

CARLOS EDUARDO GUSSO TOSIN

**UMA INFRA-ESTRUTURA REFLEXIVA PARA
APLICAÇÕES DEPENDENTES DE CONTEXTO**

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

CURITIBA

2009

CARLOS EDUARDO GUSSO TOSIN

**UMA INFRA-ESTRUTURA REFLEXIVA PARA
APLICAÇÕES DEPENDENTES DE CONTEXTO**

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

Área de Concentração: *Sistemas Distribuídos*

Orientador: Prof. Dr. Luiz Lima Jr.

CURITIBA

2009

Tosin, Carlos Eduardo Gusso

Uma infra-estrutura reflexiva para aplicações dependentes de contexto. Curitiba, 2009. 94p.

Uma Infra-Estrutura Reflexiva para Aplicações Dependentes de Contexto – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática.

1. Dependência de contexto 2. Computação ubíqua 3. Computação pervasiva 4. Infra-estrutura reflexiva. I. Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática

À minha família

Agradecimentos

Inicialmente, agradeço ao meu orientador, Luiz Lima Jr., por todas as horas de apoio e dedicação durante todo o período da elaboração da proposta e escrita de artigos e da dissertação. Agradeço também aos meus pais, Haroldo e Denise, que sempre investiram na minha educação e acreditaram que eu poderia conquistar meus sonhos. Também agradeço à minha esposa Damaris, que esteve sempre ao meu lado e soube compreender as “correrias” do meu dia-a-dia, tentando conciliar trabalho e Mestrado. Para finalizar, agradeço aos meus amigos, André e Omar, pelas discussões malucas estilo *brainstorming*, as quais ajudaram em algumas das idéias utilizadas no desenvolvimento deste trabalho.

Sumário

Lista de Figuras	x
Lista de Tabelas	xii
Lista de Símbolos	xiii
Lista de Abreviaturas	xiv
Resumo	xv
Abstract	xvi
Capítulo 1	18
Introdução	18
Capítulo 2	22
Estado da Arte	22
2.1. Contexto	22
2.1.1. Definição	22
2.1.2. Classificação	23
2.1.3. Qualidade de Contexto	25
2.2. Aplicações Dependentes de Contexto	26
2.2.1. Definição	26

2.2.2.	Classificação.....	27
2.2.3.	Exemplos	28
2.3.	Infra-estruturas para Aplicações Dependentes de Contexto	30
2.3.1.	Requisitos Desejáveis ao Trabalhar com Informações de Contexto	30
2.4.	Ontologias	33
2.5.	Plataforma CORBA	33
2.5.1.	Serviço de Notificação.....	35
2.5.2.	Serviço de <i>Trading</i>	37
2.6.	Reflexividade	38
2.7.	Conclusão.....	38
Capítulo 3		40
Trabalhos Relacionados		40
3.1.	<i>Context Toolkit</i>	40
3.2.	Infra-estrutura proposta por Sinderen <i>et al.</i> [SIN06]	42
3.3.	SOCAM	43
3.4.	MoCA	44
3.5.	Conclusão.....	45
Capítulo 4		46
Arquitetura do CxFramework		46
4.1.	Modo de Utilização.....	46
4.2.	Arquitetura Interna.....	47
4.2.1.	<i>Sensor Management Service</i>	48

4.2.2.	<i>Registry Service</i>	49
4.2.3.	<i>Context Reasoning Service</i>	50
4.2.4.	<i>Context Notification Service</i>	51
4.2.5.	<i>Proxy</i>	51
4.2.6.	<i>Security Service</i>	51
4.2.7.	<i>Application Sensor Adapter</i>	51
4.2.8.	Papéis Existentes na Infra-Estrutura.....	52
4.3.	Reflexividade e Federação de Contextos.....	53
4.4.	Conclusão.....	53
Capítulo 5		56
Cenário de Aplicação.....		56
5.1.	Problema	56
5.2.	Detalhamento do Cenário	57
5.3.	Detalhamento da Solução	59
5.3.1.	Gerenciador de Promoções.....	59
5.3.2.	Gerenciador de Filas.....	59
5.3.3.	Localizador de Carrinhos.....	60
5.3.4.	Carrinhos de Compras	60
5.4.	Ontologia e Regras de Inferência.....	61
5.5.	Utilizando Federação de Contexto.....	64
5.6.	Conclusão.....	66

Capítulo 6	68
Aspectos de Implementação e Avaliação	68
6.1. Aspectos de Implementação	68
6.2. Coleta de Lixo.....	73
6.3. Escalabilidade	75
6.4. Avaliação de Desempenho.....	76
6.5. Conclusão.....	80
Capítulo 7	82
Conclusão e Trabalhos Futuros.....	82
Referências Bibliográficas	86
Apêndice A	90
Método de Obtenção das Equações 6.1 e 6.2.....	90

Lista de Figuras

Figura 2.1: Estrutura de um framework para aplicações dependentes de contexto.....	32
Figura 2.2: Visão geral da plataforma CORBA [HEN99].....	34
Figura 2.3: O modelo <i>push</i> de entrega de eventos [HEN99].....	36
Figura 2.4: O modelo <i>pull</i> de entrega de eventos [HEN99]	36
Figura 3.1: Componentes da infra-estrutura <i>Context Toolkit</i> [DEY01]	40
Figura 3.2: Arquitetura da infra-estrutura proposta por Sinderen <i>et al.</i> [SIN06]	42
Figura 3.3: Arquitetura do SOCAM [GU05].....	43
Figura 3.4: Arquitetura da MoCA [VIT06]	44
Figura 4.1: O CxFramework do ponto de vista das aplicações	46
Figura 4.2: Estrutura do CxFramework	48
Figura 4.3: Mapeamento entre um <i>framework</i> conceitual e o CxFramework	54
Figura 5.1: Representação gráfica do cenário	58
Figura 5.2: Diversas aplicações dependentes de contexto conectadas ao CxFramework	59
Figura 5.3: Diagrama exemplificando o estado do contexto em determinado momento	62
Figura 5.4: Exemplo de ontologia expressa em formato OWL sobre XML	63
Figura 5.5: Esquematização do funcionamento da federação de contextos	65
Figura 6.1: Tarefas a serem realizadas para colocar o CxFramework em funcionamento.....	69
Figura 6.2: Arquitetura do <i>Context Notification Service</i>	71
Figura 6.3: Cenário 1, onde o número de aplicações varia.....	77
Figura 6.4: Cenário 2, onde o número de <i>sensor adapters</i> varia.....	78
Figura 6.5: Gráfico de medição de tempo em diferentes cenários	79
Figura A.1: Relação entre a quantidade de <i>sensor adapters</i> e o tempo gasto na inferência	91
Figura A.2: Relação entre a quantidade de <i>sensor adapters</i> e a memória gasta	92

Lista de Tabelas

Tabela 6.1: Arquivos de configuração do CxFramework.....	69
Tabela 6.2: Diferentes cenários de execução.....	79
Tabela A.1: Resultado das medições na inferência de uma nova ontologia.....	90

Lista de Símbolos

N	Número de <i>Sensor Adapters</i>
t	Tempo para a inferência de uma ontologia
m	Memória consumida pela ontologia

Lista de Abreviaturas

API	<i>Application Programming Interface</i>
CORBA	<i>Common Object Request Broker Architecture</i>
ECA	<i>Event-Condition-Action</i>
FIFO	<i>First In, First Out</i>
IDL	<i>Interface Definition Language</i>
OMG	<i>Object Management Group</i>
ORB	<i>Object Request Broker</i>
PDA	<i>Personal Digital Assistant</i>
QoC	<i>Quality of Context</i>
QoS	<i>Quality of Service</i>
SQL	<i>Structured Query Language</i>
TTL	<i>Time-to-Live</i>
XML	<i>Extensible Markup Language</i>

Resumo

Aplicações dependentes de contexto utilizam informações do ambiente que as cercam para adaptarem o seu comportamento. Este trabalho propõe um infra-estrutura reflexiva, com baixo acoplamento e orientada a eventos, chamada CxFramework, para auxiliar no desenvolvimento desta categoria complexa de aplicações. O CxFramework é composto de diversos tipos de serviços. A modelagem das informações de contexto é feita com o uso de ontologias, que permitem que contextos de mais alto nível sejam inferidos. As aplicações que utilizam a infra-estrutura proposta podem procurar ativamente por informações de contexto (mecanismo *pull*) ou podem ser notificadas a respeito das mudanças de contexto (mecanismo *push*), a fim de adaptarem o seu comportamento. Neste último caso, canais de eventos são utilizados para desacoplar a aplicação da infra-estrutura. Os detalhes da aquisição do contexto e seu processamento são transparentes para o desenvolvedor.

Palavras-Chave: Dependência de contexto, computação ubíqua, computação pervasiva, infra-estrutura reflexiva.

Abstract

Context-aware applications use information from their surrounding environment to adapt their behavior. This paper proposes a reflexive, loose-coupled, event-based infrastructure named CxFramework to aid the development of such complex applications. CxFramework is composed of several kinds of services. Context information is modeled using ontologies from which higher-level context information can be inferred. Applications using the infrastructure proposed can either actively look for context information (pull mechanism) or can be notified of context changes (push mechanism) in order to adapt their behavior. In the later, event channels are used to decouple the application and the infrastructure. The details of context acquisition and processing are transparent to the developer.

Keywords: Context awareness, ubiquitous computing, pervasive computing, reflexive infrastructure.

Capítulo 1

Introdução

Os seres humanos são capazes de utilizar diversos tipos de informação implícita para enriquecer sua comunicação e adaptar o seu comportamento. Esta informação implícita pode ser chamada de **contexto** da comunicação. As pessoas utilizam contexto todos os dias, mesmo sem estarem cientes disto. Como exemplo, pessoas falam mais alto em lugares com bastante barulho para que outras pessoas possam escutar. Já quando as pessoas estão em uma reunião, elas sussurram para não atrapalhar quem está ao redor.

A dependência de contexto também possui um papel importante na computação. Enquanto humanos têm uma capacidade natural de utilizar contextos enquanto se comunicam, é necessário definir, estruturar e organizar as informações de contextos para enriquecer e otimizar as interações homem-máquina e entre computadores.

Os exemplos de aplicações de contextos são vários. As mais simples normalmente estão relacionadas à apresentação de informações a usuários conforme eles entram em determinado contexto. Aplicações desta natureza podem ser usadas, por exemplo, em museus e lojas (para apresentarem informações pertinentes a usuários durante a sua locomoção nestes ambientes). Já aplicações de contexto mais complexas modificam aspectos do ambiente, reagindo ao contexto onde se encontram. É o que acontece, por exemplo, em uma aplicação que detecta a presença de pessoas em uma sala para poder controlar a intensidade da luz.

Alguns destes sistemas dependentes de contexto foram implementados com sucesso na prática, como o *Shopping Assistant* [AST94] e o *Conference Assistant* [DEY99]. O problema é

que aplicações desta natureza são difíceis de serem arquitetadas, desenvolvidas e mantidas devido ao alto grau de complexidade para lidar com sensores distribuídos, informações de contexto de alto nível, interoperabilidade entre diferentes sistemas e assim por diante. Conseqüentemente, os desenvolvedores destas aplicações precisam resolver todos estes problemas antes de resolver problemas relacionados à lógica de funcionamento da aplicação propriamente dita.

Para resolver este problema, toda a complexidade relacionada com a aquisição das informações de contexto e posterior processamento e notificação destas informações devem ser encapsuladas em uma infra-estrutura, a qual se torna responsável por estas atividades e pode também compartilhar as informações entre diferentes aplicações. Desta forma, os desenvolvedores das aplicações podem se preocupar apenas com problemas relativos à aplicação.

A infra-estrutura deve ser flexível o bastante para suportar uma grande variedade de contextos e também suportar processamento de contextos e inferência (muitas aplicações não estão interessadas em informações cruas vindas dos sensores). Neste aspecto, ontologias e inferências sobre ontologias cumprem um papel bastante importante. Uma boa infra-estrutura também deve possuir boas formas de se comunicar com as aplicações conectadas a ela. Deve ser possível às aplicações perguntarem à infra-estrutura a respeito de informações de contexto, utilizando o modo de comunicação *pull*. As aplicações também devem ser capazes de registrar o seu interesse em determinados eventos de contexto que a infra-estrutura é capaz de gerar. Logo, a infra-estrutura deve possuir um serviço de notificação o menos acoplado possível às aplicações, o qual suporta o modo de comunicação *push*.

O objetivo deste trabalho é propor uma arquitetura genérica para a integração de informações de contexto em uma plataforma distribuída através da definição de um modelo de suporte baseado em um serviço de inferência e fazer com que estas informações sejam disponibilizadas às aplicações que utilizam contexto. Este processo visa simplificar o desenvolvimento de aplicações, transferindo grande parte da complexidade de detecção de mudanças de contexto a uma infra-estrutura genérica, chamada CxFramework, a qual foi criada e desenvolvida como proposta deste trabalho.

Embora muitas infra-estruturas com o propósito de prover informações de contexto já existam atualmente, o que difere o CxFramework destas outras infra-estruturas é a

implementação da reflexão e a sua ênfase no baixo acoplamento entre a infra-estrutura e as aplicações. A reflexividade significa que as aplicações dependentes de contexto conectadas à plataforma também mudam o contexto corrente (da mesma forma que qualquer outro sensor faria) e outras aplicações podem ser avisadas destas mudanças caso estejam interessadas. Como exemplo, é possível imaginar uma pessoa **P** de posse de um PDA (*Personal Digital Assistant*) que fornece sua identificação e localização. Quando **P** entra em um supermercado, ela altera o contexto local porque outras pessoas, e até o sistema do supermercado, podem interagir com ela. Como **P** é capaz de dizer quem é e onde está, outras pessoas podem ser notificadas a respeito da sua localização e seus interesses em produtos, por exemplo. Além disso, **P** também poderia ser notificada pelo supermercado a respeito de produtos e preços.

A capacidade da reflexividade abre caminho para outra funcionalidade, também suportada pelo CxFramework: a federação de contextos. Ambientes diferentes podem ter contextos diferentes e mudanças em um ambiente podem possivelmente afetar outro ambiente. Através da federação de contextos, diversas instâncias do CxFramework podem existir, cada uma representando um contexto, ao mesmo tempo em que trocam informações a respeito das mudanças de contexto nos diversos ambientes.

O restante desta dissertação está assim organizado. No Capítulo 2, são abordados temas necessários para compreender a arquitetura proposta. São eles: contexto e dependência de contexto, ontologias, plataforma CORBA e reflexividade. No Capítulo 3, alguns trabalhos alinhados com a proposta são citados e comentados. No Capítulo 4, a arquitetura proposta é apresentada sob um aspecto conceitual. No Capítulo 5, é apresentado um cenário onde a arquitetura pode ser utilizada para resolver um problema real. No Capítulo 6 a arquitetura proposta é apresentada sob a perspectiva de implementação, juntamente com discussões a respeito de coleta de lixo, escalabilidade e avaliação de performance. O protótipo do CxFramework foi implementado utilizando CORBA (*Common Object Request Broker*) como plataforma distribuída devido a sua grande adoção, estabilidade, maturidade e conjunto de recursos adequados (serviços). Desta forma, o CxFramework integra informações de contexto à plataforma CORBA, muito embora o modelo arquitetural definido possa ser incorporado virtualmente em qualquer outra plataforma de comunicação, com maior ou menor dificuldade. Por fim, conclusões do trabalho e atividades futuras possíveis são apresentadas.

Capítulo 2

Estado da Arte

Este capítulo apresenta um resumo dos conceitos básicos e tecnologias de suporte do trabalho proposto. Assim, serão tratadas inicialmente as definições de contexto, de aplicações dependentes de contexto e de infra-estruturas para aplicações dependentes de contexto, uma vez que esses tópicos estão diretamente relacionados com a essência deste trabalho. Na seqüência, serão abordadas as ontologias, que têm se mostrado uma ferramenta útil para representação e classificação de contextos, seguido de CORBA, já que ele é o *middleware* de sistemas distribuídos utilizado pela infra-estrutura proposta, com destaque para os serviços de notificação e *trading*, diretamente utilizados pela infra-estrutura. Por fim, será abordada a reflexividade.

2.1. Contexto

2.1.1. Definição

A definição de contexto do ponto de vista computacional tem sido objeto de discussão nos últimos anos. Diversos autores possuem diversas definições que julgam como sendo ideais. Infelizmente esta falta de consenso é prejudicial, uma vez que a falta de uma definição consistente cria problemas no momento de transferir a noção de contexto para o âmbito computacional (para implementar aplicações que utilizem contexto é necessário que a noção do que é o contexto seja muito clara). A tendência é que, com o passar do tempo, o maior entendimento sobre o tema faça com que exista uma convergência sobre uma definição comum.

Schilit e Theimer [SCH94a], no trabalho que introduziu o termo *dependente de contexto*, definem contexto como sendo o local, as identidades de pessoas e objetos que estão próximos e as alterações que ocorrem nesses objetos. Numa definição mais geral, Schilit *et al.* [SCH94b] afirmam que os aspectos importantes do contexto são: onde você está, com quem você está e quais recursos estão próximos. Já Dey e Abowd [DEY00] definem contexto como sendo qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto considerado relevante na interação entre um usuário e uma aplicação, incluindo o próprio usuário e a própria aplicação.

Neste trabalho será adotada a definição de Dey e Abowd, por se tratar da definição mais genérica e consistente. A definição de Schilit e Theimer é formulada através de exemplos (pessoas, locais, etc.). Dessa forma, quando não é possível determinar se uma informação não mencionada na definição é contexto ou não, não é possível utilizar a definição de contexto para resolver o dilema [DEY00]. Já a definição de Schilit *et al.* é muito específica, já que os aspectos importantes do contexto dependem de cada situação. Em alguns casos, por exemplo, o ambiente físico pode ser totalmente relevante; já em outros casos pode não fazer diferença [DEY00].

2.1.2. Classificação

Dey e Abowd [DEY00] classificam o contexto em **primário** e **secundário**. O tipo **primário** é composto pelas informações que caracterizam a situação de uma entidade em particular. São elas:

- **Localização:** localização da entidade num espaço físico.
- **Identidade:** identificação da entidade.
- **Tempo:** informações relativas ao tempo, como hora do dia, dia da semana, etc.
- **Atividade:** representação do que está acontecendo com a entidade no momento.

O tipo **secundário** é composto por todos os outros tipos de contexto que não são considerados primários, uma vez que contextos secundários podem ser derivados dos contextos primários. Por exemplo, através do conhecimento da identidade de uma entidade e da sua localização, é possível determinar quais objetos estão próximos a ela. Ou então a partir da

informação da identidade, da localização e da hora do dia (tempo), é possível saber se determinada pessoa está participando de uma reunião.

