DIOGO ROBERTO OLSEN

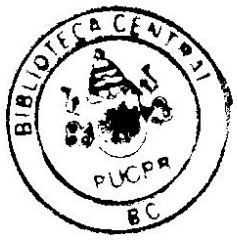


# UM GERENTE DE CONTEXTOS PARA SISTEMAS OPERACIONAIS DE PROPÓSITO GERAL

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

Curitiba PR
Fevereiro de 2012

DIOGO ROBERTO OLSEN




# UM GERENTE DE CONTEXTOS PARA SISTEMAS OPERACIONAIS DE PROPÓSITO GERAL

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

Área de concentração: *Ciência da Computação*

Orientador: Carlos Alberto Maziero

Curitiba PR
Fevereiro de 2012

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

O52g
2012

Olsen, Diogo Roberto
Um gerente de contextos para sistemas operacionais de propósito geral / Diogo Roberto Olsen ; orientador, Carlos Alberto Maziero. – 2012.
xiv, 73 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2012
Bibliografia: f. 63-65

1. Sistemas operacionais (Computadores). 2. Computação sensível a contexto. 3. Interfaces de programas aplicativos (Software). 4. Informática. I. Maziero, Carlos Alberto. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. IV. Título.

CDD 20. ed. – 005.55

Biblioteca Central
Um gerente de contextos para sistemas operacionais de
Ac. 292557 - R. 896543 Ex. 1
Doação - Doação
18/06/2012

Pontifícia Universidade Católica do Paraná
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Informática

# ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO
# PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

## DEFESA DE DISSERTAÇÃO Nº 02/2012

Aos 24 dias do mês de Fevereiro de 2012 realizou-se a sessão pública de Defesa da Dissertação **"Um Gerente de Contextos para Sistemas Operacionais de Propósito Geral"** apresentada pelo aluno **Diogo Roberto Olsen**, como requisito parcial para a obtenção do título de Mestre em Informática, perante uma Banca Examinadora composta pelos seguintes membros:

| | | |
|---|---|---|
| Prof. Dr. Carlos Alberto Maziero<br>PUCPR (Orientador) | (assinatura) | aprov.<br>(aprov/reprov.) |
| Prof. Dr. Luiz Augusto de Paula Lima Junior<br>PUCPR | | aprovado |
| Profª. Drª. Elisa Hatsue Moriya Huzita<br>UEM | | aprovado |

Conforme as normas regimentais do PPGIa e da PUCPR, o trabalho apresentado foi considerado aprovado (*aprovado/reprovado*), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora registradas no Livro de Defesas do programa.

Prof. Dr. Mauro Sérgio Pereira Fonseca
Diretor do Programa de Pós-Graduação em Informática

Programa de Pós-Graduação em Informática
PPGIa
PUCPR

Rua Imaculada Conceição, 1155 Prado Velho CEP 80215 901 Curitiba Paraná Brasil
Tel.: (41) 3271 1669 Fax: (41) 3271 2121 www.ppgia.pucpr.br

# Resumo

A computação sensível ao contexto visa tornar sistemas capazes de sentir contextos e se adaptar a eles, provendo serviços ou informações importantes para o usuário. Contexto pode ser qualquer informação usada para caracterizar de alguma forma uma pessoa, objeto ou lugar que seja de interesse do usuário. A primeira definição de computação sensível ao contexto possui quase duas décadas. Todavia, ainda hoje existem poucos trabalhos que tratam especificamente de contextos computacionais, ou seja, de informações necessárias para caracterizar sistemas computacionais, como redes e recursos disponíveis. Além disso, não encontramos trabalhos que possuem como foco principal tornar um sistema operacional sensível ao contexto. Este documento apresenta uma proposta para a criação de um modelo genérico de um gerente de contextos para sistemas operacionais de propósito geral. Este modelo inclui uma forma de definir, armazenar e gerenciar contextos computacionais, além de uma API para tornar essas informações acessíveis a aplicações do usuário.

**Palavras-chave:** Computação sensível a contexto, Sistemas operacionais, Gerente de contextos.

# Abstract

Context-aware computing aims to give systems the ability to perceive contexts and to adapt to them, providing relevant services or information to its users. Context means any information used to characterize a person, object, or place of user's interest.

The first definition of context-aware computing is around twenty-years old. However, even today there are few works that deal specifically with computing contexts, i.e. the information required to characterize computing system contexts like networks and other available resources. Furthermore, we did not find works having focus in turning standard operating systems context-aware.

This document presents a proposal to create a generic model of a context manager for general-purpose operating systems. Such model includes ways to define, store, and manage computing contexts, and an API to make such information available to user applications.

**Keywords:** Context-aware computing, Operating systems, Context manager.

# Sumário

**Resumo** vii

**Abstract** ix

**Lista de Figuras** xiii

**Lista de Abreviações** xiv

**1 Introdução** 1
- 1.1 Visão geral . . . 1
- 1.2 Proposta . . . 3
- 1.3 Organização do documento . . . 3

**2 Computação sensível ao contexto** 5
- 2.1 Introdução . . . 5
- 2.2 Sensibilidade a contextos . . . 6
- 2.3 Contexto e sistemas operacionais . . . 16
- 2.4 Considerações . . . 17

**3 Construção de Contextos** 19
- 3.1 Definição . . . 19
- 3.2 Modelo de dados . . . 21
- 3.3 Exemplos de aplicação . . . 29
- 3.4 Conclusão . . . 32

**4 O gerente de contextos** 33
- 4.1 Introdução . . . 33
- 4.2 Gerente de Contextos . . . 34
- 4.3 Contextos pré-definidos . . . 39
- 4.4 API . . . 41
- 4.5 Estudo de caso . . . 43
- 4.6 Implementação . . . 46
- 4.7 Conclusão . . . 47

**5 Resultados obtidos** 49
- 5.1 Introdução . . . 49
- 5.2 Objetivos . . . 49

5.3 Conectividade . . . . . 50
5.4 Luminosidade . . . . . 57
5.5 Conclusão . . . . . 59

**6 Considerações finais** **61**

**A Protótipo** **67**
A.1 Introdução . . . . . 67
A.2 Código Fonte . . . . . 68

# Lista de Figuras


2.1 Modelo de representação de contextos de Cheverst . . . . . 12

3.1 Contextos . . . . . 22
3.2 Processos . . . . . 22
3.3 Visão geral do modelo . . . . . 23
3.4 Coleta de dados . . . . . 25
3.5 Ciclo de vida da informação de entidade real até um contexto entregue a um processo. . . . . 26
3.6 Processo de consolidação de contextos . . . . . 27
3.7 Diagrama do modelo de dados . . . . . 28
3.8 Construção do contexto conectividade . . . . . 29
3.9 Construção do contexto conforto . . . . . 31
3.10 Construção do contexto energia . . . . . 31

4.1 Visão geral do modelo . . . . . 35
4.2 Visão geral do gerente de contextos . . . . . 36
4.3 Construção do contexto conectividade . . . . . 44
4.4 Diagrama de classe do GC . . . . . 46
4.5 Representação das tabelas . . . . . 48

5.1 Contexto conectividade . . . . . 52
5.2 Processo de construção do contexto luminosidade . . . . . 52
5.3 Estados do contexto conectividade . . . . . 56

A.1 Diagrama de classe do GC . . . . . 67

# Lista de Abreviações

| | |
|---|---|
| *API* | *Application Programming Interface* - Interface de Programação de Aplicativos |
| *at* | Atributo |
| *AT* | Conjunto de atributos |
| *C* | Contexto |
| CAC | *Context-aware computing* - Computação sensível a contextos |
| *CT* | Conjunto de contextos |
| *ic* | Informação contextual |
| *IC* | Conjunto de informações contextuais |
| *E* | Entidade |
| *ET* | Conjunto de entidades |
| *er* | Entidade real |
| *ES* | Escalonador |
| $fa$ | Função agregadora |
| *FA* | Conjunto de funções agregadoras |
| $fc$ | Função coletora |
| *FC* | Conjunto de funções coletoras |
| *GC* | Gerente de contextos |
| *GPS* | *Global Positioning System* - Sistema de posicionamento global |
| *IP* | *Internet Protocol* - Protocolo de Internet |
| *me* | Mecanismo de entrega |
| *PDA* | *Personal digital assistants* - Assistente pessoal digital |
| SO | Sistema operacional |
| $v_c$ | Valor corrente |
| $v_p$ | Valor passado |

# Capítulo 1

# Introdução

O desenvolvimento tecnológico visto nas últimas décadas tornou possível diminuir cada vez mais os dispositivos tecnológicos que usamos. Se na década de 1940 os primeiros computadores pesavam toneladas, na década de 1970 já existiam os primeiros computadores pessoais. O Kenbak-1, considerado o primeiro computador pessoal, pesava apenas 6.2 Kg e custava U$750, algo incomparável com o Eniac, que pesava 30 Toneladas.

Esta revolução tecnológica vem alterando a forma destes dispositivos, seus custos são reduzidos e a novas funções são criadas para estes aparelhos, de forma que atualmente existem smartfones que pesam poucas gramas e custam apenas algumas dezenas de dólares. Com isto, estes novos dispositivos têm se popularizado em todas as partes do mundo e são cada vez mais indispensáveis na vida moderna.

Como estes dispositivos são cada vez mais necessários eles são carregados pelos usuários para todos os lugares e oferecem cada vez mais mobilidade. Para que estes dispositivos possam assumir novas funcionalidades, é necessário que eles possuam informações sobre as situações onde em que estão inseridos. A estas informações podemos dar o nome de "contexto".

Este trabalho apresenta uma camada se software que visa tratar problemas como a criação de um modelo de dados que possa ser usado para construir contextos, a definição de uma forma de representar contextos computacionalmente e a definição de uma arquitetura para suportar este software.

Para explicar melhor a proposta deste trabalho, este capítulo está estruturado nas seguintes seções: Seção 1.1 - Visão geral: descreve superficialmente este trabalho e a área em que ele está inserido; Seção Seção 1.2 - Motivação: explica por que este tema foi escolhido; Seção 1.3 - Objetivos: apresenta os objetivos que devem ser alcançados com este trabalho; Seção 1.4 - Organização do documento: apresenta a estrutura usada no restante deste documento.

## 1.1 Visão geral

Atualmente muitos usuários de computadores usam dispositivos portáteis como notebooks e netbooks, ou dispositivos ultra-portáteis como smartfones e PDAs. Estes dispositivos são usados em vários ambientes como em casa, na universidade, no trabalho e outros. Porém, usar o mesmo dispositivo em situações diversas pode trazer alguns problemas para o usuário, como o celular que toca com o volume alto durante uma reunião ou o brilho da tela que não se adapta a luminosidade ambiente.

Nos dois exemplos citados acima ocorreram problemas por usar um mesmo dispositivo em situações diferentes, pessoas se comportam de maneiras distintas em contextos distintos, pessoas falam baixo dentro do cinema e não mostram seu álbum de fotos para todas as pessoas que estão ao seu redor. Porém os dispositivos citados acima não são capazes de fazer esta distinção de contexto.

Neste cenário, contexto pode ser definido como: "qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, um lugar ou um objeto que é considerado relevante na interação entre usuário e aplicação, incluindo a própria aplicação e usuário" [Abowd et al., 1999]

A *Computação Sensível ao Contexto* (CAC - *Context-Aware Computing*) visa tornar sistemas computacionais capazes de perceber o contexto em que estão inseridos, como por exemplo usar dados sobre a localização geográfica do usuário para determinar se o mesmo se encontra em casa ou no trabalho, ou seja, determinar parte do contexto do usuário. Para isso podem ser usados diversos sensores (como câmeras, microfones, GPSs entre outros) e algoritmos.

A CAC foi definida pela primeira vez da década de 1990 [Schilit et al., 1994], desde então diversas [Harter et al., 1999, Picco et al., 2005, Boari et al., 2006, Eagle and (Sandy) Pentland, 2006, Hong et al., 2009, Beach et al., 2010] pesquisas foram realizadas nesta área, como pode ser visto na Seção 2. Estas pesquisas tentam criar/definir aplicativos/sistemas que sejam capazes de determinar o contexto onde o usuário está inserido e com base neste contexto a aplicação/sistema deve se adaptar para melhorar de alguma forma a interação com o usuário.

A idéia básica da CAC é simples: um sistema deve ser capaz de detectar o contexto em que está inserido e com base nesse contexto o sistema deve se adaptar para melhorar a interação com o usuário. Porém, segundo o trabalho [Abowd and Mynatt, 2000], após diversas pesquisas realizadas sobre o tema, a CAC mostrou-se difícil de ser implementada.

Isso se deve em parte pela heterogeneidade dos dados usados para compor um contexto, por exemplo: para definir a localização de um usuário podem ser usados dados como redes sem fio disponíveis, dados de GPS, hora do dia, funções heurísticas, recursos computacionais disponíveis entre outros. E a definição da localização é apenas uma parte do contexto, existem mais dados que podem ser usados para compor um contexto.

Esta heterogeneidade de dados também torna complicada a representação computacional de contextos. Como representar contextos, armazenar contextos para que estes sejam usados posteriormente como dados históricos, também torna-se uma tarefa complicada. O mesmo problema ocorre quando é necessário aplicar algoritmos sobre contextos pois, existe uma quantidade grande de dados que podem ser usados para gerar um contexto.

Depois que estes problemas estejam resolvido e uma camada de software que gera contextos estiver implementada os contextos gerados podem ser usados para a classificação de usuários baseados nos contextos de cada usuário ou então estes contextos podem ser usados para modelar o comportamento de usuários e estes modelos podem ser usados para desenvolver SO mais adequados ao uso que usuários fazem dele.

Existem diversas aplicações para CAC dentro de um SO, como a auto configuração do dispositivo baseando-se em dados ambientais como luminosidade e nível de ruído. Estas aplicações podem melhorar a interação do usuário com o SO, facilitando o uso do sistema, melhorando a segurança, automatizando algumas tarefas do usuário, entre outras aplicações.

## 1.2 Proposta

O presente trabalho define de um modelo de dados capaz de construir e gerenciar contexto. Este modelo pode auxiliar na criação de diversos mecanismos capazes de prover melhorias em áreas do SO, como a interação do usuário com o SO, segurança, automatização de tarefas comuns do usuário e a configuração automática do sistema. Esta camada de software será chamada de "gerente de contextos" (GC).

Este trabalho ainda tem como requisito gerar um modelo de gerente de contextos que possa ser utilizado na maioria dos equipamentos atuais. Para que este requisito seja alcançado é necessário que o gerente não necessite de sensores especiais para funcionar (como sensores biométricos, que ainda são pouco encontrados em computadores).

Este modelo funciona sem a necessidade de infra-estruturas de rede incomuns, como a rede de sensores de presença usada em [Harter et al., 1999]. Este requisito serve para que o gerente de contextos esteja acessível à maioria dos usuários, que não possuem acesso a sensores incomuns ou redes complexas.

O modelo deve ser o mais genérico possível, desta forma será mais fácil implementar o modelo em diversos sistemas operacionais, sem que sejam necessárias muitas alterações no mesmo. Porém, mesmo que o modelo seja bem genérico, as operações de coleta de dados sempre precisarão ser alteradas para que o modelo se adapte as peculiaridades de cada SO.

Existe uma API para que outras aplicações possam acessar dados contextuais. Outra forma de acesso aos dados gerados pelo sistema é usando uma infra-estrutura de notificações automáticas sobre mudanças ocorridas nos contextos do sistema.

O gerente também armazena séries históricas dos contextos gerados para que outros processos possam usar estes dados. Esta etapa é útil para que outros processos possam analisar as transições ocorridas com os contextos. O conjunto destas funcionalidades é oferecido pelo modelo de dados que ao ser implementado constitui o GC.

## 1.3 Organização do documento

O restante desse documento está organizado da seguinte maneira: O capítulo 2 apresenta uma revisão da bibliografia sobre computação sensível a contextos; o capítulo 3 apresenta as principais definições sobre contextos e propõe um modelo de dados capaz de construir e gerenciar contextos; o capítulo 4 apresenta o gerente de contextos, sua API e um protótipo; O capítulo 5 realiza testes com o protótipo e o capítulo 6 conclui este trabalho. O código fonte do protótipo está no apêndice A deste documento.

# Capítulo 2

# Computação sensível ao contexto

Simplificadamente CAC é um paradigma da computação onde sistemas são capazes de descobrir informações contextuais como localização geográfica, recursos e/ou pessoas e/ou dispositivos próximos e/ou disponíveis, hora do dia, entre vários outros, e com base nestas informações prover informações e/ou serviços e/ou reconfigurações de serviços que sejam de interesse do usuário.

Este capítulo, usando uma revisão bibliográfica sobre o tema, irá apresentar os principais conceitos de CAC, algumas das definições já feitas sobre o tema, irá demonstrar algumas aplicações já implementadas ou propostas que utilizam contextos e posteriormente irá definir alguns conceitos que serão usados no restante deste trabalho.

## 2.1 Introdução

Desde quando foi proposta na década de 1990, no trabalho [Schilit et al., 1994], a CAC já foi definida de diversas formas, alguns autores a definem descrevendo suas funcionalidades como Schilit faz, outros autores utilizam sinônimos, como [Abowd et al., 1999] descreve em seu trabalho, outros autores por sua vez utilizam exemplos de uso das tecnologias para definir CAC, como no trabalho [Coutaz et al., 2005].

Segundo Abowd [Abowd et al., 1999], já foram utilizados os seguintes sinônimos para descrever CAC: adaptivo, reativo, responsivo, situado, sensível ao contexto e dirigido ao ambiente. Além de alguns outros, como *Context-Enabled Applications* (aplicações com contexto ativo) [Salber et al., 1999], porém o termo mais utilizado é *Context-Aware Computing*. Em português o termo mais utilizado para designar este paradigma é Computação Sensível ao Contexto, o qual será usado neste trabalho.

Existem também diversas definições para contexto, como acontece com CAC, alguns autores utilizam exemplos ou funcionalidades para definir contexto, outros utilizam sinônimos e alguns utilizam definições mais abrangentes. A mesma variedade de definições ocorre quanto a categorização de contextos, Schilit divide contextos em duas categorias: contexto computacional e contexto do usuário, [Chen and Kotz, 2000] usa além destas, uma terceira categoria: contexto temporal.

- **Contexto computacional**: Trata de dados relativos ao dispositivo, como: duração da bateria, redes disponíveis, dispositivos plugados/disponíveis, etc;

- **Contexto do usuário**: Trata de dados do usuário, como pessoas próximas, localização geográfica, temperatura, entre outros;
- **Contexto temporal**: Trata de dados temporais, ano, mês, dia hora, etc.

Dada esta diversidade de definições faz-se necessário um levantamento de como a CAC já foi definida na literatura, o mesmo ocorrerá com os contexto e suas categorias. Este levantamento será apresentado nas próximas páginas deste trabalho Posteriormente, serão apresentadas as definições de CAC e contexto adotadas neste trabalho, bem como a categorização de contexto.

## 2.2 Sensibilidade a contextos

Nesta seção serão apresentados os principais artigos referentes a CAC, para isto será seguida a ordem cronológica da publicação dos trabalhos para evidenciar a evolução ocorrida na área durante os últimos anos. Esta revisão não pretende ser exaustiva, pretende apenas levantar os principais conceitos envolvidos na área. Não serão apresentados artigos cujo foco são técnicas específicas, como a determinação de localização geográfica. Abaixo estão apresentados os trabalhos.

**Context-aware computing applications**

No trabalho [Schilit et al., 1994], Schilit define CAC como um software capaz de se adaptar de acordo com a localização, com as pessoas próximas, hosts, e dispositivos acessíveis, bem como com as mudanças deste cenário ao longo do tempo. Segundo o autor, existem três aspectos importantes do contexto: onde você está, quem está com você e quais recursos estão disponíveis. Além disto os autores definem quatro categorias de aplicações sensíveis ao contexto:

- **Detecção de proximidade**: Técnica onde objetos (dispositivos, objetos ou lugares) próximos são enfatizados ou possuem sua escolha facilitada;
- **Reconfiguração automática por contexto**: Processo de adicionar ou remover componentes; ou ainda, alterar as conexões entre componentes;
- **Comandos e informações contextuais**: Tenta explorar o fato de que as ações das pessoas são influenciadas pelas suas situações; Em uma biblioteca uma pessoa se comporta de forma diferente de quando está em uma cozinha;
- **Ações disparadas por contextos** (*context-triggered actions*): Regras simples do tipo SE-ENTÃO que definem como sistemas sensíveis ao contexto podem se adaptar;

Segundo os autores este trabalho é o primeiro sobre um dispositivo móvel construído para usar um software sensível ao contexto.

**Rapid prototyping of mobile context-aware applications: the cyberguide case study**

O trabalho [Long et al., 1996] apresenta o projeto *Cyberguide*, um guia turístico virtual que baseia-se em informações contextuais (principalmente posição e orientação) para prover informações turísticas para um visitante. Além disto, são citadas possíveis aplicações para a computação e dispositivos sensíveis ao contexto, como guias, tradutores e dispositivos de realidade aumentada.

O projeto foi dividido em 4 módulos principais: *Cartographer*, detentor das informações geográficas (como um mapa); *Librarian*, detentor de informações turísticas; *Navigator*, responsável por definir a localização do visitante; *Messenger*, responsável pela interação com o visitante.

Posteriormente, o trabalho discute o uso do *Cyberguide* em ambientes fechados e abertos, sua implementação e outras questões como o custo de implementação. Como trabalhos futuros os autores apresentam a capacidade de alterar automaticamente sua base de informações, melhorias na comunicação, no suporte a multimídias e na sensibilidade de contextos, uso da web e visão computacional, capacidade de operar *indoor* (IR) e *outdor* (GPS).

**Cyberguide: a mobile context-aware tour guide**

O trabalho [Abowd et al., 1997] reapresenta o *Cyberguide*, projeto que tenta usar informações contextuais para funcionar como um guia turístico. Além de dados relativos a localização e orientação o Cyberguide usa dados relativos ao histórico das localizações do usuário para prover serviços melhores ao usuário. Os objetivos principais deste projeto são: determinar a localização do turista; determinar para onde ele está olhando; predizer e responder questões propostas; prover interações com outros usuários.

**There is more to context than location**

Os autores do trabalho [Schmidt et al., 1998] afirmam que o contexto é a chave da interação entre humanos e computadores ao descrever "fatores externos"que adicionam significado à interação. Porém vários trabalhos utilizam como contexto apenas a localização de dado dispositivo como contexto. Para os autores, isto pode ser melhorado usando mais informações e mais sensores para criar um contexto mais sofisticado. Para estes autores, contexto pode ser estruturado, como mostrado abaixo:

- Contexto define a situação e o ambiente onde o usuário está.
- Um contexto é identificado por um nome único.
- Para cada contexto um conjunto de características é importante.
- Cada contexto define um intervalo de valores para cada característica relevante.

