UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

GUILHERME ANTONIO BORGES

Um Mecanismo Abstrato de Autoadaptação para Sistemas de Sensoriamento Urbano

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Cláudio Fernando Resin Geyer

Porto Alegre
2016

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Borges, Guilherme Antonio

Um Mecanismo de Autoadaptação Abstrato para Sistemas de Sensoriamento Urbano / Guilherme Antonio Borges. – 2016.

109 f.:il.

Orientador: Cláudio Fernando Resin Geyer.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2016.

1.Autoadaptação. 2.Computação Autonômica. 3.Sensoriamento Urbano. 4.Cidades Inteligentes. 4.Computação Ubíqua. I. Geyer, Cláudio Fernando Resin. II. Um Mecanismo de Autoadaptação para Sistemas de Sensoriamento Urbano.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Carlos e Verone, e ao meu irmão Lucas, por terem sido meu suporte permanente em todo o trajeto da minha vida até o presente momento, sendo o meu porto seguro em todos os momentos bons como ruins. Agradeço a minha namorada Michele, por ter me ajudado enormemente no último ano tanto como uma excelente companhia como pelo amor e carinho. Amo vocês.

Aos meus queridos e amados amigos Vinícius P. Lima e Ana C. Secco, pela amizade e carinho de longa data, trazendo diversos bons momentos, além de estarem presentes nos momentos mais difíceis da dissertação. Agradeço também a meus irmãos de escotismo, em especial Jean, Gleisson, Alex, Mateus, Fábio, Jansline, Letícia e Paulo que, mesmo a distância, que continuam a ser bons amigos. Sem o apoio de vocês não teria sido possível terminar este trabalho

Agradeço aos membros do grupo de pesquisa de processamento paralelo e distribuído (GPPD) da UFRGS, em especial aos membros Anubis, Rolim, Valderi, Julio, Bharbara e Hélio, que mostraram que o trabalho em equipe contribui em muito a construção do saber e da amizade. Também, ao orientador Cláudio F.R. Geyer, por ter guiado o caminho para a realização deste trabalho.

Agradeço com um carinho especial a todos os moradores e antigos moradores do apartamento 06, Rômulo, Júnior, Fábio, Silvana, Leonardo, que no decorrer destes 3 anos souberam ser pacientes e amigos em diversos momentos. Por fim, agradeço ao convênio FAPERGS/CAPES pelo financiamento de um ano de bolsa de pesquisa e ao CPD da UFRGS por mais um ano de bolsa, bem como as diversas pessoas que direta e indiretamente contribuíram para a realização deste trabalho.

RESUMO

Sensoriamento urbano e cidades inteligentes têm sido tópicos derivados da computação ubíqua em alta nos últimos anos, tanto para a academia como para a indústria, devido ao contínuo avanço tecnológico aliado à maior facilidade de acesso e aceitação pelos usuários. Na literatura pesquisada sobre plataformas que englobam tais tópicos foi constatado que diversas delas possuem algum processo autonômico utilizado para atender alguma necessidade de autoadaptação em tempo de execução. Apesar disso, nenhuma das plataformas pesquisadas focou especificamente em encontrar e propor uma solução para tratar exclusivamente a autoadaptação. Nesse contexto, esta dissertação tem por objetivo propor um mecanismo de autoadaptação para sistemas de sensoriamento urbano, além de avaliar seu comportamento. Como primeiro passo para realizar tal objetivo, foi conduzida uma pesquisa literária tendo em vistas identificar os principais casos de adaptação em sistemas de sensoriamento urbano, além de requisitos específicos da arquitetura de sensoriamento urbano UrboSenti, utilizada para implementação. Como segundo passo, a partir dos requisitos identificados, o modelo MAPE-K da computação autonômica foi escolhido como a base da construção do mecanismo de autoadaptação. A implementação deste modelo utilizou as técnicas de eventos passivos para monitoramento do ambiente, regras *Evento-Condição-Ação*, para tomada de decisão, planos estáticos para planejamento e adaptações por parâmetros e componentes para execução. Tanto o modelo como as técnicas escolhidas foram implementadas devido atenderem as necessidades dos cenários avaliados. Por fim, as avaliações aplicadas apontam resultados preliminares satisfatórios, dados os casos avaliados e os experimentos de tempo de resposta a eventos internos e interações; no entanto, tais avaliações revelarem diversos pontos que devem ser explorados em trabalhos futuros.

Palavras-chave: Autoadaptação. Computação Autonômica. Sensoriamento Urbano. Cidades Inteligentes. Computação Ubíqua.