Raz *et al.* [RAZ06] também apresenta uma forma de classificação de contextos baseada em diversos aspectos:

- **Tipos de contexto**
 - **Usuário:** informação a respeito do que está à volta do usuário (localização, mobilidade, dispositivos disponíveis, etc.), como também suas próprias características (identidade, preferências, histórico, etc.).
 - **Dispositivo:** informações do dispositivo, como endereço IP, informações de domínio, nível de bateria, etc.
 - **Rede:** informação das características da rede, como nível de segurança, QoS, recursos de rede, etc.
 - **Fluxo:** informação relativa ao fluxo na rede, como latência, taxa de erro, confiabilidade, etc.
- **Persistência**
 - **Permanente:** contexto que não muda com o tempo, mantendo-se sempre constante enquanto existir (ex: nome, número de identidade, etc.).
 - **Temporário:** o contexto muda com o passar do tempo (ex: posição, saúde, etc.).
- **Evolução**
 - **Estático:** contextos que não mudam muito rapidamente (ex: temperatura no decorrer do dia).
 - **Dinâmico:** contextos que sofrem alterações muito rapidamente (ex: posição de uma pessoa que está dirigindo um carro).
- **Meio**
 - **Físico:** informações que podem ser obtidas através de sensores (ex: localização, temperatura, nível sonoro, etc.).
 - **Intangível:** informações que não podem ser obtidas através de sensores, sendo necessária intervenção manual do usuário (ex: comida preferida, esporte predileto, etc.).

- **Relevância a um serviço ou aplicação**
 - **Necessário:** a informação do contexto precisa ser fornecida para que determinado serviço funcione corretamente
 - **Acessório:** a informação do contexto pode ser útil para que o serviço possa ser realizado de uma melhor forma, porém a informação não é obrigatória.
- **Situação temporal**
 - **Passado:** representa o contexto em algum momento do passado. Isto é possível através da manutenção de um histórico de contextos.
 - **Presente:** representa o contexto no exato momento.
 - **Futuro:** representa uma previsão futura de contexto. Esta previsão é feita através da análise dos contextos do presente e do passado.
- **Modos de Interação**
 - **Push:** a fonte do contexto informa periodicamente a situação atual do contexto a quem precisa processar a informação.
 - **Pull:** o responsável pelo processamento da informação do contexto busca a informação na fonte do contexto quando achar necessário.

2.1.3. Qualidade de Contexto

Associada às informações de contexto, existe a qualidade de contexto (QoC). Segundo Buchholz [BUC03], qualidade de contexto é qualquer informação que descreve a qualidade da informação que é utilizada como informação de contexto. Conseqüentemente, a qualidade de contexto se refere à informação propriamente dita, e não ao processo ou *hardware* que gerou a informação.

Os parâmetros mais importantes relativos à QoC são os seguintes:

- **Precisão:** mede a relação entre a informação de contexto e a realidade. Um receptor de GPS, por exemplo, é capaz de fornecer informação de localização com uma precisão de 4 metros. Já a localização baseada em uma rede de telefone celular como a GSM possui uma precisão de 500 metros.

- **Probabilidade de certeza:** é relativa à probabilidade de que determinada informação de contexto esteja correta. Como exemplo, é possível citar um sensor de temperatura, que pode ter falhas internas e começar a fornecer temperaturas erradas.
- **Confiabilidade:** é semelhante ao parâmetro de probabilidade de certeza. No entanto, a confiabilidade está relacionada ao emissor da informação de contexto. Isto é, mesmo que o emissor envie uma informação com 100% de probabilidade de certeza, o receptor dessa informação pode se basear em informações recebidas desse emissor no passado para saber se a sua fonte de informações é realmente de confiança.
- **Granularidade:** é relativa à granularidade da informação. Um sensor de temperatura localizado em uma casa, por exemplo, indica uma temperatura média. Mas a temperatura indicada pode não ser válida para todos os cômodos da casa. Devido ao número reduzido de sensores de temperatura, a fonte da informação do contexto não é capaz de fornecer a informação com uma maior granularidade.
- **Tempo de vida:** é relativo à idade da informação de contexto. Normalmente o controle do tempo de vida é feito através de um *timestamp* atrelado à informação.

2.2. Aplicações Dependentes de Contexto

2.2.1. Definição

A dependência de contexto nas aplicações e serviços computacionais está relacionada à capacidade que determinada aplicação tem em utilizar informações de contexto no seu funcionamento. Dey [DEY00] define um sistema dependente de contexto como sendo um sistema que usa o contexto para prover informações relevantes e/ou serviços para o usuário, onde a relevância depende da tarefa sendo executada pelo usuário. Essa afirmação é mais genérica do que a primeira definição de aplicações dependentes de contexto, feita por Schilit e Theimer [SCH94a]. Eles afirmavam que aplicações dependentes de contexto eram aplicações que tinham a capacidade de se adaptar de acordo com sua localização e objetos próximos, assim como as mudanças que ocorrem nesses objetos com o passar do tempo. Essa definição não é suficientemente genérica, uma vez que exclui do grupo as aplicações que apenas usam o contexto e não se adaptam a sua mudança.

2.2.2. Classificação

Dey [DEY00] propõe uma categorização de aplicações dependentes de contexto de acordo com as funcionalidades que elas podem apresentar. É dividida em três categorias:

- **Apresentação de informações e serviços ao usuário:** as aplicações obtêm informações automaticamente de acordo com o contexto. Por exemplo, uma lista de impressoras próximas ao usuário vai sendo atualizada enquanto o usuário vai caminhando pelo prédio.
- **Execução automática de um serviço:** as aplicações executam comandos automaticamente de acordo com o contexto. Por exemplo, quando o usuário entra na sala, a luz é acesa automaticamente.
- **Associação do contexto a determinado objeto para recuperação futura:** as aplicações associam informações do contexto do usuário a um objeto, com o objetivo de que a informação do contexto possa ser recuperada mais tarde. Por exemplo, o usuário cria um aviso virtual e o associa a uma mesa; outro usuário que, mais tarde, esteja próximo à mesa, é capaz de ter acesso ao conteúdo do aviso virtual.

Outra categorização de aplicações dependentes de contexto foi feita por Tuulari [TUU00]. De acordo com a sua divisão, existem aplicações que possuem internamente toda a lógica para obter e processar informações de contexto (dependência de contexto autocontida); e também existem aplicações que utilizam uma infra-estrutura externa para auxiliar na obtenção de informações de contexto (dependência de contexto baseada em infra-estrutura externa).

Chen [CHE00] também apresenta uma forma de classificar aplicações dependentes de contexto. De acordo com ele, as aplicações podem possuir:

- **Dependência de contexto ativa:** quando a aplicação se adapta automaticamente às mudanças de contexto, mudando o seu comportamento interno.
- **Dependência de contexto passiva:** quando a aplicação apresenta ao usuário informações novas ou atualizadas do contexto, ou então salva essas informações para que sejam utilizadas mais tarde.

2.2.3. Exemplos

Chen [CHE00] apresenta uma lista com algumas aplicações dependentes de contexto que foram implementadas com sucesso. Algumas delas são:

- ***Shopping Assistant***: [AST94] Esta aplicação guia os compradores dentro de uma loja, fornecendo informações relevantes, como detalhes de produtos, comparação de preços, auxílio na busca por produtos específicos, etc. A aplicação é capaz de obter a localização do cliente dentro da loja, o que possibilita a alteração do comportamento da aplicação.
- ***Conference Assistant***: [DEY99] Esta aplicação tem o objetivo de auxiliar pessoas que desejam participar de apresentações. Ela analisa o calendário e os assuntos das apresentações, localização e temas de interesse do usuário e sugere algumas apresentações a serem assistidas. Quando o usuário entra na sala de apresentação, o dispositivo móvel do usuário automaticamente mostra o nome do apresentador, o título da apresentação e outras informações relevantes. O usuário também tem acesso ao conteúdo da apresentação, comentários e questões feitas.
- ***Adaptive GSM Phone and PDA*** [SCH99]: Esta aplicação provoca algumas adaptações no telefone ou PDA do usuário de acordo com informações de contexto. No caso do PDA, o tamanho da fonte é alterado de acordo com a atividade do usuário (se ele estiver andando, uma fonte grande; se ele estiver parado, uma fonte pequena) e com as condições do ambiente (intensidade da luz, por exemplo). No caso do telefone, o modo e volume de toque do aparelho são selecionados automaticamente de acordo com o contexto do usuário (o telefone ajusta o volume do toque ou decide se vai tocar ou vibrar dependendo de onde se encontra: na mão do usuário, numa mesa, numa pasta, etc.).

Duran-Limon *et al.* [DUR04] cita mais alguns exemplos de aplicações dependentes de contexto. Elas envolvem os chamados *ambientes inteligentes*. Uma sala inteligente pode controlar a temperatura, tocar música, alterar o nível de luminosidade, tudo de acordo com as preferências do ocupante da sala. A hora do dia também pode ser levada em consideração. Um exemplo seria fazer café preto pela manhã e *cappuccino* à tarde. Caso outras pessoas estejam

presentes no ambiente, a sala inteligente tem a capacidade de analisar as preferências dessas pessoas e tomar decisões para tentar agradar a todos (temperatura ideal, quantidade de café, etc.).

Um outro exemplo de aplicação dependente de contexto citado por Duran-Limon *et al.* pode ser um sistema de controle de tráfego aéreo onde cada aeronave sabe a localização das aeronaves vizinhas e pode atuar para manter uma distância segura das mesmas, a fim de evitar colisões. Essa mesma idéia também pode ser aplicada aos carros nas ruas. Os carros podem se comunicar e cooperar para evitar colisões.

Em seu trabalho, Chen [CHE00] lista várias outras aplicações dependentes de contexto:

- **Call Forwarding:** a recepcionista sabe a localização de determinada pessoa no prédio e pode transferir a ligação para um ramal próximo dela.
- **Teleporting:** a aplicação que o usuário está usando é movida de um computador para outro conforme o usuário caminha pelo ambiente.
- **Mobisaic Web Browser:** permite a adição de informações de contexto a *links* de páginas *web*.
- **Cyberguide:** fornece informações a um turista de acordo com a sua localização atual.
- **People and Object Pager:** é capaz de enviar mensagens para uma pessoa que não têm um *pager*. A pessoa que possui um e está mais próxima do receptor da mensagem recebe a mensagem e o avisa.
- **Fieldwork:** auxilia na observação e coleta de dados em trabalhos de campo.
- **Office Assistant:** identifica visitantes que entram num escritório.
- **Location-aware Information Delivery:** permite a gravação de avisos em determinados locais, que são ouvidos pelas pessoas que passam por esses locais num outro momento.

Além dos exemplos citados, existem muitos outros. Na verdade, os exemplos de aplicações dependentes de contexto são praticamente ilimitados e podem ser aplicados em diversas áreas, uma vez que o uso de informações de contexto em aplicações abre um leque muito grande de possibilidades a serem exploradas.

2.3. Infra-estruturas para Aplicações Dependentes de Contexto

De acordo com Tuulari [TUU00], as aplicações dependentes de contexto podem implementar internamente a lógica para tratamento do contexto ou podem depender de infra-estruturas externas, que são responsáveis por tudo o que diz respeito ao contexto. As duas abordagens são possíveis, mas a primeira delas apresenta algumas desvantagens que devem ser consideradas. Uma delas é que, caso seja necessário adicionar outra fonte de informação de contexto, por exemplo, o código da aplicação tem que ser alterado. E a outra desvantagem, talvez a principal delas, é a de não permitir reusabilidade do código de tratamento do contexto. Se cada aplicação implementar internamente seu próprio método para trabalhar com informações de contexto, teoricamente existirão diversas aplicações realizando a mesma tarefa de uma maneira não padronizada. Uma solução melhor é abstrair o tratamento do contexto e removê-lo de dentro do código da aplicação, tornando-o reusável entre diversas aplicações de uma forma padronizada, através de uma infra-estrutura independente das aplicações que a utilizam.

2.3.1. Requisitos Desejáveis ao Trabalhar com Informações de Contexto

Após o entendimento da importância da existência de uma infra-estrutura externa para realizar o tratamento do contexto, é necessário compreender o que uma infra-estrutura dessa natureza deve prover. Dey [DEY01] enumerou algumas capacidades que uma infra-estrutura para aplicações dependentes de contexto deve possuir:

- **Separação de conceitos:** está relacionada com a separação entre a obtenção do contexto e seu uso. Essa separação permite que as aplicações utilizem a informação do contexto sem se preocuparem com a forma como ela foi obtida (normalmente o código de obtenção de contexto é complexo, sendo necessária uma comunicação direta com os sensores).
- **Interpretação de contexto:** está relacionada com a combinação de informações de contexto de baixo nível a fim de gerar informações de maior utilidade para as aplicações. Por exemplo, uma aplicação pode estar interessada na informação de que uma reunião está ocorrendo. Para que essa informação seja obtida, é necessário combinar informações de localização de diversas pessoas, hora do dia, nível sonoro da sala onde estão as

pessoas, etc. Essa interpretação também poderia ser feita diretamente pela aplicação, mas se for feita a nível de infra-estrutura, ela pode ser utilizada por diversas aplicações.

- **Comunicações distribuídas e transparentes:** está relacionada com a natureza distribuída das informações de contexto. Normalmente, os sensores que fornecem informações de contexto estão distribuídos na rede. Dessa forma, é necessário que a infra-estrutura dê suporte a essa distribuição e a deixe transparente para as aplicações, uma vez que o desenvolvedor da aplicação não deve se preocupar com questões de distribuição na aquisição da informação de contexto.
- **Disponibilidade constante de aquisição de contexto:** está relacionada com a capacidade que os componentes que obtêm informações de contexto têm de estarem sempre disponíveis. As aplicações que utilizam uma infra-estrutura que provê informações de contexto devem ser capazes de utilizar componentes já existentes ao invés de criar seus próprios componentes. Dessa forma, diversas aplicações podem utilizar os mesmos componentes da infra-estrutura, que devem ter um ciclo de vida independente da aplicação que os utiliza.
- **Armazenamento de contexto:** está relacionado com a capacidade de armazenar de alguma forma informações de contexto já obtidas, criando um histórico. Ter um histórico de informações de contexto é importante por dois motivos. O primeiro é que as aplicações podem ter necessidade de recuperar alguma informação do passado. O segundo é que, baseado em informações históricas de contexto, em alguns casos é possível fazer uma previsão do contexto futuro, o que pode ser desejável em alguns casos.
- **Descoberta de recursos:** está relacionada com o fornecimento de um serviço para as aplicações de descoberta de recursos da infra-estrutura. Quando as aplicações desejam se comunicar com os sensores, por exemplo, é importante que a infra-estrutura forneça uma maneira fácil e transparente para realizar essa tarefa. Esse componente tem a responsabilidade de descobrir sensores, comunicar-se com eles e fornecer um meio para que eles e as aplicações possam interagir, tornando transparente ao desenvolvedor questões como protocolos, linguagens, etc.

Para Costa [COS04], uma infra-estrutura para aplicações dependentes de contexto deve ser capaz de suportar os seguintes aspectos:

- **Reação a estímulos do ambiente:** as aplicações dependentes de contexto devem ser avisadas quando mudanças no contexto ocorrem.
- **Encapsulamento da rede de sensores:** o desenvolvedor não deve ter a preocupação de buscar os sensores ou como extrair a informação diretamente deles. A infra-estrutura deve tornar isso transparente.
- **Suporte a aspectos de distribuição:** aplicações dependentes de contexto são distribuídas por natureza. Logo, o suporte à distribuição é necessário e deve ser transparente ao desenvolvedor.
- **Suporte à representação de contexto:** é necessário que haja uma representação consistente do contexto, que não permita ambigüidades na sua compreensão. Só dessa forma as aplicações e a infra-estrutura poderão interagir adequadamente.

Uma infra-estrutura para aplicações dependentes de contexto pertence às categorias *middleware based systems* e *context-server based systems*, apresentadas em [BAL07]. A Figura 2.1 mostra a estrutura de um *framework* conceitual separado em camadas.

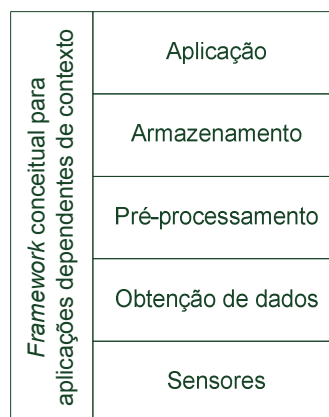


Figura 2.1: Estrutura de um framework para aplicações dependentes de contexto

2.4. Ontologias

Uma ontologia define um conjunto de conceitos utilizados para descrever e representar um domínio [POW03]. Quatro elementos compõem uma ontologia:

- **Indivíduos:** podem ser objetos concretos (ex: cadeira, computador, telefone) ou abstratos (ex: números, palavras). São os elementos básicos de uma ontologia.
- **Classes:** agrupam objetos comuns. Podem conter indivíduos e/ou outras classes.
- **Atributos:** descrevem os indivíduos de uma ontologia. Cada atributo é composto por um nome e um valor.
- **Relacionamentos:** descrevem como os objetos de uma ontologia se relacionam. Os relacionamentos mais importantes são *é-um* (ex: telefone sem fio *é-um* equipamento eletrônico) e *parte-de* (ex: antena *é parte-de* um telefone sem fio). O relacionamento do tipo *é-um* cria a noção de relacionamento hierárquico entre os objetos.

O propósito geral de uma ontologia é apresentar um meio de classificação aos indivíduos. Segundo Baldauf *et al.* [BAL07], ontologias representam uma descrição de conceitos e relacionamentos. Dessa forma, são um instrumento promissor para modelar informações de contexto devido ao seu alto grau de expressividade. Várias infra-estruturas propostas utilizam ontologias para modelagem de contexto, como será visto no **Erro! Fonte de referência não encontrada.**

2.5. Plataforma CORBA

No mundo onde vivemos existe uma grande heterogeneidade de dispositivos e uma necessidade cada vez maior de comunicação entre eles. Levando essa afirmação para o contexto da informática, existem hoje diversos tipos de redes, sistemas operacionais, computadores, linguagens de programação, e é necessário fazer com que essa gama de diversidades interaja.

Como os aumentos da heterogeneidade e da comunicação são inevitáveis, é necessário que haja algum mecanismo que facilite o desenvolvimento de aplicações que têm essa necessidade. Esse mecanismo deve deixar transparente para o desenvolvedor questões que

envolvem a heterogeneidade, a fim de que ele possa se concentrar exclusivamente no desenvolvimento das regras de negócio das aplicações.

Com esse intuito, a OMG [OMG08] criou a plataforma CORBA. CORBA visa facilitar a criação de aplicações distribuídas, tornando transparente ao desenvolvedor questões como localização dos objetos na rede, diferenças de linguagem de programação, diferenças de sistema operacional e diferenças de plataforma de *hardware*. O elemento na arquitetura CORBA responsável pela transparência no uso distribuído das aplicações é o ORB. O ORB é responsável por tornar as informações acessíveis de forma que o desenvolvedor possa programar como se todos os recursos estivessem localizados no seu próprio computador.

CORBA é uma arquitetura no estilo cliente-servidor. Além do ORB, diversos outros elementos fazem parte da infra-estrutura, como é mostrado na Figura 2.2.