O grande desafio é identificar o conjunto de características que devem/podem ser usadas/capturadas para serem usadas em um contexto. Além disto, o histórico passado dos contextos deve ser usado para se aproximar de dada situação/ambiente. Contextos podem ser adquiridos implicitamente, monitorando a atividade do usuário, ou explicitamente, solicitando dados ao usuário.

O trabalho apresenta algumas das tecnologias de sensores que podem ser usados em dispositivos ultra-portáteis, como sensores ópticos, de áudio, de movimento, localização, bio-sensores e sensores especializados, como termômetros. Usando alguns destes sensores é possível criar sensibilidade a contextos baseada em sensores, e isto torna possível criar, por exemplo, interfaces de usuário adaptáveis.

Em seguida os autores apresentam algumas técnicas para utilizar dados oriundos de diversos sensores para melhorar a sensibilidade a contextos, chamada de *Sensor Fusion for Context Awareness*. Para isto eles propõe uma arquitetura multi-nível descrita a seguir:

- **Sensors**: neste nível existem dois tipos de sensores:
  - **sensores físicos** obtém dados ambientais diretamente de sensores implementados em hardware, e então criam
  - **sensores lógicos** responsáveis por abstrair sensores físicos e armazenar os dados dos sensores físicos;
- **Cues**: nível que provê uma abstração dos sensores lógicos e físicos, onde cada *cue* obtem dados de um único sensor e provê um dado simbólico ou sub-simbólico, baseado no valor do sensor. Cada *cue* está associado a apenas um sensor, porém podem existir vários *cues* associados a um mesmo sensor;
- **Contexts**: nível abstrato onde um contexto descreve dada situação e cada contexto é baseado em *cues* disponíveis;
- **Scripting**: nível que provê informações contextuais a cada aplicação;

O trabalho então apresenta um dispositivo experimental para testar os sensores e uma forma de derivar contextos usando dados originados pelos sensores. Como exemplo é possível concluir se o dispositivo encontra-se no escritório ou não, baseando-se em dados sobre luminosidade, movimentação, temperatura, etc.

### The anatomy of a context-aware application

Já em 1999, o trabalho [Harter et al., 1999] apresenta uma plataforma de computação sensível a contextos que permite que um usuário, carregando um sensor, seja seguido por suas aplicações ao se mover dentro de uma construção (*Follow-me Applications*), de forma que as aplicações estejam disponíveis em qualquer equipamento/rede disponível no ambiente. A plataforma ainda cria dinamicamente um modelo do ambiente usando como base os dados sobre localização oriundos dos sensores e um software de telemetria.

A arquitetura proposta contém cinco componentes principais:

- Um sistema de localização/identificação de objetos.
- Um modelo que descreve entidades essenciais envolvidas nas aplicações móveis do mundo real.
- Um sistema persistente e distribuído de objetos que provê informações do modelo para as aplicações.

- Monitores de recursos que executam sobre a rede, informando o status dos equipamentos a um repositório central.
- Um serviço de monitoramento espacial, que habilita eventos em aplicações sensíveis a localização.

Posteriormente o trabalho apresenta mais alguns módulos da sua plataforma, como a sensibilidade à localização, baseada em tecnologias ultra-sônicas emitidas por um sensor que deve ser carregado pelo usuários e captadas por receptores instalados na construção; a capacidade de sentir o ambiente real; a infra-estrutura usada.

**The context toolkit: aiding the development of context-enabled applications**

Também em 1999, o trabalho [Salber et al., 1999] fala sobre o conceito de *context widgets*: partes de um toolkit que faz a mediação entre o ambiente (contextos) e as aplicações, facilitando o desenvolvimento de soluções reusáveis para usar informações contextuais em aplicações interativas. Para os autores faltam modelos conceituais e ferramentas para solucionar problemas que envolvem contextos.

Segundo os autores as dificuldades de trabalhar com informações contextuais originam-se das diversas naturezas da informação contextual, entre as maiores dificuldades os autores citam: uso de sensores não convencionais (como o usado no trabalho [Harter et al., 1999]); dados que devem ser abstratos para serem usados em aplicações; dados adquiridos de fontes distribuídas e heterogêneas; dados dinâmicos.

O desenvolvimento deste toolkit usa o conceito dos *widgets*, de interfaces gráficas, os quais fazem a mediação entre aplicativos e usuários. Da mesma forma os *context widgets* realizam a mediação entre as aplicações e o ambiente. Os autores definem um *context widget* como sendo um componente de software que aprove acesso para aplicações à informações contextuais sobre o seu ambiente de operação. As vantagens deste modelo são:

- Ocultar a complexidade de sensores.
- Abstrair informações contextuais.
- Provêr códigos reusáveis/customizáveis.

*Context widget* possuem um estado (conjunto de atributos) e um comportamento (*triggers callbacks*), funcionam como um bloco que gerencia a sensibilidade de dada parte do contexto. Além disto o toolkit permite a composição de widgets através da união de outros widgets. *Context widget* possuem duas grandes diferenças com GUI widgets: *Context widget.* operam em uma arquitetura distribuída, pois necessitam de dados externos e *Context widget* monitoram informações ambientais que podem ser solicitadas a qualquer momento.

Então o trabalho apresenta alguns exemplos de como foram construídos alguns widgets, outros exemplos de como construir aplicações usando o toolkit e futuras extensões para o toolkit. O trabalho também apresenta alguns detalhes de implementação: como trabalhar com distribuição, composição, heterogeneidade e dinamismo. Para que um sistema seja sensível a contextos é importante a capacidade de transferir dados pela rede, desta forma um widget é constituído de três partes: um gerador que adquire dados dos sensores; um interpretador, que cria abstrações de alto nível com base nos dados dos sensores; um servidor para prover tais dados.

**Towards a better understanding of context and context-awareness**

Ainda em 1999, o trabalho [Abowd et al., 1999] apresenta um *survey* sobre a computação sensível a contextos priorizando usar definições e categorias de contextos e *computação sensível a contextos* (CAC). Após apresentar uma introdução ao tema, o trabalho apresenta algumas definições. Segundo os autores, vários trabalhos possuem dificuldades para definir o que é contexto. O primeiro trabalho a introduzir CAC foi [Schilit et al., 1994], porém o trabalho (como vários dos seus sucessores) apresenta exemplos para explicar contextos, e não uma definição.

Para os autores, a melhor definição de contexto é: "Contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, um lugar ou um objeto que é considerado relevante na interação entre usuário e aplicação, incluindo a própria aplicação e usuário". Os autores afirmam que existem algumas categorias de contextos mais importantes, são elas: Localização, identidade, atividade e tempo.

As primeiras definições de *context-aware* restringem-se a aplicativos que, após informados de um dado contexto, adaptam-se a este. Ainda surgiram diversos sinônimos para o termo, dentre eles: *adaptive, reactive, responsive, situated, context-sensitive e environment-directed.* Tais definições podem ser divididas em duas categorias: aplicações que usam contextos e aplicações que se adaptam a contextos. Segundo os autores, *context-aware* pode ser definido como: "Um sistema é sensível a contextos se ele usa contextos para prover informações relevantes e/ou serviços para o usuário, onde relevância depende da tarefa do usuário".

Posteriormente o trabalho apresenta um catálogo dos principais trabalhos da área, comenta sobre o desenvolvimento de aplicações sensíveis a contexto. Os autores também criam uma classificação das características de aplicações sensíveis a contextos, como pode ser visto a seguir:

- **Apresentação** de informações/serviços ao usuário.
- **Execução** automática de dados serviços.
- **Tagging** de informações contextuais para uso posterior.

**Charting past, present, and future research in ubiquitous computing**

Já no ano 2000, o trabalho [Abowd and Mynatt, 2000] avalia o passado, presente e futuro da computação ubíqua (*Ubiquitous Computing*). Na década de 1990 a computação ubíqua (*ubicomp*) possuiu três interesses principais: interfaces naturais (*natural interfaces*): que é a capacidade de um computador oferecer formas comuns de comunicação humana, como gestos e palavras; aplicações sensíveis a contextos (*context-aware applications*); *automated capure and access*: que é a capacidade de um computador automaticamente capturar experiências humanas e prover um acesso universal a estas experiências ao usuário.

Para os autores, uma aplicação ubíqua deve ser sensível a contextos. Os primeiros trabalhos sobre computação sensível a contextos (como: [Schilit et al., 1994] e [Pascoe, 1998]) baseiam-se em uma pequena parcela de contextos, a localização, que é um dos contextos mais usados nesta área. Porém após vários trabalhos sobre o uso da localização como contexto demonstraram que o mesmo é difícil de ser implementado. Para os autores contexto pode ser completamente definido com os "cinco W": *Who, What, Where, When, Why.*

Outro ponto importante da computação sensível a contextos é a definição de uma forma de representação de contextos, a qual ainda não foi satisfatoriamente resolvida. Além disto, faz-se necessário utilizar técnicas de fusão de contextos para aumentar a confiança de dado contexto. A união das técnicas de sensibilidade a contextos e interações naturais podem gerar técnicas capazes de criar uma realidade aumentada.

**Developing a context-aware electronic tourist guide: some issues and experiences**

Cheverst apresenta um guia turístico inteligente e eletrônico [Cheverst et al., 2000b], chamado GUIDE que combina tecnologias de computação móvel com uma infra-estrutura *wireless* para apresentar a visitantes de uma cidade informações baseadas em seus contextos ambiental e pessoal. Os focos principais deste trabalho são: Requisitos necessários para implementar o guia; design e customização de um navegador Web para permitir a navegação de um usuário; análise do guia para melhorar a experiência do usuário com o sistema.

Segundo os autores, existem dois tipos de informações que podem ser vistas como contextos: contextos pessoais e contextos ambientais. Como exemplos de contextos pessoais, os autores usam localização do usuário, histórico dos seus interesses e preferências pessoais. Como exemplos de contextos ambientais são usados a hora do dia e o horário de abertura de atrações da cidade. Os autores afirmam que os dois tipos de contexto devem ser usados juntos para adaptar a apresentação de dados ao visitante.

Quanto às informações apresentadas pelo guia, os autores as dividem em três tipos: Informações geográficas; informações no formato de hipertexto; Componentes ativos que podem gerar eventos. O modelo não é melhor explicado neste trabalho, sendo expandido no trabalho [Cheverst et al., 2000a]. Além disto os autores afirmam que não existe um modelo para representar dados heterogêneos como estes de forma adequada.

**Experiences of developing and deploying a context-aware tourist guide: the guide project**

Ainda no ano 2000 Cheverst expande o artigo [Cheverst et al., 2000b], no trabalho [Cheverst et al., 2000a] que trata de um guia turístico inteligente que trabalha com sensibilidade a contextos, chamado de GUIDE. Segundo os autores os principais requisitos da aplicação são: flexibilidade para o visitante poder criar seu próprio roteiro turístico; suporte a informações baseadas em contextos para melhorar a interação do turista com o guia; suporte para informações dinâmicas; suporte para serviços interativos.

Baseado em informações contextuais, o guia fornece informações extraídas da Web e personalizadas para cada turista. Segundo os autores, contextos podem ser divididos em duas categorias, como pode ser visto no artigo [Cheverst et al., 2000b], contextos pessoais e contextos ambientais. O modelo de dados adotado para representar estes contextos trabalha com quatro tipos de dados, como visto a seguir:

- Informações que representam o contexto atual, como "Está Chovendo" ou dados históricos do usuário;
- Informações geográficas, como coordenadas ou dados simbólicos, como "Próximo à";
- Informações em hipertexto, como páginas WEB armazenadas localmente;

- Componentes ativos: entidades capazes de armazenar estados, como preferências do usuário;

Segundo os autores, não existe um modelo capaz de armazenar todos estes tipos de dados de forma eficiente, por isso foi criado um modelo baseado na relação de dois componentes: objetos ativos e informações em hipertexto. Este modelo permite a criação de relacionamentos entre informações geográficas (objetos ativos) e pontos de navegação do guia. Este modelo pode ser visto na figura 2.1.

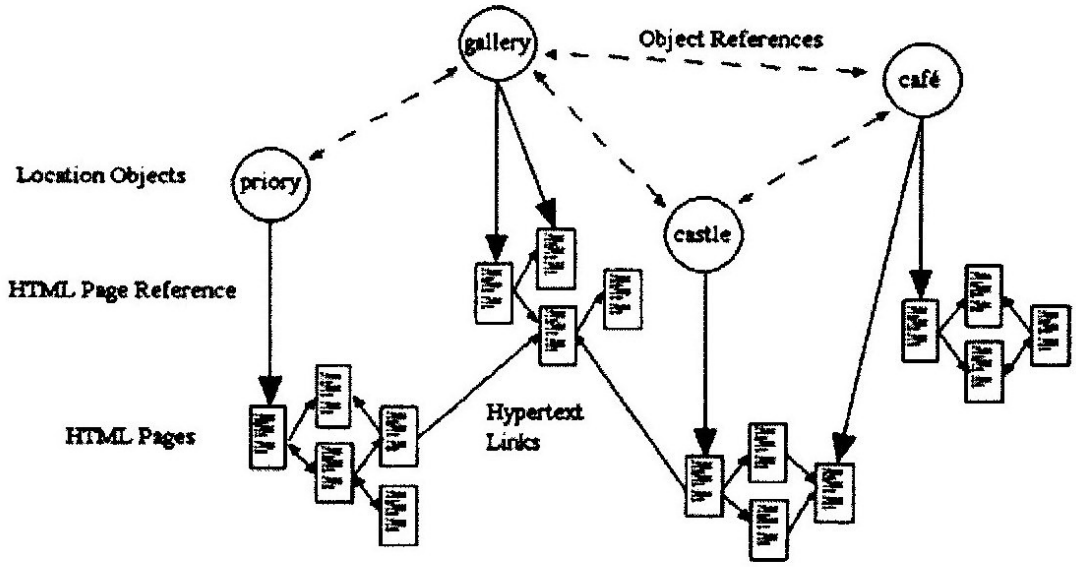



Figura 2.1: Modelo de representação de contextos adotado em [Cheverst et al., 2000a]

### Understanding and using context

O trabalho [Dey, 2001] complementa o trabalho [Salber et al., 1999] analisando definições de contexto criadas anteriormente, por outros autores, para então criar sua própria definição de contexto: "Contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, um lugar ou um objeto que é considerado relevante na interação entre usuário e aplicação, incluindo a própria aplicação e usuário".

Os autores definem computação sensível a contextos como: Um sistema é sensível a contextos se ele usa contextos para prover informações relevantes e/ou serviços para o usuário, onde a relevância depende da tarefa do usuário. Além disto, classificam as funcionalidades de uma aplicação sensível a contextos em três categorias: apresentação de informações/serviços ao usuário; execução automática de serviços para o usuário; armazenamento de informações contextuais para uso posterior.

Neste trabalho, os autores propoem a criação de uma "arquitetura" chamada *Context Toolkit*, capaz de facilitar a criação de aplicações sensíveis a contexto. Tal aplicação teria cinco funcionalidades principais: *Context Widget* responsável por adquirir e distribuir informações sobre contextos; *Storage* responsável por armazenar contextos para utilização posterior; *Context Interpreter* responsável por interpretar dados sobre contextos; *Aggregators* responsável por

coletar/agregar todos os contextos interessantes para uma certa entidade; *Situation* responsável por determinar, automaticamente, quais dados são relevantes para uma certa aplicação.

**An infrastructure approach to context-aware computing**

Para os autores do trabalho [Hong and Landay, 2001] muitas pesquisas foram feitas sobre computação sensível a contextos, e muitos protótipos foram criados, demonstrando o potencial desta área, porém também demonstrou-se as dificuldades de design, desenvolvimento e manutenção de aplicações sensíveis a contextos. Para os autores, a forma mais simples de criar e manter sistemas sensíveis a contexto é distribuir parte do esforço computacional através de camadas intermediárias acessíveis pela rede, para que então as aplicações acessem estes serviços e obtenham os contextos.

Para demonstrar seu ponto de vista, os autores analisaram as formas mais comuns de prover sistemas que adquirem contextos para prove-los a aplicações:

- **Bibliotecas de funções** são um conjunto de funções relacionadas com o foco no reuso, que poderiam ser usadas, por exemplo, no armazenamento e análise de contextos.
- **Frameworks** também são um conjunto de funções com foco em reuso, porém diferem-se de bibliotecas por assumir o controle de certos pontos, enquanto bibliotecas deixam todo o controle nas aplicações que as utilizam. Frameworks também podem ser configurados para necessidades específicas.
- **Toolkits** são componentes construídos dentro de frameworks para prover componentes reusáveis de funcionalidades comuns. Dentro de um framework de computação sensível a contextos um toolkit poderia ser usado para oferecer acesso a vários tipos de sensores.
- **Infra-estruturas** são produtos de softwares capazes de mediar a interação de outros softwares. Estas estruturas poderiam ser acessíveis pela rede e assim prover diversos serviços para outras aplicações.

Segundo os autores a abordagem de infra-estrutura possui algumas vantagens sobre as outras abordagens, tais como: independência de hardware, sistema operacional e linguagem de programação; melhores capacidades de manutenção e evolução, visto que sensores e serviços podem ser alterados de forma independente e dinâmica, sem alterar o comportamento das aplicações que são executadas; A abordagem de infra-estrutura de rede também permite o compartilhamento de sensores, poder de processamento, dados e serviços.

**A survey of context-aware mobile computing research**

Segundo o artigo [Chen and Kotz, 2000] a computação sensível a contextos é um paradigma da computação móvel no qual aplicações podem descobrir e usar informações contextuais como localização do usuário, hora do dia, etc. Quando foi proposta na década de 1990 pelos pioneiros da Olivetti research e da Xerox PARC [Schilit et al., 1994], a computação sensível a contextos era vista como uma área da computação ubíqua, ou pervasiva.

Tal paradigma só tornou-se possível apoiando-se em outras duas tecnologias: tecnologias de comunicação sem fio e tecnologias que tornaram os computadores portáteis e apesar

de várias pesquisas demonstrarem a utilidade desta nova área, ela ainda não está disponível para ser usada por usuários comuns.

Para os autores diversas definições de contexto já foram escritas sem que sejam satisfatórias. Por isso alguns autores definem contexto dividindo-o em categorias e exemplificando, sendo que para os autores a melhor definição usando categorias de contextos é a que pode ser vista abaixo:

- **Contexto computacional**: dados como conectividade de rede, custos de comunicação, bandas, recursos próximos;
- **Contexto de usuário**: como a localização do usuário, contexto social;
- **Contexto físico**: como nível de ruído, luminosidade, temperatura;
- **Contexto temporal**: como hora do dia, dia da semana, mês, ano;
- **Contexto histórico**: obtido usando os outros tipos de contexto;

A definição de contexto do trabalho é: "Contexto é um conjunto de estados ambientais e um conjunto de configurações que determinam o comportamento de uma aplicação ou ocorrência de um evento de uma aplicação que é de interesse do usuário". Diversos valores de contextos devem ser unidos para gerar um entendimento melhor da situação atual do usuário. Contextos primários, como localização, entidades, atividade e tempo atuam como índices de outras informações contextuais.

Baseados na definição de contexto os autores afirmam que existem duas formas de uma aplicação usar contextos: adaptando seu comportamento automaticamente de acordo com contextos descobertos (*using active context*); apresentando o contexto para o usuário ou armazenando contextos para uso posterior (*using passive context*). Com isso os autores oferecem duas definições de contexto:

- **Active context awareness**: Uma aplicação se adapta automaticamente ao descobrir um novo contexto, alterando seu comportamento;
- **Passive context awareness**: Uma aplicação apresenta um novo contexto ao usuário ou torna o contexto persistente para que o usuário o use posteriormente;

Segundo os autores, para que uma aplicação consiga usar contextos deve existir um mecanismo que adquira o contexto atual e o entregue à aplicação. Para os autores, os dados (ditos de baixo nível) mais comumente usados são: localização; tempo; objetos próximos; banda de rede; orientação e aceleração. Além disto podem ser enviados contextos mais sofisticados (ditos de alto nível) para a aplicação, como a "atividade atual" do usuário, estes contextos são adquiridos usando técnicas como visão computacional ou inteligência artificial para combinar contextos de baixo nível.

Portanto, faz-se necessário perceber quando dado contexto é alterado para então notificar as aplicações de tais mudanças. Como existem dados sobre contextos de diversas naturezas, existem diversas formas de representá-los, por exemplo, dados sobre localização podem ser expressados de uma forma simbólica, usando contextos como: "perto de", "em casa"; ou então usando coordenadas geográficas. Existem outros fatores que devem ser levados em consideração como granularidade, escalabilidade e disponibilidade de sinais.

Faz-se necessário também separar aplicações do código responsável por adquirir contextos usando uma camada intermediária que adquira/processe contextos e os envie às aplicações. Para isso existem duas abordagens que podem ser seguidas: uma arquitetura centralizada, que provê dados sobre contextos para aplicações, abordagem que pode apresentar problemas de escalabilidade; ou uma arquitetura distribuída, que obtém dados sobre contextos de diversos lugares e os envia às aplicações.

**Context is key**

Segundo o trabalho [Coutaz et al., 2005] contexto não é simplesmente um estado de um ambiente pré-definido com um conjunto fixo de interações, mas sim parte do processo de interação com um ambiente que está sempre mudando composto de recursos reconfiguráveis, migratórios e distribuídos. A computação ubiqua faz parte de um modelo no qual usuários, serviços e recursos descobrem outros usuários serviços e recursos, e integram-se com estes para formar uma nova experiência. Para isso existem dois processos cruciais: reconhecer os objetivos e atividades do usuário e mapear estes objetivos e atividades em uma população de serviços e recursos.

Posteriormente os autores levantam três questões sobre contextos: Contexto não é um simples estado, e sim parte de um processo. Não é suficiente para um sistema saber um contexto correto em um dado instante, este contexto deve estar correto enquanto o usuário está envolvido em um processo; A "melhor adaptação possível", normalmente, é determinada pela fusão de diversas informações; Há o risco de existirem diferenças entre o modelo de interação do sistema (forma como o sistema deve ser usado), como o modelo mental que o usuário possui do sistema (forma como o usuário tenta usar o sistema).

O trabalho propõe um framework conceitual para sistemas sensíveis a contexto baseado em dois componentes: Um *Ontological Foundation* que modela contextos como um grafo direcionado de estados onde cada nó é um contexto e as ligações denotam as mudanças ocorridas no contexto e cada contexto é definido por um conjunto de entidades, um conjunto de regras (que as entidades devem satisfazer) e um conjunto de relacionamentos entre as entidades e um *Runtime Infrastructure Model* responsável por adquirir/prover contextos como um "utilitário público"para outros serviços.