ABSTRACT

In the last years, urban sensing and smart cities have been popular topics derived from the ubiquitous computing, for both the academia and the industry, due to its continuous technological development combined with greater facilities of access and acceptance by the users. The reviewed literature about platforms that encompass such topics showed that many of them have some kind of autonomic process used to meet any need for self-adaptation at runtime. Despite this, none of the researched platforms focused in proposing a solution to exclusively meet the self-adaptation properties. In this way, this dissertation aims to propose a self-adaptive mechanism to urban sensing systems, as well as evaluating its behavior. As the first step to achieving such goal, a literature review was performed aiming to identify the main adaptation cases in urban sensing systems, as well to identify the specific requirements of the UrboSenti architecture for urban sensing. As the second step, the autonomic computing MAPE-K model was chosen to compose the foundation of the self-adaptive mechanism based on the identified requirements. The implementation of this model used the techniques of passive events for monitoring, rules Event-Condition-Action for decision making, static plans for planning and parameter and component adaptations for execution were used in the proposed implementation to meet the evaluated scenario needs. Lastly, the applied evaluations indicate satisfactory results, given the assessed cases and the experiments of scalability at the response of internal events and interactions. However, they have left many open points that should be explored in future works.

Keywords: Self-adaptation. Autonomic Computing. Urban Sensing. Smart Cities. Ubiquitous Computing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

2G	Segunda Geração
3G	Terceira Geração
4G	Quarta Geração
ADB	Android Debug Bridge
AVD	Android Virtual Devices
BAA	Broad Agency Announcement
CPU	Unidade Central de Processamento
DARPA	Defense Advanced Research Projects Agency
DTN	Delay Tolerant Network
DVM	Dalvik Virtual Machine
ECA	Evento, Condição e Ação
EDGE	Explicit Data Graph Execution
GB	Gigabyte
GC	Garbage Collector
GCM	Google Cloud Messaging
GHz	Gigahertz
GPPD	Grupo de Processamento Paralelo e Distribuído
GPRS	Serviço de Rádio de Pacote Geral
GPS	Global Positioning System
HD	Hard Disk
HTTP	Hypertext Transfer Protocol
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IO	Input/Output
Java SE	Java Standard Edition
JVM	Java Virtual Machine
MANET	Mobile Ad Hoc Network
MAPE	Monitor, Analyse, Plan and Execute
MAPE-K	Monitor, Analyse, Plan, Execute and Knowledge
MER	Modelo Entidade Relacionamento
PC	Computador Pessoal
QoS	Qualidade de Serviço
RAM	Random Access Memory

SGBD	Sistema de Gerenciamento de Banco de Dados
SO	Sistema Operacional
SOA	Services Oriented Architecture
SPL	Software Product Line
SQL	Structured Query Language
SSID	Service Set Identifier
TCP	Transmission Control Protocol
UbiArch	Ubiquitous Architecture for Context Management and Application Development
UDP	User Datagram Protocol
UFRGS	Universidade Federal do Rio Grande do Sul
UML	Linguagem de Modelagem Unificada
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Mark Weiser, em 1991, conceituou a computação ubíqua como um paradigma computacional futurista onde as tecnologias se tornariam parte constante da vida cotidiana, tornando-se indivisível a partir dela – efeito por ele considerado a terceira era computacional. Neste paradigma, a tecnologia é centrada no usuário, buscando assim atender as suas necessidades e preferências em qualquer hora e local de forma transparente. Entretanto, os cientistas e engenheiros ainda terão muito trabalho a fazer até seu total potencial ser alcançado de forma factível e acessível a todos.

Todavia, apesar deste longo caminho existir, a tecnologia atual já propicia que a computação ubíqua entre na vida das pessoas por intermédio de telefone celulares, tablets, relógios e televisões inteligentes, entre outros dispositivos cotidianos, facilitando assim a interação com usuários e a coleta de informações. Tal mudança contextual é, conforme Cuff, Hansen e Kang (2008), o principal motivador do surgimento do "Sensoriamento Urbano".

O Sensoriamento Urbano é uma recente área que busca coletar dados oriundos do ambiente urbano em larga escala através do uso de dispositivos fixos e móveis distribuídos pela cidade (CAMPBELL et al., 2006). Estas informações coletadas podem ser utilizadas em diversos tipos de domínios de aplicações, tais como *healthcare*, redes sociais, segurança, monitoramento do ambiente e transporte (WU et al., 2013).