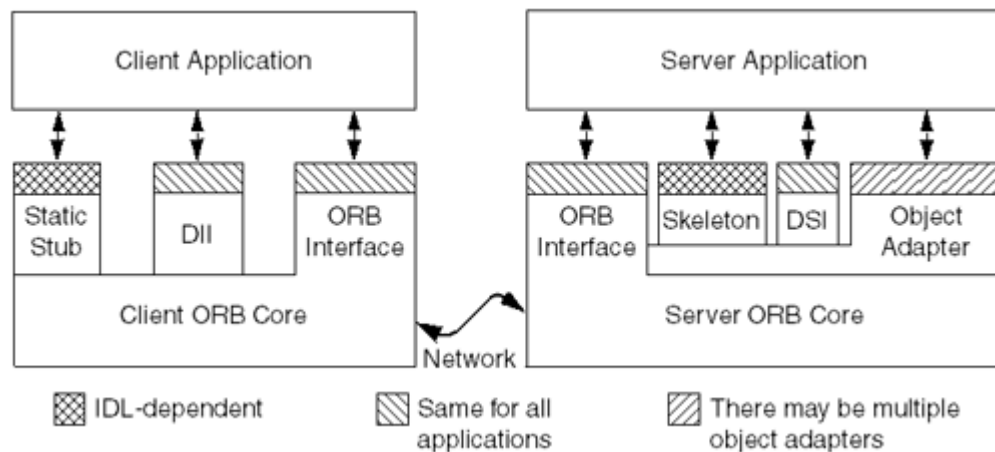


Figura 2.2: Visão geral da plataforma CORBA [HEN99]

Além do ORB, outro elemento importante da arquitetura CORBA é a IDL. A IDL é uma espécie de linguagem para definição de interfaces CORBA. A IDL, apesar de ter uma sintaxe similar ao C++ [STR07], não é baseada em nenhuma linguagem de programação existente. Isso permite que códigos escritos em diversas linguagens de programação possam se comunicar de uma forma uniforme, garantindo a interoperabilidade entre eles.

2.5.1. Serviço de Notificação

A plataforma CORBA possui um sistema de notificação de eventos chamado *Notification Service* [SIE00], que é uma extensão do *Event Service*, o primeiro serviço de eventos em CORBA. A existência de um serviço dessa natureza em CORBA possibilita que a plataforma dê suporte à comunicação assíncrona.

A comunicação assíncrona permite uma clara separação entre produtores e consumidores da informação, sendo que produtores e consumidores não precisam necessariamente ter qualquer referência entre si. Isto é, a comunicação assíncrona permite que alguém crie a informação sem saber quem está interessado nela, assim como também permite que alguém utilize a informação sem saber quem foi que a criou.

A existência de um mecanismo que permita comunicações assíncronas é muito importante, sobretudo em aplicações distribuídas, onde são comuns os casos de falha ou atraso na comunicação entre os componentes da aplicação.

No modelo do serviço de notificação do CORBA, os produtores (*suppliers*) são responsáveis pela criação dos eventos, enquanto os consumidores (*consumers*) são responsáveis pela recepção dos eventos [HEN99]. Produtores e consumidores se conectam a um canal de eventos (*event channel*), que possui as seguintes responsabilidades:

- Registrar produtores e consumidores.
- Entregar a informação de forma confiável a todos os consumidores que desejam recebê-la.
- Implementar lógica de tratamento de erro quando a entrega da informação a um ou mais consumidores falha.

Existem dois modelos para a entrega de eventos no serviço de notificação do CORBA: *push* e *pull*.

No modelo *push*, o produtor coloca o evento no canal de eventos e este envia a informação ao consumidor. O envio é feito através da chamada de uma função de *callback* no consumidor, que é informada no momento que o consumidor se registra no canal de eventos. A Figura 2.3 mostra o esquema de funcionamento do modelo *push*.

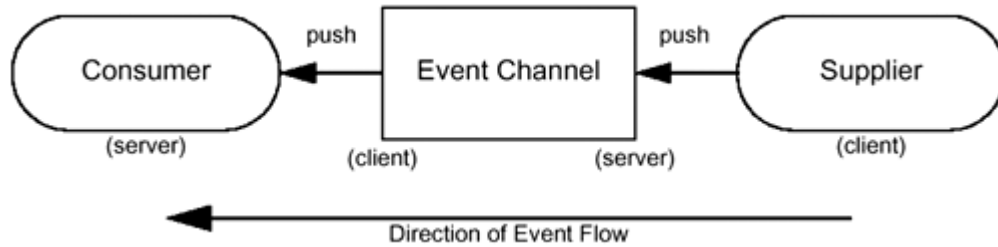


Figura 2.3: O modelo *push* de entrega de eventos [HEN99]

Já no modelo *pull*, o consumidor toma a iniciativa de buscar pela ocorrência de determinado evento no canal de eventos. Por sua vez, o canal de eventos também busca no produtor o evento desejado. A Figura 2.4 mostra o esquema de funcionamento do modelo *pull*.

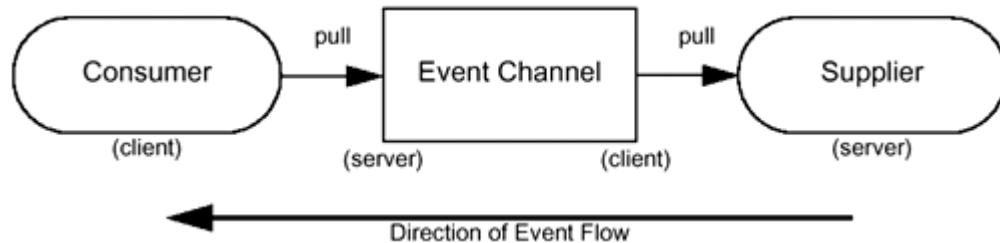


Figura 2.4: O modelo *pull* de entrega de eventos [HEN99]

O serviço de notificação também suporta combinações entre esses dois modelos de entrega de eventos, uma vez que possibilita que diversos produtores e consumidores estejam ligados a um mesmo canal de eventos e que cada um utilize o seu próprio modelo de entrega. Essa característica faz com que o serviço de notificação de CORBA seja flexível.

Além de transmitir os eventos aos consumidores, o serviço de notificação também é capaz de desconectar os consumidores que estejam conectados em determinado canal, tanto no modo *push* quanto no modo *pull*. Neste caso, o CORBA avisa os consumidores através da chamada de um método de *callback*, dando a chance para que eles tratem a desconexão de forma adequada.

Além do que já foi mencionado, o serviço de notificação do CORBA adicionou algumas características novas ao seu antecessor *Event Service*. As principais são:

- Possibilidade da utilização de parâmetros de qualidade de serviço (QoS) associados aos eventos.
- Estrutura bem definida para representação de eventos, a fim de organizar os dados e os metadados do evento.
- Filtragem dinâmica de eventos baseado no tipo e em parâmetros de QoS.

2.5.2. Serviço de *Trading*

A plataforma CORBA possui um serviço de nomes (*naming service*), o qual permite a localização de um objeto a partir de um nome conhecido. Funciona da seguinte forma: a referência de determinado objeto é armazenada no serviço de nomes juntamente com um nome único para esse objeto. Mais tarde, quando for necessário recuperar a referência do objeto, basta fazer uma procura pelo seu nome. Esse serviço é semelhante ao serviço de páginas brancas de uma lista telefônica.

O serviço de nomes é útil e muito importante na plataforma CORBA, mas possui uma desvantagem: o nome do objeto tem que ser conhecido de antemão para que seja possível localizá-lo. Com o objetivo de prover um serviço semelhante ao serviço de nomes, mas com maior poder de busca pelos objetos, foi criado o serviço de *trading* [HEN99].

O serviço de *trading* funciona de forma semelhante ao serviço de nomes, mas com uma diferença: ao invés da busca ser feita por nomes de objetos, ela é feita pelas características dos objetos. Isto é, com esse serviço é possível obter referência(s) a objeto(s) sem que o nome seja conhecido, apenas através das suas características. Esse serviço é semelhante ao serviço de páginas amarelas de uma lista telefônica.

A principal entidade do serviço de *trading* é o *trader*. Ele é responsável pelo registro e busca de objetos que atendem a determinado critério. O registro é feito associando um conjunto de pares propriedade/valor à referência do objeto. A busca é baseada no valor das propriedades armazenadas utilizando uma linguagem chamada *trader constraint language*. Essa linguagem permite fazer buscas complexas baseadas em propriedades dos objetos e lembra muito o SQL, utilizado em consultas de tabelas de bancos de dados.

2.6. Reflexividade

Reflexividade é um mecanismo utilizado por um programa para obter informações sobre o ambiente de execução onde ele está inserido e modificá-lo caso seja necessário [GOL98]. Na reflexão, um programa consegue obter informações a respeito da sua própria estrutura. Normalmente a termo reflexividade refere-se à reflexão dinâmica, que ocorre em tempo de execução. Mas também existe a reflexão estática, que ocorre em tempo de compilação.

A reflexão pode ser utilizada em cenários como os de auto-otimização e auto-modificação de um programa. Neste caso, o programa poderá otimizar-se ou modificar-se de acordo com a tarefa que ele está realizando em um determinado momento. No caso da reflexividade dinâmica, isto pode ser feito em tempo de execução, isto é, enquanto o programa está executando.

2.7. Conclusão

O CxFramework utilizará ontologias para modelar as informações de contexto e será construído sobre a plataforma CORBA.

Os principais motivos que fazem com que ontologias sejam escolhidas para a representação de contexto no CxFramework são a sua própria natureza (ontologias são uma forma de classificação de indivíduos) e a capacidade de inferência de novas ontologias a partir de uma inicial. Outras tecnologias, como a utilização de modelagem orientada a objetos ou pares de chave e valor, também são opções para representar contexto, mas são mais limitadas (não possuem a mesma expressividade). Além disso, não possuem mecanismo de inferência já previsto, o qual deve ser implementado manualmente.

Já os principais motivos que levam o CxFramework a adotar o CORBA é a possibilidade de aproveitar serviços já existentes na plataforma, como o *Naming Service*, *Trading Service* e *Notification Service*. CORBA também já possui suporte a componentes distribuídos, além de ser uma plataforma madura e estável.

O próximo capítulo apresentará algumas infra-estruturas para aplicações dependentes de contexto existentes que estão relacionadas com a proposta deste trabalho.

Capítulo 3

Trabalhos Relacionados

As seções a seguir tratam dos trabalhos na literatura mais alinhados com esta proposta.

3.1. *Context Toolkit*

Uma das primeiras infra-estruturas criadas com a finalidade de suportar aplicações dependentes de contexto foi o *Context Toolkit* [DEY01]. Ela foi utilizada como base de algumas outras infra-estruturas que surgiram depois. A Figura 3.1 mostra os diversos componentes que compõem a arquitetura do *Context Toolkit*.

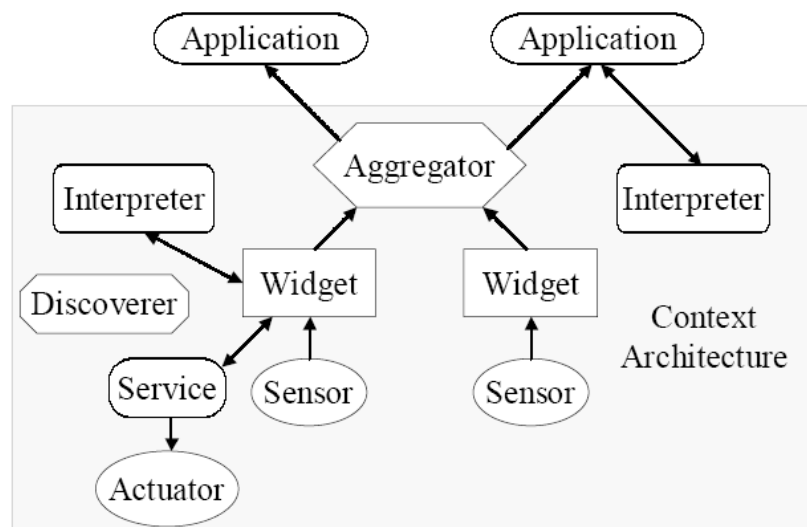


Figura 3.1: Componentes da infra-estrutura *Context Toolkit* [DEY01]

Os componentes centrais da infra-estrutura são os chamados *context widgets*. Esses componentes possibilitam que as aplicações obtenham informações de contexto através de uma interface comum, independente de como a informação do contexto foi obtida. Os *context widgets* são responsáveis pela comunicação direta com os sensores para obtenção da informação de contexto, as quais são modeladas através da utilização de pares de chave e valor.

Os componentes *interpreters* têm como responsabilidade interpretar o contexto obtido pelos *widgets*, gerando uma informação de mais alto nível. Por exemplo, um *interpreter* pode ser responsável por informar se uma reunião está acontecendo, analisando algumas informações de contexto, descobre que diversas pessoas estão no mesmo ambiente e o nível sonoro é baixo.

Os componentes *aggregators* agrupam *widgets* que possuem alguma relação lógica. Por exemplo, determinada ação deve ocorrer se um indivíduo está em sua sala, em silêncio e sem se movimentar. Essas informações podem ser agrupadas em um *aggregator* para facilitar o acesso. Caso contrário, as aplicações deveriam acessar diversas informações para chegar a essa conclusão.

Os componentes *services* têm como responsabilidade a execução de ações na infra-estrutura, ao contrário dos outros componentes já mencionados, responsáveis apenas por ler informações de contexto. Por exemplo, a ação de “acender a luz” pode ser realizada por um serviço da infra-estrutura (as aplicações que usam a infra-estrutura podem compartilhar o mesmo serviço).

Por último, o componente *discoverer* tem o registro de todos os componentes da infra-estrutura. É com esse componente que as aplicações interagem para procurar por *widgets*, *interpreters*, *aggregators* e *services*. A busca pode ser feita por nome ou características dos componentes.

Para resolver o problema da distribuição dos sensores, a infra-estrutura implementa um mecanismo próprio de comunicação, que utiliza XML sobre HTTP. Essa forma de comunicação também é utilizada pelo serviço de eventos, que notifica as aplicações registradas sobre as alterações no contexto (comunicação assíncrona) e também pelo serviço de obtenção de informações através de *queries* (comunicação síncrona).

Uma desvantagem desta infra-estrutura é não possuir um mecanismo centralizado para buscar informações de contexto: é necessário que a aplicação tenha referência a todos os *widgets*,

interpreters e/ou *aggregators* necessários, o que causa grande dependência entre objetos e, após certo ponto, impossibilita garantir a escalabilidade. No entanto, essa arquitetura introduz um conceito muito importante: a separação da aquisição da informação de contexto e da sua representação (através dos *widgets*, que funcionam como *wrappers* de contexto).

3.2. Infra-estrutura proposta por Sinderen *et al.* [SIN06]

Sinderen *et al.* [SIN06] também propôs uma infra-estrutura para aplicações dependentes de contexto. A Figura 3.2 mostra a arquitetura proposta.

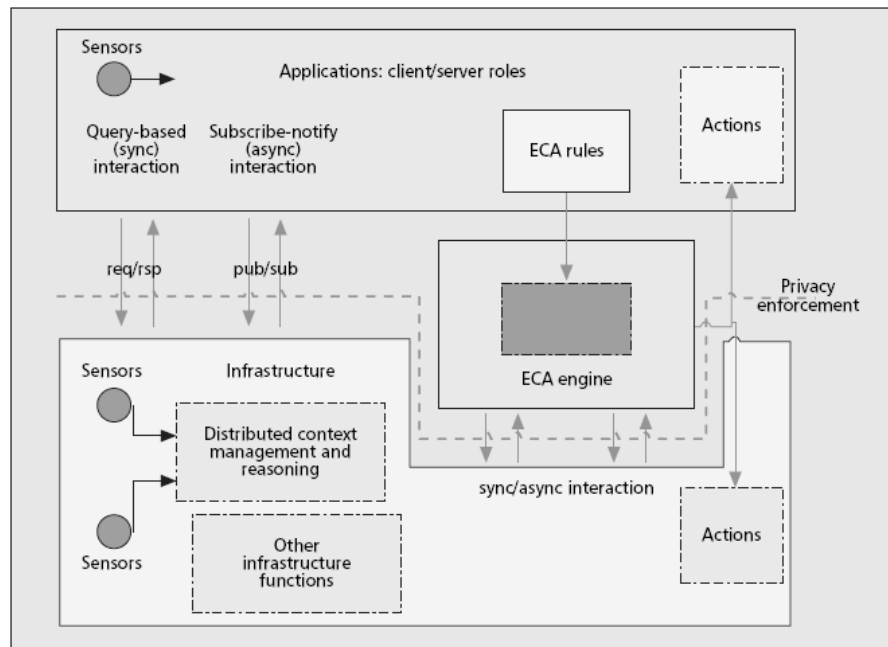


Figura 3.2: Arquitetura da infra-estrutura proposta por Sinderen *et al.* [SIN06]

Esta infra-estrutura possui um módulo chamado *ECA engine*, que implementa o padrão ECA (*Event-Condition-Action*) [COS04]. Através da utilização desse módulo, as aplicações podem registrar interesse em determinadas mudanças de contexto.

Quanto à modelagem das informações de contexto, esta infra-estrutura utiliza ontologias. A inferência de contexto é feita através de aprendizagem de máquina e resolução de ontologias.

No que diz respeito à segurança, a infra-estrutura possui mecanismos para lidar com a confiabilidade e privacidade das informações.

3.3. SOCAM

O SOCAM (*Service Context-Aware Middleware*) [GU05] é uma infra-estrutura para aplicações dependentes de contexto também focada na utilização de ontologias como forma de modelagem de informações de contexto. Sua arquitetura pode ser vista na Figura 3.3.

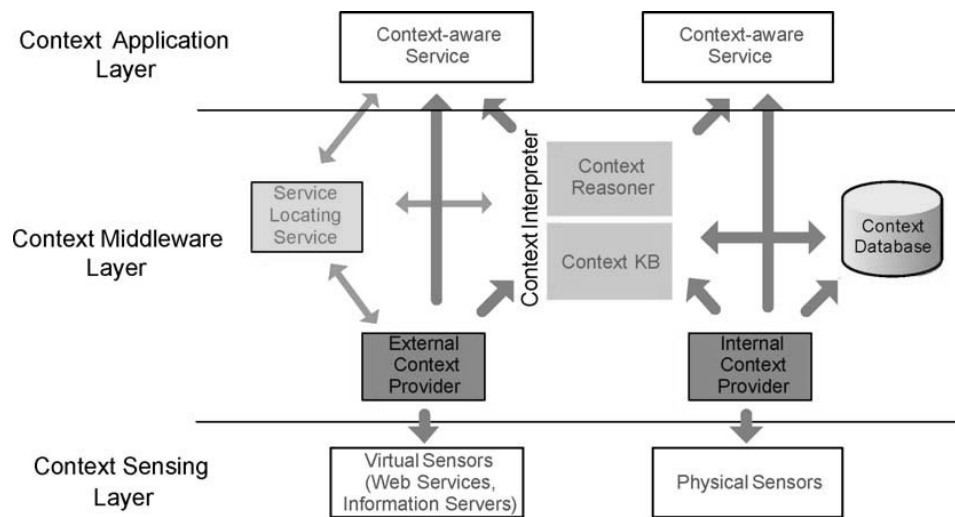


Figura 3.3: Arquitetura do SOCAM [GU05]

Nesta infra-estrutura, as ontologias são divididas em dois níveis: comum e específica de um domínio. Na primeira, estão as ontologias mais genéricas, comuns a todos os domínios. Já na segunda, como o próprio nome diz, estão as ontologias que fazem sentido apenas dentro de um domínio. A separação das ontologias em dois níveis tem como objetivo reduzir o seu tamanho e, conseqüentemente, reduzir o tempo de inferências de novas ontologias. Outra característica do SOCAM é introduzir um método para classificação e dependência entre contextos.