**Contextphone: A prototyping platform for context-aware mobile applications**

[Raento et al., 2005] apresentam um protótipo de uma plataforma para aplicações móveis sensíveis a contextos chamada de *ContextPhone*. Esta plataforma foi desenvolvida usando C++ e baseia-se em um conjunto de bibliotecas que oferecem funcionalidades como:

- Prover contexto como um recurso que pode ser entendido e alterado por humanos;
- Incorporar aplicações existentes, como aplicações de mensagens instantâneas;
- Oferecer uma interação rápida;
- Ser robusta, se recuperando de faltas de energia ou rede;
- Permitir que os usuários controlem relações como as geradas entre a carga da bateria e o uso da rede;

- Enfatizar linhas do tempo dos contextos;
- Permitir um desenvolvimento rápido de aplicações.

A arquitetura do *contextphone* baseia-se em 4 módulos: um módulo responsável pelos sensores; um módulo responsável pela comunicação, que usa IP; um módulo responsável pela customização de outras aplicações; e um módulo responsável pelo lançamento automático de serviços baseados em contextos.

#### Marcopolo

O [Symonds, 2007] é um software para Mac OS capaz de detectar diversos contextos do dispositivo, como: presença de dispositivos *USB*, portas *FireWire* ligadas, monitores plugados, redes sem fio disponíveis, hora do dia, etc. Com base nestes dados o software consegue configurar alguns aspectos do dispositivo, como: fundo de tela, impressora padrão, regras de *firewall* entre outros.

#### ControlPlane

O software [Rue, 2012] baseia-se no código-fonte do Marcopolo para detectar diversas informações contextuais e com base nestas informações é possível configurar algumas caracteristicas do dispositivo onde ele está instalado. Este programa difere do Marcolopo por ser mais atualizado.

#### Sidekick

O [Oomph Inc, 2012] é um software para Mac OS que consegue atualizar algumas configurações de notebooks baseando-se na localização geográfica do dispositivo. Para isso é necessário instalar o software e configurar os locais onde o notebook é mais usado, como casa, escritório, etc. Após detectar um determinado local o software é capaz de configurar alguns serviços, como o lançamento de aplicativos.

## 2.3 Contexto e sistemas operacionais

Como visto existem várias pesquisas sobre CAC, porém a maioria destas usa como contexto principalmente a localização do usuário ou entidades próximas. Poucos trabalhos tratam de contextos computacionais e nenhum dos trabalhos pesquisados para este trabalho apresentou pesquisas sobre contextos computacionais em SO.

Existem alguns artigos que descrevem o que é um contexto computacional [Schilit et al., 1994, Chen and Kotz, 2000], enquanto outros trabalhos [Hong and Landay, 2001] discutem formas de criar aplicações sensíveis a contextos. O primeiro trabalho a definir CAC [Schilit et al., 1994], divide contextos em três categorias: contexto computacional, do usuário e contexto físico.

Quando Schilit definiu contexto computacional o foco de sua definição estava em redes e dispositivos próximos, porém este trabalho assume que existem mais dados sobre o contexto computacional que podem ser usados. Posteriormente [Chen and Kotz, 2000] adicionou duas novas categorias a estas três categorias de Schilit: contexto temporal e contexto histórico.

Usando os contextos computacional, temporal e histórico é possível criar novos contextos, e com estes criar sistemas sensíveis a contexto capazes de reconfigurar automaticamente alguns comportamentos do SO de forma que a interação com o usuário fique mais eficiente.

Segundo Schilit existem quatro categorias de aplicações sensíveis a contextos, dentre elas duas podem ser usadas em SO para reconfigura-lo, a reconfiguração automática por contextos e as ações disparadas automaticamente por contextos. Usando estas técnicas e os contextos acima mencionados em um SO é possível criar uma interação mais sofisticada entre o usuário e o sistema.

É possível extrair diversos dados de SO para caracterizar diversos contextos. Por exemplo no Linux [Torvalds, 1991], o pseudo sistema de arquivos `/proc` [Killian, 1984, Faulkner and Gomes, 1991] contém diversos dados sobre a situação atual o SO e seus processos, estes dados poderiam ser lidos, analisados e compilados em contextos sobre o SO.

O `/proc` pode ser visto como uma janela de comunicação entre o espaço de usuários do sistema (*user level*) e o núcleo (*kernel level*), onde processos do espaço de usuário podem obter dados sobre o estado do sistema. O `/proc` apresenta dados de dois tipos: dados sobre o estado interno do núcleo Linux e dados sobre cada um do processos que atualmente estão em execução no sistema.

Sobre o estado SO o `/proc` provê diversas informações, como: número que arquivos abertos, número de trocas de contexto, número de processos em execução, informações sobre redes disponíveis, sistemas de arquivos disponíveis, dispositivos plugados, uso do processador, uso da memória, etc.

No `/proc` existem também diversas informações sobre cada um dos processos que são executados no SO, como arquivos abertos pelo processo, uso de memória pelo processo, status do processo, prioridades dos processos, número de threads, estado das threads, entre outras informações.

Todas estas informações sobre o núcleo do SO e seus processos podem ser organizadas para gerar contextos computacionais que facilitem a interação do usuário com o sistema. Estes e outros contextos podem ser usados pelo próprio SO ou podem ser oferecido pelo SO como um serviço, para outros aplicativos.

## 2.4 Considerações

Como visto, a CAC foi definida pela primeira vez na década de 1990 [Schilit et al., 1994], porém existem diversos autores que usam definições diferentes para CAC. Para dar continuidade a este trabalho faz-se necessária a definição de CAC, contexto e a categorização de contextos, para que posteriormente seja possível delimitar seu escopo.

Dentre todas as definições citadas na seção 2.2, algumas limitam-se a definir contexto usando funcionalidades de CAC para isso. As definições que usam funcionalidades, como: "um software capaz de se adaptar de acordo com a localização de uso"[Schilit et al., 1994], costumam ser pouco precisas, portanto não devem ser usadas para delimitar um escopo.

Outras definições usam como parte da definição de contextos, a configuração das aplicações que usam dados sobre contexto, como a definição de [Chen and Kotz, 2000]: "Contexto é um conjunto de estados ambientais e um conjunto de configurações que determinam o comportamento de uma aplicação ou ocorrência de um evento de uma aplicação que é de interesse do usuário".

Neste caso, ao atrelar à definição de contexto ao comportamento da aplicação que usa contextos, o autor está afirmando que duas aplicações que possuam comportamentos diferentes perante um mesmo conjunto de informações contextuais, estariam recebendo contextos diferentes, pois seus comportamentos são diferentes.

Visto todas as definições estudadas para este trabalho, a mais apropriada é a definição dada no trabalho [Abowd et al., 1999], que define contexto como: "Contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, um lugar ou um objeto que é considerado relevante na interação entre usuário e aplicação, incluindo a própria aplicação e usuário".

Uma das vantagens desta definição é sua abrangência, ao definir contexto como qualquer informação que caracterize a situação de uma entidade. Além disto, ao incluir na definição de contexto o estado da própria aplicação, a definição permite que um SO possa usar seu estado interno como parte do contexto.

O trabalho [Abowd et al., 1999] define CAC da seguinte maneira: "Um sistema é sensível a contextos se ele usa contextos para prover informações relevantes e/ou serviços para o usuário, onde relevância depende da tarefa do usuário". Definição esta que também será adotada para este trabalho.

Para categorizar contextos, dentre todas as classificações estudadas neste trabalho a classificação dada por [Chen and Kotz, 2000] foi a que melhor se adequou a este trabalho, por incluir categorias para contexto computacional, temporal e históricos. A classificação apresentada no trabalho de Chen pode ser vista abaixo:

- **Contexto computacional**: dados como informações de rede, custos de comunicação, bandas, recursos próximos;
- **Contexto de usuário**: como a localização do usuário, contexto social;
- **Contexto físico**: como nível de ruído, luminosidade, temperatura;
- **Contexto temporal** como hora do dia, dia da semana, mês, ano;
- **Contexto histórico** deduzido a partir de outros tipos de contexto.

Como visto neste capítulo, a CAC ainda não foi completamente definida, existem várias definições de CAC, e em alguns casos, um único autor apresenta definições diferentes [Abowd et al., 1999, Abowd and Mynatt, 2000]. Por isso foram adotadas definições de CAC e contexto e a categorização de contextos.

Pela revisão bibliográfica anteriormente apresentada percebe-se também que vários trabalhos usam como contexto apenas a localização do usuário [Harter et al., 1999, Cheverst et al., 2000b]. Porém poucos trabalhos tratam especificamente de contextos computacionais e nenhum destes trabalhos pesquisou profundamente sobre uma forma de associar CAC com um SO. Esta área pouco explorada da CAC será o tema do próximo capítulo deste trabalho.

# Capítulo 3

# Construção de Contextos

Após apresentar uma revisão bibliográfica sobre CAC, este trabalho define o que é contexto, explica o que é uma série histórica de contextos, apresenta o mecanismo responsável por construir contextos e por fim apresenta alguns estudos de caso para demonstrar como as definições apresentadas podem ser aplicadas.

## 3.1 Definição

Segundo [Abowd et al., 1999] contextos são quaisquer informações que podem ser usadas para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, um lugar ou um objeto considerado relevante na interação entre o usuário e a aplicação, incluindo a própria aplicação e o usuário.

No trabalho [Chen and Kotz, 2000] contexto é um conjunto de estados ambientais e um conjunto de configurações que determinam o comportamento de uma aplicação ou ocorrência de um evento de uma aplicação que é de interesse do usuário.

Com base na definição de Chen, este trabalho entende que um contexto pode ser visto como um conjunto de valores, aqui chamados de *informações contextuais* (*ic*). Estas informações contextuais devem estar relacionadas entre si de forma que, conforme Abowd, caracterizem a situação de uma entidade.

Cada informação contextual é composta por um conjunto de valores associados a intervalos de tempo, este conjunto de tuplas $(tempo, valor)$ são chamados de *séries históricas*. As séries históricas compõem o conjunto de valores que determinada informação contextual assumiu ao longo de um determinado intervalo de tempo.

Além disso, as informações contextuais devem ser construídas em um nível de abstração adequado para ser usado por outras aplicações interessadas em contextos, as quais tratariam da adaptação (segundo Chen, configurações que determinam o comportamento do sistema) ou ocorrência de eventos no sistema. Esta adaptação ou ocorrência de eventos, baseada em contextos é, em última instância, o que interessa ao usuário.

Assim, neste trabalho, contexto ($C$) pode ser definido como sendo um conjunto de informações contextuais (*ic*) relacionadas entre si e construídas em um nível de abstração adequado para ser usados por outros processos computacionais e aplicações interessadas em contextos.

Uma informação contextual (*ic*) é um conjunto de dados que representa a abstração de uma ou mais entidades correlatas do mundo real que são mapeadas no mundo virtual ou com-

putacional. Desta forma contexto e informação contextual podem ser definidos formalmente como:

$$C = \{ic_1, ic_2, ic_3, \ldots, ic_n\}$$

*onde:*

*C : Contexto*

*ic : Informação Contextual*

$$ic = \{< v_1, t_1 >, < v_2, t_2 >, \ldots, < v_n, t_n >\}$$

*com:*

$$t_0 < t_1 < \ldots < t_n$$

$$t_i - t_{i-1} = \Delta_t$$

$$n, i \in \mathbb{N}$$

$$1 \leq i \leq n$$

*onde:*

*ic : Informação contextual*

*v : Valor que a ic assumiu em determinado instante de tempo*

*t : Instante de tempo*

$\Delta_t$ *: Intervalo de tempo*

$t_n$ *: Instante de tempo atual*

*n : Número de valores armazenados*

$< v,t >$ *: Par formado pelo valor (v) da informação contextual (ic) e o instante de tempo (t) onde este valor foi considerado válido.*

Para determinar o intervalo de tempo ($\Delta_t$) onde determinado par ($< v_n, t_n >$) foi considerado válido é necessário determinar o intervalo de tempo $\Delta_t$ entre o par $< v_n, t_n >$ e o próximo par da seqüencia $< v_{n+1}, t_{n+1} >$. Desta forma, determinado valor ($v$) foi considerado válido no intervalo de tempo: $\Delta_t = [t_n, t_{n+1})$. A partir deste momento, o valor considerado válido é $v_{n+1}$.

Vale ressaltar que dentre todas as tuplas armazenada=s na série história, o último valor da série, com o instante de tempo $t$ mais próximo do instante de tempo atual é chamado de *valor*

*corrente* enquanto os outros valores são chamados de *valor passado*. Portanto valores correntes e passados podem ser definidos por:

**Valor Corrente** ($v_c$) : Valor associado a determinada informação contextual que é considerado válido para o instante de tempo atual;

**Valor Passado** ($v_p$) : É o valor associado a determinada informação contextual e a um determinado instante de tempo onde o mesmo foi considerado um valor corrente ($v_c$).

Com o mapeamento de parte do mundo real para dentro do mundo virtual, cada característica de uma ou mais entidades são representadas pelas informações contextuais (*ic*), as quais são agrupadas em contextos ($C$). Além disso, vale a pena ressaltar algumas características deste modelo:

- Neste modelo, uma informação contextual pode estar presente em diversos contextos. Para explicar esta situação pode ser usada a informação contextual "hora do dia". Esta informação pode estar associada com diversos contextos, como posição geográfica ou a disponibilidade de algum recurso ou serviço.

- Uma informação contextual também pode existir sem que ela esteja contida em qualquer contexto. Neste caso a informação contextual está apenas definida dentro do modelo, porém não está sendo usada para compor nenhum contexto neste instante, mas pode ser necessária para outro contexto em outro momento.

Esta situação será exemplificada no final deste capítulo, em um estudo de caso. A próxima seção deste capítulo visa mostrar de onde os dados que são usados para consolidar contextos são obtidos e como este dados são processados para que seja possível construir contextos, constituindo assim um modelo de dados.

## 3.2 Modelo de dados

Esta seção irá definir a origem dos dados usados neste trabalho, a forma como estes dados são processados para que sejam usados na construção de contextos. Para fazer isto, será usado como ponto de partida o conceito de contextos e será definido todo o processamento realizado até que seja possível chegar na origem e no destino dos dados usados como contextos.

Já foram definidos o que são contextos e informações contextuais, estas estruturas podem ser vistas na figura 3.1. Nesta figura existem três contextos $C_1$, $C_2$ e $C_3$ que são representados por círculos e sete informações contextuais ($ic_1, ic_2, ic_3, ic_4, ic_5, ic_6, ic_7$) que são representadas textualmente no diagrama.

Neste diagrama, os contextos são constituídos pelas seguintes informações contextuais: $C_1 = \{ic_1, ic_2, ic_5\}$, $C_2 = \{ic_1, ic_4, ic_5, ic_6\}$ e $C_3 = \{ic_7\}$. Vale ressaltar que as informações contextuais $ic_1$ e $ic_5$ são compartilhadas pelos dois contextos ($C_1$ e $C_2$) enquanto a $ic_2$ não está contida em nenhum contexto.

O objetivo de construir contextos é poder entregá-los aos processos que irão utilizá-los. No diagrama 3.2 podem ser vistos os contextos $C_1$ e $C_2$ sendo entregues aos processos $P_1$ e $P_2$. Vale ressaltar que neste modelo um processo pode receber mais de um contexto, como ocorre com $P_2$ e um contexto pode ser entregue a vários processos, como ocorre com $C_1$.

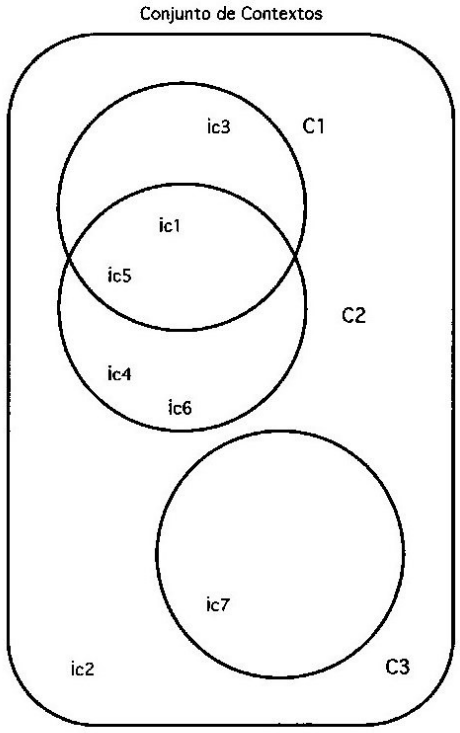



Figura 3.1: Modelo que representa contextos e informações contextuais

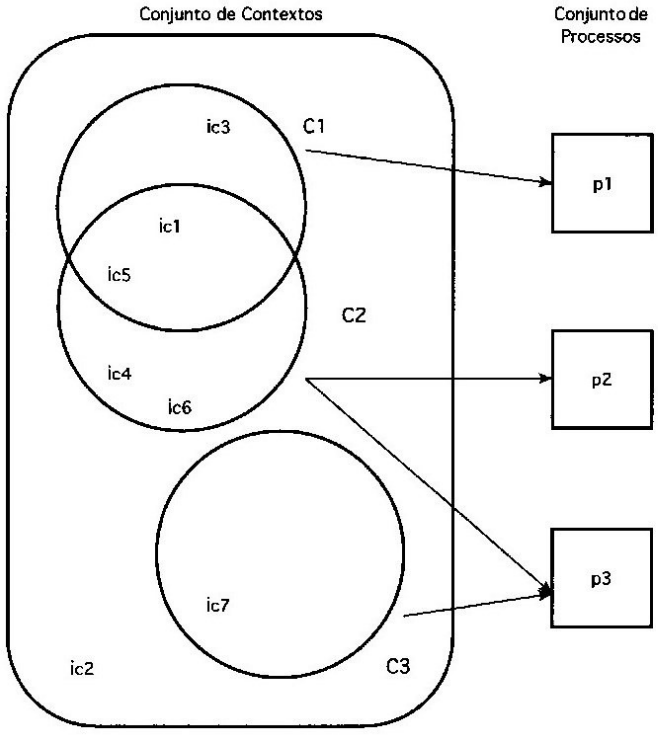



Figura 3.2: Modelo que representa contextos sendo entregues a processos

Fornecer um contexto a um processo significa tornar disponível o conjunto de valores das informações contextuais que estão associadas a determinado contexto. Diversos mecanismos podem ser usados para fornecer dados sobre contextos aos processos, eles serão discutidos

no capítulo 4. Vale ressaltar que, conforme a definição de contextos, estas informações devem possuir um nível de abstração adequado para serem fornecidos aos processos.

Neste caso a informação contextual $ic_2$ não pode ser fornecida a nenhum processo, pois não é um contexto mas apenas uma informação contextual. Caso algum processo necessite da $ic_2$, ela deve ser associada a qualquer contexto, mesmo que o contexto contenha apenas esta informação contextual e então poderá ser fornecida ao processo.

Uma informação contextual *ic* que não esteja associada a nenhum contexto é dita uma *informação contextual inativa*, pois está apenas representada no modelo, sem possuir valor atualizado. A partir do momento que esta *ic* esteja associada a um contexto, seu valor passa a ser calculado e atualizado e esta *ic* passa a ser uma *informação contextual ativa*. A diferenciação entre informações contextuais ativas e inativas pode ser útil na implementação do modelo, pois somente as informações ativas precisam ser periodicamente atualizadas.

Depois de definir o que é um contexto e qual é o seu destino, é o momento de definir qual é a origem dos dados usados para construi-lo. Para isto é necessário retomar a definição proposta para informação contextual: "uma informação contextual (*ic*) é um conjunto de valores que representa uma abstração de uma ou mais entidades correlatas do mundo real que são mapeadas no mundo virtual".

Assim, uma informação contextual é definida como uma abstração de uma entidade do mundo real, ou seja a origem de todos os dados é o mundo real, de onde os dados são coletados, por funções denominadas *funções coletoras* ($fc$). Vale ressaltar que no mundo real podem ser incluídos o usuário e também o próprio sistema que está coletando os dados.

Os dados coletados pelas funções coletoras são armazenados em abstrações do mundo real, aqui chamadas de *entidades*. Cada entidade é composta por um conjunto de valores correlatos denominados *atributos*. São os atributos que armazenam os valores de retorno das funções coletoras. Posteriormente os atributos das entidades são processados, por *funções agregadoras* e então os contextos são construídos e entregues aos processos. conforme indica a figura 3.3

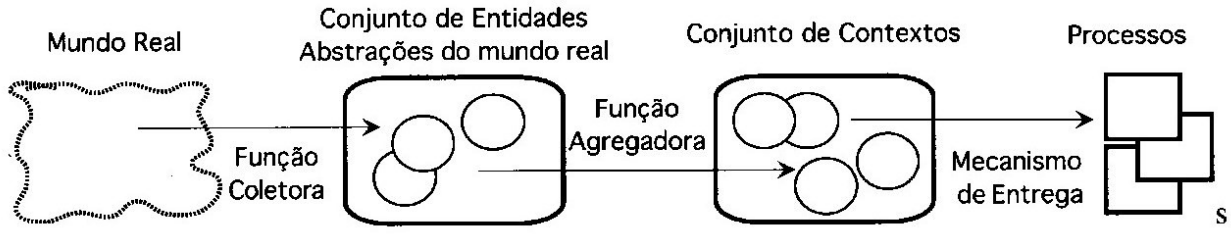



Figura 3.3: Visão geral do modelo

Após apresentar a visão geral do modelo cabe explicar melhor cada etapa do processo, iniciando pelas funções coletoras. Uma *função coletora* ($fc()$) é uma função responsável por coletar dados do mundo real, usando software e/ou sensores de hardware. Os dados coletados são os valores correntes de um ou mais atributos de determinada entidade.

A entrada de uma função coletora, sobre a qual a função coletora irá gerar suas saídas, será chamada de *entidade real* (*er*). Uma função coletora também pode obter dados de outras entidades e seus atributos ou de contextos e suas informações contextuais. Vale ressaltar neste momento que apenas contextos e informações contextuais podem ser entregues a processos, entidades e atributos não.

A principal diferença entre atributos e informações contextuais é o nível de abstração, pois informações contextuais devem possuir um nível de abstração adequado para deter-

minado problema. Como uma informação pode possuir um nível de abstração adequado para determinado problema e inadequado para outro, uma informação contextual também pode ser considerada um atributo e uma informação contextual simultaneamente.

As funções coletoras são ativadas seguindo uma periodicidade de leitura dos atributos de entidades ou informações contextuais onde o mesmo é usado, ou quando ocorre alguma mudança no mundo real de interesse do atributo que o mapeia e que foi percebida por sensores ou pelo sistema operacional. Uma função coletora pode ler dados de qualquer lugar, desde que retorne valores para serem armazenados em apenas uma variável.

Após uma função coletora adquirir dados, estes dados são armazenados em entidades ($E$). Segundo Chen, uma entidade é algo relevante na interação entre o usuário e o sistema. Como modelo de dados deste trabalho, uma entidade é definida como um conjunto de atributos ($at$) correlatos que visam representar aspectos do mundo real que sejam de interesse do sistema e/ou suas aplicações. Uma função coletora pode ser definida por:

---

$$at = fc(er_1, er_2, er_3, \ldots)$$

*onde:*

$at$ : *Atributo.*

$fc$ : *Função coletora.*

$er$ : *Entidades do mundo real que serão mapeadas no atributo* $at_x$.