Através da coleta e análise de dados é possível retornar benefícios para o usuário, que podem ser desde em escala coletiva, tal como o melhoramento do planejamento urbano de sua cidade, como em escala individual ao recomendar o aumento de atividades físicas definidas por um especialista do serviço utilizado, ou mesmo a remuneração pelas informações compartilhadas, como é o caso da aplicação Google Opinion Rewards[1]. Tal efeito é denominado por autores como Campbell et al. (2006), Abdelzaher et al. (2007) e Ra et al. (2012) como o humano no ciclo ou, em inglês, *human-in-the-loop*.

Apesar do ser humano exercer esse importante papel na área de sensoriamento urbano sua participação deve ser alocada para tarefas onde a automatização não é possível, como o próprio ato de compartilhar uma informação participativa, ou mesmo para especificação de políticas de alto nível, os quais podem ser utilizados por sistemas autônomos. Segundo Sanchez et al. (2014), esta alocação de tarefa é necessária, pois a intervenção do humano, sozinha, não é o suficiente para prover resposta a todos os eventos e tratamentos a falhas

[1] https://play.google.com/store/apps/details?id=com.google.android.apps.paidtasks

devido à escala e a variedade de eventos a serem gerenciados em escala urbana. Desta forma, certo grau de automatização é necessário.

Tendo em vista esta observação, diversos trabalhos de sensoriamento urbano, apresentados na Seção 3.2, optaram por adotar alguns processos de adaptação para otimizar, tratar erros, controlar o acesso e reconfiguração de forma autônoma, isto é, que uma vez em funcionamento não necessitem da intervenção do usuário ou da aplicação para operar. Apesar disso, estes trabalhos propõem formas de tratamento, em geral, diretamente nos componentes de negócio, ou considerando poucos contextos de outros componentes. Tais questões, de acordo com Krupitzer et al. (2015), dificultam a separação de interesses e o gerenciamento através de políticas de alto nível visando um comportamento unificado para o sistema autoadaptativo. Além disso, essa dificuldade é agravada à medida que a complexidade e escala do sistema aumenta.

Como solução para esse problema, o modelo de sistema autoadaptativo proposto pela IBM em 2001, a Computação Autonômica, pode ser utilizado (Ganek e Corbi, 2003). Este modelo propõe que gerentes autonômicos gerenciem o sistema monitorando e realizando adaptações em artefatos gerenciáveis, os quais podem ser representações para as funções, componentes ou mesmo dispositivos de sensoriamento que podem ser autogerenciados em tempo de execução. O principal objetivo desta separação, segundo Kephart e Chess (2003), é justamente conferir ao sistema autonômico a capacidade de tomada de decisão para agir sobre o nível inferior do sistema, minimizando a intervenção do usuário nestas tarefas.

No âmbito desta área, o presente trabalho busca propor um mecanismo de autoadaptação com base no modelo da computação autonômica para prover comportamento autoadaptativo em sistemas de sensoriamento urbano. Como principal questão de pesquisa, quais funções e eventos encontrados nos dispositivos de sensoriamento podem ser tratados por um mecanismo autoadaptativo? Por segundo, qual o impacto dessa modificação no uso de recursos feito pela aplicação de sensoriamento em um dispositivo?

Em ordem a responder tais questões, o mecanismo proposto seria melhor aplicado em uma arquitetura de sensoriamento urbano modular, aberta e que possibilite uma visão geral de todos os elementos gerenciados, além de proporcionar interfaces para alterá-los. Com base nessas definições, a arquitetura UrboSenti, proposta por Rolim et al. (2014), foi selecionada para ser a base da implementação. Como diferencial da literatura abordada na Seção 3.2, a UrboSenti é construída seguindo a concepção de arquiteturas orientadas a serviço (SOA), além de possibilitar uma visão mais modular dos componentes internos ao adicionar o

diferencial de considerar que componentes sob demanda/opcionais existam para que diferentes requisitos dos dispositivos e de suas aplicações de sensoriamento.

Além disso, entre os componentes considerados pela UrboSenti um barramento de eventos central é proporcionado para receber e encaminhar eventos para tratadores por meio do método *Publish/Subscribe*, proporcionando uma visão unificada do que ocorre em todo o dispositivo. Entre os componentes sob demanda da UrboSenti, o componente chamado de adaptação foi proposto conceitualmente com o objetivo realizar as ações de gerenciamento e adaptação de forma autonômica dos componentes gerenciáveis utilizando o barramento de eventos para tanto. Apesar de considerar tal componente, a arquitetura o representa somente superficialmente e conceitualmente, não especificando como os eventos podem tratados e as adaptações aplicadas no ambiente de sensoriamento urbano.