Quanto às aplicações dependentes de contexto, estas podem obter informações de contexto através dos métodos *pull* e *push*. No caso do modo de comunicação *push*, as aplicações são notificadas através de um método de *callback*. Este mecanismo faz com que a infra-estrutura e a aplicação fiquem fortemente acopladas, o que é uma desvantagem em um cenário distribuído.

O SOCAM é implementado em Java, utilizando RMI. Como RMI é uma tecnologia atrelada ao Java, não é possível fazer com que os componentes do SOCAM se comuniquem com outros componentes externos, e até aplicações, que não sejam escritos na linguagem Java.

3.4. MoCA

A MoCA (*Mobile Collaboration Architecture*) [VIT06] é uma arquitetura para suportar aplicações dependentes de contexto que envolvem usuários móveis.

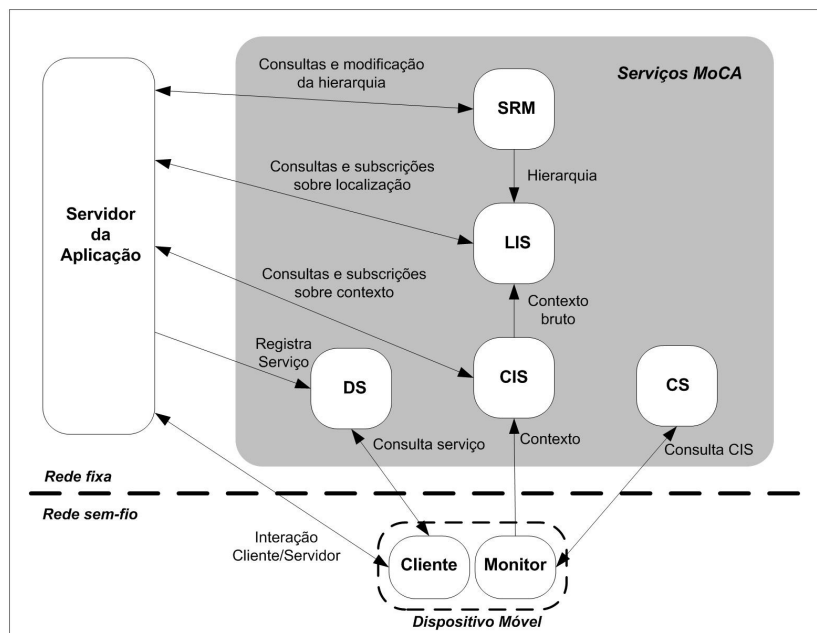


Figura 3.4: Arquitetura da MoCA [VIT06]

Esta arquitetura é composta por serviços como o CIS (*Context Information Service*), que armazena e processa informações de contexto, e o LIS (*Location Inference Service*), que infere informações de localização utilizando contextos armazenados no CIS. O componente *Monitor* é executado no dispositivo móvel do cliente, e é deste componente a responsabilidade de fornecer informações de contexto (como carga da bateria do dispositivo, memória livre, uso da CPU, etc.).

A notificação às aplicações pode ser feita nos modos *push* e *pull*. Em ambos os casos, são utilizados os protocolos TCP e UDP, o que acopla as aplicações à infra-estrutura. Além disso, a quantidade de contexto que a infra-estrutura é capaz de gerenciar é limitada (resume-se a algumas

informações vindas do dispositivo móvel) e não pode ser amplamente estendida. Desta forma, o mecanismo de inferência também é limitado, além de ser atrelado ao código da infra-estrutura.

3.5. Conclusão

As infra-estruturas mencionadas neste Capítulo possuem algumas características interessantes e que podem ser aproveitadas na construção de uma nova infra-estrutura para aplicações dependentes de contexto. Dentre elas, é possível destacar o uso de ontologias (que são uma ferramenta poderosa para representar contexto) e a separação da obtenção e da representação do contexto (que permite que os sensores sejam enxergados de forma homogênea).

No entanto, algumas características presentes nas infra-estruturas mencionadas poderiam ser melhoradas. É o caso da existência do acoplamento entre aplicações e infra-estrutura no que diz respeito à notificação, ao invés do uso de um mecanismo baseado em eventos, que permitiria o desacoplamento, muitas vezes desejado em sistemas distribuídos. Além disso, nenhuma das infra-estruturas mencionadas considera que a própria aplicação poderia alterar o contexto. Esta abordagem permitiria a representação de uma gama ainda maior de cenários onde a dependência de contexto pode ser utilizada.

No próximo capítulo será abordada a arquitetura do CxFramework, a infra-estrutura para aplicações dependentes de contexto proposta neste trabalho, que busca não só aproveitar as características importantes de infra-estruturas já existentes como também introduzir outros conceitos ainda não explorados.

Capítulo 4

Arquitetura do CxFramework

4.1. Modo de Utilização

O CxFramework é uma infra-estrutura baseada em serviços que suporta aplicações dependentes de contexto. A Figura 4.1 mostra o CxFramework do ponto de vista das aplicações.

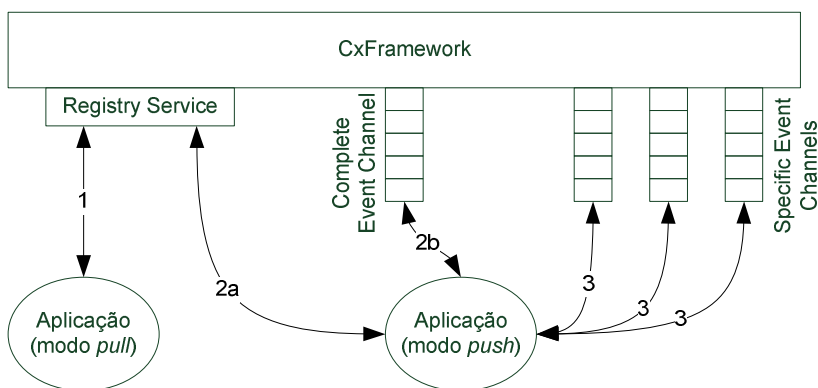


Figura 4.1: O CxFramework do ponto de vista das aplicações

As aplicações podem usar os modos de comunicação *pull* e *push*. No modo de comunicação *pull*, as aplicações solicitam à infra-estrutura (usando o *Registry Service*, que será detalhado mais tarde neste capítulo) informações de contexto (isto é representado pelo evento 1 na Figura 4.1). Neste caso, a informação de contexto desejada deve ser previamente conhecida.

No modo de comunicação *push*, as aplicações são notificadas de forma assíncrona a respeito das mudanças de contexto que as interessam (um método de *callback* na aplicação é

chamado). A comunicação entre o CxFramework e as aplicações é baseada em eventos, o que permite um baixo acoplamento. Isto significa que a notificação pode ser entregue mesmo que o emissor ou o receptor não estejam disponíveis ao mesmo tempo, o que é particularmente importante em arquiteturas distribuídas móveis. As aplicações precisam obter uma referência ao *Specific Event Channel* do seu interesse (por cada *Specific Event Channel* trafega um tipo de informação de contexto diferente). Desta forma, elas podem se registrar nestes canais e serem notificadas a respeito de mudanças de contexto (a Seção 6.1 explica o que são exatamente estas referências). Isto pode ser feito de duas formas.

A primeira é solicitar à infra-estrutura (usando novamente o *Registry Service*) eventos de contexto existentes baseado em características (evento 2a na Figura 4.1). Como exemplo, uma aplicação poderia solicitar à infra-estrutura todas as informações de contexto relacionadas à Sala 1 que a infra-estrutura é capaz de gerar. Como resultado, a aplicação receberia “temperatura” e “nível sonoro”, que são informações de contexto relacionadas à Sala 1.

A segunda abordagem é se conectar ao *Complete Event Channel* (evento 2b na Figura 4.1). Todos os eventos de contexto gerados pela infra-estrutura trafegam por este canal. Conseqüentemente, aplicações conectadas a ele são notificadas de todas as mudanças e podem escolher os tipos de contexto que as interessam.

Usando tanto a primeira quanto a segunda abordagem, as aplicações recebem uma lista de alterações de contexto suportadas pela infra-estrutura. Adicionalmente, cada mudança de contexto está atrelada a um representante de um *Specific Event Channel*, o qual pode ser usado para que a aplicação se registre no canal (evento 3 na Figura 4.1). Cada *Specific Event Channel*, responsável por apenas um tipo de evento, notifica todas as aplicações conectadas a ele. Como exemplo, uma aplicação interessada em ser notificada quando a temperatura ou o nível sonoro da Sala 1 mudar se registraria em ambos os canais, o que permitiria o recebimento de notificação dos tipos de mudança no contexto.

4.2. Arquitetura Interna

Internamente, o CxFramework é composto de cinco serviços e um módulo coletor de lixo (*garbage collector*). A Figura 4.2 mostra a arquitetura do CxFramework. As seções seguintes descrevem mais detalhadamente cada um dos componentes da infra-estrutura.

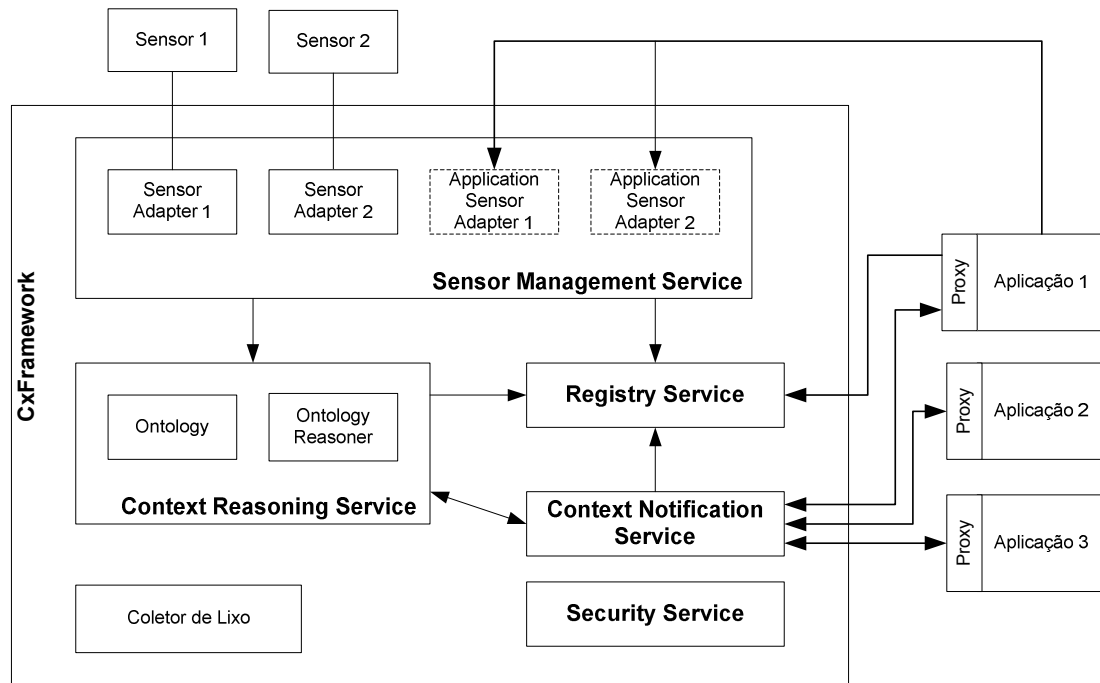


Figura 4.2: Estrutura do CxFramework

4.2.1. *Sensor Management Service*

O *Sensor Management Service* é responsável pela comunicação entre a infra-estrutura e os sensores externos (representados pelos *sensors* na Figura 4.2) e possui um conjunto de *Sensor Adapters*. Os sensores são dispositivos físicos (como sensores de temperatura, calendários, relógios, etc.) ou “virtuais” (outras aplicações) capazes de fornecer informações de contexto. O *Sensor Adapter* é responsável por encapsular o código de comunicação com um sensor externo, agindo como um representante do sensor dentro da infra-estrutura. O *Sensor Adapter Provider* (veja Seção 4.2.8) é responsável por implementar o código de comunicação com os sensores e também por registrá-los. Os *Application Sensor Adapters* são *Sensor Adapters* especiais. Seu funcionamento será explicado na Seção 4.2.7.

Um *Sensor Adapter* pode ser implementado de diversas formas. Se o sensor é capaz de fornecer seu estado para uma aplicação externa, o *Sensor Adapter* pode ser implementado de forma a aguardar notificações vindas do sensor. Se o sensor é capaz de fornecer informações apenas se for solicitado, o *Sensor Adapter* pode ser implementado de forma a executar um *loop*

solicitando informações do sensor de tempos em tempos. Além disso, alguns sensores podem suportar as duas abordagens: notificar seu estado e também retornar a informação quando solicitado. Neste caso, o *Sensor Adapter* poderia tirar vantagem destes dois tipos de comportamento.

O principal propósito de um *Sensor Adapter* é encapsular o código necessário para comunicação com os sensores, que podem ser totalmente heterogêneos e dos mais diversos tipos, escondendo estes detalhes da infra-estrutura. Esta abordagem expõe à infra-estrutura um conjunto de componentes homogêneos (os *Sensor Adapters*), a qual não precisa conhecer detalhes de implementação e dos sensores propriamente ditos. Isto ajuda na manutenibilidade (todo o código de comunicação com sensores está contido nos *Sensor Adapters*) e na extensibilidade (novos *Sensor Adapters* podem ser facilmente adicionados para suportar novos tipos de sensores).

Quando um *Sensor Adapter* detecta uma mudança de estado, ele deve realizar apenas uma ação: notificar o *Context Reasoning Service* (veja a Seção 4.2.3) que o estado do sensor mudou. O *Sensor Adapter* deve informar seu ID único e qual o novo estado do sensor representado por ele.

4.2.2. *Registry Service*

O *Registry Service* mantém o registro dos seguintes componentes da infra-estrutura:

- **Serviços da infra-estrutura:** todos os serviços da infra-estrutura se registram no *Registry Service*. A razão para isto é que os serviços precisam se comunicar e o *Registry Service* é responsável por fornecer as referências aos serviços desejados (cada serviço possui um nome único).
- **Eventos de mudanças de contexto:** todos os eventos de mudança de contexto suportados pela infra-estrutura devem ser registrados no *Registry Service*. Aplicações dependentes de contexto procuram por eventos de contexto que podem ser do seu interesse para que possam ser notificadas quando mudanças ocorrem. Por este motivo estes eventos de mudança de contexto devem estar registrados. A Seção 6.1 explica o que são estes eventos de contexto.

4.2.3. *Context Reasoning Service*

O *Context Reasoning Service* contém a inteligência da infra-estrutura. Este é o serviço que recebe eventos vindos dos *Sensor Adapters* e os processa, inferindo informações de contexto de mais alto nível. O *CxFramework* utiliza ontologias (veja a Seção 2.4) para modelar informações de contexto e inferir informações de contexto de alto nível. O *ontology provider* é responsável por prover a ontologia e as regras de inferência.

O motivo da escolha de ontologias como forma de modelagem de informações de contexto é devido ao fato que é fácil inferir novas ontologias a partir de uma ontologia original, utilizando regras de inferência. Esta característica já vem embutida no conceito de ontologia, que é uma forma de classificação de indivíduos. Além disso, ontologias podem ser criadas por pessoas que não estão envolvidas com a área de programação, através de ferramentas gráficas que, muitas vezes, são intuitivas e fáceis de usar. Outras abordagens para modelagem de contexto se mostraram limitadas (como a utilização de pares de chave e valor) ou fortemente acopladas com a programação da infra-estrutura (como a modelagem de contexto de forma orientada a objetos). Em ambos os casos, o mecanismo de inferência deve ser implementado pelo próprio programador, o que aumenta a complexidade e diminui a flexibilidade da solução.

Os *Sensor Adapters* notificam o *Context Reasoning Service* quando o estado dos sensores muda. Quando notificado, o *Context Reasoning Service* muda a ontologia corrente baseada no novo estado do sensor enviado pelo *Sensor Adapter*. O mapeamento entre os *Sensor Adapters* e a ontologia está descrito na *Event Table*. A partir do ID do *Sensor Adapter*, a *Event Table* fornece a informação de qual relação da ontologia é afetada pelo *Sensor Adapter* que detectou a mudança do estado do sensor.

Quando a ontologia muda, o *Context Reasoning Service* utiliza as regras de inferência para inferir uma nova ontologia. Todos os relacionamentos da ontologia que são alterados são enviados para o *Context Notification Service* (veja a Seção 4.2.4), a qual é responsável por notificar as aplicações a respeito das mudanças no contexto.

4.2.4. Context Notification Service

O *Context Notification Service* notifica aplicações a respeito de mudanças de contexto. Este serviço gerencia o *Complete Event Channel* e os *Specific Event Channels*. A Seção 6.1 explica como o *Context Notification Service* funciona sob o aspecto de implementação.

4.2.5. Proxy

O *proxy* é outro componente importante na arquitetura. Ele não está diretamente inserido na infra-estrutura, mas é parte da aplicação conectada ao CxFramework. O *proxy* é uma API importada pela aplicação cuja responsabilidade é esconder código específico da plataforma (isto é, da plataforma distribuída usada na implementação do CxFramework) da aplicação. Toda a comunicação entre a aplicação dependente de contexto e a infra-estrutura deve, obrigatoriamente, ser feita através do *proxy*.

O *proxy* também cumpre um papel importante em permitir que a infra-estrutura seja suficientemente genérica para ser construída sobre outras plataformas. Se as aplicações invocassem código específico de uma plataforma ao invés de utilizar o *proxy*, elas teriam que ser alteradas se o CxFramework fosse implementado sobre uma plataforma diferente.

4.2.6. Security Service

O *Security Service* é o serviço responsável por controlar o acesso às informações de contexto. Apesar de estar previsto na arquitetura do CxFramework, consideramos o detalhamento e implementação deste serviço como estando fora do escopo deste trabalho.