---

Vale ressaltar algumas características desta parte do modelo como pode ser visto na figura 3.4. Uma função coletora pode obter dados de diversas origens, porém cada função coletora deve retornar valores para apenas um atributo de determinada entidade, esta característica permite que atributos diferentes possuam periodicidades de atualização diferentes.

Isto pode ser visto na $fc_3()$, que obtém dados sobre temperatura e umidade, porém armazena valores em apenas um atributo $at_3$. Também pode ocorrer de mais de uma função coletora acessar a mesma estrutura do mundo real, como ocorre com as funções coletoras $fc_4()$, $fc_5()$ e $fc_6()$ que processam dados sobre a luminosidade do ambiente.

Entidades podem possuir atributos em comum, porém cada atributo deve possuir uma e apenas uma função coletora, o que garante a periodicidade de cada atributo e permite que eles sejam organizados em entidades de forma independente. Um caso onde um mesmo atributo é compartilhado é o caso da data, onde diversas entidades podem conter o mesmo atributo (data) sem precisar recalcular o mesmo valor.

O fato de atributos terem a mesma origem, como ocorreu com os atributos $at_4$, $at_5$ e $at_6$ e o fato de entidades compartilharem atributos, como ocorreu com as entidades $E1$ e $E2$ que compartilham o atributo $at_3$ ocorre por que neste modelo a semântica destes atributos não é considerada, o mesmo ocorre na relação entre informações contextuais e contextos.

Após a coleta dos valores dos atributos das entidades, a próxima etapa necessária para gerar contextos é feita pelas funções agregadoras. Uma função agregadora ($fa()$) é responsável por construir a informação contextual a partir dos atributos das entidades. As informações

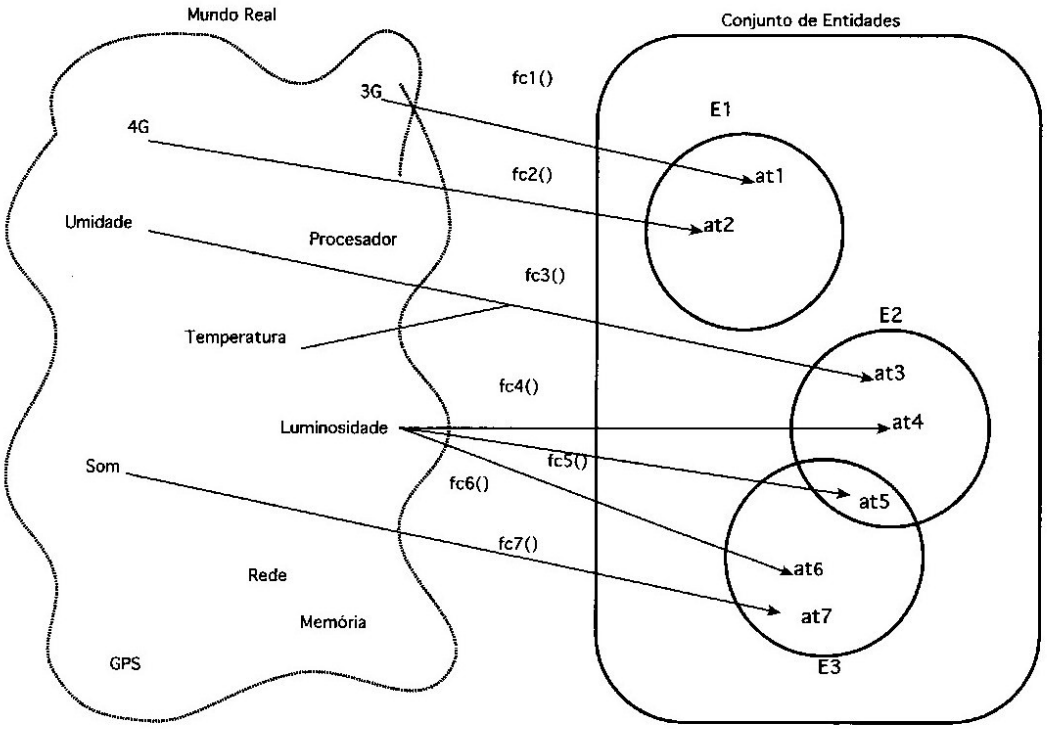



Figura 3.4: Coleta de Dados

contextuais devem estar construídas em um nível de abstração adequado para uso pelos processos que estão interessados em contextos. Estas funções são ativadas sempre que for necessário atualizar alguma informação contextual de algum contexto.Uma função agregadora pode ser definida por:

$$ic = fa(at_1, at_2, at_3, \ldots)$$

*onde:*

*ic : Informação contextual.*

*fa : Função agregadora.*

*at : Atributos do conjunto de entidades ET.*

Os atributos das entidades são considerados dados de baixo nível por não representarem necessariamente as informações no nível de abstração desejado pelos processos que solicitam contextos, logo a idéia é que as funções coletoras obtenham dados brutos e as funções agregadoras sejam responsáveis por transformar estes dados em contextos. As funções agregadoras tendem a ser funções baseadas em inteligência artificial, funções heurísticas, funções de tomada de decisões, algoritmos de reconhecimento de padrões, mineração de dados, ontologias, entre outros tipos de funções.

Assim como as informações contextuais, os atributos também são constituídos de séries históricas de valores que o atributo assumiu ao longo de determinado intervalo de tempo. Vale comentar que as principais diferenças entre atributos e informações contextuais são o nível de abstração dos dados e o fato de que atributos não podem ser entregues diretamente a processos e informações contextuais (*ic*) podem.

Atributos podem ser organizados em um conjunto de todos os atributos $AT$, assim cada entidade $E$ é um subconjunto deste conjuto $AT$, ou seja, $E_j \subset AT \forall j$. Da mesma forma $IC$ é o conjunto de todas as informações contextuais e cada contexto C é um subconjunto do conjunto $IC$, logo $C_j \subset IC \forall j$.

A figura 3.5 apresenta o ciclo de vida da informação de uma entidade real, até que possa ser entregue a um processo. Na figura *er* é uma entidade real, *at* um atributo, $E$ uma entidade, *ic* informação contextual, $C$ contextos e $p$ um processo que recebe um contexto.

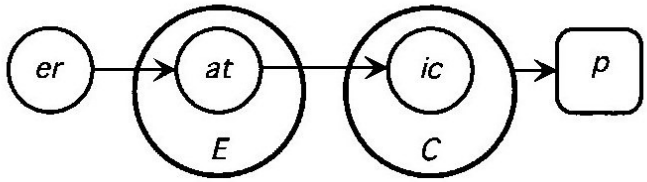



Figura 3.5: Ciclo de vida da informação de entidade real até um contexto entregue a um processo.

Como pode ser visto na figura 3.6, diversas funções agregadoras podem acessar um mesmo atributo de uma entidade, como ocorre com o atributo $at_2$ que é acessado pelas funções agregadoras $fa_2()$ e $fa_3()$. Além disto uma função agregadora pode acessar vários atributos, como ocorre com a $fa_4()$ que acessa os atributos $at_3$ e $at_4$. A figura 3.7 apresenta o diagrama completo do modelo de dados usado neste trabalho.

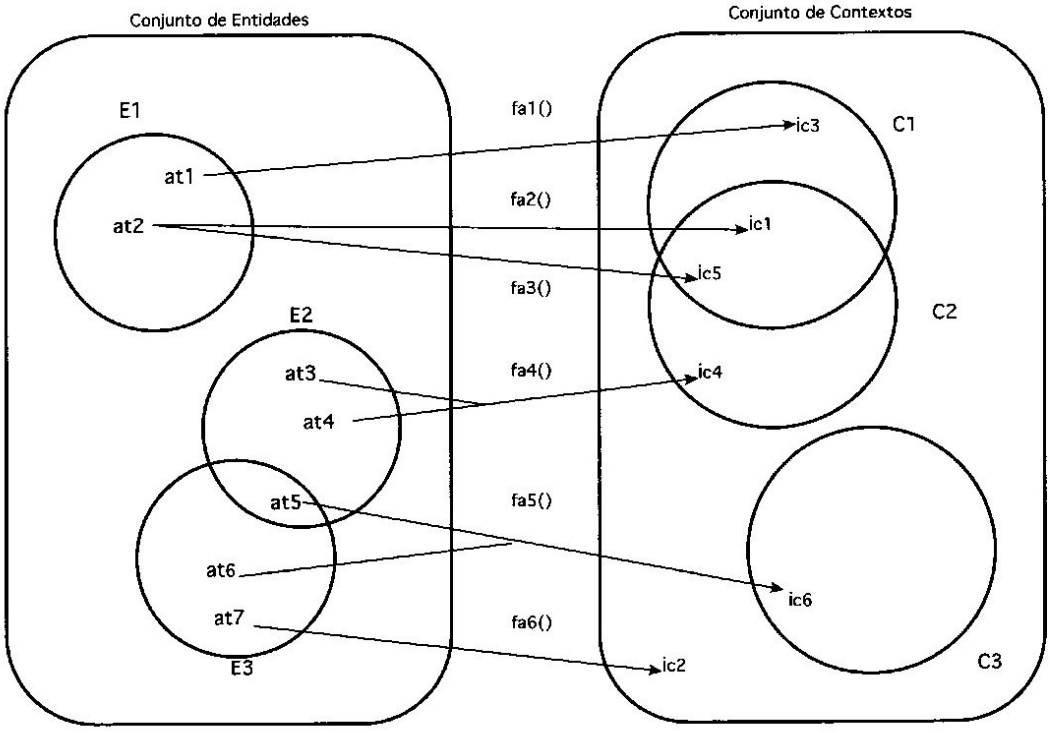



Figura 3.6: Processo de consolidação de contextos

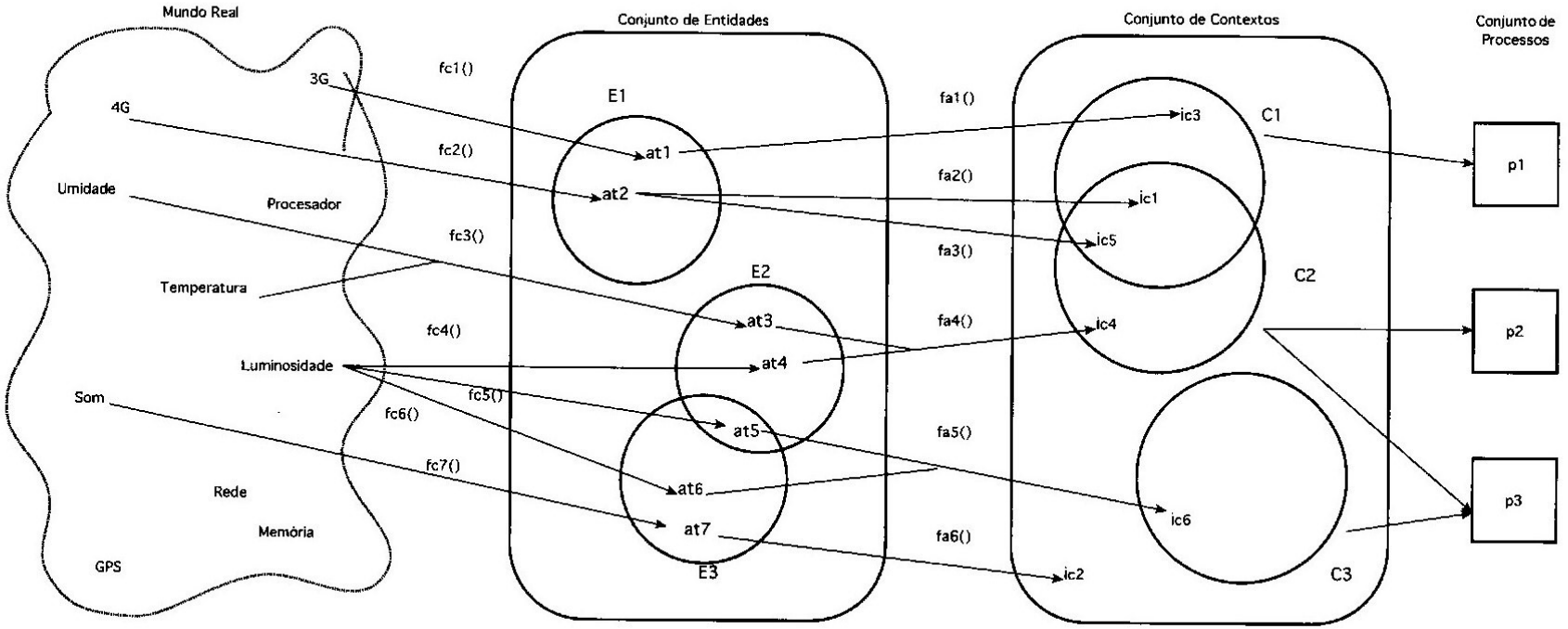



Figura 3.7: Diagrama completo do modelo de dados

## 3.3 Exemplos de aplicação

Nesta seção serão apresentados três estudos de caso com o objetivo de demonstrar como o modelo proposto pode ser aplicado. No primeiro exemplo será construído um contexto sobre conectividade, no segundo um contexto sobre o nível de conforto de usuários em determinado ambiente e no terceiro um histórico de contextos sobre a carga da bateria.

### Exemplo 1: Conectividade

O objetivo deste estudo de caso é a construção de um contexto que informe sobre o estado da conectividade de determinado dispositivo. Este contexto deve verificar quais das seguintes tecnologias de rede está disponível em determinado momento: 3G, redes sem fio e redes cabeadas. Com base nestas redes o contexto deve assumir um dos seguintes estados:

- **Nula**: Quando nenhuma rede estiver disponível;
- **Celular**: Quando apenas a rede 3G estiver disponível;
- **Wireless**: Quando a rede Wireless estiver disponível e a cabeada não;
- **Cabeada**: Quando a rede cabeada estiver disponível.

Os estados que este contexto pode assumir podem ser usados para configurar o dispositivo para sempre usar a conexão que normalmente é mais rápida, barata e gasta menos energia, desativando as outras conexões para economizar recursos computacionais e financeiros (no caso de conexões 3G que são normalmente as mais caras e lentas).

O primeiro passo para a construção deste contexto é a definição de uma entidade, chamada *Redes Disponíveis*, com 3 atributos do tipo booleano: *3g*, *wireless* e *cabeada*. Estes atributos irão assumir o valor verdadeiro para a disponibilidade de determinada rede e falso quando a rede não estiver disponível. O papel da função coletora de cada atributo será verificar a disponibilidade de cada rede, conforme o diagrama 3.8.

Após a criação da entidade e a atribuição de valores aos seus atributos, a próxima etapa do processo é a definição do contexto Conectividade que possuirá uma informação contextual também chamado de conectividade, o qual assumirá algum dos valores enumerados acima. A função agregadora conectividade será a responsável por atribuir o valor a esta informação, usando como base os valores dos atributos da entidade *redes disponíveis*.

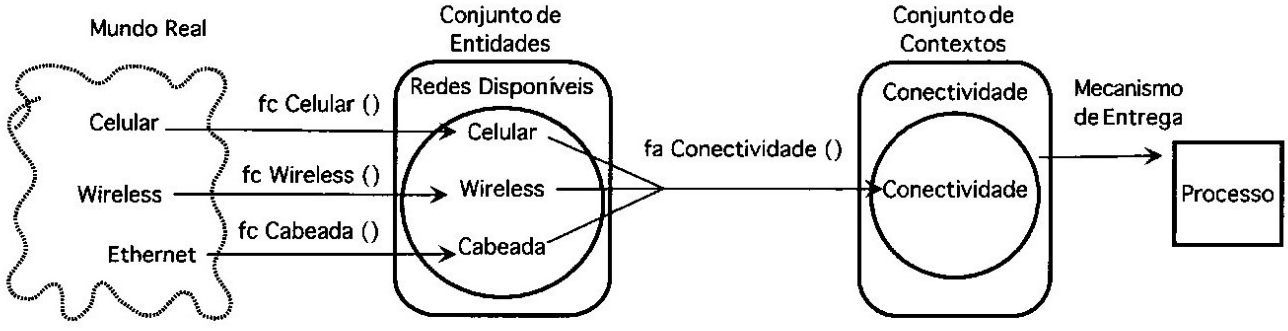



Figura 3.8: Processo de construção do contexto *conectividade*

As funções coletoras ($Fc_{Celular}()$, $Fc_{Wireless}()$ e $fc_{Cabeada}()$) podem ser executadas com uma granularidade alta e ativadas automaticamente pelo sistema operacional quando alguma rede que estava disponível tornar-se indisponível ou quando uma nova rede for detectada. Isto pode ser feito para economizar processamento e energia do dispositivo.

Além disto, caso a função agregadora ($fa_{conectividade}()$) detecte que a rede cabeada está disponível, não é necessário verificar as outras redes e caso a rede cabeada esteja indisponível e a rede wireless esteja disponível, não é preciso preciso verificar a rede 3G. Após a execução da função agregadora, conforme o código a seguir, a informação contextual estará atualizada e pode ser entregue aos processos interessados.

**Algoritmo 1** $fa_{conectividade}()$

```
if (at_cabeada == "ativa") then
    ic_conectividade ← "cabeada"
else if (at_wireless == "ativa") then
    ic_conectividade ← "semfio"
else if (at_celular == "ativa") then
    ic_conectividade ← "celular"
else
    ic_conectividade ← "nula"
end if
```

## Exemplo 2: Meio Ambiente

Pode ser construído um contexto sobre o ambiente onde o usuário/dispositivo está inserido, para este estudo de caso pode ser criado um cenário onde exista um conjunto de sensores de temperatura, umidade relativa do ar, luminosidade do ambiente, nível de ruído e presença de pessoas, que verifica a presença de pessoas pela ocorrência ou não de movimento.

Neste cenário são criadas duas entidades: *Ambiente*, que contem os atributos *luminosidade*, *nível de ruído*, *temperatura* e *umidade relativa do ar* e a entidade *Presença* que contém um atributo que assume verdadeiro quando detectar a presença/movimento de pessoas e falso quando não detectar ninguém.

Com base nestas entidades é possível construir contextos sobre o nível de conforto a que o usuário está submetido, para isto pode ser levado em consideração se a temperatura e a umidade estão em níveis adequados, se o nível de ruído não está alto e se a luminosidade ambiente não está muito alta ou baixa para determinado período do dia. Além disto, é possível criar um contexto *presença* que verifica se existe alguém ou não neste ambiente.

O processo de construção destes contextos pode ser acompanhado pela figura 3.9. Um processo poderia usar estes dois contextos para regular o nível de conforto de ambientes apenas quando existisse a presença de alguém no ambiente, economizando energia quando o ambiente estiver desocupado.

## Exemplo 3: Energia e Histórico de Contextos

O objetivo deste estudo de caso é a construção de um contexto sobre o estado da energia de um dispositivo. Além de construir um contexto sobre o estado da energia atual de

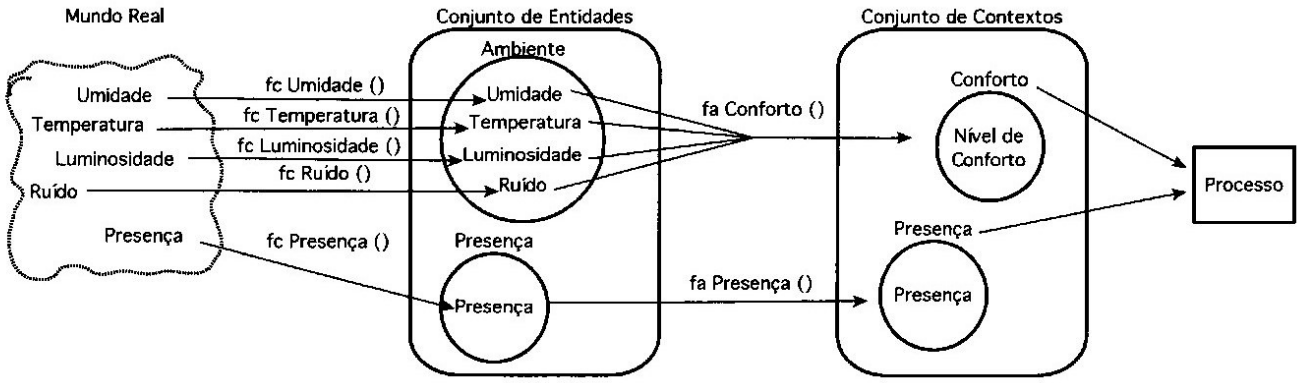



Figura 3.9: Processo de construção do contexto conforto

determinado dispositivo, pode ser interessante armazenar algumas informações para uso posterior. Estas informações poderiam ser usadas para verificar problemas na bateria, verificar a vida útil de determinada bateria ou até para traçar o perfil de consumo de energia do usuário/dispositivo.

Inicialmente é necessário construir o contexto sobre energia, para isto é necessário coletar informações sobre a forma como o dispositivo está sendo alimentado, se o dispositivo está na tomada ou não e se ele está conectado a uma bateria ou não. Também é necessário coletar informações sobre o estado da bateria, se esta estiver disponível. Podem ser coletadas informações como carga total, carga disponível, consumo imediato, etc.

O processo de coleta de informações pode ser visto na figura 3.10. A função coletora $fc_{Tomada}()$ verifica se o dispositivo está conectado na tomada ou não, a função $fc_{Bateria}()$ verifica se a bateria está disponível e se o dispositivo está carregando ou consumindo a energia da bateria. A função coletora $fc_{InfoBateria}()$ coleta as informações sobre a bateria, conforme sugerido anteriormente. Estas informações são armazenadas em duas entidades: Alimentação e Bateria.

Posteriormente a função agregadora $fc_{Energia}()$ usa as informações coletadas pelas funções coletoras e armazenadas nos atributos das entidades alimentação e bateria para criar um contexto sobre o estado da energia do dispositivo. Este contexto pode ser criado de diversas formas, apresentando a porcentagem da bateria que está disponível, o tempo restante de uso, em níveis de carga como: Tomada, Carregada, Crítica ou Vazia.