Por fim, o mecanismo autoadaptativo proposto nesta dissertação é aplicado no componente de adaptação da UrboSenti, proporcionando as especificações preliminares necessárias para tratar os eventos de forma autonômica em ambientes de sensoriamento urbano. Ainda neste capítulo o contexto em que este trabalho está inserido, a explicitação dos objetivos visados e a descrição da estrutura do trabalho serão apresentadas.

1.1 Contexto

O projeto UbiArch (*Ubiquitous Architecture for Context Management and Application Development*), desenvolvido no Grupo de Processamento Paralelo e Distribuído (GPPD) nas dependências da UFRGS, tem como foco de pesquisa a concepção de arquiteturas de software e de aplicações que sejam capazes de se adaptar ao contexto no qual estão inseridas.

Uma das contribuições buscada por este projeto toma por base o cenário de cidades inteligentes, que também engloba o conceito de sensoriamento urbano, no qual surgem desafios complexos, relacionados principalmente à necessidade de proporcionar não somente novos tipos de serviços que busquem auxiliar na organização da cidade e no bem-estar de seus habitantes, mas também de promover alternativas sustentáveis, que diminuam o consumo de recursos energéticos naturais, através do uso de energias renováveis, e que reduzam a emissão de gases nocivos na atmosfera. De acordo com Schaffers et al (2011) o conceito de Cidades Inteligentes (Smart Cities) é uma resposta para tais desafios.

Nesse contexto, foi proposta a arquitetura UrboSenti (ROLIM et al., 2014), a qual difere das demais soluções existentes por prover um modelo conceitual orientado a serviços, na integração de sensores tradicionais com redes sociais e na disponibilização de serviços para

auxiliar no desenvolvimento de novas aplicações de sensoriamento. Internamente em seu módulo de sensoriamento o componente chamado de adaptação é o objeto de estudo desta dissertação.

1.2 Objetivos

1.2.1 Geral

Projetar um mecanismo de autoadaptação para plataformas de sensoriamento urbano utilizando a perspectiva da computação autonômica adicionando-o ao componente de adaptação da UrboSenti.

1.2.2 Específicos

- Identificar os principais casos de autoadaptação que ocorrem em sistemas de sensoriamento urbano;
- Propor um modelo conceitual para a arquitetura que atenda tais casos de autoadaptação;
- Implementar o protótipo com suporte a múltiplas plataformas;
- Aplicar experimentos para avaliar a capacidade do mecanismo de atender a demanda.

1.3 Estrutura do Texto

O texto apresentado nesta dissertação é dividido da seguinte forma:

- Capítulo 2: apresenta o levantamento bibliográfico necessário para compreensão dos elementos utilizados no decorrer do trabalho;
- Capítulo 3: apresenta uma visão geral da arquitetura UrboSenti, bem como os trabalhos relacionados de sensoriamento urbano e sua análise comparada segundo a visão da computação autonômica;
- Capítulo 4: apresenta a proposta do mecanismo de autoadaptação com base nos requisitos elencados pelos trabalhos de sensoriamento urbano apresentados, pela arquitetura UrboSenti e pela especificação da computação autonômica;

- Capítulo 5: especifica os elementos utilizados na implementação da UrboSenti, descreve os casos de estudo de adaptação bem como os resultados dos experimentos realizados na arquitetura e no mecanismo de adaptação;
- Capítulo 6: apresenta os trabalhos relacionados sobre sistemas e frameworks autoadaptativos, comparando-os com o mecanismo proposto;
- Capítulo 7: conclui o trabalho e elenca os trabalhos futuros.

2 REFERENCIAL CONCEITUAL

Neste capítulo são definidos e delimitados os paradigmas e características dos temas abordados. Para tanto, foram realizadas consultas nas principais literaturas da área, compondo assim o conhecimento inicial e necessário para este trabalho que está relacionado à computação ubíqua, computação autonômica e sensoriamento urbano.

2.1 Computação Ubíqua

A computação ubíqua, também conhecida como computação pervasiva, foi conceituada por Mark Weiser em 1991, o qual propaga a ideia de que uma pessoa terá muitos computadores compostos por múltiplas formas e funções a fim de se adequar a diferentes tarefas. Esses múltiplos computadores por pessoa estarão em itens cotidianos, como em livros, roupas, carros e vidros, e irão interagir com o usuário, o qual não distinguirá se essa interação é ou não com um computador. Essa noção psicológica é relativa à segunda mudança que Weiser (1991, p. 1, tradução nossa) previu, onde os computadores irão desaparecer em um efeito descrito em suas palavras: "*tecem-se no tecido da vida cotidiana até que serão indistinguíveis a partir dela*".