4.2.7. Application Sensor Adapter

Após detalhar a estrutura do CxFramework, é importante destacar a presença dos *Application Sensor Adapters*, destacados na Figura 4.2. O *Application Sensor Adapter* funciona como qualquer outro *Sensor Adapter*, exceto pelo fato de que os *Application Sensor Adapters* são criados e gerenciados pelas aplicações dependentes de contexto conectadas ao CxFramework. Quando uma aplicação se registra na infra-estrutura, ela também assume o papel de sensor. Devido a isso, aplicações podem alterar o contexto do ambiente (usando seus *Application Sensor Adapters*) da mesma forma que qualquer outro sensor faria. Este recurso faz com que o

CxFramework seja uma infra-estrutura **reflexiva**, isto é, a própria infra-estrutura é influenciada pelo ambiente que ela representa (veja a Seção 4.3).

4.2.8. Papéis Existentes na Infra-Estrutura

Existem, no CxFramework, quatro papéis, cada um com tarefas bem definidas.

O primeiro papel é o do *Sensor Adapter Provider*, que é responsável por implementar o código dos *sensor adapters*. O CxFramework é capaz de trabalhar com *sensor adapters* implementados por diferentes *providers*, já que todos os *sensor adapters* possuem uma interface comum. A grande vantagem disso é que quem faz esta implementação conhece o mecanismo de funcionamento do sensor que será representado pelo *Sensor Adapter*, podendo implementá-lo da forma mais correta. O CxFramework deve apenas utilizar os *Sensor Adapters* já criados, sem se preocupar em como a comunicação com os sensores é realizada.

O segundo papel é do *Ontology Provider*, que é responsável por criar a ontologia e as regras de inferência que serão utilizadas pelo CxFramework para representar o contexto.

O terceiro papel é do *CxFramework Administrator*, o administrador da infra-estrutura. É dele a responsabilidade de configurar todos os serviços corretamente, utilizando os arquivos de configuração necessários (a Seção 6.1 detalha mais a respeito dos arquivos de configuração utilizados pela infra-estrutura). O administrador utiliza o que é fornecido pelos *Sensor Adapter Providers* e pelo *Ontology Provider* para configurar a infra-estrutura corretamente, inclusive devendo interagir com eles para garantir o bom funcionamento da infra-estrutura.

Por último, existe o papel das aplicações dependentes de contexto (*context-aware applications*). São elas que se conectam a infra-estrutura e recebem as informações de contexto sentidas e processadas pela infra-estrutura. As aplicações também podem fornecer informação de contexto, através dos *Application Sensor Adapters*, já abordados anteriormente.

É possível perceber que o maior esforço de configuração se encontra na administração da infra-estrutura em si, e não nas aplicações. Isto é justificável, uma vez que a infra-estrutura tem por objetivo encapsular todo o código referente a contexto, tirando essa responsabilidade das aplicações.

4.3. Reflexividade e Federação de Contextos

As aplicações dependentes de contexto, ao se conectarem à infra-estrutura, assumem também o papel de sensores. Utilizando os seus *Application Sensor Adapters*, elas podem fazer alterações no contexto que se refletem na infra-estrutura. Em outras palavras, a infra-estrutura não só influencia as aplicações como também é influenciada por elas, caracterizando a **reflexividade**. Como exemplo de reflexividade, uma aplicação poderia estar interessada em saber quando outra aplicação (possivelmente um serviço) se conecta à infra-estrutura. Note que, neste cenário, a presença de uma aplicação muda o contexto corrente, caracterizando a reflexão.

Utilizando a reflexividade, é possível montar uma **federação de contextos**. A federação de contextos significa fazer com que dois ou mais contextos distintos, representados por instâncias diferentes de CxFrameworks, possam trocar informações, fazendo com que alterações em um contexto possam alterar outros. A idéia por trás disso é ter uma aplicação que se conecta à duas ou mais infra-estruturas e faz o papel de “ponte”. Ela é notificada a respeito de mudanças de contexto em uma delas e repassa esta informação para as outras utilizando um *Application Sensor Adapter*.

O Capítulo 5 mostrará o uso de reflexividade e federação de contextos num cenário prático.

4.4. Conclusão

Este capítulo descreveu a arquitetura do CxFramework, detalhando cada um dos serviços que o compõem. Foram abordados também a reflexividade e a federação de contextos, que abrem ainda mais o leque de possibilidades de aplicações dependentes de contexto suportadas pela infra-estrutura.

A infra-estrutura CxFramework se encaixa nas categorias *middleware based systems* e *context-server based systems* apresentadas em [BAL07]. A Figura 4.3 mostra o mapeamento entre um *framework* conceitual separado em camadas, mostrado anteriormente na Figura 2.1, e o CxFramework.

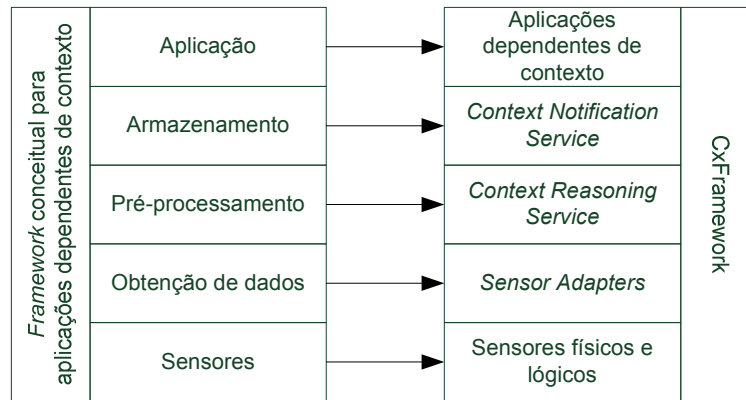


Figura 4.3: Mapeamento entre um *framework* conceitual e o CxFramework

No próximo capítulo será mostrado um exemplo de utilização do CxFramework em um cenário prático.

Capítulo 5

Cenário de Aplicação

Após o detalhamento da arquitetura do CxFramework e a abordagem de aspectos relacionados à implementação, este capítulo apresenta um cenário prático de aplicação do CxFramework na resolução de um problema.

5.1. Problema

Os gerentes de um grande supermercado identificaram a necessidade de automação de diversos pontos dentro do estabelecimento. Dentre esses pontos, foi decidido que, num primeiro momento, uma solução para automatizar três deles seria necessária, com os objetivos de melhorar as ações de marketing e, conseqüentemente, ampliar as vendas, e aumentar a satisfação dos clientes.

O primeiro ponto visa comunicar promoções de produtos aos clientes, uma vez que atualmente o supermercado não possui uma forma eficiente para fazê-lo. Assim, é necessário tornar esta comunicação mais eficiente.

O segundo ponto é relacionado às filas dos caixas. Atualmente, os clientes do supermercado perdem muito tempo procurando um caixa para pagar suas compras e, quando acham, às vezes os caixas estão cheios e o atendimento demora mais tempo do que deveria. Logo, é preciso implantar alguma solução que procure otimizar o tempo nas filas, fazendo o cliente esperar o mínimo possível.

O terceiro ponto é relacionado ao abandono dos carrinhos dentro do supermercado. Por diversas vezes, os clientes abandonam os carrinhos nos mais diversos lugares, o que gera dois problemas. O primeiro deles é que os carrinhos abandonados algumas vezes atrapalham a passagem de outros clientes dentro do supermercado; o segundo é que estes carrinhos abandonados ficam indisponíveis para novos clientes que chegam ao supermercado. Logo, é necessário que haja uma forma fácil de identificar a localização destes carrinhos abandonados para que os funcionários do supermercado possam removê-los dos locais onde se encontram.

Para resolver estes três problemas, foi decidido também que as aplicações a serem desenvolvidas não devem necessitar da interação direta do cliente para poderem funcionar adequadamente. Em outras palavras, estas aplicações devem se comportar em função das mudanças do ambiente onde estão inseridas, como se “adivinhassem” o que está acontecendo no supermercado para tomarem decisões. A idéia por trás disso é simplificar ao máximo a utilização destas aplicações por todos os clientes do supermercado, independentemente do seu nível de conhecimento a respeito de tecnologia.

A partir da descrição dos problemas a serem resolvidos, é possível concluir que as soluções devem ser baseadas em aplicações dependentes de contexto, isto é, aplicações que alteram seu comportamento de acordo com o ambiente onde estão inseridas. Mas desenvolver todo o código destas aplicações é uma tarefa complicada, já que seria necessário manusear informações de baixo nível de diversos sensores, além da impossibilidade de compartilhar as informações de contexto entre as três (ou talvez, no futuro, até mais) aplicações. Neste caso, fica claro que a melhor opção é utilizar uma infra-estrutura de suporte a aplicações dependentes de contexto, uma vez que ela tira das aplicações a responsabilidade de tratar questões referentes ao contexto. Desta forma, o CxFramework pode ser utilizado para atender as necessidades do supermercado.

5.2. Detalhamento do Cenário

A Figura 5.1 mostra um esquema do supermercado, e será usada como base para o entendimento do cenário onde o CxFramework será aplicado.

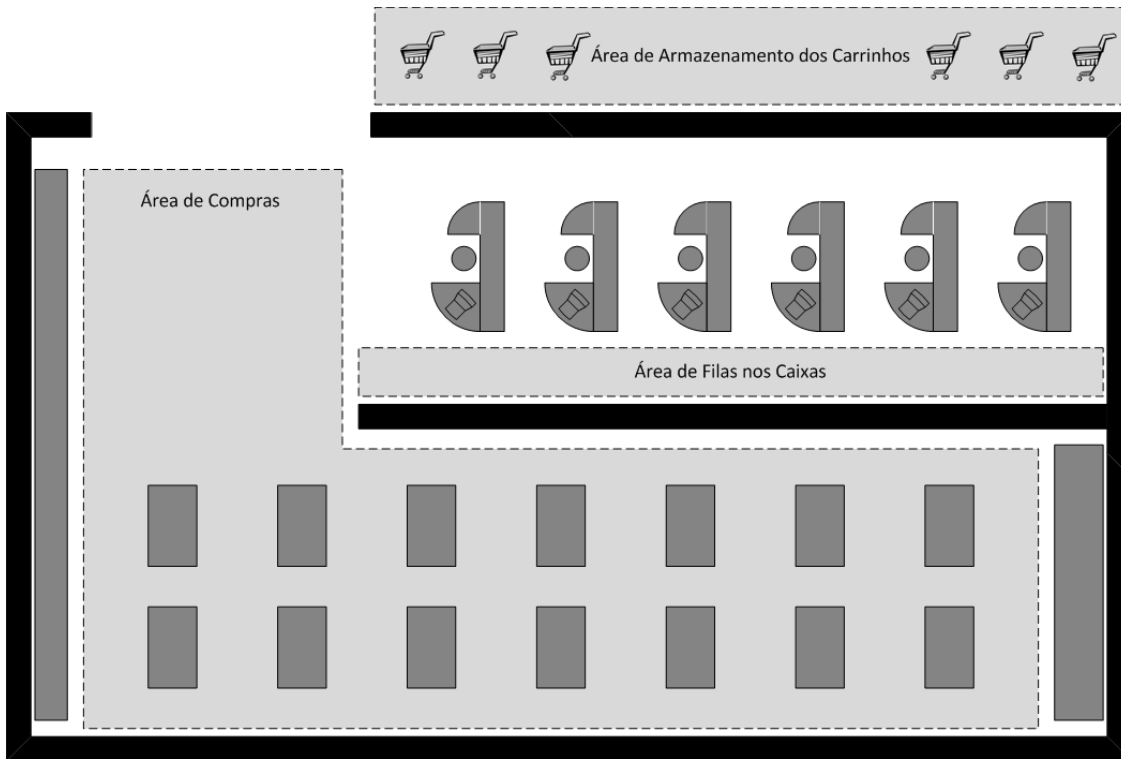


Figura 5.1: Representação gráfica do cenário

Cada carrinho do supermercado é equipado com um *display* capaz de transmitir algumas informações ao cliente. Os carrinhos que não estão sendo utilizados permanecem na área de armazenamento, mostrada na Figura 5.1.

Quando um cliente pega um carrinho, entra no supermercado para iniciar suas compras e começa a andar pelo supermercado (locomovendo-se pela área de compras, conforme mostra a Figura 5.1), o visor do carrinho mostra informações a respeito das promoções dos produtos localizados próximos ao local do cliente no supermercado.

Após terminar suas compras, o cliente se dirige para a área de filas dos caixas, também mostrada na Figura 5.1. Ao adentrar esta área, o visor do carrinho informa o cliente a respeito de qual caixa ele deve se dirigir, de forma a tentar fazer com que o seu atendimento seja o mais rápido possível.

Já quando clientes abandonam os carrinhos dentro do supermercado, os funcionários devem ser avisados através de seus PDAs a respeito da localização destes carrinhos, de forma que possam rapidamente devolvê-los à área de armazenamento de carrinhos.

5.3. Detalhamento da Solução

A solução para o problema proposto envolve diversas aplicações dependentes de contexto, as quais devem estar conectadas ao CxFramework. A Figura 5.2 mostra este cenário.

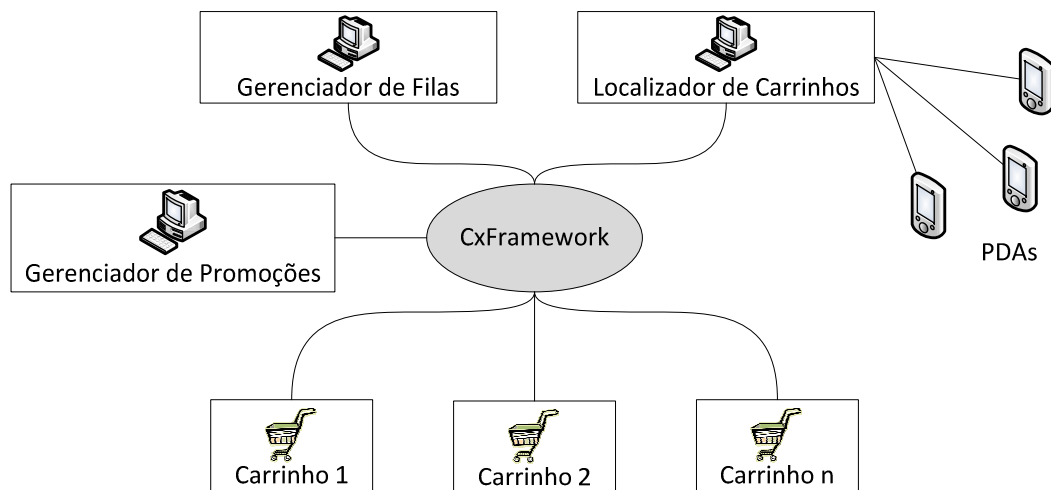


Figura 5.2: Diversas aplicações dependentes de contexto conectadas ao CxFramework

5.3.1. Gerenciador de Promoções

O Gerenciador de Promoções é a aplicação responsável por comunicar os clientes a respeito das promoções de produtos próximas ao local onde eles se encontram.

Para que esta aplicação funcione, ela deve cruzar informações existentes sobre as promoções atuais com a localização dos clientes dentro do supermercado. O CxFramework notifica esta aplicação sempre que os clientes se aproximam de determinada seção. Ao receber esta informação, o Gerenciador de Promoções busca as promoções pertinentes e notifica o CxFramework (através do seu *Application Sensor Adapter*), que por sua vez notificará o carrinho correspondente a respeito da mudança no contexto.

5.3.2. Gerenciador de Filas

O Gerenciador de Filas é a aplicação responsável por analisar o estado atual das filas nos caixas (levando em consideração critérios como tempo médio de atendimento, número médio de

pessoas aguardando, etc.) e direcionar o cliente para um caixa onde o tempo para o atendimento será supostamente menor.

Toda vez que um cliente adentra a área de filas nos caixas, o CxFramework notifica o Gerenciador de Filas a respeito deste fato. Ao ser notificada, a aplicação obtém o panorama atual e os dados estatísticos das filas, o que permite que ela chegue a uma conclusão a respeito de qual fila o cliente deve entrar. Quando o Gerenciador de Filas chega nesta conclusão, ele envia ao CxFramework (através do seu *Application Sensor Adapter*) a informação de que determinado cliente deve se dirigir à determinada fila. Na seqüência, o CxFramework notifica o carrinho correspondente (que também é considerada uma aplicação dependente de contexto) a respeito da mudança de contexto, isto é, para qual fila o cliente deve seguir. Esta informação é mostrada no visor do carrinho.

5.3.3. Localizador de Carrinhos

O Localizador de Carrinhos é a aplicação responsável por identificar carrinhos abandonados em diversas áreas do supermercado e avisar aos funcionários, a fim de que eles possam retorná-los à área de armazenamento de carrinhos.

Para que a aplicação funcione adequadamente, o CxFramework deve notificá-la toda vez que determinado carrinho for considerado como abandonado (a Seção 5.3.4 abordará o processo utilizado pelo CxFramework para chegar a esta conclusão). Uma vez notificado, o Localizador de Carrinhos envia as informações do local onde o carrinho foi abandonado para os PDAs dos funcionários do supermercado, para que eles possam remover o carrinho do local.

5.3.4. Carrinhos de Compras

Cada carrinho do supermercado é considerado, neste cenário, uma aplicação dependente de contexto conectada ao CxFramework. Eles devem ser configurados para serem notificados nestas situações:

- quando o Gerenciador de Promoções envia avisos de promoções;
- quando o Gerenciador de Filas envia um aviso informando para qual fila de caixa o cliente deve se dirigir.

Nos dois casos acima, as informações vindas do CxFramework são traduzidas pelo software do carrinho como avisos para o cliente no visor.

Para que a solução funcione, cada carrinho deve ser capaz de fornecer três informações ao CxFramework. A primeira delas é a localização do carrinho dentro do supermercado, em forma de coordenadas. Esta informação é importante para que o CxFramework possa expressar a localização do carrinho em mais alto nível, concluindo a respeito da seção onde ele está ou mesmo a área onde se encontra (área de compras, área de filas dos caixas, etc.). A segunda informação é o tempo que o carrinho está parado, que auxiliará na tarefa de determinar se o carrinho foi abandonado. A terceira informação é a massa total do carrinho (massa do carrinho somada com a massa dos produtos dentro dele). Esta informação, combinada com a informação de localização e o tempo que o carrinho está parado, permitirá saber se o carrinho foi abandonado pelo cliente. Por exemplo, o CxFramework é capaz de concluir que, se um carrinho possui massa total igual a sua própria massa, o carrinho está vazio. Além disso, se o carrinho está localizado na área de compras e a sua localização não foi alterada nos últimos minutos, o CxFramework pode concluir que o carrinho foi realmente abandonado e disparar uma notificação para o Localizador de Carrinhos. As informações de localização e massa total devem ser atualizadas conforme o cliente se locomove pelo supermercado e coloca produtos no carrinho. Já a informação do tempo que o carrinho está parado deve ser atualizada de acordo com uma taxa fixa de atualização (a cada 60 segundos, por exemplo). Os carrinhos utilizam os seus *Application Sensor Adapters* para avisarem o CxFramework a respeito das mudanças no contexto.

5.4. Ontologia e Regras de Inferência

Para modelar as informações de contexto neste cenário, foi criada uma ontologia. O estado do contexto em um determinado momento para um carrinho é exemplificado na Figura 5.3, que é uma representação gráfica da ontologia.