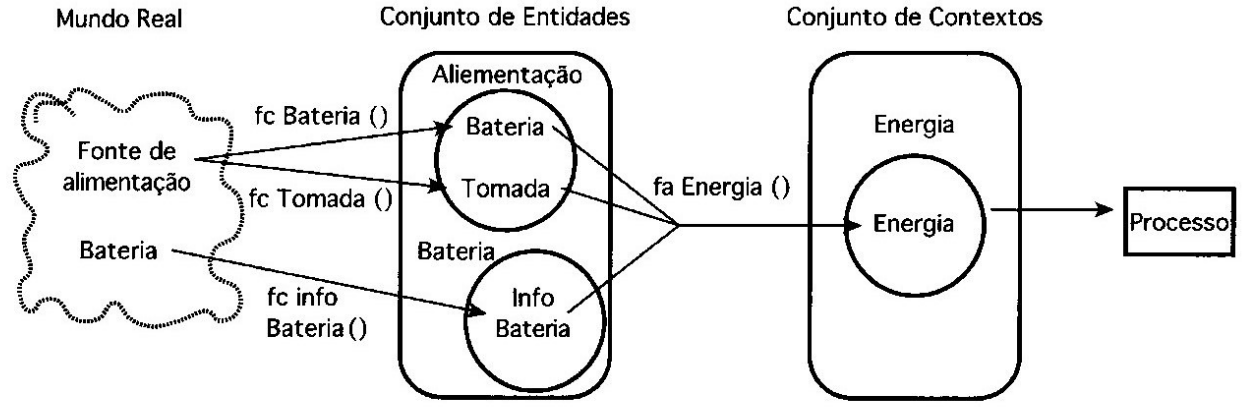



Figura 3.10: Processo de construção do contexto energia

Como cada atributo/informação contextual armazena uma série histórica dos valores que assumiu, caso um processo precise de dados sobre os valores antigos de algum atributo/informação contextua basta usar a API do *GC* para obter acesso aos dados, de forma que cada valor de medições estará organizado em pares com o instante em que a medição foi realizada.

## 3.4 Conclusão

Este capítulo definiu o que é um contexto ($C$) para este trabalho, para isto definiu as estruturas: informação contextual ($ic$), entidade ($E$), atributo ($at$), atributo do mundo real ($er$) e as funções: função coletora ($fc()$) e função agregadora ($fa()$). Este capítulo também apresentou o modelo de dados que é usado para gerar contextos e o processo de construção de históricos de contextos, para isto foram definidos contextos correntes e passados.

Foram apresentados três estudos de caso para demonstrar como o modelo pode ser aplicado em situações reais. Os estudos de caso envolveram um contexto sobre conectividade, conforto ambiental de um usuário, estado da energia de um dispositivo e um contexto histórico sobre a energia do dispositivo. O próximo capítulo irá usar as definições apresentadas neste capítulo para definir o que é o Gerente de Contextos, qual seu papel e como ele pode ser implementado.

# Capítulo 4

# O gerente de contextos

Como visto no capítulo 2, existem poucos trabalhos que tratam de contextos computacionais e não foi encontrado nenhum trabalho que tratasse especificamente da relação de contextos com o *sistema operacional* (SO). Como foi apresentado na seção 2.3 existem diversas utilidades para um sistema sensível a contextos integrado a um SO e diversas pesquisas podem ser feitas nesta área.

O capítulo 3 definiu o que são contextos e diversas estruturas que são necessárias para a consolidação de contextos, o processo de consolidação de contextos, o modelo de dados que pode ser usado para construir contextos e também apresentou três estudos de caso sobre como podem ser criados contextos. Assim as próximas etapas deste trabalho são definir o que é o *gerente de contextos* (GC) e propor uma forma de implementar este modelo dentro de um SO.

Este capítulo define o que é o GC e apresenta a proposta de um modelo para a criação de um GC para um SO de propósito geral capaz de construir e manter contextos. Este capítulo está dividido nas seguintes seções:Introdução; Gerente de contextos: define como funciona o GC; Contextos pré-definidos: enumera o conjunto de contextos que tornam o GC funcional; API: apresenta a API de acesso ao GC; Implementação: apresenta o protótipo implementado; Conclusão.

## 4.1 Introdução

Atualmente diversos SOs trabalham com informações que poderiam ser tratadas como contextos, porém estas informações estão dispersas sobre diversas APIs de acesso aos sistemas e suas bibliotecas. Esta seção irá apresentar algumas destas informações, explicar como estas informações estão disponíveis para usuários e programadores e irá mostrar como estas informações poderiam estar disponíveis como contextos.

O MAC OS Lion, último sistema operacional para computadores da Apple [Apple Inc, 2012] possui algumas características que poderiam ser tratadas como contextos, uma delas é a sensibilidade do SO à luminosidade ambiente. Usando a câmera que está disponível nos notebooks da marca, o SO capta a imagens para calcular a luminosidade do ambiente onde o usuário está.

Com estas informações, o SO consegue calcular e adequar o brilho da tela do notebook para que a luz emitida pela tela não seja incomoda ao usuário e para evitar que o computador gaste energia desnecessária. Quanto maior a luminosidade do ambiente maior é o brilho da tela,

de forma que em um ambiente escuro é necessário menos luz da tela para que o usuário consiga usar o computador com conforto.

Além de usar a luminosidade ambiente para calcular a o brilho da tela, o MAC OS Lion também usa a luminosidade para ativar luzes de fundo do teclado, que são usadas para melhorar a visibilidade das teclas quando existe pouca luz ambiente. Enquanto o ambiente estiver bem iluminado as luzes ficam apagadas, mas quando a luminosidade ambiente fica baixa as luzes do teclado são ativadas.

Diversos dispositivos portáteis, que rodam os sistemas operacionais IOS da Apple ou o Android [Google Inc, 2012] também possuem diversas informações que poderiam ser tratadas como contextos. A presença de fone de ouvidos ou não nos dispositivos que possuem diversos níveis de volume de som, dependendo da condição do aparelho, da presença de um fone e se o volume é de músicas ou chamadas.

A maioria dos dispositivos portáteis (como celulares, tablets e computadores de mão) também são sensíveis ao sentido do aparelho, se o mesmo está posicionado na vertical ou na horizontal, se está em pé ou deitado, em movimento ou parado, etc. Quando estes aparelhos tiram fotos estas fotos podem ser preenchidas com informações sobre localização, data e até sentido em que a câmera estava apontando.

Todas estas informações poderiam ser vistas como contextos e poderiam ser construídas pelo modelo apresentado no capítulo anterior. Porém atualmente estas informações estão acessíveis para os programadores nas APIs de programação dos dispositivos sem nenhum tipo de padronização e de forma descentralizada, o que dificulta o acesso à informação.

Pode-se usar como exemplo o caso de uma foto que possui informações sobre onde foi tirada, o sentido do dispositivo, o sentido da câmera e a data e hora em que foi tirada. Para inserir estas informações na foto, seria necessário conhecer as APIs de vários hardwares diferentes (relógio, bússola, GPS e acelerômetro) e aprender a manipular os dados de baixo nível oferecido pelo hardware, o que dificulta muito o trabalho do programador da aplicação.

Caso estas informações estivessem todas disponíveis na forma de contextos, centralizadas em uma única infraestrutura, seria necessário apenas uma API com a qual seriam solicitadas todas estas informações, que poderiam ser entregues ao aplicativo em um nível de abstração adequado para ser usado, e não em dados de baixo nível.

Esta camada de software, responsável por construir contextos e entregar estes contextos às aplicações interessadas é chamada de *gerente de contextos* (*GC*); e na próxima seção é apresentada uma proposta de um modelo para sua criação.

## 4.2 Gerente de Contextos

Esta seção irá definir quais são os objetivos do GC, quais vantagens a implementação deste GC pode oferecer, quais são os contextos que ele pode construir por padrão e como inserir novos contextos. Também irá apresentar uma arquitetura proposta para o GC, irá propor uma API de acesso ao GC e irá propor uma forma de como estes contextos podem ser fornecidos aos aplicativos interessados.

O GC é uma camada de software capaz de construir contextos segundo o modelo apresentado na seção 3.2. Esta camada centraliza a construção de contextos, o que facilitaria a implementação de softwares capazes de sentir contextos, pois todas as informações relevantes estarão disponíveis em uma única API, independente de hardware ou versão de outros softwares.

Esta camada de software também evita que diversos aplicativos precisem obter e tratar dados de baixo nível, como os oferecidos pelo harware, pois o GC já oferece dados em um nível de abstração adequado ao uso pelos aplicativos interessados, como descrito no capítulo 3. O GC segue o mesmo modelo apresentado na figura 3.3 que é reapresentada abaixo.

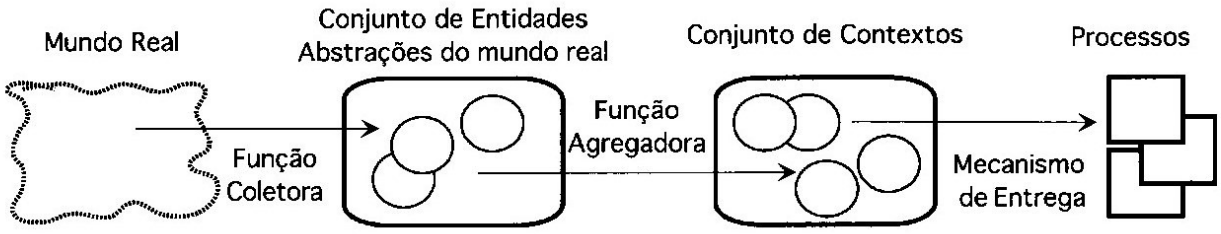



Figura 4.1: Visão geral do modelo

Para definir o que é o GC, é necessário ter em mente o modelo de dados apresentado na seção 3.2, como pode ser visto na figura 4.1. O GC é uma camada de software que implementa o modelo apresentado anteriormente e pode ser definido pela figura 4.2 que é apresentada a seguir.

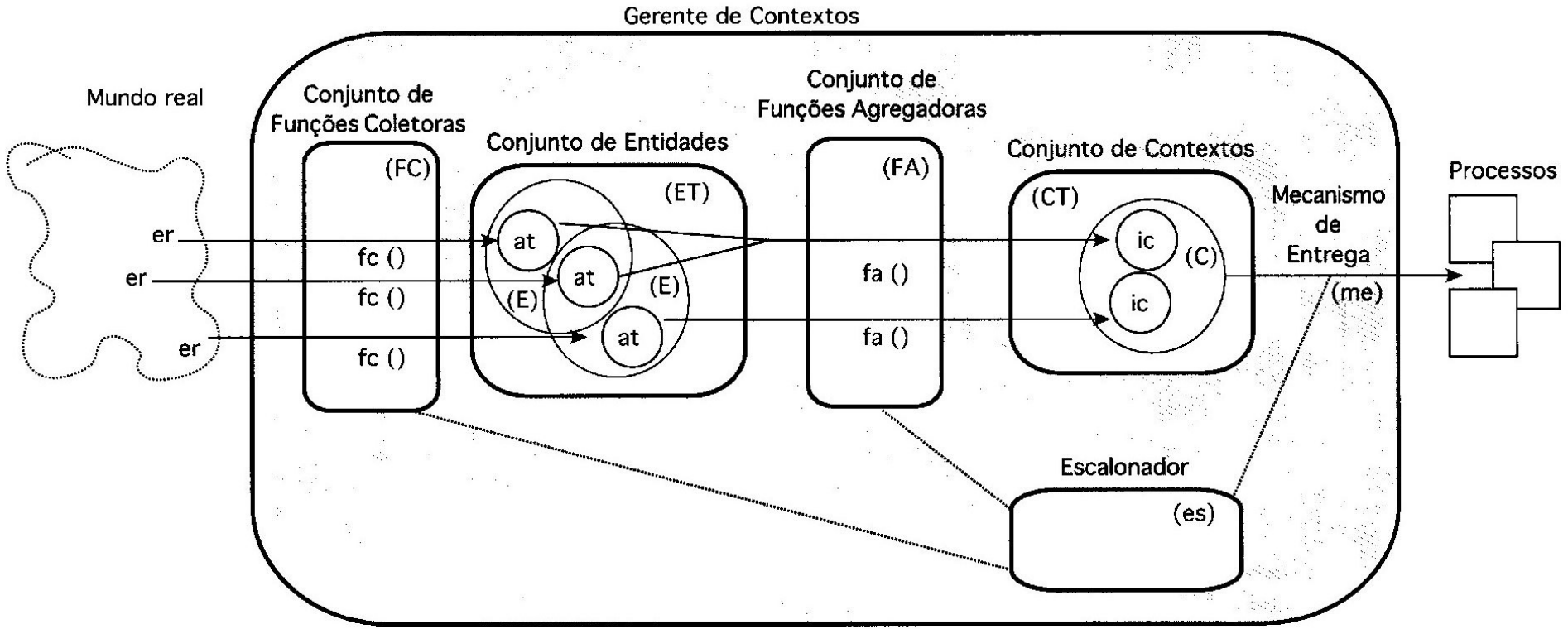



Figura 4.2: Visão geral do gerente de contextos

Como pode ser visto no diagrama, o GC é constituído pelos seguintes componentes: conjunto de funções coletoras ($FC$) e suas funções coletoras ($fc()$), conjunto das entidades ($ET$) suas entidades ($E$) e atributos ($at$), conjunto de funções agregadoras ($FA$) e suas funções coletoras ($fa()$), conjunto de contextos ($CT$) seus contextos ($C$) e informações contextuais ($ic$), escalonador ($es$) além da API. Cada um destes componentes está descrito a seguir.

- $FC$ é o conjunto formado por todas as funções coletoras que atuam no sistema. Estas funções possuem como entrada entidades do mundo real ($er$) e as saídas são os atributos ($at$) das entidades ($E$) do conjunto de entidades ($ET$).
- $ET$ é o conjunto de entidades representado no modelo de dados, onde cada entidade ($E$) é constituída por um conjunto de atributos ($at$).
- Atributos $at$ são uma série histórica de todos os valores que a função coletora associada a este atributo assumiu em determinado intervalo de tempo.
- O conjunto $FA$ é formado pelas funções agregadoras ($fa()$) que foram apresentadas no modelo de dados. Estas funções possuem como entrada os atributos ($at$) das entidades ($E$) e a saída destas funções são as informações contextuais ($ic$) dos contextos ($C$).
- O $CT$ é representado no modelo de dados por seus contextos ($C$) e informações contextuais ($ic$).
- Cada informação contextual ($ic$) é uma série histórica de todos os valores que a função coletora ($fa()$) associada a ele assumiu em determinado intervalo de tempo.
- O escalonador ($es$) é o mecanismo responsável por executar as funções $FC$ e $FA$ para que estas possam obter os valores dos atributo ($at$) e com estes construir os contextos ($C$). O escalonador funciona executando determinada função e posteriormente agendando sua próxima execução para um próximo intervalo de tempo pré-definido.
- O *mecanismo de entrega* ($me$) é a interface do GC com algum mecanismo que seja capaz de fornecer os contextos que foram construídos aos processos que estão interessados.
- O GC também precisa de uma $API$ que permita aos usuários e processos configurarem o sistema. Esta $API$ será usada para configurar o sistema, criar e excluir contextos ($C$), informações contextuais ($ic$), funções agragadoras ($fa()$), entidades ($E$), atributos ($at$) e funções coletoras ($fc()$). Também cabe acrescentar um conjunto de contextos padrão ao GC, este conjunto de contextos será apresentado na seção 4.3.

Ao centralizar todo o tratamento de contextos sobre uma mesma API, o uso de contextos por processos e usuários fica facilitado, pois o desenvolvedor precisa conhecer apenas uma API homogênea de acesso ao GC e não diversas APIs dos sistemas subjacentes. O desenvolvedor também não precisa trabalhar com diversas versões de hardware ou software, visto que o GC deve tratar disto.

Outra característica obtida com a implementação do GC é que em vez de diversos aplicativos tentarem acessar um dispositivo, como um GPS, temos apenas um (o GC) que se responsabiliza por acessar os dispositivos, processar os dados e entregar o resultado para os aplicativos interessados. Este comportamento pode melhorar o desempenho de alguns SOs,

hardwares ou até economizar bateria, pois evita que determinado dispositivo seja usado sem necessidade.

Além disto, como os contextos devem possuir um nível de abstração adequado para ser usado por processos e pelo usuário o desenvolvedor não precisará se preocupar com dados de baixo nível oriundo de dispositivos de hardware, pois o GC deve apresentar apenas dados prontos para serem usados.

Quando um contexto estiver pronto para ser entregue aos processos interessados, o *mecanismo de entrega* (*me*) realiza a entrega dos contextos aos processos. Diversos mecanismos de comunicação entre processos podem ser usados para fornecer estes dados aos processos interessados, como: RPC (*Remote Procedure Calls* - Chamada Remota de Procedimentos), DBus, Corba, Sockets, Pipes, memória compartilhada entre outros.

Estas tecnologias podem ser usadas de diversas formas, como no modelo cliente-servidor, onde os processos atuariam como clientes solicitando ao gerente, que atuaria como servidor, os contextos que precisarem. Neste modelo o cliente deveria conhecer o conjunto de contextos disponíveis, usando para isso a API do gerente, e então solicitando os contextos desejados, no momento em que necessitasse co contexto.

Usando filas de mensagens, o gerente atuaria empilhando contextos em uma fila, e o processo interessado atuaria realizando leituras nesta fila, onde cada nova leitura do processo seria um novo contexto construído pelo gerente. Caso o gerente use memória compartilhada, o gerente e o processo interessado estariam compartilhando os regiões de memória onde os contextos seriam armazenados.

Também podem ser usados canais de eventos como mecanismo de entrega (*me*), o que viabiliza usar o gerente de contextos de forma que um processo qualquer do sistema, usando a API do gerente, obtênha uma lista de de todos os contextos disponíveis no gerente, e se inscreve para receber todas as alterações de determinado contexto. Desta forma todo novo valor gerado para determinado contexto é entregue ao processo que o solicitou.

Podem ser usados mecanismos para realizar a troca de contextos de forma distribuída, assim determinado dispositivo pode receber contextos de outros dispositivos/sistemas e adequar-se a estes contextos. Dispositivos sem sensores de algum tipo, também poderiam receber contextos baseados nestes sensores, porém processados em outros dispositivos.

Ao realizar o processo de construção de contextos em duas etapas, uma de aquisição e outra de processamento, é possível que um dado coletado possa ser usado em diversos contextos. Ao coletar, apenas uma vez as redes sem fio disponíveis é possível gerar vários contextos como: recursos computacionais disponíveis, localização do usuário, contextos sobre nível de segurança contra ataques, presença de determinado dispositivo, etc.

Caso este processo fosse realizado de uma forma descentralizada, cada contexto gerado e baseado nas redes disponíveis precisaria adquirir o mesmo dado, consumindo recursos computacionais desnecessários. Ao centralizar o processo em um gerente e permitir que atributos sejam compartilhados, o tempo de processamento do gerente possui um aumento linear, ou menor, de processamento conforme aumenta o número de contextos.

Em um ambiente real, o gerente teria um número indefinido de contextos para serem construídos, dependendo da necessidade do usuário/aplicação e dos recursos/sensores disponíveis, etc. O desempenho computacional do gerente dependerá da quantidade de contextos que estão sendo gerados, dos algoritmos que estão gerando os contextos e da periodicidade com que estes contextos são construídos.

Tendo a estrutura básica do GC sido apresentada, a próxima seção irá apresentar a uma sugestão de um conjunto de contextos que poderia ser incluída no GC de forma que este possua uma funcionalidade mínima para auxiliar desenvolvedores de aplicações sensíveis a contextos.

## 4.3 Contextos pré-definidos

Além da estrutura básica, cabe ao GC possuir um conjunto pré-definido de contextos. Esse conjunto visa oferecer um ponto de partida par desenvolvedores de aplicativos sensíveis a contextos. Esta lista visa ser uma lista de sugestões de contextos e não ser uma lista exaustiva de possíveis contextos. Outros contextos podem ser criados com a ajuda da API do GC.

Vale informar que nem todos estes contextos foram implementados junto com o protótipo deste trabalho. O contextos sugeridos serão apresentados usando a divisão de de categorias de contextos sugerida por [Chen and Kotz, 2000] e pode ser vista a seguir:

### Contextos de usuário

Contextos do usuário são um conjunto de contextos que fazem referência ao estado do usuário. Estão listados a seguir um conjunto de possíveis contextos do usuário que poderiam ser gerados.

**Localização** : A localização do usuário pode ser calculada com a ajuda de sistemas de GPS, usando uma entidade GPS que receba dados de um dispositivo ou com a ajuda das redes disponíveis, com a ajuda de uma entidade redes, visto que a localização de algumas redes é conhecida. O contexto localização não precisa trabalhar necessariamente com dados de posição geográficos, mas pode também informar dados mais abstratos como: casa, escola, escritório, local público, etc.

**Tipo de atividade** : Podem ser usados dados sobre o processamento, carga da memória, quantidade de entrada e saída de dados para classificar o tipo de atividade que o usuário está realizando. Com estas informações o computador pode ser configurado para atender melhor as necessidades do usuário. Algumas entidades envolvidas neste contexto também seriam usadas para gerar contextos computacionais.

**Comunicação** : As formas possíveis de comunicação poderiam ser informadas, junto com seus respectivos custos. Estes dados podem ser usados para configurar o comportamento do dispositivo de forma a atender melhor o usuário e otimizar o uso de energia e recursos financeiros.

**Movimentação** : Informações sobre a movimentação do usuário poderiam ser consolidadas em um contextos. Dados como: velocidade, sentido e direção do movimento, sentido do dispositivo, etc. Estes dados podem ser adquiridos de acelerômetros, bússolas e sistemas de GPS presentes nos dispositivos.

### Contextos físicos

Contextos físicos são contextos que fazem referência ao ambiente físico onde o usuário está inserido. Diversos destes dados também poderiam ser usados para configurar o ambiente, como aparelhos de ar condicionado ou a abertura de janelas.

**Luminosidade** : A luminosidade do ambiente poderia ser calculada com a ajuda de uma câmera em uma entidade ambiente. Com este dado o brilho da tela pode ser ajustado para que fique mais confortável para o usuário usar o dispositivo. A luminosidade estaria associada a uma entidade ambiente.

**Nível de ruído** : O nível de ruído pode ser calculado para ajustar o volume do dispositivo. Um celular poderia aumentar o volume quando houver muito ruído no ambiente e diminuir quanto estiver em um ambiente silencioso. Este valor estaria associado a uma entidade ambiente em um atributo som.

**Temperatura** : A temperatura ambiente pode ser adquirida com a ajuda de um termômetro. Este atributo também estaria associado à entidade ambiente.

**Previsão do tempo** : Dados sobre a previsão do tempo poderiam ser obtidos de serviços especializado pela internet e poderiam ser oferecidos para o usuário e aplicações pelo *GC*. Estes dados também estariam associados à entidade ambiente.

## Contextos temporais

Os contextos temporais fazem referência ao tempo, os principais contextos são: data, hora, período do dia, estação do ano, dia da semana. Todos estes dados poderiam estar associados a uma entidade tempo e poderiam ser construídos apenas quando solicitados, sem que seja necessário gastar processamento construindo contextos desnecessários.

## Contexto computacional

O *GC* deve ser capaz de acessar dados internos do SO como número de processos atuais do sistema, redes em que o sistema está conectado, recursos disponíveis, etc para que com base nestes dados sejam construídos contextos computacionais.