Apesar de tal funcionalidade da computação ubíqua ainda ser distante da realidade atual, ela tem inspirado diversas iniciativas em diferentes áreas da computação a contribuir passo-a-passo para alcançar sua concretização. Exemplo disso pode ser encontrado no estudo de Salim e Haque (2015), o qual mostram que o uso de *smartphones*, *displays* públicos, sistemas físicos cibernéticos (que incluem sistemas embarcados, sensores e atuadores) e da internet das coisas (IoT) tem cada vez mais contribuído para a aproximação do usuário com a tecnologia. Essa participação cada vez mais integrada entre os computadores e os seus usuários tem proporcionado que áreas emergentes como o Sensoriamento Urbano – melhor explorado na Seção 2.5 - ganhem mais espaço para pesquisa e desenvolvimento tecnológico.

Nesse contexto, a computação orientada a contexto exerce o importante papel de adquirir as informações necessárias para que aplicações sensíveis ao contexto possam se adaptar às mudanças contextuais (BETTINI et al, 2010). Na computação ubíqua, a orientação a contextos – explorada na Seção 2.6 - exerce um papel extremamente importante, pois ele dá o sentido necessário para que a informação seja interpretada pelos dispositivos de um ambiente de forma imperceptível para o usuário. Por exemplo, as preferências de um usuário para trabalhar em seu escritório envolvem uma temperatura de 20º, úmido, bastante iluminado

e ao som ambiente de uma cachoeira; com base nessas informações contextuais, assim que o usuário for identificado que está chegando ao escritório, em uma situação contextual de trabalho, os dispositivos presentes no escritório procurariam aplicar as condições ideais com base nas preferências desse usuário. Se este usuário mudar de local de trabalho, o sistema poderia consultar se quer manter as mesmas preferências, as quais seriam aplicadas no próximo ambiente da mesma forma. Tais informações de forma pura, isto é, sem relação a contextos dificultariam consideravelmente a construção de um sistema ubíquo.

Além dos contextos relacionados com o usuário, contextos sobre os dispositivos e o sistema também podem ser utilizados para melhor uso dos recursos e funcionalidades. Exemplo disso seriam os estados internos de um dispositivo sobre o contexto de execução, a conectividade, o nível de bateria, entre outros, que, de acordo com Mizouni et al (2014), podem ser convenientemente utilizadas por um sistema autoadaptativo para fazer alterações tendo em vistas configurações mais eficientes dos recursos utilizados sem a necessidade da intervenção do usuário. A forma como esses sistemas realizam tais tarefas e como são definidos é abordada em maiores detalhes nas Seções 2.2, 2.3 e 2.4.

2.2 Computação Autonômica

Diversos trabalhos, motivados por fins tanto acadêmicos quanto industriais, buscaram tornar um software autoadaptativo atribuindo-o a capacidade adaptar-se, com o mínimo de intervenção humana seja em resposta a mudanças internas, como por exemplo, identificar se o estado atual de execução está consistente, como em resposta a mudanças ambientais, assim como detectar se um provedor entre vários está atendendo a sua demanda.

Existem diversas definições para software autoadaptativo, sendo uma das primeiras, talvez uma das mais utilizadas, provida pela agência DARPA (Defense Advanced Research Projects Agency) Broad Agency Announcement (BAA) por Laddaga, Robertson e Shrore (1997): software autoadaptativos avaliam seus comportamentos conhecidos e trocam de comportamento quando notam que não conseguirão realizar o que haviam pretendido fazer, ou quando percebem que é possível realizar a melhor funcionalidade ou obter a melhor performance possível.

Contudo, com o passar dos anos, diversas outras inspirações para a construção de sistemas autoadaptativos têm instigado autores a conceituarem outros nomes, mantendo uma definição similar. Um dos mais significativos nomes associados foi conceituado pela IBM em 2001, através do manifesto de Horn, o qual sugere o conceito de computação autonômica,

comparando o autogerenciamento com o sistema nervoso humano, que executa tarefas biológicas de forma inconsciente (HUEBSCHER; MCCANN, 2008 apud HORN, 2001).

A ideia de Horn baseia-se em que, a partir da incorporação de mecanismos regulatórios do corpo, os quais não necessitam de pensamento consciente, é possível sugerir a construção de mecanismos que também permitam a um sistema de computador tornar-se autogerenciável (DOBSON et al., 2010). Uma das grandes motivações para a construção deste tipo de sistemas consiste na busca para eliminar ou ao menos diminuir a necessidade de intervenção humana perante o software, tendo como consequência a redução de custos.