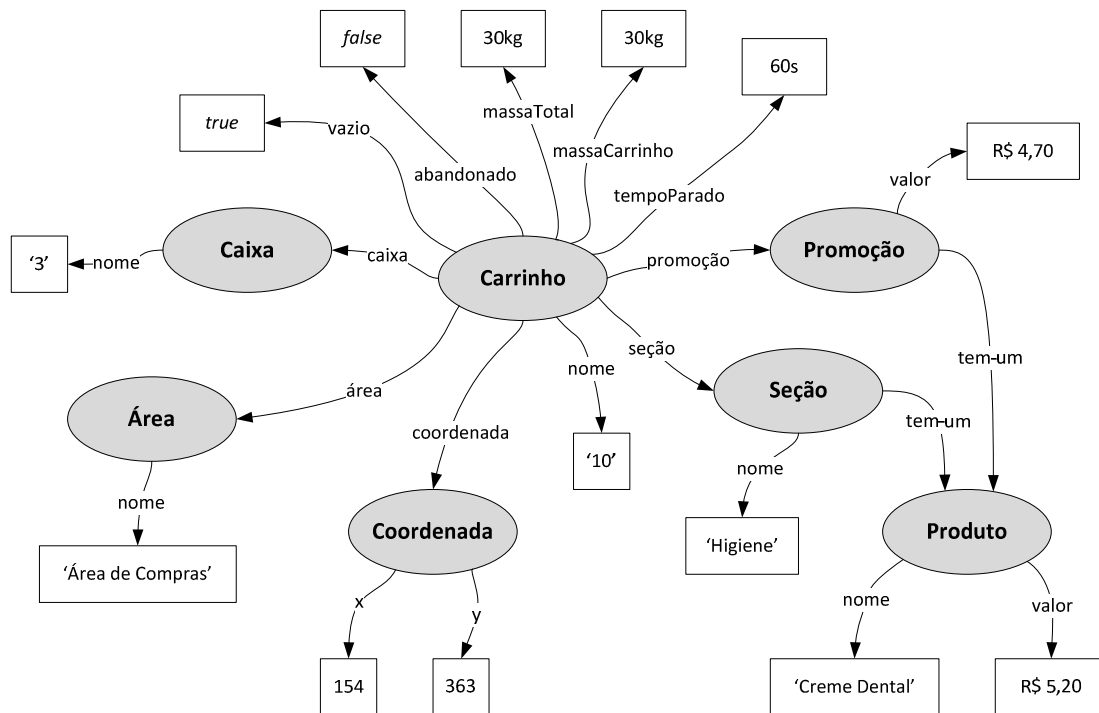


Figura 5.3: Diagrama exemplificando o estado do contexto em determinado momento

É possível perceber que a ontologia é composta por sete classes, cada uma possuindo seus atributos e seus relacionamentos com outras classes. É importante perceber que a Figura 5.3 representa o estado de um carrinho específico num determinado momento (no caso, o Carrinho 10, de acordo com o seu atributo **nome**), o que significa que cada carrinho poderá possuir também os mesmos atributos e relacionamentos, mas provavelmente com valores diferentes.

Na Figura 5.4 é possível visualizar parte do mesmo estado representado na Figura 5.3, mas desta vez em formato OWL formatado em XML.


```

<owl:Class rdf:ID="Carrinho" />
<owl:Class rdf:ID="Coordenada" />
<owl:FunctionalProperty rdf:ID="vazio">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdfs:domain rdf:resource="#Carrinho"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="coordenada">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#Carrinho"/>
  <rdfs:range rdf:resource="#Coordenada"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="nome">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Área"/>
        <owl:Class rdf:about="#Seção"/>
        <owl:Class rdf:about="#Carrinho"/>
        <owl:Class rdf:about="#Produto"/>
        <owl:Class rdf:about="#Caixa"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>

```

Figura 5.4: Exemplo de ontologia expressa em formato OWL sobre XML

A linguagem OWL não tem a intenção de ser uma linguagem simples para ser entendida pelas pessoas. Apesar disso, em ontologias menos complexas, é possível compreendê-la de maneira fácil. O trecho da linguagem mostrado na Figura 5.4 inicia com a definição das classes **Carrinho** e **Coordenada**. Na seqüência, são definidas as propriedades **vazio**, **coordenada** e **nome**, juntamente com outras informações como o *domain*, o *range* e o *type* de cada uma delas.

Juntamente com a ontologia, este cenário também deve possuir regras de inferência bem definidas. Estas regras serão utilizadas pelo *Context Reasoning Service* para inferir as informações de contexto de alto nível que efetivamente interessam às aplicações dependentes de contexto, a partir das informações vindas dos *Sensor Adapters* (incluindo também os *Application Sensor Adapters*).

Como exemplo, duas possíveis regras de inferência para um cenário como este poderiam ser as seguintes:

```
[defineAreaCompras:
  (?carrinho ns:coordenada ?coord),
  (?coord ns:x ?posX),
  (?coord ns:y ?posY),
  greaterThan(?posX, 200),
  lessThan(?posX, 300),
  greaterThan(?posY, 200),
  lessThan(?posY, 300)
->
  (?carrinho ns:area ns:AreaCompras)]

[defineAbandono:
  (?carrinho ns:area ns:AreaCompras),
  (?carrinho ns:vazio "true"),
  (?carrinho ns:tempoParado ?t),
  greaterThan(?t, 300)
->
  (?carrinho ns:abandonado "true")]
```

As regras acima estão especificadas no formato das regras de inferência suportado pelo Jena. A tradução de cada uma das regras poderia ser escrita da seguinte forma:

- Regra **defineAreaCompras**: verifique a coordenada X e Y do carrinho e, se a coordenada X estiver entre 200 e 300 e a coordenada Y estiver entre 200 e 300, defina a área do carrinho como “Área de Compras”.
- Regra **defineAbandono**: se a área do carrinho está definida como “Área de Compras”, e se o carrinho está vazio, e se o tempo que o carrinho está parado for superior a 300 (segundos), defina o atributo **abandonado** do carrinho como verdadeiro.

Estas regras são responsáveis, respectivamente, por definir dentro de qual área o carrinho está localizado e se o carrinho foi abandonado por um cliente. Fica claro que, para o sistema funcionar corretamente, outras regras de inferência precisam definidas, além das duas já exemplificadas nesta Seção.

5.5. Utilizando Federação de Contexto

O cenário de utilização do CxFramework deste Capítulo pode ser estendido para aproveitar outra característica importante proporcionada pela infra-estrutura: a federação de contextos.

Além do ambiente do supermercado já retratado, o estabelecimento conta ainda com um estacionamento de veículos no subsolo. É razoável supor que estes dois ambientes (supermercado e estacionamento) sejam bastante distintos. Em outras palavras, são ambientes diferentes que possuem contextos igualmente diferentes agindo sobre eles.

Muitos dos clientes que finalizam suas compras se dirigem ao estacionamento com seus carrinhos, a fim de poderem colocar as compras em seus carros. Normalmente, após fazerem isto, estes clientes abandonam os carrinhos no próprio ambiente do estacionamento.

O abandono do carrinho no estacionamento cria um novo obstáculo à solução já proposta. Como o estacionamento é um ambiente totalmente diferente do supermercado, notificar o Localizador de Carrinhos a respeito de carrinhos abandonados no estacionamento não é possível num primeiro momento. Para tornar isto possível é necessário que, de alguma forma, os ambientes troquem informações, mesmo sendo diferentes.

Para possibilitar este tipo de comunicação, o CxFramework lança mão da federação de contextos. A Figura 5.5 mostra como a federação de contextos pode ser aplicada num cenário como este.

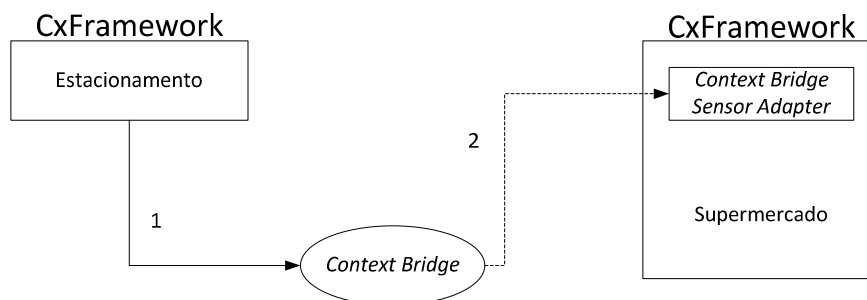


Figura 5.5: Esquematização do funcionamento da federação de contextos

O funcionamento da federação de contextos nada mais é do que utilizar a capacidade de reflexão da infra-estrutura de outra forma. Uma aplicação dependente de contexto, aqui chamada de *Context Bridge*, é responsável por fazer uma “ponte” entre os dois contextos. Ela se conecta aos dois ambientes e repassa os dados que recebe do primeiro ambiente (através de notificação) para o segundo ambiente (através do seu *application sensor adapter*).

Considerando os cenários do estacionamento e do supermercado, o *Context Bridge* inicialmente se registra nos dois CxFrameworks e é configurado para receber notificações a respeito de carrinhos abandonados no estacionamento. Uma vez que um evento deste tipo é recebido, ele utiliza o seu *Application Sensor Adapter* para alterar o contexto do supermercado, informando que determinado carrinho foi abandonado. Desta forma, o ambiente do supermercado passa a conhecer este novo fato e é capaz de notificar o Localizador de Carrinhos de forma adequada.

5.6. Conclusão

Este capítulo mostrou um cenário onde o CxFramework pode ser utilizado de forma a resolver um problema real. O problema foi levantado e a solução foi proposta e detalhada, de maneira a aproveitar todos os recursos da infra-estrutura.

É importante notar que o cenário proposto utiliza a reflexividade de forma constante. É fácil observar isso ao atentar para o fato de que os carrinhos do supermercado, considerados aplicações dependentes de contexto, modificam o contexto onde estão inseridos conforme são movidos pelo cliente dentro do supermercado, por exemplo. Além disso, a federação de contextos, utilizada para comunicar os ambientes do supermercado e do estacionamento, só é possível devido à capacidade de reflexão. Desta forma, uma infra-estrutura não-reflexiva não seria capaz de reproduzir este mesmo cenário de forma satisfatória, já que infra-estruturas desta natureza não levam em consideração o fato de que as próprias aplicações conectadas a ela são capazes de mudar o contexto.

No próximo capítulo serão mostrados os aspectos de implementação acerca dos componentes do CxFramework. Além disso, serão discutidos os tópicos de coleta de lixo e escalabilidade, juntamente com a apresentação de uma avaliação de performance da infra-estrutura.

Capítulo 6

Aspectos de Implementação e Avaliação

6.1. Aspectos de Implementação

O CxFramework foi implementado na linguagem Java sobre o OpenORB [OPE08], uma implementação em Java da especificação CORBA. Cada serviço da infra-estrutura possui sua própria IDL e é representada por um objeto CORBA.

Antes de colocar a infra-estrutura em funcionamento, o administrador do CxFramework precisa configurar e iniciar cada um dos serviços adequadamente. Além disso, os serviços não podem ser iniciados de forma aleatória, pois existem dependências entre eles. A Figura 6.1 mostra um grafo onde os vértices representam cada uma das tarefas que devem ser completadas pelo administrador para colocar a infra-estrutura toda em execução. Basicamente, as tarefas se resumem a escrever arquivos de configuração e iniciar os serviços. As arestas do grafo indicam relações de dependência entre os vértices sucessores e predecessores. Por exemplo, o *Context Reasoning Service* só pode ser iniciado após o *Registry Service* ter sido iniciado e todos os cinco arquivos de configuração do serviço terem sido corretamente criados. O CxFramework só poderá ser totalmente inicializado se todas as tarefas forem realizadas de acordo com as regras de dependência já mencionadas.

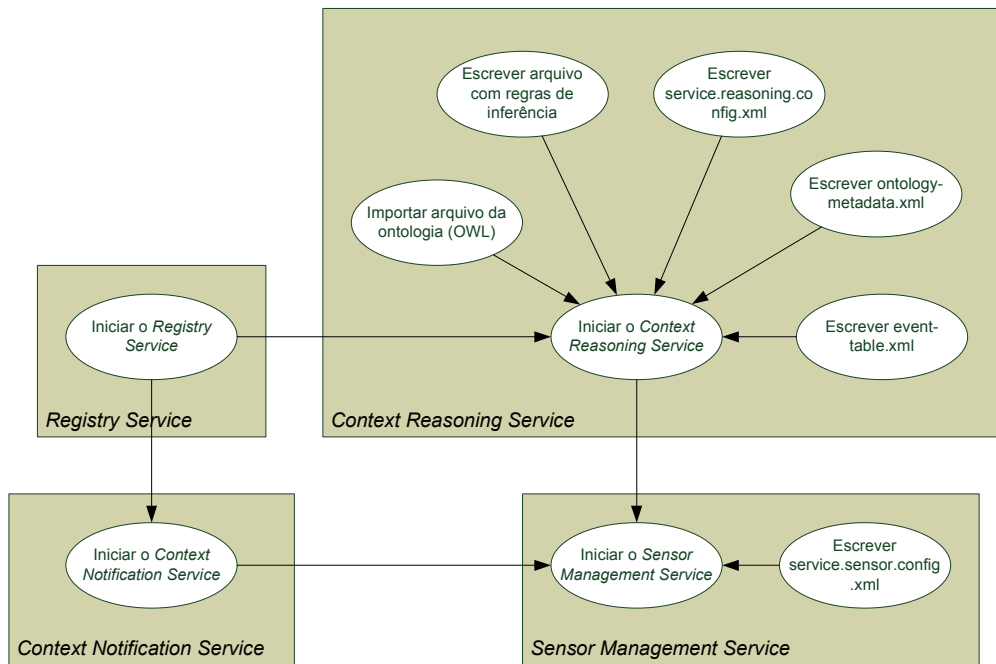


Figura 6.1: Tarefas a serem realizadas para colocar o CxFramework em funcionamento

Cada um dos arquivos de configuração citados na Figura 6.1 é explicado na Tabela 6.1.

Tabela 6.1: Arquivos de configuração do CxFramework

Serviço	Arquivo	Formato	Descrição
Context Reasoning Service	service.reasoning.config.xml	XML	Configuração de algumas propriedades relacionadas ao serviço.
Context Reasoning Service	ontology-metadata.xml	XML	Lista de todos os eventos de contexto suportados pela infraestrutura.
Context Reasoning Service	event-table.xml	XML	Mapeamento entre <i>sensor adapters</i> e elementos da ontologia.
Context Reasoning Service	Arquivo com a ontologia	OWL formatado em XML	Ontologia suportada pela infraestrutura.

Context Reasoning Service	Arquivo com as regras de inferência	Formato proprietário do Jena [JEN08]	Regras de inferência a serem aplicadas sobre a ontologia.
Sensor Management Service	service.sensor.config.xml	XML	Configuração dos <i>sensor adapters</i> que fazem parte da infra-estrutura.

O *Registry Service* encapsula o CORBA *Naming Service* e *Trading Service*. Desta forma, tudo o que é registrado no *Registry Service* é internamente registrado em um desses dois serviços.

O *Sensor Adapter provider* deve implementar os *Sensor Adapters*, os quais também são objetos CORBA, e registrá-los no *Sensor Management Service*. Existe um arquivo de configuração em formato XML que contém a lista de *Sensor Adapters* registrados no serviço. Cada *Sensor Adapter* deve ter seu próprio ID único e uma classe Java que o representa (esta classe deve implementar uma interface comum a todos os *Sensor Adapters*). Quando o serviço é iniciado, o arquivo de configuração é lido e os *Sensor Adapters* são instanciados e iniciados.

O *ontology provider* deve fornecer a ontologia e as regras de inferência para configurar o *Context Reasoning Service*. O CxFramework suporta ontologias especificadas em linguagem OWL [WOR08] e formatadas em XML. Ferramentas desenvolvidas por terceiros, como o Protégé [PRO08] podem ser utilizadas para criar a ontologia. Ferramentas desta natureza normalmente são capazes de exportar as ontologias criadas em formato OWL. O motor de inferência de ontologias utilizado pelo CxFramework é o framework Jena [JEN08]. Conseqüentemente, o formato das regras de inferência fornecidas pelo administrador da infra-estrutura devem seguir o formato estabelecido pelo Jena.

Outro componente importante, que deve ser fornecido pelo administrador na configuração do *Context Reasoning Service*, é a *Event Table*. A *Event Table* é uma tabela que mapeia um *Sensor Adapter* a uma propriedade na ontologia. Por exemplo, um *Sensor Adapter* chamado **S1** é responsável por fornecer a temperatura de uma **sala**. Neste cenário, a *Event Table* mapearia **S1** a um indivíduo da ontologia chamado **sala** e a um atributo chamado **temperatura**. Através da utilização desta tabela, o serviço é capaz de saber qual parte da ontologia será alterada quando o

Sensor Adapter enviar uma notificação de mudança de contexto. A *Event Table* também é configurada por um arquivo XML.

É importante perceber que, apesar das tarefas do administrador da infra-estrutura demandarem grande esforço, esta abordagem simplifica o desenvolvimento de aplicações dependentes de contexto, já que todo código relacionado ao contexto é encapsulado pela infra-estrutura.

O *Context Notification Service* é responsável por notificar a aplicação a respeito de mudanças no contexto. A Figura 6.2 mostra seus componentes.