O contexto gerado deve ser tratado para que possa ser armazenado e usado posteriormente, funcionando como contexto histórico ou passivo. Contextos históricos podem ser usados para modelar o comportamento do usuário e/ou do sistema, para automatizar algumas ações como o lançamento de certos aplicativos, para otimizar o uso de recursos, para melhorar a segurança do sistema entre outros usos.

Outro uso ativo dos contextos criados pelo gerente é o lançamento automático de aplicativos ou serviços. Como exemplos deste uso podem ser citados o lançamento automático de um certo aplicativo cada vez que o SO detectar que está conectado em uma certa rede, ou no caso de ativar um serviço de *backup* todo dia útil em uma certa hora ou condição. Segue uma lista de possíveis contextos.

**Energia** : Diversos dados sobre a energia poderiam ser obtidos, como presença de bateria, status da bateria (descarregando, carregando ou completa), carga da bateria, tempo restante de bateria, tempo restante para terminar de carregar a bateria, etc. Todos estes dados poderiam estar associado a uma entidade bateria.

**Comunicação** : Tipos de rede disponível (ethernet, wireless, 3G/4G, ...), banda disponível, custo, quantidade de dados enviados/recebidos, velocidade atual e média da rede, dados sobre a as interfaces (IP, mascara, portas, ...). Estes dados poderiam ser agregados em uma entidade rede.

**Armazenamento** : Uma entidade armazenamento poderia tratar dados como: volumes disponíveis, uso dos volumes, tipos de dados armazenados, etc.

**Dispositivos** : Uma lista de todos os dispositivos disponíveis para uso pelo usuário, presença de fone de ouvido, tamanho e tipo da tela, dispositivos de entrada de dados, etc. Estes valores podem ser usados, tanto pelo usuário quanto pelo SO para se configurar.

**Sistema Operacional** : Diversos dados do SO poderiam ser oferecidos como contextos, número de processos, carga do processador, carga da memória, dados do SO, e outros dados, como ocorre no sistema de arquivos /proc do Linux.

### Contextos históricos

Segundo Chen contextos históricos são contextos gerados a partir de outros contextos e poderiam ser armazenados conforme a necessidade do usuário ou de outros processos do sistema. Vale ressaltar que contextos históricos e séries históricas não são a mesma coisa. Contextos históricos são contextos gerados de outros contextos, séries históricas são o conjunto de valores que um atributo ou informação contextual assumiu em determinado intervalo de tempo.

## 4.4 API

A API oferecida pelo gerenciador de contextos às aplicações/processos que desejem registrar e fazer uso de contextos é definida por oito operações básicas: *cadastrar*, *associar*, *ativar*, *receber valor*, *receber contexto*, *desassociar*, *desativar* e *descadastrar*.

Estas funções são o conjunto mínimo de operações, porém nada impede que outras funções sejam adicionadas à *API* para torná-la mais sofisticada. Porém, com estas operações o uso da *API* fica simplificado, o que facilita o uso de contextos por processos e pelo usuário. As operações estão descritas a seguir:

### cadastrar

A primeira operação básica necessária para que o *GC* funcione é a adição de um novo atributo (*at*) ou informação contextual (*ic*), para o gerente. Esta função pode ser usada tanto para adicionar um atributo (*at*) quanto uma informação contextual (*ic*). Esta operação precisa dos seguintes argumentos para ser executada:

**Identificador** : identificador do atributo(*at*) ou informação contextual(*ic*), este pode ser o nome do atributo, com o qual o mesmo será identificado e localizado dentro do *GC*;

**Função** : função coletora ($fc()$) ou agregadora ($fa()$) responsável por obter o valor do item;

**Parâmetros** : este é o conjunto de parâmetros necessários para que a função acima descrita possa ser executada;

**Descrição** : descrição textual do item que foi adicionado ao GC.

Ao adicionar um item (*at* ou *ic*) é nescessário adicionar também uma função ($fc()$ ou $fa()$) que gere o valor deste item, seus argumentos e período. Assim esta operação é a principal operação do *GC*, com ela atributos (*at*), informações conetxtuais (*ic*), funções coletoras ($fc()$) e funções agregadoras ($fa()$) são adicionadas ao modelo.

## associar

Esta operação associa um atributo (*at*) a uma entidade($E$) ou uma informação contextual (*ic*) a um contexto($C$). Vale ressaltar de um atributo (*at*) pode estar associado a mais de uma entidade ($E$), o mesmo ocorrem com as informações contextuais(*ic*) e contextos ($C$). Esta operação precisa receber como parâmetros o identificador do atributo (*at*) e da entidade ($E$) ou da informação contextual (*ic*) e do contexto ($C$).

## ativar

Como descrito no capítulo 3 uma entidade ($E$) ou uma informação contextual (*ic*) podem estar ativas, quanto a função associada está sendo executada ou possui uma execução agendada, ou inativa, quando não apresenta um valor atualizado. Portanto é necessário criar uma função de ativação e outra de desativação na (*API*).

No momento que esta operação é executada a função associada ($fa()$ ou $fc()$) também é executada e uma próxima execução é agendada para o período associado. Desta forma o atributo (*at*) ou a informação contextual (*ic*) associada não poderá a apresentar um valor inválido. Os parâmetros necessários são: identificador e período e o intervalo total de tempo que deve ser armazenado na série histórica.

O período indica a periodicidade com que a função ($fc()$ ou $fa()$) será executada e o intervalo indica o período total de tempo que será armazenado na série histórica. Caso o valor informado para o período seja zero, esta função não terá sua execução agendada pelo escalonador e só será executada caso outra função a execute.

O intervalo total indica o tamanho da série histórica, por exemplo, caso uma função possua um período de 5 segundos e o intervalo da série possua um valor de 1 minuto (60 segundos), a série histórica terá um tamanho de 12 amostras ($60 \div 5 = 12$). Caso o valor informado para o intervalo total seja zero, apenas o último valor será armazenado.

## receber valor

Esta é a operação responsável por retornar o valor de determinado atributo (*at*). Esta operação é usada por funções agregadoras ($fa()$) para obter os dados dos atributos (*at*) das entidades ($E$). Esta operação precisa de apenas dois parâmetros: o identificador do atributo desejado e o intervalo de tempo desejado.

O intervalo de tempo informado será o intervalo de tempo que será buscado dentro da série histórica. Caso o atributo possua uma série histórica de uma hora de duração, é possível solicitar apenas os últimos dez minutos desta série informando o instante inicial e final do intervalo desejado e assim todos os dados dentro deste intervalo serão retornados.

Caso não seja informado um intervalo de tempo, toda a série histórica será retornada. Caso seja informado um intervalo de tempo que esteja completamente fora do intervalo de tempo da série histórica, apenas o valor mais próximo será retornado.

### receber contexto

Esta operação precisa receber como parâmetro o identificador de determinado contexto ($C$) e com esta informação ele irá buscar todas as informações contextuais ($ic$) associadas, reunir seus valores em uma estrutura e retornar esta estrutura. Também é necesário informar o intervalo de tempo desejado para a série histórica. Caso não seja informado um intervalo de tempo, toda a série histórica será retornada

### desativar

Esta operação desativa um atributo ($at$) ou informação contextual ($ic$) do $GC$. Ao desativar este item, a função ($fc()$ ou $fa()$) associada a ele para de ser executada e caso este seja o último item de uma entidade($E$) ou contexto ($C$) este também é excluído. O único parâmetro necessário para esta operação é o identificador do item a ser desativado.

### desassociar

Esta é a operação que desassocia um atributo ($at$) ou informação contextual ($ic$) de uma entidade ($E$) ou contexto ($C$). Os parâmetros necessários são os identificadores do atributo ($at$) e da entidade($E$) ou da informação contextual ($ic$) e do contexto($C$).

### descadastrar

Esta operação remove um atributo ($at$) ou informação contextual ($ic$) definitivamente do $GC$, se necessário as desativa e desassocia de todas as entidades e contextos.

## 4.5 Estudo de caso

Nesta seção será apresentado um estudo de caso com o objetivo de demonstrar como a *API* do *GC* pode ser usada para cadastrar e usar contextos. Para isso será usado um cenário onde o *GC* esteja executando e pronto para ser usado por qualquer aplicação que o necessite. Além disso será levado em consideração que o desenvolvedor já implementou as funções coletoras e agregadoras necessárias.

O contexto que será cadastrado é o contexto sobre conectividade, presente na seção 3.3. Este contexto informa como está a conectividade do computador, assumindo um destes estados: nula, paga, wireless e cabeada. Para relembrar o contexto conectividade a figura 3.8 será reexibida nesta seção.

O primeiro passo necessário para construir o contexto conectividade é realizar o cadastro dos atributos ($at$): 3$G$, *Wireless* e *Cabeada*. Além disso é necessário cadastrar a informação contextual ($ic$) conectividade. Isso pode ser feito com as seguintes operações:

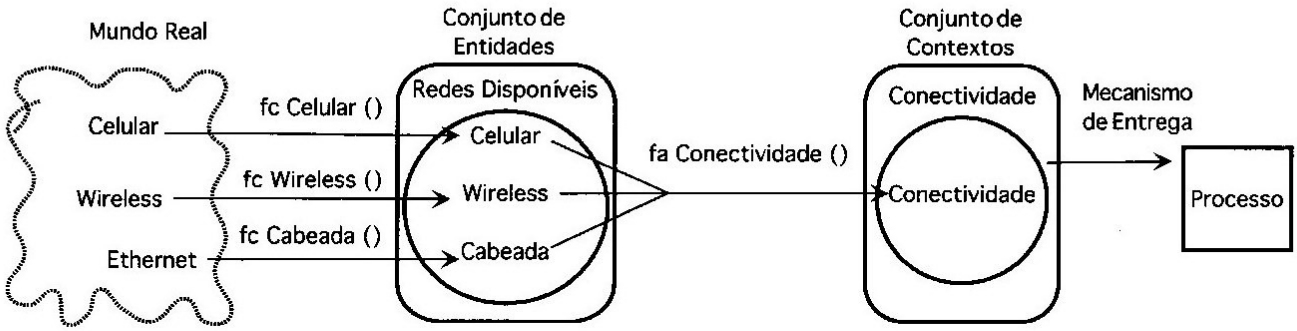



Figura 4.3: Processo de construção do contexto *conectividade*

---

**cadastrar** *("Celular", $f_c$Celular, NULL, "Rede Celular")*

**cadastrar** *("Wireless", $f_c$Wireless, NULL, "Rede sem fio")*

**cadastrar** *("Cabeada", $f_c$Cabeada, NULL, "Rede cabeada")*

**cadastrar** *("Conectividade", $f_a$Conectividade, NULL, "Ic Conectividade")*

---

Posteriormente associa-se os atributos (*at*) a uma entidade (*E*) "*Redes Disponíveis*"e a informação contextual (*ic*) ao contexto (*C*) *conectividade*. Para isso as seguintes operações devem ser executadas:

---

**associar** *("Celular", "Redes Disponíveis")*

**associar** *("Wireless", "Redes Disponíveis")*

**associar** *("Cabeada", "Redes Disponíveis")*

**associar** *("Conectividade", "Conectividade")*

---

Então é necessário ativar os três atributos e a informação contextual para que suas respectivas funções coletoras e agregadoras passem a obter os dados necessários para a série histórica. Como neste caso não é necessário armazenar séries históricas, o intervalo total a ser armazenado valerá zero. A seguinte lista de operações ativa os atributos com um período de 10 minutos (600 segundos) e sem armazenar séries históricas, ou seja, o intervalo vale zero:

**ativar** *("Celular", 600, 0)*

**ativar** *("Wireless", 600, 0)*

**ativar** *("Cabeada", 600, 0)*

**ativar** *("Conectividade", 600, 0)*

Feito isso, falta apenas obter o valor do contexto com a seguinte operação:

**receber contexto** *("Conectividade")*

## 4.6 Implementação

O objetivo desta implementação é a criação de um protótipo funcional para demonstrar a funcionalidade do modelo. Esta implementação não pretende apresentar todas as funcionalidades sugeridas na seção 4.3 nem foi concebida pensando em desempenho ou segurança, mas sim como uma forma de provar que o modelo proposto funciona como o esperado.

O protótipo foi desenvolvido sobre um sistema operacional Mac OS Lion (versão 10.7) na linguagem *Python* [python, 2012] (versão 2.7) no paradigma orientado a objetos. Esta linguagem foi escolhida por possuir um desenvolvimento ágil. Além disso o desenvolvedor deste trabalho possui experiência nesta linguagem e neste SO.

A implementação do GC inicia com a classe Gerente que possui dois objetos, *conjunto de entidades* ($ET$), *conjunto de contextos* ($CT$). Os conjunto de entidade ($ET$) e de contextos ($CT$) são o próprio conjunto de entidades ($ET$) e contextos ($CT$) presentes no modelo de dados. Ambos os conjuntos são inicializados no construtor da classe Gerente e herdam de uma classe chamada tabela, como pode ser visto na figura 4.4.

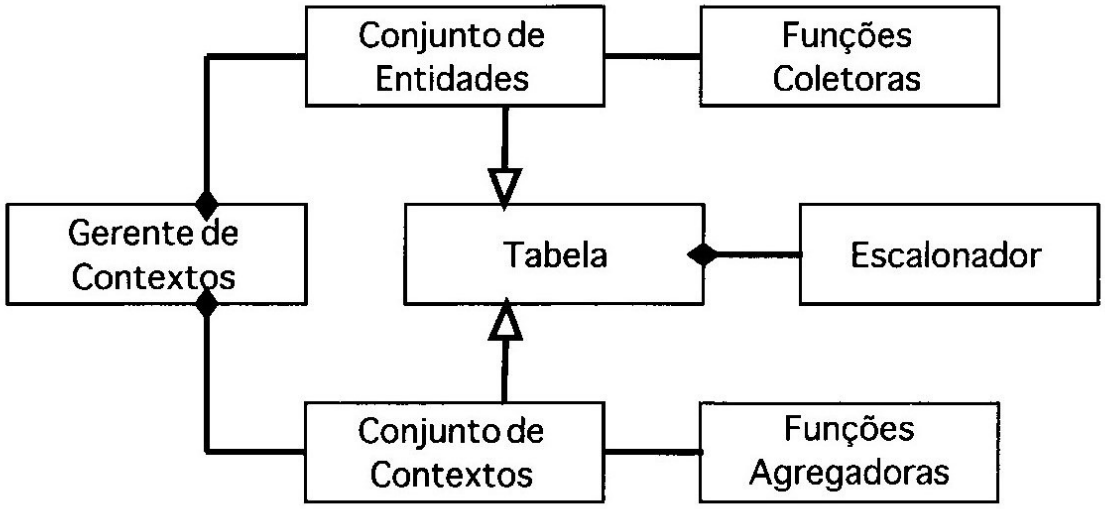



Figura 4.4: Diagrama de classe do GC

Como o nome sugere, a classe *tabela* funciona como uma tabela e foi implementada usando os dicionários da linguagem *Python*. Os dicionários são estruturas de dados capazes de armazenar pares de $(chave, valor)$, onde a chave é um identficador único e o valor que foi armazenado é uma tupla contendo um conjunto de valores como descrito a seguir.

**chave** : Identificador único de uma entrada no dicionário,
esta chave possui o mesmo valor do *id_item*;

**identificador** : identificador de um atributo ou informação contextual;

**identificador de grupo** : identifica a entidade ($E$) ou contexto ($C$)
associada a esta informação;

**função** : função coletora ou agregadora associada a esta informação;

**período** : intervalo de tempo entre execuções simultâneas
da função associada;

**intervalo** : intervalo de tempo total em que devem ser
armazenados valores na série histórica;

**argumentos** : argumentos necessários para a execução desta função;

**série histórica** : série de todos os valores retornados
pela função associada em determinado período de tempo;

**descrição** : descrição textual do item.

Desta forma, a tabela (super classe) e os conjuntos de entidades ($ET$) e contextos ($CT$) podem ser representados graficamente pela figura A.2. A classe tabela possui um objeto *escalonador* (*es*), o qual é responsável pelo processos de execução e agendamento de execuções das funções coletoras ou agregadoras ($fc()$ ou $fa()$) associadas a cada item (*at* ou *ic*) da tabela.

Com a estrutura do conjunto de entidades ($ET$) e do conjunto de contextos ($CT$) apresentada é possível definir a forma como o *GC* opera. Ao inicializar o *GC* ele inicializa duas tabelas ($ET$ e $CT$). Após este momento é possível incluir novos atributos (*at*) e informações contextuais (*ic*) no *GC*.

Ao incluir itens (*at* ou *ic*), estes serão armazenadas nas linhas das tabelas ($ET$ e $CT$) e suas funções ($fc()$ ou $fa()$) são executadas, gerando um valor para o item e suas próximas execuções são agendas pelo escalonador (*es*) para atualizar seus valores.

O desempenho desta aplicação dependerá do número de atributos e informações contextuais cadastradas, da forma como suas funções obtêm estes valores (suas funções coletoras $fc()$ e agregadoras $fa()$) e da periodicidade com que estas funções são executadas. O código implementado está no apêndice A deste documento.

## 4.7 Conclusão

Este capítulo definiu o que é o *GC*, o conjunto de funções coletoras ($FC$), o conjunto de entidades ($ET$), o conjunto de funções agragadoras ($FA$), o conjunto de contextos ($CT$), o mecanismo de entrega (*me*), a API (*API*) e o escalonador (*es*). Apresentou uma forma de como

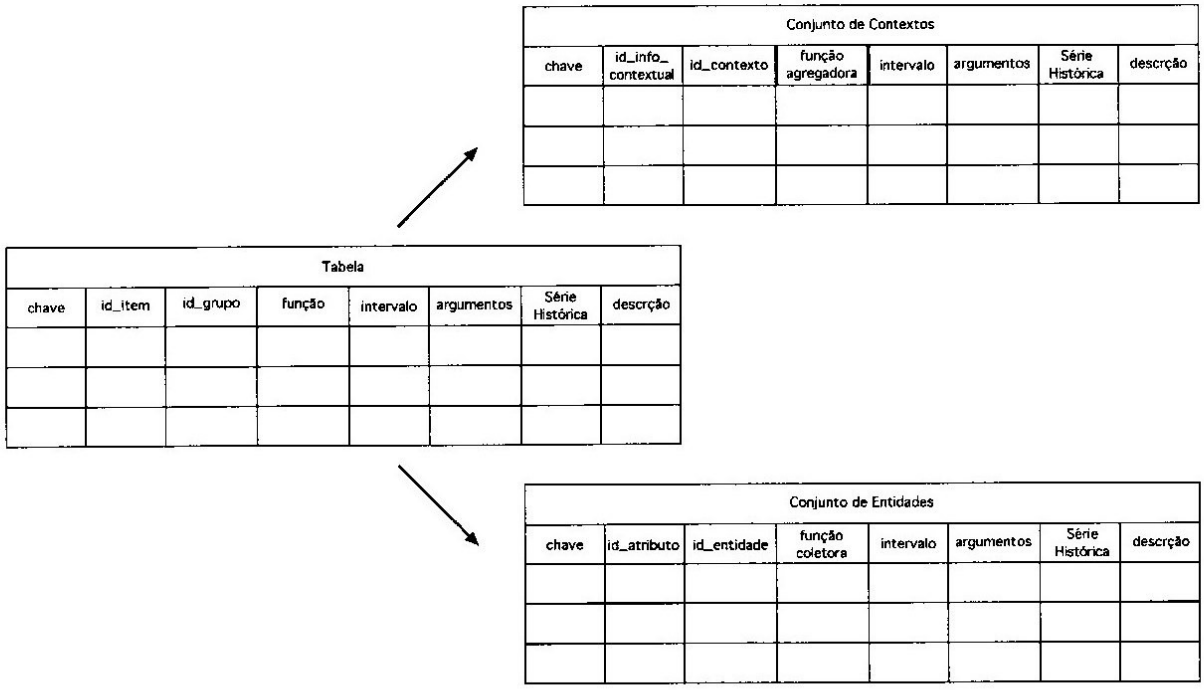



Figura 4.5: Representação das tabelas

um protótipo poderia ser implementado e também apresentou um conjunto de contextos pré-definidos. O próximo capítulo irá tratar dos resultados obtidos com o protótipo implementado.

# Capítulo 5

# Resultados obtidos

O capítulo 2 apresentou uma revisão bibliográfica sobre CAC delimitando o que é CAC e o que são contextos. O capítulo 3 definiu o que é contexto e apresentou um modelo de dados que pode ser usado para construir contextos. Posteriormente o capítulo 4 definiu o que é o GC e também apresentou um protótipo que implementa o modelo apresentado no capítulo 3.

Neste capítulo são apresentados alguns experimentos realizados com o protótipo apresentado no capítulo 4 e os dados obtidos destes experimentos. Este capítulo também irá apresentar os objetivos dos experimentos, o ambiente e as condições onde estes experimentos foram executados. Posteriormente será feita uma análise dos resultados obtidos.

## 5.1 Introdução

Este capítulo visa demonstrar que o modelo proposto no capítulo 3 e implementado no capítulo 4 atende os requisitos de funcionamento esperados, para verificar isso serão realizados alguns experimentos, dos quais serão extraídos alguns dados. Por fim serão elaboradas as conclusões gerais sobre o funcionamento, utilidade e efetividade do GC com base nos resultados obtidos.

O experimento consiste em executar o GC, cadastrar alguns atributos ($at$) e entidades ($E$) para que seja possível construir informações contextuais ($ic$) e contextos ($C$). Os valores de cada um destes itens serão obtidos e avaliados para verificar se GC está executando conforme o esperado. Esta verificação será feita confrontando os dados obtidos com o modelo apresentado no capítulo 3.

Também serão colhidos dados sobre o consumo de recursos computacionais pelo gerente de contextos, para verificar a eficiência do modelo em relação ao consumo de recursos. Além disso será analisado se o modelo facilita a atividade de desenvolver aplicações sensíveis a contextos ao comparar o desenvolvimento de aplicações sensíveis a contextos usando o GC ou não. A próxima seção estabelece os objetivos dos experimentos realizados.

## 5.2 Objetivos

Esta seção irá descrever os objetivos dos experimentos a serem realizados. O objetivo destes experimentos é coletar dados que permitam analisar o modelo e o protótipo implemen-

tado sobre três perspectivas: utilidade, efetividade e eficiência. A forma usada para analisar os dados sobre cada perspectiva está descrita a seguir:

**Utilidade** : Verifica se o modelo proposto realmente é útil para o desenvolvedor de aplicativos sensíveis a contextos, verifica se o modelo realmente facilita a atividade do programador diminuindo a necessidade de: usar dados de baixo nível; conhecer bibliotecas diferentes para cada contexto; acessar dados heterogêneos; armazenar valores históricos; acessar hardwares distintos;

**Efetividade** : Analisa se o protótipo implementado executa as operações descritas no modelo de dados apresentado no capítulo 3. Esta análise será feita confrontando o modelo apresentado com os valores que cada atributo ($at$) e informação contextual ($ic$) irá assumir dentro do GC. Além disso será verificada a origem e os destinos das funções coletoras ($fc()$) e agregadoras ($fa()$) do GC.

