjani, A., Tarr, P., and Hailpern, B. (2003). Web Services: Promises and Compromises. *Queue*, 1(1):48–58.
- Microsystems, S. (2002a). Java Message Service. Available at <http://java.sun.com/products/jms/>.
- Microsystems, S. (2002b). *Java Remote Method Invocation Specification*. Sun Microsystems. version 1.4.
- Minneman, S., Harrison, S., Janssen, B., Kurtenbach, G., Moran, T., Smith, I., and van Melle, B. (1995). A confederation of tools for capturing and accessing collaborative activity. In *Proceedings of the third ACM International Conference on Multimedia*, pages 523–534. ACM Press.
- Mukhopadhyay, S. and Smith, B. (1999). Passive capture and structuring of lectures. In *ACM Multimedia*, pages 477–487.
- Narayanan, S. and McIlraith, S. A. (2002). Simulation, Verification and Automated Composition of Web Services. In *Proceedings of the eleventh international conference on World Wide Web*, pages 77–88. ACM Press.
- Norman, D. A. (1998). The Invisible Computer. *MIT Press*, pages 62–63.
- Object Management Group (2001). CORBA/IIOP Specification. OMG publications. http://www.omg.org/technology/documents/formal/corba_iiop.htm.
- Pascoe, J. (1998). Adding generic contextual capabilities to wearable computers. In *International Symposium on Wearable Computers*, pages 92–99.
- Pedersen, E. R., McCall, K., Moran, T. P., and Halasz, F. (1993). Tivoli: an electronic whiteboard for informal workgroup meetings. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 391–398. ACM Press.
- Pimentel, M. G. C., Abowd, G. D., and Ishiguro, Y. (2000). Linking by interacting: a paradigm for authoring hypertext. In *ACM Conference on Hypertext*, pages 39–48.
- Pimentel, M. G. C., Fortes, R. P. M., and Moreira, E. S. (2001a). Development and understanding of automated capture environments to support long-term use: Proposal 1999. Relatório Técnico 149, ICMC-USP.

- Pimentel, M. G. C., Izeki, C. A., and Arruda Jr, C. R. E. (2001b). An XML-based Infrastructure Supporting Collaborative Annotations as First-class Hyperdocuments. In *Proceedings of VII Brazilian Symposium of Multimedia and Hypermedia Systems*, pages 173–186.
- Pimentel, M. G. C., Kerimbaev, Y. I. B., Abowd, G. D., and Guzdial, M. (2001c). Supporting Long-term Educational Activities Through Dynamic Web Interfaces. *Interacting With Computers Journal*, 13:353–374.
- Postel, J. (1980). RFC 768: User Datagram Protocol. <http://www.ietf.org/rfc/rfc768.txt>.
- Postel, J. (1981). RFC 793: Transmission Control Protocol. <http://www.ietf.org/rfc/rfc793.txt>.
- Rhodes, B. J. (1997). The wearable remembrance agent: A system for augmented memory. In *Proceedings of The First International Symposium on Wearable Computers*, pages 123–128, Cambridge, Mass., USA.
- Rodden, T., Cheverst, K., Davies, N., and Dix, A. (1998). Exploiting context in hci design for mobile systems. In *Workshop on Human Computer Interaction with Mobile Devices*.
- Rose, M. (2001). RFC 3080: The Blocks Extensible Exchange Protocol Core. <http://www.ietf.org/rfc/rfc3080.txt>.
- Ryan, N. S., Pascoe, J., and Morse, D. R. (1997). Enhanced reality fieldwork: the context-aware archaeological assistant. In Gaffney, V., van Leusen, M., and Exxon, S., editors, *Computer Applications in Archaeology*, British Archaeological Reports, Oxford. Tempus Reparatum.
- Salber, D., Dey, A., and Abowd, G. (1998). Ubiquitous computing: Defining an hci research agenda for an emerging interaction paradigm. Git-gvu-98-01, Georgia Institute of Technology.
- Salber, D., Dey, A. K., and Abowd, G. D. (1999). The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 434–441.
- Schilit, B. and Theimer, M. (1994). Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32.