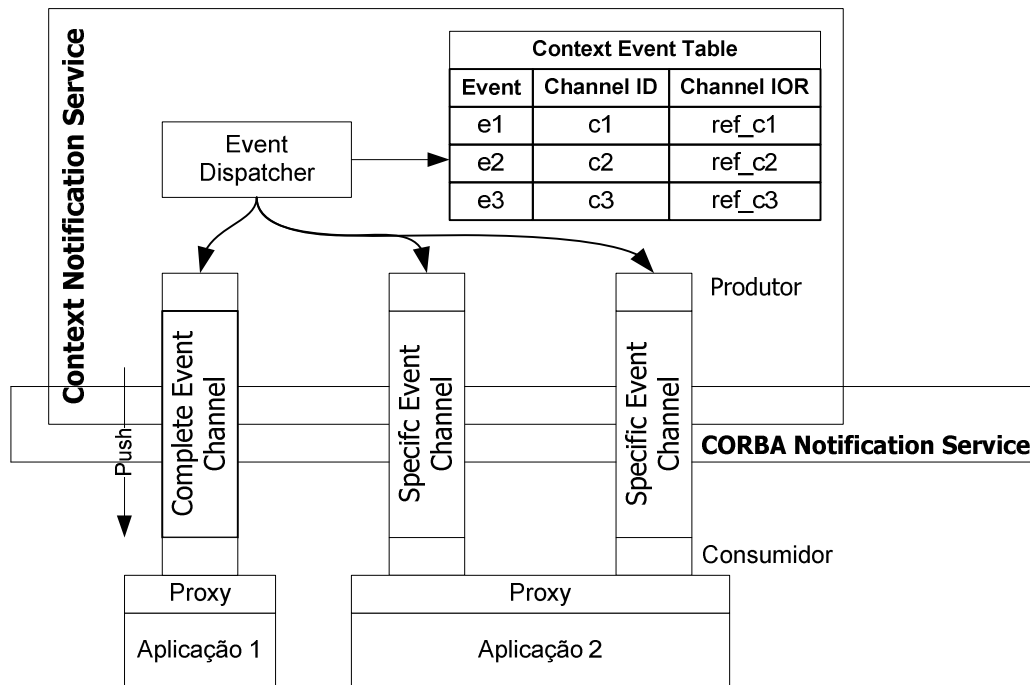


Figura 6.2: Arquitetura do *Context Notification Service*

A comunicação entre o *Context Notification Service* e as aplicações acontece através de canais de eventos do *CORBA Notification Service* [SIE00], onde o *Context Notification Service* é o produtor e a aplicação é o consumidor da informação. Os canais de eventos possuem a disciplina de acesso FIFO, onde a primeira informação a ser produzida é a primeira a ser consumida, garantindo uma ordem de entrega dos eventos. As aplicações podem ser conectadas

ao *Complete Event Channel* (como a *Application 1* mostrada na Figura 6.2). Quando isto acontece, a aplicação será notificada a respeito de qualquer mudança de contexto. Já quando conectada a um *Specific Event Channel*, a aplicação será notificada apenas da mudança de contexto que o canal gerencia (os *Specific Event Channels* são responsáveis apenas por um tipo de mudança de contexto). É importante mencionar novamente que existe um *proxy* entre a aplicação e os canais de eventos por questões de transparência. Além disso, por questões de otimização, os *Specific Event Channels* são criados apenas quando alguma aplicação solicita o registro no canal. Isto significa que estes canais são criados dinamicamente sob demanda.

Quando o *Context Notification Service* recebe uma mudança de contexto vinda do *Context Reasoning Service*, ele imediatamente coloca o evento de contexto no *Complete Event Channel*. Depois disso, o serviço procura na *Context Event Table* qual é o canal responsável por entregar às aplicações o evento de contexto ocorrido. Se nenhuma entrada na tabela for encontrada, significa que nenhuma aplicação está interessada no evento. Neste caso, o evento é descartado. Já se existir a entrada, significa que alguém está interessado em ser notificado a respeito do evento. As entradas na tabela são criadas no momento em que as aplicações registram o interesse em determinado evento. Esta entrada associa o evento de contexto com um ID único de canal de eventos. Além disso, este novo evento é registrado no *Registry Service*, permitindo que as aplicações que usam o *Registry Service* para procurar por eventos de contexto sejam capazes de encontrá-lo. Por exemplo, um possível evento de contexto poderia ser o recurso **sala** (indivíduo da ontologia) ligado à propriedade **temperatura** (atributo da ontologia), o que poderia ser descrito como **temperatura da sala**. O *Registry Service* registra os objetos de eventos de contexto no *CORBA Trading Service*.

Se o evento for encontrado na tabela, o serviço verifica se o canal já foi criado. Caso não tenha sido, nada acontece. Caso contrário, o evento de contexto é colocado no *Specific Event Channel* associado.

Como explicado no Capítulo 4, as aplicações possuem duas formas de obter referências aos *Specific Event Channels* quando o modo de comunicação *push* é utilizado. Quando elas buscam eventos de contexto utilizando o *Registry Service*, são retornados eventos de contexto de acordo com as características desejadas, juntamente com as devidas referências aos *Specific Event Channel Proxies*. O mesmo acontece quando as aplicações se registram no *Complete Event*

Channel. Eventos de contexto colocados neste canal também estão associados com seus *Specific Event Channel Proxies* correspondentes. Logo, em ambas as situações, as aplicações são capazes de se conectar aos canais de eventos que as interessam. Quando o modo de comunicação *pull* é utilizado, os canais de eventos não são utilizados. Neste caso, as aplicações buscam diretamente um evento de contexto previamente conhecido e recebem seu valor corrente de forma síncrona. Outros componentes importantes, que devem ser fornecidos pelo administrador na configuração do *Context Reasoning Service*, são a *Event Table* e eventos de contexto suportados pela infraestrutura. A *Event Table* é uma tabela que mapeia um *Sensor Adapter* a uma propriedade na ontologia. Por exemplo, um *Sensor Adapter* chamado **S1** é responsável por fornecer a temperatura de uma **sala**. Neste cenário, a *Event Table* mapearia **S1** a um indivíduo da ontologia chamado **sala** e a um atributo chamado **temperatura**. Através da utilização desta tabela, o serviço é capaz de saber qual parte da ontologia será alterada quando o *Sensor Adapter* enviar uma notificação de mudança de contexto. A *Event Table* também é configurada por um arquivo XML. Já os eventos de contexto suportados pela infra-estrutura (definidos no arquivo *ontology-metadata.xml*) precisam ser informados a fim de que as aplicações dependentes de contexto possam buscar por eventos nesta lista para que possam registrar o interesse de serem notificadas. Ao ser iniciado, o *Context Reasoning Service* lê este arquivo e armazena os eventos de contexto suportados no *Registry Service*, que por sua vez registra, internamente, os objetos de eventos de contexto no *CORBA Trading Service*, possibilitando a busca destes objetos através de suas características.

É importante perceber que, apesar das tarefas do administrador da infra-estrutura demandarem grande esforço, esta abordagem simplifica o desenvolvimento de aplicações dependentes de contexto, já que todo código relacionado à contexto é encapsulado pela infra-estrutura.

6.2. Coleta de Lixo

O CxFramework funciona num cenário dinâmico. Conseqüentemente, um mecanismo eficiente de coleta de lixo é necessário.

O coletor de lixo (*garbage collector*) não é considerado um serviço da infra-estrutura por não expor nenhuma interface. Ele é um agente que age em diversas partes da infra-estrutura, de tempos em tempos, e coleta tudo o que não está mais sendo usado pelas aplicações.

O primeiro cenário é aplicado aos *Specific Event Channels*. Cada canal tem um TTL (tempo de vida ou *time to live*), baseado no último acesso. Quando o TTL é alcançado, o *garbage collector* pede ao *Context Notification Service* para destruir o canal. Mas antes de fazer isto, o *Context Notification Service* avisa as aplicações conectadas ao canal que o canal está prestes a ser destruído, permitindo que elas decidam o que querem fazer (esta funcionalidade é suportada pelo *CORBA Notification Service*, conforme descrito na Seção 2.5.1). Se as aplicações decidirem se conectar novamente ao canal, o canal será criado de novo. Já se nenhuma aplicação estiver conectada ou solicitar reconexão ao canal, o canal permanecerá inexistente.

O segundo cenário é aplicado aos eventos contidos na *Context Event Table* (que faz parte do *Context Notification Service*) e eventos registrados no *Registry Service*. Estes eventos também possuem um TTL associado. Eventos cujo TTL expirou e que estão associados a canais de eventos que não estão criados, podem ser removidos do *Context Event Table* e do *Registry Service* porque eles estão ativos por bastante tempo e nenhuma aplicação registrou interesse nestes eventos. Este cenário está diretamente ligado ao primeiro cenário explicado anteriormente. O coletor de lixo primeiro deve solicitar a destruição do canal (como explicado no primeiro cenário) e depois fazer a checagem dos eventos que podem ser destruídos.

O terceiro cenário é relacionado aos *Application Sensor Adapters*, os quais são dinâmicos, isto é, existem durante o período que a aplicação está conectada ao CxFramework. Quando as aplicações desconectam, seus *Application Sensor Adapters* devem ser destruídos, juntamente com os canais gerados pra notificar seus eventos, já que eles não serão mais utilizados. Nesta situação, aplicações devem marcar seus *Application Sensor Adapters* como destrutíveis antes de desconectar do CxFramework. Fazendo isto, o coletor de lixo é capaz de identificar quais *Application Sensor Adapters* podem ser destruídos (a destruição é feita pelo *Sensor Management Service* a pedido do coletor de lixo).

6.3. Escalabilidade

Escalabilidade é outro importante requisito relacionado à coleta de lixo de infra-estruturas dependentes de contexto que fornecem informações de contexto para diversas aplicações. Conseqüentemente, o CxFramework deve suportar um grande número de aplicações com um impacto mínimo de performance.

Conforme o número de aplicações conectadas ao CxFramework aumenta, é razoável imaginar que o *Context Notification Service* deverá aumentar a quantidade de notificações. Este aspecto depende bastante do *middleware* utilizado na implementação da infra-estrutura. Isto não deve ser um problema para o CORBA, já que o *CORBA Notification Service* foi desenhado para suportar uma grande quantidade de clientes conectados aos seus canais [GOR01]. Um problema maior está relacionado à grande quantidade de *Application Sensor Adapters* que serão criados quando as aplicações se conectarem ao CxFramework.

A quantidade de *Application Sensor Adapters*, e também de *Sensor Adapters* em geral, tem impacto sobre o tamanho da ontologia. Ontologias maiores implicam em maior tempo para inferir novas ontologias derivadas. Este é o mesmo impacto que existe quando a ontologia é muito complexa. De acordo com [GU05], o tempo de inferência é diretamente proporcional ao tamanho da ontologia. Isto também impacta nos tamanhos da *Context Event Table* e dos eventos registrados no *Registry Service*, que deverão armazenar uma quantidade maior de informações.

A Equação 4.1, obtida a partir de simulações de diferentes cenários, mostra a relação entre o número de *sensor adapters* (N) e o tempo de inferência de uma ontologia (t). É possível perceber que o aumento do tempo acontece de forma linear e é diretamente proporcional à quantidade de *sensor adapters*. Já a Equação 4.2, também obtida através de simulações, mostra a relação entre o número de *sensor adapters* (N) e a quantidade de memória ocupada pela ontologia (m). Esta relação, assim como na relação anterior, também possui crescimento linear. O método de obtenção das equações está detalhado no Apêndice A.

$$t = 0,4N \quad (6.1)$$

$$m = 2,2N \quad (6.2)$$

A primeira coisa a ser levada em consideração é que a noção de contexto no campo computacional deve ser considerada limitada, isto é, ela representa o contexto em um ambiente limitado. Isto se deve ao fato de que a dependência de contexto possui tipicamente um caráter local, ou, em outras palavras, limitada a um determinado espaço físico. Como o CxFramework é capaz de representar dados de contexto de um ambiente específico, é esperado que a quantidade de regras de inferência de ontologias sejam também limitadas (a quantidade de regras de inferências está diretamente relacionada com a quantidade de ontologias que podem ser inferidas). Além disso, num ambiente limitado é razoável supor que centenas de aplicações podem estar conectadas simultaneamente à infra-estrutura, mas não milhares ou milhões. Mesmo que o tempo de inferência aumente quando aplicações se conectam à infra-estrutura, ainda assim o tempo de inferência seria aceitável considerando que as aplicações dependentes de contexto não são aplicações onde o tempo é um fator crítico (como em aplicações de tempo real, por exemplo).

6.4. Avaliação de Desempenho

Para demonstrar como o CxFramework se comporta de acordo com a variação de *sensor adapters* e aplicações, foram realizadas medições de tempo considerando dois cenários diferentes. Para todos os cenários, os *sensor adapters* geraram eventos em intervalos de tempo aleatórios, variando entre 0 e 5 segundos. Já as aplicações foram configuradas de forma a serem notificadas a respeito de todas as mudanças de contexto inferidas. A medição de tempo relativa a cada cenário foi realizada três vezes, e o tempo final foi obtido através da média aritmética entre eles. Além disso, para todos os cenários foi utilizado o mesmo conjunto de regras de inferência, composto por oito regras. As medições foram feitas com todos os serviços da infra-estrutura sendo executados num único computador, um PC com processador Core 2 DUO 2 GHz com 2 GB RAM. É importante perceber que todas as medições relativas aos *sensor adapters* valem também para os *application sensor adapters*, que são gerenciados pelas aplicações e possibilitam a reflexividade.

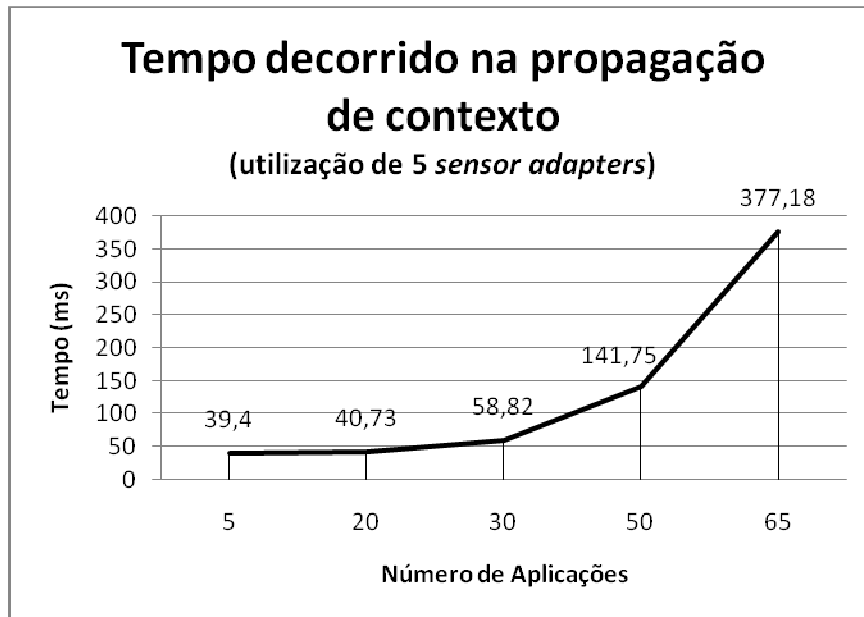


Figura 6.3: Cenário 1, onde o número de aplicações varia

A Figura 6.3 mostra os tempos medidos no Cenário 1. Os tempos do gráfico representam o tempo médio (em milissegundos) decorrido para que uma mudança de estado de um sensor seja propagada em forma de informação de contexto de alto nível até as aplicações dependentes de contexto interessadas na mudança do contexto. Em outras palavras, este é o tempo decorrido para que uma alteração sentida por sensor em uma ponta chegue até a aplicação localizada na outra ponta da infra-estrutura.

Neste cenário, o número de *sensor adapters* foi mantido constante (foram utilizados 5 *sensor adapters*), enquanto o número de aplicações foi sendo alterado para cada medição. O objetivo desta simulação é compreender qual o impacto que o aumento no número de aplicações causa na infra-estrutura.

É possível perceber que a curva no gráfico é exponencial, mas mesmo para 65 aplicações conectadas à infra-estrutura, o tempo ainda pode ser considerado pequeno (377,18 ms). Logo, é possível concluir que o aumento no número de aplicações não causa queda de desempenho significativa. Isto se deve ao fato de que a notificação das aplicações a respeito das mudanças de contexto é feita pelo *CORBA Notification Service*, que é um serviço que já foi criado com o intuito de ser escalável.

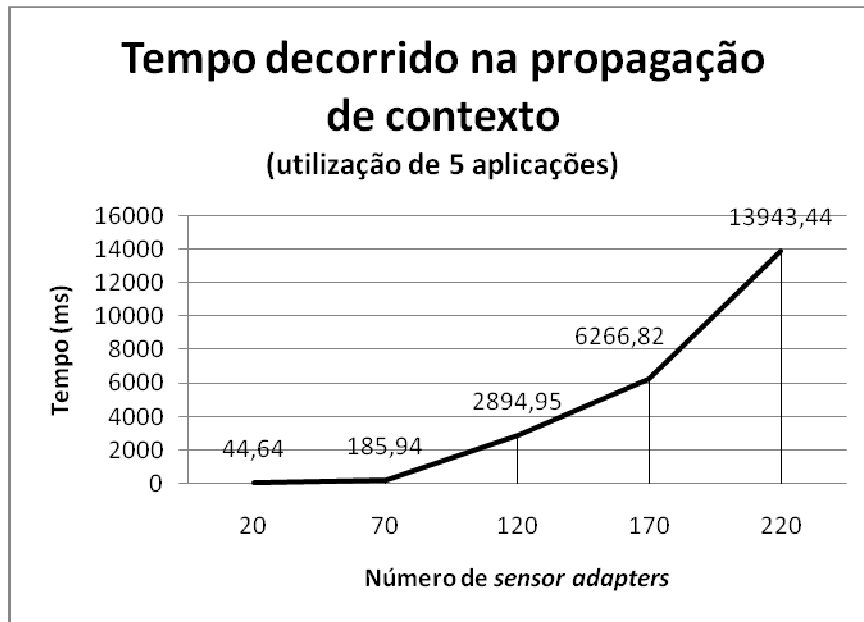


Figura 6.4: Cenário 2, onde o número de *sensor adapters* varia

Já o Cenário 2 mostra uma situação inversa à do Cenário 1. Enquanto o número de aplicações foi mantido constante (foram utilizadas 5 aplicações), o número de *sensor adapters* foi sendo alterado a cada medição. O objetivo desta simulação é visualizar o comportamento da infra-estrutura conforme novos *sensor adapters* vão sendo adicionados. Os resultados podem ser vistos na Figura 6.4.

Neste caso, diferente do que aconteceu no Cenário 1, a queda de desempenho foi sentida rapidamente. A curva exponencial no gráfico que representa o tempo aumenta de forma muito mais agressiva do que no Cenário 1, o que mostra que o aumento de *sensor adapters* provoca uma queda de desempenho muito maior do que o aumento de aplicações. A explicação deste fenômeno está diretamente ligada à inferência de novas ontologias. Quanto maior o número de *sensor adapters* gerando eventos, mais vezes novas ontologias serão inferidas. Isto faz com que a inferência, que é um processo relativamente pesado computacionalmente, vire um gargalo.

Além das simulações considerando os cenários anteriores, foram realizadas também medições de tempo procurando representar ambientes de pequeno, médio e grande porte. O

objetivo destas novas simulações são demonstrar a viabilidade de utilização do CxFramework em ambientes reais. A Tabela 6.2 detalha como cada um destes cenários foi configurado.

Tabela 6.2: Diferentes cenários de execução

Cenário	Núm. <i>Sensor Adapters</i>	Núm. Aplicações
Ambiente de pequeno porte (PP)	50	5
Ambiente de médio porte (MP1)	100	10
Ambiente de grande porte (MP2)	200	20
Ambiente de grande porte (GP)	400	40

Para a execução desta nova simulação, valem os mesmos critérios e configurações detalhados no início desta seção.