Muitas vezes é possível encontrar termos diferentes relacionados com a computação autonômica, tal como o termo sistemas autogerenciados (*self-management*), o qual Kramer e Magee (2007), além de Sterritt (2005) afirmam ser claramente um sistema autonômico. Autoadaptação (*self-adaptation*) ou software autoadaptativos são outros termos muito utilizados que, de acordo com HUEBSCHER e MCCANN (2008), possuem um domínio mais limitado que a computação autonômica, estando incluídos nela. Apesar de nomes diferentes, os conceitos destes domínios são extremamente relacionados e, em muitos casos, eles podem ser usados como sinônimo (SALEIHIE; TAHVILDARI, 2009). O caso das propriedades *self-** e os ciclos de adaptação são os elementos mais comuns abordados por esses tipos de trabalhos, ambos apresentados nas próximas seções.

2.3 Propriedades Self-*

As propriedades *self-** representam as propriedades que delimitam os objetivos de adaptação aplicados em um sistema que o tornam um sistema autoadaptativo. De acordo com Kephart e Chess (2003), além de Saleihie e Tahvildari (2009), essas propriedades foram definidas em correspondência ao mecanismo biológico de autoadaptação, sendo elas:

- *Autoconfiguração (self-configuring)*: Confere a capacidade de se reconfigurar automaticamente e dinamicamente em respostas às mudanças, instalando, atualizando e integrando, além de compor e decompor entidades de software. Por exemplo: em tempo de execução, o sistema faria suporte dinâmico para adicionar ou remover servidores em uma infraestrutura sem intervenção humana. Similarmente a uma nova célula a ser integrada em um corpo, um novo elemento autonômico deve ser capaz de se integrar à infraestrutura do sistema, e o sistema existente deve ser capaz de adaptar-se ao novo elemento;

- *Autocura (self-healing)*: Refere-se à capacidade autoadaptativa de detectar, diagnosticar, evitar e reparar problemas resultantes de bugs, defeitos ou falhas no software e hardware. Métodos de recuperação podem incluir o uso de recursos para encontrar alternativas, download de atualizações de software, substituição de componentes de hardware que falharam, reiniciando os elementos que falharam ou simplesmente enviando uma notificação a um administrador humano. Similarmente à forma como o cérebro usa áreas não danificadas para reimplementar funções perdidas, sistemas autonômicos podem integrar dinamicamente componentes redundantes ou subutilizados para substituir peças defeituosas, maximizando sua disponibilidade;
- *Auto-otimização (self-optimising)*: Software autoadaptativos com essa capacidade buscam continuamente formas de melhorar seu funcionamento, e se tornarem mais eficientes no custo, desempenho ou qualidade de serviço (QoS) para satisfazer os objetivos pré-definidos, que podem ser além de centrados no software, também centrados no usuário. Por exemplo, adaptar-se para consumir o mínimo possível de bateria de um smartphone e encontrar as melhores rotas para diminuir a latência das mensagens é uma tarefa de auto-otimização. Assim como os músculos de um organismo tornam-se mais eficientes com exercício, um sistema autonômico deve ser capaz de otimizar a si mesmo ao longo do tempo, aprendendo e melhorando as várias combinações de valores para os parâmetros internos;
- *Autoproteção (self-protecting)*: Indica que o software vai buscar se proteger de ataques maliciosos e de falhas em cascata inadvertidas. Esta propriedade busca antecipar os problemas e tomar medidas para evitá-los ou minimizar seus efeitos, bem como sistemas de defesa contra diversos ataques. Ações de autoproteção podem incluir tornar recursos off-line quando há detecção de intrusos, aumentando as verificações de segurança ao suspeitar de potenciais ameaças, ou, mais comumente, pode alertar os administradores do sistema. Semelhante a um sistema imunológico biológico, os sistemas autonômicos podem fazer uso de um "sistema imunológico digital" para protegê-los, enquanto exigem um mínimo ou zero intervenção ou ciência do usuário.