- Schilit, B. N., Adams, N., Gold, R., Tso, M., and Want, R. (1993). The ParcTab Mobile Computing System. In *Proceedings of the Workshop on Workstation Operating Systems*, pages 34–39.
- Schilit, B. N., Adams, N., and Want, R. (1994). Context-aware computing applications. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90.
- Schilit, B. N., Golovchinsky, G., and Price, M. N. (1998). Beyond paper: Supporting active reading with free form digital ink annotations. In *Conference on Human Factors in Computing Systems*, pages 249–256.
- Schmandt, C. and Arons, B. (1984). Phoneslave: a graphical telecommunications system. In *Society for Information Display International Symposium*, volume 25, pages 146–149.
- Stal, M. (2002). Web services: beyond component-based computing. *Communications of the ACM*, 45(10):71–76.
- Starner, T., Schiele, B., and Pentland, A. (1998). Visual Contextual Awareness in Wearable Computing. In *International Symposium on Wearable Computers*, pages 50–57.
- Streitz, N. A., G., J., Haake, J. M., and Hol, J. (1994). Dolphin: integrated meeting support across local and remote desktop environments and liveboards. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 345–358. ACM Press.
- Treese, W. (2002). XML, web services, and XML. *netWorker*, 6(3):9–12.
- Truong, K. N., Abowd, G. D., and Brotherton, J. A. (2001). Who, What, When, Where, How: Design Issues of Capture & Access Applications. In *Proceedings of the International Conference on Ubiquitous Computing*, pages 209–224.
- W3C (1999a). Namespaces in XML. Available online in <http://www.w3.org/TR/REC-xml-names/>.
- W3C (1999b). XML Path Language (XPath). Available online in <http://www.w3.org/TR/xpath>.
- W3C (2001). XML Schema, W3C Recommendation, 2 may 2001. Available online in <http://www.w3.org/TR/xmlschema-0/>.

- W3C (2002). Extensible Markup Language (XML). Available online in <http://www.w3.org/TR/xml11/>.
- W3C (2002a). Web Services Activity. Available online in <http://www.w3.org/2002/ws>.
- W3C (2002b). Web Services Architecture. Available online in <http://www.w3.org/TR/ws-arch/>.
- W3C (2002c). XML Signature Syntax and Processing Recommendation, 12 february 2002. Available online in <http://www.w3.org/TR/xmlsig-core/>.
- W3C (2002). XSL Transformations (XSLT). Available online in <http://www.w3.org/TR/xslt>.
- W3C (2003). Web Services Description Language (WSDL). Available online in <http://www.w3.org/TR/wsdl12>.
- Wactlar, H., Kanade, T., Smith, M., and Stevens, S. (1996). Intelligent access to digital video: Informedia project. *Pervasive Computing*, 29(5):46-52.
- Want, R., Hopper, A., Falco, V., and Gibbons, J. (1992). The active badge location system. *ACM Transactions on Information Systems*, 10(1):91-102.
- Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, 265(3):94-104.
- Weiser, M. (1993). Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 6(7):75-84.
- Weiser, M. (1994). The World is not a Desktop. *Interactions*, 1(1):7-8.
- Weiser, M. and Brown, J. S. (1996). The Coming Age of Calm Technology. Technical report, Xerox PARC.
- White, S. A., Gupta, A., Grudin, J., Chesley, H., Kimberly, G., and Sanocki, E. (2000). Evolving use of a system for education at a distance. In *33rd Hawaii International Conference on System Sciences*.
- Witbrock, M. J. and Hauptmann, A. G. (1998). Speech recognition for a digital video library. *Journal of the American Society of Information Science*, 49(7):619-632.

Yellin, D. M. (2001). Stuck in the middle: Challenges and trends in optimizing middleware. In *Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems*, pages 175–180. ACM Press.