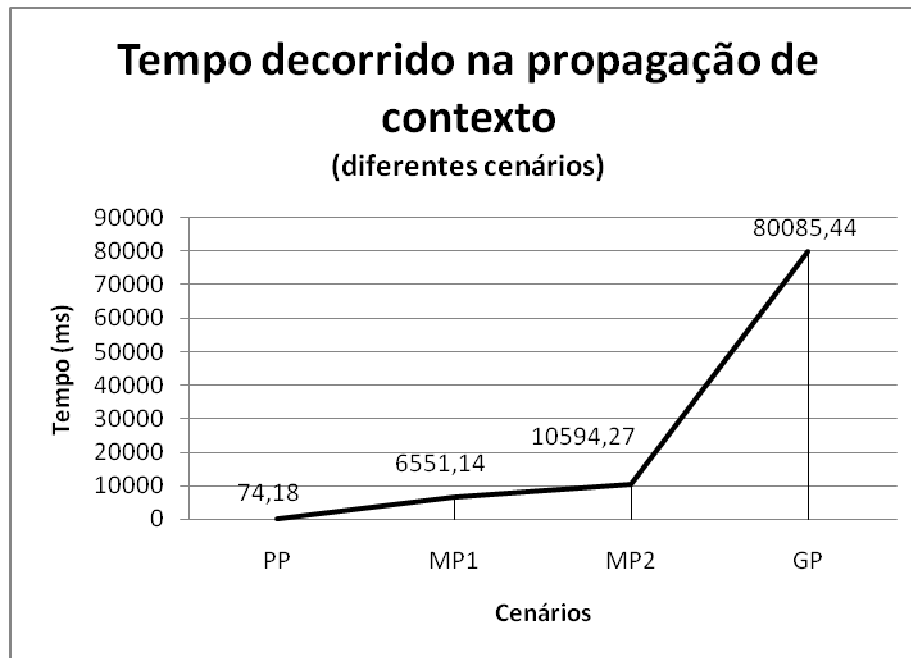


Figura 6.5: Gráfico de medição de tempo em diferentes cenários

A Figura 6.5 mostra o tempo médio (em milissegundos) que um evento leva para ser gerado pelo *sensor adapter* até o momento que a aplicação é notificada a respeito da mudança de contexto, da mesma forma como nas simulações anteriores.

No cenário que representa o ambiente de pequeno porte (PP), o tempo médio de execução foi bastante pequeno (74,18 ms). Já nos ambientes de médio porte (MP1 e MP2), os tempos subiram para 6551,14 ms e 10594,27 ms, respectivamente. Mesmo sendo valores bem maiores do que o obtido no ambiente de pequeno porte, eles ainda podem ser considerados pequenos se o contexto representado é estático, isto é, não muda rapidamente (a classificação de contexto é explicada na Seção 2.1.2). Isto mostra a viabilidade da utilização do CxFramework em ambientes de pequeno e médio porte, os mais típicos. Já no cenário que representa um ambiente de grande porte, o tempo médio foi bem maior (80085,44 ms). Desta forma, para representar ambientes maiores, seria necessário utilizar um hardware melhor ou separar os serviços do CxFramework entre diversas máquinas interligadas em rede, o que aumentaria a capacidade de processamento (esta capacidade de distribuição dos serviços na rede já está prevista na arquitetura). Também poderia ser utilizada a federação de contextos como forma de evitar o gargalo do processamento das informações. Atitudes como estas ajudariam a reduzir sensivelmente o tempo médio do fluxo de um evento de contexto dentro da infra-estrutura.

6.5. Conclusão

Este Capítulo apresentou o CxFramework sobre um ponto de vista de implementação. Também foram discutidos assuntos como coleta de lixo e escalabilidade, além da realização de testes para medir o desempenho e a viabilidade da utilização na infra-estrutura na prática. Com a análise dos resultados, foi possível concluir que o CxFramework é escalável e pode ser utilizado num ambiente real com uma performance satisfatória para cenários onde o tempo de resposta não é um fator crítico. A realização de testes em um ambiente de rede não foi realizada porque o tempo de latência adicionado não seria significativo diante do tempo gasto no processo de inferência de novas ontologias, que deve ser realizado por apenas um computador. Além disso, a tolerância a possíveis falhas nos sensores foi considerada como estando fora do escopo do trabalho.

Capítulo 7

Conclusão e Trabalhos Futuros

Com o passar do tempo, várias infra-estruturas de suporte a aplicações dependentes de contexto foram desenvolvidas. Este trabalho propôs a arquitetura do CxFramework, uma infra-estrutura reflexiva para dar suporte a aplicações dependentes de contexto, através de mecanismos de obtenção, processamento e notificação de contexto. A existência de uma infra-estrutura desta natureza significa que as aplicações não precisam se preocupar em como obter e processar informações de contexto, já que estas complexidades são deslocadas para a infra-estrutura. Outra vantagem importante é que diversas aplicações podem compartilhar a mesma infra-estrutura, o que permite reuso do código. Como aplicações dependentes de contexto frequentemente utilizam dispositivos heterogêneos (em termos de hardware, sistemas operacionais, etc.), a plataforma escolhida para a implementação do CxFramework foi CORBA, a qual é amplamente conhecida e já alcançou um alto grau de maturidade. No entanto, os serviços fundamentais definidos podem ser aplicados a outras plataformas com maior ou menor grau de dificuldade, dependendo da plataforma.

Apesar de existirem diversas infra-estruturas como o mesmo propósito do CxFramework, é possível enumerar diversos aspectos onde ele se sobressai. Na questão da notificação de aplicações a respeito das mudanças no contexto, todas as infra-estruturas estudadas implementam seus próprios mecanismos de notificação, sendo que todos eles acoplam as aplicações dependentes de contexto à infra-estrutura. O mecanismo utilizado pelo CxFramework é baseado em comunicação baseada em eventos, onde existe um canal de eventos responsável por gerenciar

o transporte da informação da infra-estrutura para as aplicações interessadas. A existência deste canal proporciona o desacoplamento, isto é, a infra-estrutura e as aplicações só referenciam o canal de eventos. Em ambientes distribuídos, onde nem sempre todos os componentes estão disponíveis, o desacoplamento é de fundamental importância.

Outra característica inovadora do CxFramework, não presente em nenhuma infra-estrutura pesquisada, é o suporte à reflexividade. A reflexividade possibilita que o CxFramework seja capaz de ser influenciado pelo contexto que ele representa. Isto representa um grande benefício e confere ao CxFramework a capacidade de representar cenários que outras infra-estruturas não teriam condições. Além disso, o suporte à reflexividade permite a utilização de federação de contextos, onde contextos diferentes (representados por instâncias de CxFramework diferentes) podem se comunicar.

Mesmo sendo uma infra-estrutura completa e capaz de suportar aplicações dependentes de contexto, o CxFramework ainda pode ser melhorado em diversos aspectos.

Atualmente, é responsabilidade do administrador informar (via arquivo de configuração) quais são os contextos suportados pela infra-estrutura. Esta lista de contextos é utilizada pelas aplicações que desejam obter todos os tipos de notificações possíveis para poderem escolher nas quais se registrarem. Delegar esta responsabilidade ao administrador é uma tarefa que poderia ser evitada se houvesse um mecanismo automático para descobrir os eventos de contexto possíveis de serem gerados pela infra-estrutura. Um mecanismo desta natureza deve levar em consideração a ontologia e as regras de inferência, a fim de poder concluir quais são estes eventos.

Para melhorar a qualidade da informação de contexto, é possível implementar os conceitos de qualidade de contexto (QoC) na infra-estrutura. Além disso, a incorporação de privacidade, segurança e restrição às informações de contexto ao CxFramework podem ser tópicos de grande importância para alguns cenários, além do suporte à tempo real.

Outro aspecto que pode ser ainda explorado é a redução da quantidade de dados transmitidos na comunicação entre aplicações dependentes de contexto, já que um vocabulário comum entre elas já está estabelecido. Este é um assunto de grande interesse, particularmente na área de dispositivos móveis (embora não seja restrito somente a ela).

Ainda sobre as aplicações, é possível também trabalhar em um modelo onde as aplicações são ativas, isto é, são capazes de interagir com os sensores. Neste cenário, os sensores seriam

considerados como serviços que possuem uma interface de comunicação. Como exemplo, um usuário poderia receber eventos de contexto de serviços disponíveis em um prédio, como um serviço de impressão. Mais do que apenas receber estes eventos, o usuário poderia também interagir ativamente com o serviço através de uma interface conhecida, enviando um documento para a impressora.

Referências Bibliográficas

- [AST94] ASTHANA, A.; CRAVATTS, M.; KRZYZANOWSKI, P. *An indoor wireless system for personalized shopping assistance*. IEEE Workshop on Mobile Computing Systems and Applications, 1994.
- [BAL07] BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. *A Survey on Context-Aware Systems*. Information Systems Institute, Vienna University of Technology, Austria, 2007.
- [BUC03] BUCHHOLZ, T.; KÜPPER, A.; SCHIFFERS, M. *Quality of Context: What It Is and Why We Need It*. Workshop HP OpenView Univ. Assn, 2003.
- [CHE00] CHEN, G.; KOTZ, D. *A Survey of Context-Aware Mobile Computing Research*. Department of Computer Science, Dartmouth College: Hanover, NH, 2000.
- [COS04] COSTA, P. D.; PIRES, L. F.; SINDEREN, M. V. *Designing a Configurable Services Platform for Mobile Context-Aware Applications*. International Journal of Pervasive Computing and Communications, 2004.
- [DEY00] DEY, A. K.; ABOWD, G. D. *Towards a Better Understanding of Context and Context-Awareness*. Conf. Human Factors in Comp. Sys, 2000.
- [DEY01] DEY, A. K.; ABOWD, G. D. *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*. Human-Computer Interaction, 2001.

- [DEY99] DEY, A. K.; SALBER, D.; ABOWD, G. D.; FUTAKAWA, M. *The Conference Assistant: Combining Context-Awareness with Wearable Computing*. Proceedings of the 3rd International Symposium on Wearable Computers, 1999.
- [DUR04] DURAN-LIMON, H. A.; BLAIR, G. S.; FRIDAY, A.; GRACE, P.; SAMARTZIDIS, G.; SIVAHARAN, T; WU, M. *Context-Aware Middleware for Pervasive and Ad Hoc Environments*. Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing. Toronto, Ontario, Canada, 2004.
- [GOL98] GOLM, M. ; KLEINODER, J. *metaXa and the future of reflection*. Proceedings of Workshop on Reflective Programming in C++ and Java, 1998.
- [GOR01] GORE, P.; CITRON, R; SCHIMIDT, D.; O'RYAN, C. *Designing and Optimizing a Scalable CORBA Notification Service*. Workshop on Optimization of Middleware and Distributed Systems, ACM SIGPLAN, 2001.
- [GU05] GU, T.; PUNG, H.; ZHANG, D. *A service-oriented middleware for building context-aware services*. Journal of Network and Computer Applications, 2005.
- [HEN 99] HENNING, M.; VINOSKI, S. *Advanced CORBA Programming with C++*. Addison Wesley, 1999.
- [JEN08] JENA. *Jena: A Semantic Web Framework for Java*. Disponível em: <http://jena.sourceforge.net>. Acesso em: 03/2007.
- [OMG08] OBJECT MANAGEMENT GROUP. Disponível em: <http://www.omg.org>. Acesso em: 03/2007.
- [OPE08] OPENORB. Disponível em: <http://openorb.sourceforge.net>. Acesso em: 03/2007.

- [POW03] POWERS, S. *Practical RDF*. O' Reilly, 2003.
- [PRO08] PROTÉGÉ. Disponível em: <http://protege.stanford.edu/overview/protege-owl.html>. Acesso em: 05/2007.
- [RAZ06] RAZ, D.; et al. *Fast and Efficient Context-Aware Services*. John Wiley & Sons, 2006.
- [SCH94a] SCHILIT, B.; THEIMER, M. *Disseminating Active Map Information to Mobile Hosts*. IEEE Network. 1994. p. 22-32.
- [SCH94b] SCHILIT, B.; ADAMS, N.; WANT, R. *Context-Aware Computing Applications*. 1st International Workshop on Mobile Computing Systems and Applications, 1994.
- [SCH99] SCHMIDT, A.; AIDOO, K. A.; TAKALUOMA, A.; TUOMELA, U.; LAERHOVEN, K. V.; VELDE, W. V. *Advanced Interaction in Context*. Proceedings of 1st International Symposium on Handheld and Ubiquitous Computing. Karlsruhe, Germany: Springer Verlag, 1999.
- [SIE00] SIEGEL, J. *CORBA 3 Fundamentals and Programming*. Wiley & Sons Inc, 2nd edition, 2000.
- [SIN06] SINDEREN, M. J. V.; HALTEREN A. T. V.; WEGDAM, M.; MEEUWISSEN, H. B.; EERTINK, E. H. *Supporting Context-Aware Mobile Applications: An Infrastructure Approach*. IEEE Communications Magazine, 2006.
- [STR07] STROUSTRUP, B. *The C++ Programming Language*. Disponível em: <http://www.research.att.com/~bs/C++.html>. Acesso em: 01/2007.
- [TUU00] TUULARI, E. *Context Aware Hand-Held Devices*. VTT Electronics. VTT Publications, 2000.

[VIT06] VITERBO, J.; SACRAMENTO, V.; ROCHA, R.; ENDLER, M. *MoCA: Uma arquitetura para o desenvolvimento de aplicações sensíveis ao contexto para dispositivos móveis*. 24º Simpósio Brasileiro de Redes de Computadores, 2006.

[WOR08] WORLD WIDE WEB CONSORTIUM. *Web Ontology Language (OWL) Guide*. Disponível em: <http://www.w3.org/TR/owl-guide>. Acesso em: 03/2007.

Apêndice A

Método de Obtenção das Equações 6.1 e 6.2

Este Apêndice demonstra como as equações 6.1 e 6.2 foram obtidas. Estas equações mostram a relação entre a quantidade de *sensor adapters* e o tempo de inferência de novas ontologias ($t = 0,4N$), bem como a memória gasta para armazená-las ($m = 2,2N$).

A obtenção das equações foi feita através de simulações. Para cada medição, o número de *sensor adapters* presentes foi sendo aumentado e o tempo gasto para inferir uma nova ontologia, além da quantidade de memória necessária para armazená-la, foram sendo anotados. O resultado destas medições pode ser visto na Tabela A.1.

Tabela A.1: Resultado das medições na inferência de uma nova ontologia

Número de <i>sensor adapters</i>	Tempo p/ inferir uma ontologia (ms)	Memória necessária (KB)
100	79	1081,63
400	204	1519,32
700	266	1950,20
1000	359	2269,62
1300	453	2878,66
1600	562	3267,72
1900	672	3656,78
2200	843	4110,95
2500	1062	4694,74
2800	1280	4,96

O resultado das medições foram plotados em dois gráficos, um deles representando a relação entre o aumento no número de *sensor adapters* e o tempo gasto na inferência de uma ontologia e outro representando a relação entre o aumento no número de *sensor adapters* e a memória utilizada pela ontologia inferida. Ambos os gráficos podem ser vistos nas Figura A.1 e Figura A.2.

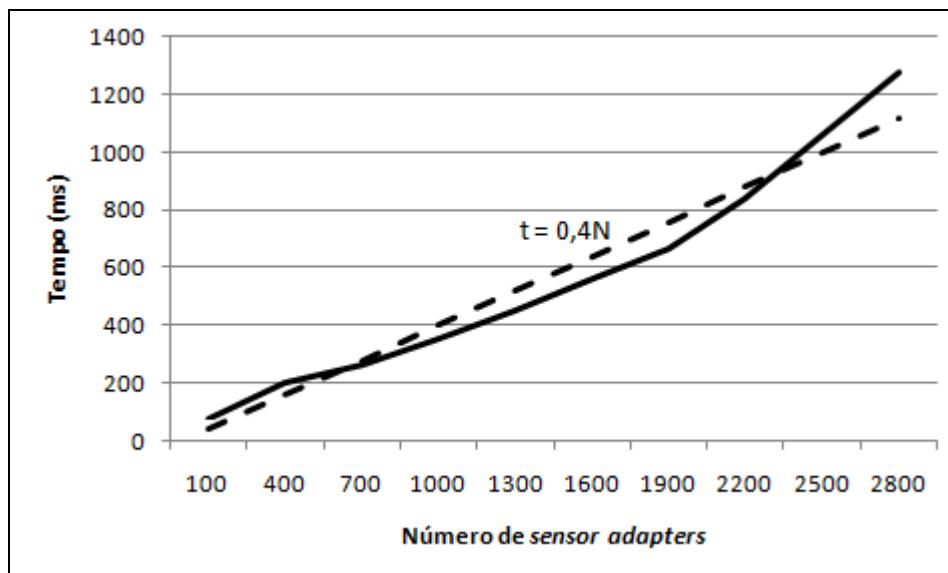


Figura A.1: Relação entre a quantidade de *sensor adapters* e o tempo gasto na inferência

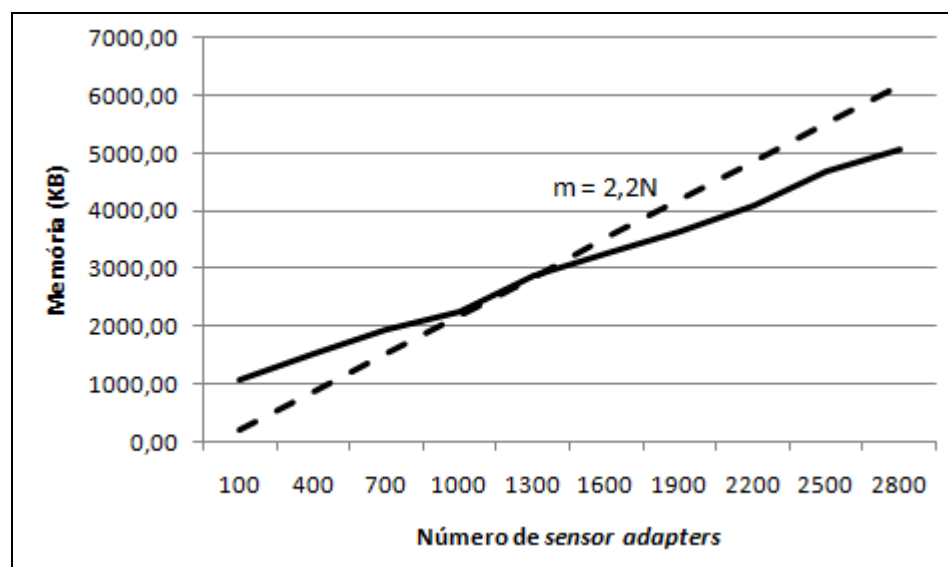


Figura A.2: Relação entre a quantidade de *sensor adapters* e a memória gasta

Nos gráficos, as linhas que representam as medições são as contínuas. É possível perceber que conforme o número de *sensor adapters* vai sendo aumentado, o tempo e a memória gastos também aumentam de forma diretamente proporcional, apresentando um crescimento linear. A partir do comportamento dos gráficos, foi possível encontrar um coeficiente que, se utilizado numa equação, pudesse representar um comportamento aproximado ao visto na prática. Desta forma, surgiram os coeficientes 0,4 e 2,2, o que possibilitou a criação das duas equações utilizadas para representar as relações entre o número de *sensor adapters* e o tempo e memória gastos na inferência de uma nova ontologia. O comportamento destas equações que resultaram neste processo também pode ser visto nos gráficos representados pelas Figura A.1 e Figura A.2, desta vez através das linhas pontilhadas.