Apesar das quatro propriedades *self-** abrangerem a maioria dos possíveis usos em um sistema autoadaptativo, desde que a computação autonômica foi conceituada em 2001, diversos autores propuseram outras propriedades de forma mais específica ou mesmo equivalentes. Segundo Lalanda, McCann e Diaconescu (2013) a maioria dessas propriedades

estendidas são necessárias para alcançar os recursos de auto-chop (*self-configuring*, *self-healing*, s*elf-optimising* e *self-protecting*) fundamentais e podem, portanto, incluir elas. Podemos citar, por exemplo, os casos encontrados nos artigos utilizados para comparação:

- *Self-adaptation (autoadaptação)*: Indica que o software possui a habilidade de se modificar em reação as mudanças de contexto internas ou externas (FLOCH et al., 2006; NALLUR e HAHSOON, 2012). De acordo com Saleihie e Tahvildari (2009), se um software possui alguma propriedade *self-** qualquer ele já pode ser considerado sendo *self-adaptation*, em suma, ele é o termo mais genérico;
- *Self-adjusting*: Sistema é capaz de se modificar enquanto em execução, incluindo modificações da estrutura interna, configuração ou comportamento (AYALA, AMOR e FUENTES, 2012). Esta relacionada, pela definição dada, mais especificamente com *self-optimising* e *self-configuring;*
- *Self-management*: Indica que o software possui a habilidade de *self-adaptation* ou que é autonômico (AYALA, AMOR e FUENTES, 2012);
- *Self-reconfigure* ou *self-reconfigurable*: O software se autoreconfigura em resposta a mudanças nos requisitos e das condições ambientais (DALPIAZ, GIORGINI e MYLOPOULOS, 2013). Esta estando inclusa na propriedade *self-configuring;*
- *Self-recovering*: Indica que o sistema possui a habilidade de recuperar-se parcial ou completamente de falhas (LALANDA, MCCANN; DIACONESCU, 2013). Esta propriedade está incluída, pela definição conceitual, em *self-healing;*
- *Self-chop*: Refere-se às propriedades *self-** fundamentais: *self-configuration*, *self-healing*, *self-optimisation* e *self-protection* (WEYNS, MALEK e ANDERSSON, 2012; LALANDA, MCCANN e DIACONESCU, 2013).

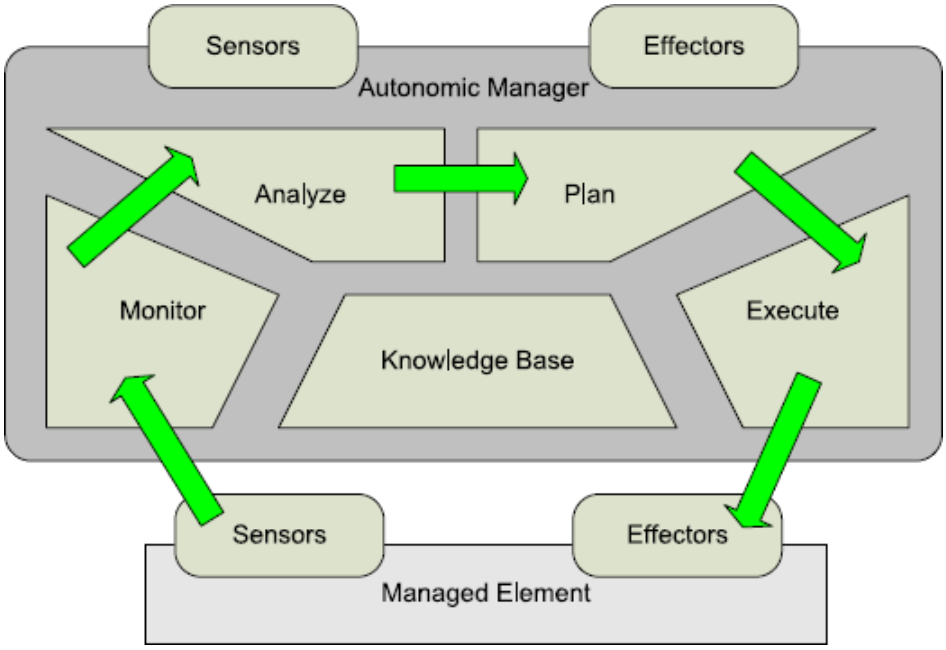
2.4 Loop de Adaptação

O loop de adaptação, ou loop de controle, é utilizado em muitos trabalhos sobre sistemas autoadaptativos, sendo inspirado em modelos de loop fechado da teoria do controle, o que possibilita encontrar trabalhos com propostas para controle em formato parecido, embora menos abrangentes, em anos anteriores do surgimento do conceito da computação autonômica e dos sistemas autoadaptativos. Um exemplo disto é proposto por Schöner e Dose (1992), os quais utilizam um esquema de loop fechado dotado de três etapas: sensoriamento; planejamento; e atuação; utilizadas para controlar um veículo autônomo.