**Eficiência** : Verifica a quantidade de recursos que o modelo necessita para executar, principalmente memória e processador. Além disso irá analisar se o GC é capaz de ajudar o SO a economizar recursos de alguma forma.

Para este capítulo serão executados dois experimentos com o GC, no primeiro será construído um contexto sobre a conectividade do dispositivo; no segundo será captada a luminosidade do ambiente. As próximas seções descrevem como estes experimentos são realizados, em quais condições cada experimento foi executado e quais dados foram obtidos de cada etapa de cada experimento.

## 5.3 Conectividade

Este experimento baseia-se em construir um contexto que informe quais tecnologias estão disponíveis para o dispositivo se comunicar. O contexto ($C$) *conectividade* pode assumir um dos seguintes valores: *Nula*, sem redes disponíveis; *Sem Fio*, quando apenas a rede sem fio está disponível; *Cabeada*, quando apenas a rede cabeada está disponível; *Total*, quando ambas as redes estão disponíveis.

Para construir este contexto foi cadastrada uma entidade ($E$) chamada de *redes* com dois atributos ($at_1$): *ethernet* e ($at_2$): *wireless*. Estes atributos estão associados a funções coletoras que verificam se a rede, cabeada ou sem fio, está disponível ou não no momento e então estes atributos assumem um dos valores: *Ativa* ou *Inativa*.

As funções coletoras ($fc()$) *fc_ethernet_conected* e *fc_wireless_conected* são as responsáveis por verificar se a rede cabeada ou sem fio está disponível, para isso elas usam o SO e o comando *ifconfig* para verificar a disponibilidade da rede. Os atributos *ethernet* e *wireless* possuem seus valores atualizados por estas funções. Este processo pode ser acompanhado pela figura 5.3.

Após as funções coletoras ($fc()$) atribuirem um valor aos atributos ($at$) ethernet e wireless da entidade ($E$) redes, a função agregadora ($fa()$) *fa_conectividade* acessa os valores dos atributos ($at$) da entidade ($E$) redes e com base nestes valores atribui um valor à informação contextual ($ic$) *Conectividade* do contexto ($C$) *Conectividade*. Após esta etapa o contexto pode ser entregue a processos interessados.

Como o GC deve possuir um conjunto de contextos pré definidos, presume-se que as funções coletoras e agregadoras($fc()$ e $fa()$) associadas a este contexto já estejam defindas no GC, portanto para construir este contexto não é necessário implementar estas funções, é necessário apenas usar a API do GC da seguinte forma:

---

**cadastrar** *("ethernet", fc_ethernet_conected, (), "Verifica rede cabeada")*

**cadastrar** *("wireless", fc_wireless_conected, (), "Verifica rede sem fio")*

**associar** *("ethernet", "redes")*

**associar** *("wireless", "redes")*

**ativar** *("ethernet", 10, 1000)*

**ativar** *("wireless", 10, 1000)*

**cadastrar** *("conectividade", fa_conectividade, (), "Verifica a conectividade")*

**associar** *("conectividade", "conectividade")*

**ativar** *("conectividade", 10, 1000)*

---

Caso o desenvolvedor precisasse desenvolver algum contexto que não está definido no GC, seria necessário escrever as funções coletoras e agregadoras associadas ao contexto e também cadastrá-las no GC. Porém o GC se encarregaria de executá-las e de armazenar as séries históricas, conforme indicado na função de ativação da API do GC.

Neste experimento, a função de ativação das funções coletoras e agregadoras ($fc()$ e $fa()$) está informando uma periodicidade de 10 segundos e um intervalo total de 1000 segundos, isso indica que o GC irá armazenar dados sobre os últimos 1000 segundos destes atributos ($at$) e informações contextuais ($ic$), ou seja, irá armazenar os últimos 100 resultados ($1000 \div 10 = 100$).

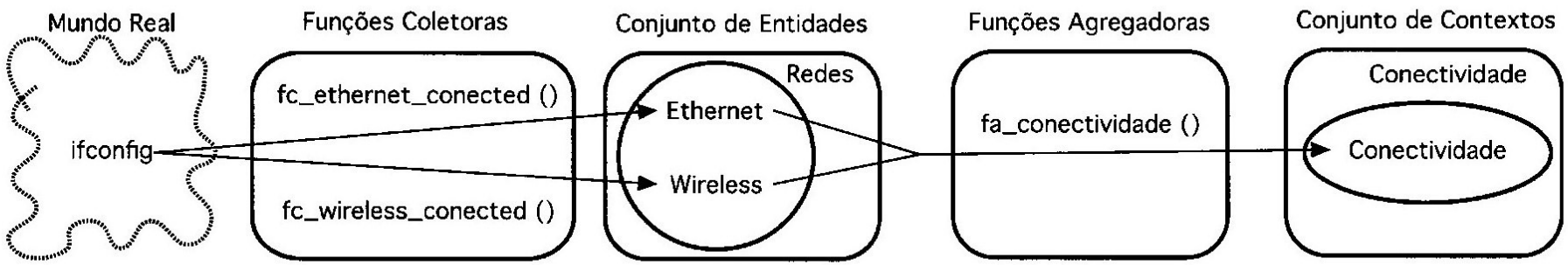



Figura 5.1: Processo de construção do contexto conectividade

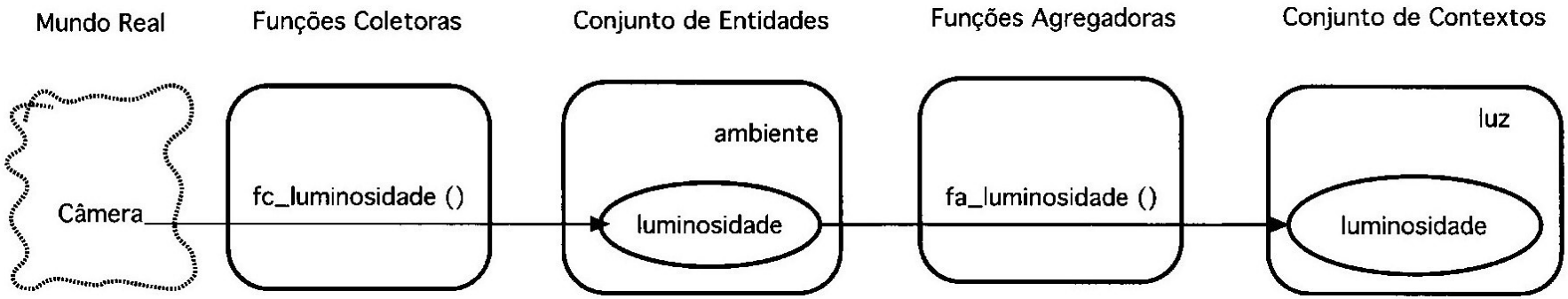



Figura 5.2: Processo de construção do contexto luminosidade

## Ambiente

Este experimento foi realizado em um computador da Apple, rodando o MAC OS X, versão 10.7.2 com um processador de 2.26 GHz e 4GB de memória. Este computator possui um interface de rede cabeada e outra sem fio. O GC está implementado em Python, versão 2.7 e seu código, incluindo o código deste experimento são apresentados no apêndice A deste documento.

As entidades do mundo real que foram acessadas neste experimento são os valores retornados pelo comando `ifconfig`. O comando `ifconfig` está presente em diversos sistemas operacionais baseados em UNIX e a seguir pode ser visto um exemplo de como este comando opera. As funções coletoras executam o comando `ifconfig` e retornam o status da interface (en0 ou en1) para o atributo (*ethernet* ou *wireless*) associado à interface.

```
# ifconfig
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST>
        options=27<RXCSUM,TXCSUM,VLAN_MTU,TSO4>
        ether 34:15:9e:0f:d0:fc
        media: autoselect
        status: inactive
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST>
        ether f8:1e:df:e5:72:18
        inet6 fe80::fa1e:dfff:fee5:7218%en1 prefixlen
64 scopeid 0x6
        inet 192.168.25.11 netmask 0xffffff00 broadcast
192.168.25.255
        media: autoselect
        status: active
```

## Execução

Este experimento foi realizado conforme descrito nesta seção, com os valores de cada atributo e informação contextual (*at* e *ic*) sendo atualizados de 10 em 10 segundos e uma série histórica de 1000 segundos, o que gera uma lista que armazena as últimas 100 medições.Durante a realização deste experimento as conexões do computador foram ativadas e desativadas para verificar se o modelo perceberia as alterações do ambiente.

A seguir serão apresentadas partes dos dados obtidos neste experimento para cada atributo e informação contextual (*at* e *ic*) do modelo. Não será apresentada toda a seqüência de 100 valores para cada atributo pois estes valores são apenas repetições deste padrão. O primeiro valor apresentado é o instante de tempo onde foi coletado o valor pelo GC, o outro valor é o valor do atributo/informação contextual (*at* e *ic*).

```
Interface cabeada:
    ((2012, 1, 22, 11, 32, 53, 290304), 'inactive');
    ((2012, 1, 22, 11, 33, 3,  296146), 'inactive');
    ((2012, 1, 22, 11, 33, 13, 305048), 'active');
    ((2012, 1, 22, 11, 33, 23, 311769), 'active');
    ((2012, 1, 22, 11, 33, 33, 318886), 'active');
    ((2012, 1, 22, 11, 33, 43, 326111), 'active');
    ((2012, 1, 22, 11, 33, 53, 333325), 'active');
    ((2012, 1, 22, 11, 34, 3,  341417), 'active');
    ((2012, 1, 22, 11, 34, 13, 354802), 'inactive');
    ((2012, 1, 22, 11, 34, 23, 363942), 'inactive');
    ((2012, 1, 22, 11, 34, 33, 369789), 'inactive')
```

```
Interface sem fio:
    ((2012, 1, 22, 11, 32, 53, 295008), 'inactive');
    ((2012, 1, 22, 11, 33, 3,  299919), 'inactive');
    ((2012, 1, 22, 11, 33, 13, 306306), 'inactive');
    ((2012, 1, 22, 11, 33, 23, 311918), 'inactive');
    ((2012, 1, 22, 11, 33, 33, 319006), 'inactive');
    ((2012, 1, 22, 11, 33, 43, 325999), 'active');
    ((2012, 1, 22, 11, 33, 53, 333549), 'active');
    ((2012, 1, 22, 11, 34, 3,  341285), 'active');
    ((2012, 1, 22, 11, 34, 13, 354382), 'active');
    ((2012, 1, 22, 11, 34, 23, 441207), 'active');
    ((2012, 1, 22, 11, 34, 33, 447186), 'active')
```

```
Contexto conectividade:
    ((2012, 1, 22, 11, 32, 53, 298895), 'Nula');
    ((2012, 1, 22, 11, 33, 3,  300034), 'Nula');
    ((2012, 1, 22, 11, 33, 13, 303301), 'Nula');
    ((2012, 1, 22, 11, 33, 23, 304913), 'Cabeada');
    ((2012, 1, 22, 11, 33, 33, 306741), 'Cabeada');
    ((2012, 1, 22, 11, 33, 43, 308583), 'Cabeada');
    ((2012, 1, 22, 11, 33, 53, 310413), 'Total');
    ((2012, 1, 22, 11, 34, 3,  312277), 'Total');
    ((2012, 1, 22, 11, 34, 13, 313897), 'Total');
    ((2012, 1, 22, 11, 34, 23, 315581), 'Sem Fio');
    ((2012, 1, 22, 11, 34, 33, 317160), 'Sem Fio')
```

O formato dos dados coletados é uma tupla, onde o primeiro item representa o instante em que o dado foi coletado. Por padrão a biblioteca do *python* utilizada no GC usa o formado apresentado acima para imprimir datas, Trata-se de outra tupla com informações sobre ano, mês, dia, hora, minuto, segundo e microsegundos em que o dado foi construído. A última string é o valor do atributo ou informação contextual (*at* e *ic*).

Durante a execução deste experimento também foram coletados dados sobre o consumo de recursos do GC para verificar sua eficiência. Para coletar estes dados foram aguardados 17 minutos ($1000 \div 60 = 16,66$), para que o GC tivesse coletado e armazenado todas os 100 valores de cada atributo e informação contextual (*at* e *ic*). Além disso foram coletados dados sobre um interpretador python que não estivesse executando programa algum, para comparar com a execução do GC.

| Processo | Uso da CPU | # Threads | Memória |
|---|---|---|---|
| Interpretador Python | 0.0% | 1 | 2628 Kb |
| Gerente de Contextos | 0.5% | 4 | 3316 Kb |

Tabela 5.1: Valores coletados sobre consumo de recursos.

A tabela 5.1 mostra dados coletados com o comando `top`, presente em diversos sistemas baseados no UNIX. O primeiro valor da tabela mostra a porcentagem do uso da CPU pelo processo, o segundo valor é o número de *threads* do processo e o último valor a quantidade de memória do sistema que está sendo usada pelo processo. A seguir será feita uma análise dos dados.

## Análise

A análise dos resultado será realizada levando em conta cada um dos objetivos descritos na seção 5.2 e posteriormente será verificado se o objetivo da proposta foi atingido. A primeira etapa desta análise irá verificar se o GC proposto e implementado possui utilidade, ou seja, facilita o processo de criação de aplicações sensíveis a contextos.

Para isso será retomado o código da API necessário para construir o contexto conectividade, apresentado no inicio desta seção. Para este experimento assume-se que suas funções coletoras e agregadoras ($fa()$ e $fc()$) já estejam cadastradas no GC, dentro do conjunto básico de contextos pré-definidos descrito na seção 4.3, o caso onde estas funções não existem será discutido posteriormente.

Para implementar o contexto conectividade é necessário usar 9 linhas e 3 funções. Não é necessário conhecer dados sobre interfaces de rede, protocolos de comunicação, ou qualquer dado referente a comunicação entre computadores. Não é necessário que o desenvolvedor conheça outras bibliotecas ou acesse dados heterogêneos, visto que o contexto é apresentado de forma textual.

O desenvolvedor também não precisou desenvolver códigos para o hardware, ou para armazenar dados históricos do seu contexto. O GC pode também ser usado como um daemon, verificando dados contextuais mesmo que o programa do desenvolvedor não esteja sendo executado, pois o GC pode entregar a série histórica de valores ao processo em outro momento.

Caso o programador precise de um contexto que não foi previamente definido e inserido no CG, isso significa que o desenvolvedor deverá escrever as funções coletoras e agregadoras ($fc()$ e $fa()$) associadas ao contexto ($C$) desejado. Neste caso o desenvolvedor não precisará

desenvolver o processo de criação de threads e escalonamento, normalmente necessários para tornar uma aplicativo sensível a contextos, pois o GC se encarrega destas tarefas.

Este desenvolvedor também não precisará desenvolver *daemons* responsáveis por coletar informações contextuais ou monitorar determinadas condições, pois o próprio GC pode se encarregar desta tarefa. Assim como o GC se encarrega de armazenar séries históricas de todos os valores necessários para o desenvolvedor. Existe ainda a possibilidade de integrar os dados gerados pelo desenvolvedor com os dados já cadastrados no GC.

Para verificar se o modelo teórico e o modelo implementado são coerentes, será confrontada a API usada para criar o contexto com o modelo teórico. Na API, a função *cadastrar* é responsável por criar atributos e formações contextuais (*at* e *ic*) no modelo. Esta mesma operação é responsável por cadastrar no GC as funções coletoras e agregadoras ($fc()$ e $fa()$) associadas aos itens.

Estes itens estão num primeiro momento inativos, e não associados a nenhuma entidade ou contexto ($E$ ou $C$). Depois de cadastrado é possível usar a função *associar*, que irá criar entidades ou contextos ($E$ ou $C$) e associá-los aos atributos e informações contextuais (*at* e *ic*). O próximo passo é a ativação dos atributos e informações contextuais (*at* e *ic*) para que estes possam executar suas funções coletoras e agregadoras ($fc()$ e $fa()$) e atualizar seus valores.

Para analisar se o CG está se comportando como esperado, pode-se analisar os dados coletados e apresentados nesta seção. Todos os dados iniciam em um mesmo instante e estão divididos em doze amostras. Neste momento os dados serão referenciados como 12 instantes (de $T0$ a $T11$). O progresso destes dados pode ser visto na figura 5.3.

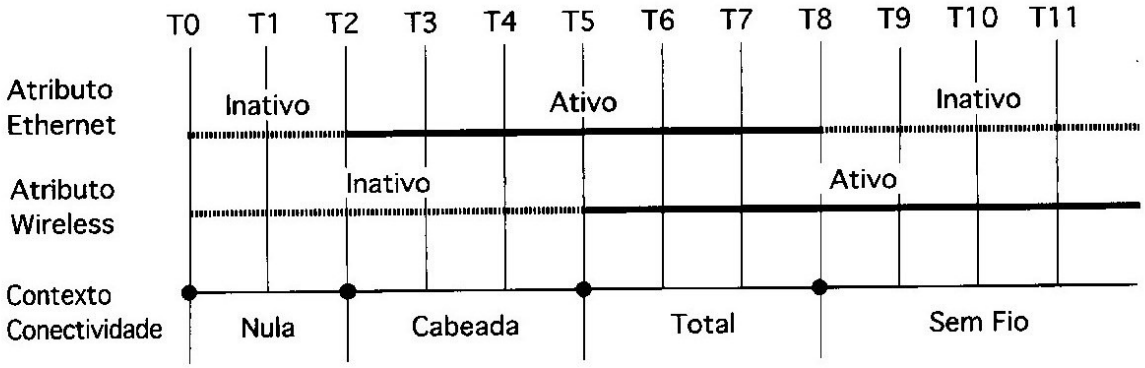



Figura 5.3: Diagrama contendos os estados do contexto conectividade

No primeiro intervalo de tempo $\Delta_1 = [T0, T2)$ tanto o atributo (*at*) ethernet quanto o wireless possuem o valor "inativo". Isto ocorre por que as funções coletoras associadas a estes atributos acessaram a entidade real (*er*), que é o comando ifconfig, e como estas interfaces não estavam ativas retornaram o valor "inativo"para seus respectivos atributos. Neste intervalo de tempo o contexto conectividade assumiu o valor "Nula", informando que a conectividade do dispositivo era nula.

No instante de tempo $T2$ a função coletora *fc_ethernet_conected* verificou que a interface cabeada estava ativa, neste instante o atributo (*at*) Ethernet assumiu o valor "ativo"e a informação contextual (*ic*) conectividade assumiu o valor "Cabeada", pois apenas o atributo (*at*) Ethernet estava ativo. Este mesmo estado se manteve durante o intervalo de tempo $\Delta_2 = [T2, T5)$.

Em $T5$ a função coletora *fc_wireless_conected* verificou que a interface de rede sem fio estava ativa, o status do atributo (*at*) Wireless mudou para "ativo"e conforme esperado a

informação contextual (*ic*) conectividade também mudou de estado para "Total", visto que ambas as interfaces estavam ativadas. Este estado manteve-se durante as amostras $T6$ e $T7$ ($\Delta_3 = [T5, T8)$).

No instante $T8$ o atributo ethernet mudou de estado para "Inativo", o que fez com que a informação contextual (*ic*) conectividade assumisse o valor "Sem Fio", visto que apenas a conexão sem fio estava ativada. Pode-se verificar que o GC comportou-se como esperado durante todo o intervalo estudado.

Sob o ponto de vista da eficiência no consumo de recursos computacionais, o GC consumiu cerca de $0,5\%$ do processador durante o período de tempo estudado. Vale ressaltar que este valor irá depender do número de funções ($fc()$ e $fa()$) a serem executadas, da periodicidade destas funções e dos códigos internos das funções. Funções mais numerosas, complexas e com um período menor farão o consumo de processador aumentar.

Neste experimento o GC usou 4 threads: o próprio GC e outra thread para cada função ($fc()$ ou $fa()$) ativa no sistema. No caso duas funções coletoras *fc_ethernet_conected* e *fc_wireless_conected* e uma função agregadora *fa_conectividade*. Quanto ao uso de memória, o GC usou $3316Kb$ de memória para ser executado, visto que o interpretador python sozinho precisa de $2628Kb$, o GC usou $688Kb$ para armazenar seus códigos e os 300 valores das séries históricas.

Neste experimento o GC conseguiu construir corretamente o valor dos atributos (*at*), informações contextuais (*ic*) e contextos ($E$); usou uma quantidade de recursos computacionais viável para a tarefa que executou e apresentou uma utilização fácil para que um desenvolvedor possa criar aplicações sensíveis a contextos.

## 5.4 Luminosidade

Este experimento visa calcular a *luminosidade* ambiente, usando uma câmera para capturar imagens, pois este é um dispositivo presente em diversos computadores, celulares, tablets, etc. Usando a imagem captada pela câmera é possível calcular a luminosidade do ambiente para que com estes dados o dispositivo possa ser reconfigurado, ajustando o brilho da tela e a luz de fundo do teclado, por exemplo.

Uma forma simples de calcular a luminosidade é transformar a imagem capturada pela câmera em imagens em níveis de cinza, e então somar o valor de todos os pixels, que normalmente valem de 0 (escuridão total) a 255 (claridade total). Assim é possível calcular a porcentagem de luz presente na imagem, sendo que se todos os pixels forem preto absoluto a imagem é 0% clara e se todos os pixels forem branco absoluto a imagem é 100% clara.

### Ambiente

Tendo estabelecido a forma de calcular a luminosidade, usando como entrada as imagens obtidos por uma câmera, esta imagens serão a entidade do mundo real (*er*) usada neste experimento. A câmera usada neste experimento possui uma resolução de $640x480$ pixels e é ligada por uma interface USB a um computador.

O computador onde este experimento foi executado executa como SO o Linux, possui um processador de 2 Ghz e 2GB de memória. Neste experimento foi usado o mesmo código do GC que foi usado no outro experimento, no MAC OS X, pois o python pode ser executado em ambas as plataformas.

## Realização

Durante a realização deste experimento o computador foi ligado com a câmera em um ambiente com luz artificial, durante o experimento a câmera foi tampada e outras vezes iluminada diretamente com outra fonte de luz para verificar o funcionamento do GC. A API usada para cadastrar, associar e ativar as entidades necessárias para gerar o contexto luminosidade pode ser vista a seguir:

**cadastrar** *("luminosidade", fc_luminosidade, (), "Verifica Luminosidade")*

**associar** *("luminosidade", "ambiente")*

**ativar** *("luminosidade", 20, 1800)*

**cadastrar** *("luminosidade", fa_luminosidade, (), "IC luminosidade")*

**associar** *("luminosidade", "luz")*

**ativar** *("luminosidade", 20, 1800)*

A primeira linha cadastra o atributo (*at*) luminosidade, junto com a função coletora (*fc*()) *fc_luminosidade*. A segunda linha cria a entidade (*E*) ambiente e a associa com o atributo luminosidade. Na terceira linha o atributo luminosidade é ativado. Na quarta linha é criada a informação contextual (*ic*) luminosidade, na quarta ela é associada com o contexto (*C*) luz e a última linha ativa a informação contextual luminosidade. Este processo pode ser visto no diagrama 5.2. Alguns dos dados obtidos pelo experimento podem ser vistos a seguir:

```
Contexto luminosidade:
    ((2012, 1, 20, 23, 40, 11, 724675), 0.57374514910130714),
    ((2012, 1, 20, 23, 40, 31, 725516), 0.5688853145424837),
    ((2012, 1, 20, 23, 40, 51, 726375), 0.57160437091503269),
    ((2012, 1, 20, 23, 41, 11, 727216), 0.56591733047385617),
    ((2012, 1, 20, 23, 41, 31, 728094), 0.58262357026143796),
    ((2012, 1, 20, 23, 41, 51, 728970), 0.57293964460784319),
    ((2012, 1, 20, 23, 42, 11, 729807), 0.59061606413398693),
    ((2012, 1, 20, 23, 42, 31, 730658), 0.566241574754902),
    ((2012, 1, 20, 23, 42, 51, 731617), 0.41568882761437909),
    ((2012, 1, 20, 23, 43, 11, 732562), 0.63413245506535942)
```

Como no experimento anterior, o primeiro conjunto de dados representa o instante de tempo onde o contexto foi construído e o segundo, o valor do contexto, que neste caso representa a quantidade de luminosidade ambiente.

### Análise

Para verificar a utilidade neste experimento é necessário retomar o processo usado para construir este contexto. Foi necessário utilizar uma biblioteca para acessar a câmera e obter as imagens, outra para processar as imagens, transformando-a em escala de cinza além de conhecimentos sobre o processamento digital de imagens.

Caso um desenvolvedor precise de um contexto sobre a luminosidade, seria necessário todo este processo para construi-lo, além de conhecimentos sobre threads e escalonamento para conseguir que estes dados fossem atualizados constantemente. Como foi descrito nesta seção, usando o GC proposto foi necessário usar apenas uma biblioteca e seis linhas de código para obter o contexto luminosidade.

Quanto à efetividade do GC no experimento, o contexto foi gerado corretamente e respondeu corretamente quando tinha a lente da câmera tampada, indicando uma luminosidade muito próxima de 0% e quando iluminada diretamente com outra fonte de luz a luminosidade indicada era próxima de 100%.

Quanto a eficiência computacional do GC neste experimento, o GC usou $95680KB$ de memória. Este aumento no consumo de recursos é esperado, visto que o consumo de recursos pelo GC depende do número de funções ($fc()$ e $fa()$) a serem executadas, da periodicidade destas funções e dos códigos internos das funções. Como processamento digital de imagens consome mais recursos computacionais, o GC também consumiu mais recursos.

Neste experimento o GC conseguiu gerar corretamente os valores esperados para o contexto luminosidade, porém ficou claro que o desempenho do GC e do SO que o executa é afetado diretamente pelas funções ($fc()$ e $fa()$) que o GC executa. Neste caso o GC pode ajudar a economizar recursos computacionais, caso dois processos precisem do contexto Luminosidade, o GC pode calcular este valor apenas uma vez, evitando assim retrabalho dentro do sistema.

## 5.5 Conclusão

Este capítulo realizou dois experimentos com o GC, no primeiro foi construído um contexto sobre conectividade, no segundo um contexto sobre a luminosidade ambiente. Ambos os experimentos foram analisados para verificar se o modelo e o protótipo possuem utilidade no auxilio ao desenvolvimento de aplicações sensíveis a contextos, são efetivos ao calcular contextos e como o GC consome recursos computacionais.

# Capítulo 6

# Considerações finais

A idéia básica da computação sensível a contextos é simples: um sistema deve ser capaz de detectar o contexto em que está inserido e com base neste contexto o sistema deve se adaptar para melhorar a interação com o usuário. Diversos dados podem ser usados por dispositivos para que o dispositivo possa entender o contexto em que está inserido, dados de GPS, redes disponíveis, câmeras, microfones, etc.

Porém, com esta heterogeneidade dos dados que podem ser usados para construir contextos, a construção de aplicativos sensíveis a contextos torna-se muito complexa, pois é necessário obter dados de baixo nível, oriundos do hardware, acessar diversas bibliotecas que não seguem um padrão, conhecer dados de diversas áreas da computação como processamento de som, imagem, sinais, etc.

O objetivo deste trabalho é propor um modelo de dados capaz de gerenciar contextos para viabilizar que desenvolvedores criem aplicativos sensíveis a contextos sem que seja necessário enfrentar as dificuldades mencionadas acima. Para cumprir este objetivo este trabalho apresentou as seguintes etapas:

- No inicio deste trabalho foi feita uma descrição do problema, da motivação e dos objetivos deste trabalho. Também foi apresentada uma revisão bibliográfica da computação sensível a contextos;
- Em seguida foi definido o que é contexto ($C$), atributo real ($er$), função coletora ($fc()$), atributo ($at$), entidade ($E$) função agregadora ($fa()$), informação contextual ($Ic$) e mecanismo de entrega ($me$).
- Com as definições anteriores foi proposto um modelo de dados capaz de construir e gerenciar contextos. Com este modelo foram realizados três estudos de caso que demonstram como o modelo de dados proposto pode ser usado para construir contextos;
- Foi proposto um gerente de contextos ($GC$) que usa o modelo de dados proposto para construir e gerenciar contextos. Com este GC foi apresentado um conjunto de contextos pré-definidos que podem ser usados como ponto de partida para desenvolvedores de aplicativos sensíveis a contextos. Também foi definida a API de acesso ao gerente de contextos;
- Foi apresentada a implementação de um protótipo que implementa o modelo de dados proposto e alguns estudos de caso que usam o GC para desenvolver aplicações sensíveis a contextos.

Posteriormente foram propostos e realizados experimentos para verificar se o protótipo implementado seria útil para desenvolvedores, se o mesmo executa as operações como proposto no modelo de dados e qual a eficiência do protótipo no uso de recursos computacionais para que este consiga gerenciar contextos. Os experimentos realizados estão apresentados a seguir:

- Criação de um contexto *conectividade*, que verifica se o dispositivo possui conectividade nula, cabeada, sem fio ou ambas (cabeada e sem fio). Neste experimento o GC conseguiu construir e gerenciar contextos conforme o esperado e consumindo uma quantidade aceitável de recursos computacionais.

- Criação de um contexto *luminosidade*, que verifica a luminosidade do ambiente onde o dispositivo está. Neste experimento os contextos também foram construídos e gerenciados de um forma satisfatória. Como esperado o consumo de recursos computacionais deste experimento foi superior ao experimento anterior, já que o processamento de imagens necessita de mais recursos.

Com a realização destes experimentos pode-se observar que desenvolver aplicações sensíveis a contextos é mais simples, mais organizado e menos trabalhoso usando o GC do que construir aplicações sensíveis a contextos sem usar o GC. Portanto o consumo de recursos computacionais pelo GC é aceitável, dada a utilidade do modelo para o desenvolvedor.

Vários exemplos de CAC em um SO podem ser encontradas neste trabalho e em diversos softwares como [Symonds, 2007, Rue, 2012, Oomph Inc, 2012], entre outros softwares que implementam a sensibilidade a contextos. Esses softwares visam coletar alguns contextos pré-determinados e adaptar o dispositivo baseando-se nesses dados

Outros softwares como o [Salber et al., 1999] oferecem um conjunto de contextos pré-definidos para desenvolvedores adicionarem sensibilidade a contextos, porém não facilitam o processo de criação de novos tipos de contextos nem a possibilidade de armazenar séries históricas de contexto automaticamente, para uso posterior, como o GC faz.

Como trabalhos futuros, diversos contextos podem ser propostos e implementados para tornar os sistemas computacionais mais adaptados às pessoas e não o contrário, além disso o GC pode ser expandido para que possa se comunicar com outros gerentes de contextos e assim tornar os GCs capazes de entender o contexto de outros dispositivos e se adaptar conforme estes dados.

Depois que um gerente de contextos está implementado, os contextos gerados podem ser aproveitados em outras trabalhos, para classificar usuários baseando-se nos contextos, ou então, os contextos podem ser usados para modelar o comportamento dos usuários e estes modelos podem ajudar a desenvolver SOs mais adequados aos usuários.

# Referências Bibliográficas


[Abowd et al., 1997] Abowd, G. D., Atkeson, C. G., Hong, J., Long, S., Kooper, R., and Pinkerton, M. (1997). Cyberguide: a mobile context-aware tour guide. *Wireless network*, 3(5):421–433.

[Abowd et al., 1999] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK. Springer-Verlag.

[Abowd and Mynatt, 2000] Abowd, G. D. and Mynatt, E. D. (2000). Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58.

[Apple Inc, 2012] Apple Inc (2012). Mac os lion. http://www.apple.com/.

[Beach et al., 2010] Beach, A., Gartrell, M., Xing, X., Han, R., Lv, Q., Mishra, S., and Seada, K. (2010). Fusing mobile, sensor, and social data to fully enable context-aware computing. In Dalton, A. and Want, R., editors, *HotMobile*, pages 60–65. ACM.

[Boari et al., 2006] Boari, M., Lodolo, E., Monti, S., and Pasini, S. (2006). Middleware for automatic dynamic reconfiguration of context-driven services. Washington, DC, USA. IEEE Computer Society.

[Chen and Kotz, 2000] Chen, G. and Kotz, D. (2000). A survey of context-aware mobile computing research. Technical report, Hanover, NH, USA.

[Cheverst et al., 2000a] Cheverst, K., Davies, N., Mitchell, K., and Friday, A. (2000a). Experiences of developing and deploying a context-aware tourist guide: the guide project. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 20–31, New York, NY, USA. ACM.

[Cheverst et al., 2000b] Cheverst, K., Davies, N., Mitchell, K., Friday, A., and Efstratiou, C. (2000b). Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24, New York, NY, USA. ACM.

[Coutaz et al., 2005] Coutaz, J., Crowley, J. L., Dobson, S., and Garlan, D. (2005). Context is key. *Communications of the ACM*, 48(3):49–53.

[Dey, 2001] Dey, A. K. (2001). Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7.

[Eagle and (Sandy) Pentland, 2006] Eagle, N. and (Sandy) Pentland, A. (2006). Reality mining: sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4).

[Faulkner and Gomes, 1991] Faulkner, R. and Gomes, R. (1991). The process file system and process model in UNIX System V. In *USENIX Conference Proceedings*.

[Google Inc, 2012] Google Inc (2012). Android. http://www.android.com/.

[Harter et al., 1999] Harter, A., Hopper, A., Steggles, P., Ward, A., and Webster, P. (1999). The anatomy of a context-aware application. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 59–68, New York, NY, USA. ACM.

[Hong et al., 2009] Hong, J., Suh, E., and Kim, S. (2009). Context-aware systems: A literature review and classification. *Expert Systems with Applications*, 36(4):8509 – 8522.

[Hong and Landay, 2001] Hong, J. I. and Landay, J. A. (2001). An infrastructure approach to context-aware computing. *ACM Transactions on Computer-Human Interaction*, 16(2):287–303.

[Killian, 1984] Killian, T. J. (1984). Processes as files. In *USENIX Software Tools Users Group Summer Conference*.

[Long et al., 1996] Long, S., Kooper, R., Abowd, G. D., and Atkeson, C. G. (1996). Rapid prototyping of mobile context-aware applications: the cyberguide case study. In *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 97–107, New York, NY, USA. ACM.

[Oomph Inc, 2012] Oomph Inc (2012). Sidekick. http://oomphalot.com/sidekick/.

[Pascoe, 1998] Pascoe, M. J. (1998). Adding generic contextual capabilities to wearable computers. In *ISWC '98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, page 92, Washington, DC, USA. IEEE Computer Society.

[Picco et al., 2005] Picco, G. P., Balzarotti, D., and Costa, P. (2005). Lights: a lightweight, customizable tuple space supporting context-aware applications. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 413–419, New York, NY, USA. ACM.

[python, 2012] python (2012). python. http://www.python.org/.

[Raento et al., 2005] Raento, M., Oulasvirta, A., Petit, R., and Toivonen, H. (2005). Contextphone: A prototyping platform for context-aware mobile applications. *IEEE Pervasive Computing*, 4(2):51–59.

[Rue, 2012] Rue, D. (2012). Controlplane. http://www.controlplaneapp.com/.

[Salber et al., 1999] Salber, D., Dey, A. K., and Abowd, G. D. (1999). The context toolkit: aiding the development of context-enabled applications. In *CHI 99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA. ACM.

[Schilit et al., 1994] Schilit, B., Adams, N., and Want, R. (1994). Context-aware computing applications. In *WMCSA '94: Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, pages 85–90, Washington, DC, USA. IEEE Computer Society.

[Schmidt et al., 1998] Schmidt, A., Beigl, M., and w. Gellersen, H. (1998). There is more to context than location. *Computers and Graphics*, 23:893–901.

[Symonds, 2007] Symonds, D. (2007). Marcopolo. http://www.symonds.id.au/marcopolo/.

[Torvalds, 1991] Torvalds, L. (1991). The linux kernel.

# Apêndice A

# Protótipo

## A.1 Introdução

Este capítulo visa apresentar o código fonte que implementa o protótipo deste trabalho. A figura a seguir foi apresentada no capítulo 4.4 e apresenta o diagrama UML simplificado deste protótipo. Nesta figura estão presentes as classes Gerente de contextos; Tabela; Conjunto de entidades; Conjunto de contextos; Funções coletoras; Funções agregadoras. Estas são as classes implementadas no protótipo.

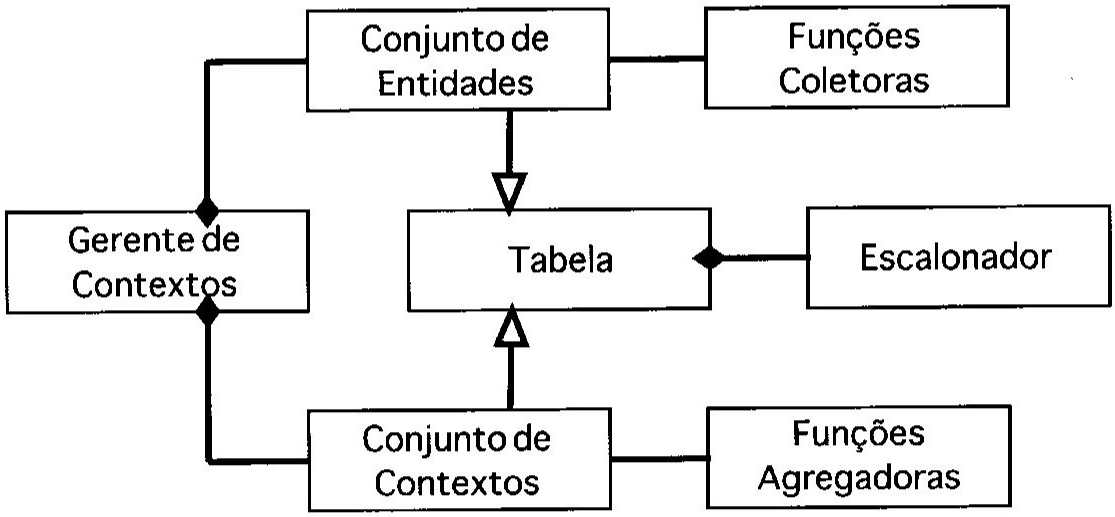



Figura A.1: Diagrama de classe do GC

Os arquivos estão organizados da seguinte forma: *gerente.py*: implementa a classe gerente de contextos; *tabela.py*: implementa as classes tabela, conjunto de entidades e conjunto de contextos; *dispatcher.py*: implementa o escalonador; *func_coletoras.py*: implementa as funções coletoras; *func_agregadoras.py*: implementa as funções agregadoras. A seguir é apresentado o código fonte do protótipo.

## A.2 Código Fonte

Listing A.1: Gerente de Contextos - gerente.py

```
from func_coletoras import *
from func_agregadoras import *
from tabela import *
import time

class Gerente (object):
        def __init__ (self):
                self.conectividade ()

        def conectividade (self):
                self.ce = conjunto_entidades ("")
                self.ce.cadastrar ("ethernet", fc_ethernet_conected, ('',), "Verifica a
                    ethernet")
                self.ce.cadastrar ("wireless", fc_wireless_conected, ('',), "Verifica a
                    wireless")
                self.ce.associar ("ethernet", "redes")
                self.ce.associar ("wireless", "redes")
                self.ce.ativar ("ethernet", 1, 100)
                self.ce.ativar ("wireless", 1, 100)

                self.cc = conjunto_contextos ()
                self.cc.cadastrar ("conectividade", fa_conectividade2, (self.ce, ""), "informacao
                     contextual conectividade")
                self.cc.associar ("conectividade", "conectividade")
                self.cc.ativar ("conectividade", 1, 100)

        def luminosidade (self):
                self.ce = conjunto_entidades ("")
                self.ce.cadastrar ("luminosidade", fc_luminosidade, ('',), "Verifica
                    Luminosidade")
                self.ce.associar ("luminosidade", "ambiente")
                self.ce.ativar ("luminosidade", 20, 1800)

                self.cc = conjunto_contextos ()
                self.cc.cadastrar ("luminosidade", fa_luminosidade, (self.ce, ""), "informacao
                    contextual luminosidade")
                self.cc.associar ("luminosidade", "luz")
                self.cc.ativar ("luminosidade", 20, 1800)

gc = Gerente ()
```

Listing A.2: Tabela, Conjunto de Entidades e Conjunto de Contextos - tabela.py

```
from func_agregadoras import *
from func_coletoras import *
from dispatcher import *

import datetime
import time

class Tabela (object):

    def __init__ (self):
        self.__dic__ = {}

    def cadastrar (self, id_item, func, args, desc = ""):

        item = (id_item, (), func, args, None, desc)
        self.__dic__[id_item] = item

    def associar (self, id_item, id_grupo):
        if not ( id_item in self.__dic__ ):
            return "Item nao cadastrado."

        novo_grupo = self.__dic__[id_item][1] + (id_grupo,)
        nova_tupla = ( self.__dic__[id_item ][0], novo_grupo, self.__dic__[id_item ][2], self.
            __dic__[id_item ][3], self.__dic__[id_item ][4], self.__dic__[id_item ][5] )
        self.__dic__[id_item] = nova_tupla

    def ativar (self, id_item, periodo, intervalo_total ):
        if not ( id_item in self.__dic__ ):
            return "Item nao cadastrado."

        item = self.__dic__[id_item]
        disp = Dispatcher (item[2], periodo, intervalo_total , item[3])
        nova_tupla = (item[0], item[1], item[2], item[3], item[4], disp )
        self.__dic__[id_item] = nova_tupla

        disp.run_and_dispatch ()

    def receber_valor (self, id_item, intervalo_i = float('inf'), intervalo_f = float('inf')):
        if not id_item in self.__dic__:
            return "Item nao encontrado"

        item = self.__dic__[id_item]
        disp = item[5]
        valor = disp.get_valor ( intervalo_i , intervalo_f )

        return valor

    def __get_estado__ (self, id_grupo, intervalo_i = float('inf'), intervalo_f = float('inf'))
        :
        valores = []

        for chave in self.__dic__:
            item = self.__dic__[chave]
```

```
                        if id_grupo in item[1]:
                            disp = item[5]
                            valores.append(disp.get_valor ( intervalo_i , intervalo_f ))

                if len(valores) == 1:
                    return valores[0]

                else:
                    return tuple(valores)

class conjunto_entidades (Tabela):

    def get_contexto (self, id_grupo, intervalo_i , intervalo_f ):
        return "Operacao nao permitida"

class conjunto_contextos (Tabela):

    def get_contexto (self, id_grupo, intervalo_i = None, intervalo_f = float('inf')):
        return super(conjunto_contextos, self).__get_estado__ (id_grupo, intervalo_i ,
            intervalo_f )
```

Listing A.3: Escalonador - dispatcher.py

```
import time
import datetime
from threading import Timer

class Dispatcher:
    def __init__ (self, func, time, full, args):
        self.__func__ = func
        self.__time__ = time
        self.__full__ = full
        self.__args__ = args

        self.__valor__ = []

    def run_and_dispatch (self):
        try:
            tupla = ( datetime.datetime.now(), self.__func__ (* self.__args__) )

            if len( self.__valor__) >= ( self.__full__ / self.__time__):
                self.__valor__.pop(0

            self.__valor__.append( tupla )
        except TypeError, erro:
            print "Um erro ocorreu ao executar a funcao: %s" % erro

        try:
            Timer ( self.__time__, self.run_and_dispatch, () ). start ()
        except TypeError, erro:
            print "Um erro ocorreu ao agendar a execucao da funcao: %s"
                % erro

    def get_valor (self, start = None, end = None):
        result = []

        if ( start == None) and (end == None):
            return self.__valor__[0]
        if ( start == None) and (end == float ('inf')):
            return self.__valor__
        if ( start == float ('inf')) and (end == float ('inf')):
            return self.__valor__[-1]
        for i in self.__valor__:
            if start <= i[0] <= end:
                result.append(i)

        return ( result )
```

Listing A.4: Funções Coletoras - func_coletoras.py

```
import os
import Image
import cv

def fc_luminosidade ():
        camcapture = cv.CreateCameraCapture(0)

        cv.SetCaptureProperty(camcapture,cv.CV_CAP_PROP_FRAME_WIDTH, 640)
        cv.SetCaptureProperty(camcapture,cv.CV_CAP_PROP_FRAME_HEIGHT, 480)

        if not camcapture:
            sys.exit(1)

        frame = cv.QueryFrame(camcapture)
        gray = cv.CreateImage ((640, 480), cv.IPL_DEPTH_8U, 1)
        cv.CvtColor(frame, gray, cv.CV_BGR2GRAY)

        total = 0

        for x in range(480/10):
            for y in range(640/10):
                total = total + int(gray[(x*10),(y*10)])

        percent = total / float ((480/10)*(640/10)*255)

        return percent

def fc_ethernet_conected ():
        saida = os.popen("ifconfig en0")
        for linha in saida:
                if "inactive" in linha:
                        return "inactive"
        return "active"


def fc_wireless_conected ():
        saida = os.popen("ifconfig en1")
        for linha in saida:
                if "inactive" in linha:
                        return "inactive"
        return "active"
```

Listing A.5: Funções Agregadoras - func_agregadoras.py

```
def fa_luminosidade (ce):

    lz = ce.receber_valor ("luminosidade", float('inf'), float('inf'))

    return lz[-1]


def fa_conectividade (ce):
    et = ce.receber_valor ("ethernet", float('inf'), float('inf'))

    wi = ce.receber_valor ("wireless", float('inf'), float('inf'))

    if et[1] == "inactive" and wi[1] == "inactive":
        return "Nula"
    elif et[1] == "inactive":
        return "Sem Fio"
    elif wi[1] == "inactive":
        return "Cabeada"
    else:
        return "Total"
```