A maioria dos trabalhos sobre autoadaptação segue o padrão comum de loop fechado: Monitorar (*Monitor*), Analisar (*Analyse*), Planejar (*Plan*) e Executar (*Execute*), ou MAPE, conectados por um *feedback* (NALLUR e BAHSOON, 2012). Este também é conhecido por MAPE-K, onde o K é o conhecimento obtido durante a execução. O padrão MAPE foi introduzido pela IBM e pode ser ilustrado pela Figura 2.1 que demonstra os componentes e o processo, contemplando desde o sensoriamento até a efetivação da adaptação. A função de cada um dos componentes, segundo IBM (2005), além de Saleihie e Tahvildari (2009), é descrita a seguir:

- *Sensors* (Sensores): São entidades de software que geram uma coleção de dados refletindo o estado do sistema. Por exemplo, a latência na comunicação com serviços, o consumo de energia e a informação de desconexão são informações monitoradas por sensores de um dispositivo móvel;
- *Effectors* (Efetores ou Executores): Realizam as ações de adaptação que podem ser, por exemplo, desde uma simples alteração de um parâmetro do elemento autonômico, até como configurar uma nova instância de serviço em um agrupamento de servidores de web o qual envolve a alocação e gerenciamento de diversos recursos;
- *Monitor* (Monitor): Esta função provê os mecanismos para coletar, agregar, filtrar e reportar detalhes (como métricas e topologias) coletados dos recursos gerenciados;
- *Analyse* (Analisar): A função de análise provê os mecanismos que correlacionam e modelam situações complexas (por exemplo, previsão de séries temporais e modelos de filas). Esses mecanismos permitem que o gerente autonômico (autonomic manager) aprenda sobre o ambiente e ajude a predizer situações futuras;
- *Plan* (Planejar): Provê os mecanismos que constroem as ações necessárias para alcançar as metas e objetivos. Esta etapa utiliza políticas para guiar esta tarefa.
- *Execute* (Executar): Provê os mecanismos que controlam a execução do plano considerando atualizações dinâmicas;
- *Knowledge Base* (Base de conhecimento): Contém os dados usados pelas quatro funções autonômicas do gerente, as quais armazenam e acessam o conhecimento de forma compartilhada. Este conhecimento pode incluir dados tais como informação sobre topologia, logs de histórico, sintomas e políticas.

Figura 2.1 - Loop de adaptação MAPE-K



Fonte: Adaptado de IBM (2005)

Durante o processo do loop um gerente autonômico, também conhecido como mecanismo de adaptação (*adaptation engine*) e agente adaptativo (*adaptation agent*), pode acessar os *Sensors* e os *Effectors* para agregar mais informações para tomada de decisão, o que caracteriza uma forma de monitoramento adaptativo.

Outras publicações utilizam diferentes modelos de loop como Garlan et al. (2004) e Saleihie; Tahvildari (2009), os quais utilizam um padrão monitorar, detectar, decidir e atuar, baseados no conhecimento de adaptação. Dobson et al. (2006) representam um loop similar como o loop de controle autonômico simular ao da comunicação autonômica, que inclui coletar, analisar, decidir e atuar. Oreizy et al. (1999) utilizam um loop de gerenciamento de adaptação, que é composto por alguns processos que visam coletar observações, avaliar e monitorar observações coletadas, planejar mudanças, implantar as descrições de mudança, além de colocar as mudanças em prática.

Todos os modelos acima possuem em comum a noção de que há um processo externo da lógica de negócio que continuamente monitora elementos adaptáveis controlando-os por um loop com *feedback*. Contudo, em benefício desta pesquisa, escolheu-se somente detalhar as etapas de loop da computação autonômica por possuir uma maior descrição bibliográfica e especificação, por parte da IBM. Os processos de monitoramento, análise, planejamento e execução (MAPE) são descritos com maiores detalhes nas próximas seções.

2.4.1 Monitoramento

A função de monitoramento coleta, agrega, filtra e reporta detalhes, tais como métricas e topologias, oriundos de recursos gerenciados para o gerente autonômico (IBM, 2005). Tal função é uma característica essencial dos elementos autonômicos, pois o processo de monitorar a si mesmo e os recursos gerenciados garante que sejam cumpridos os objetivos visados, além de registrar informações que servem de base para a adaptação, a otimização e a reconfiguração (KEPHART; CHESS, 2003). Assim, a largura de banda e a latência da rede poderiam ser usadas como medidas de desempenho de servidores de web, enquanto a duração da bateria poderia ser usada como uma medida relevante para um smartphone.

Huebscher e Mccann (2008) identificaram dois tipos de monitoramento em sistemas autonômicos, os quais podem ser combinados para proporcionar ao gerente autonômico informações mais ricas sobre o estado do artefato, sendo eles:

- *Monitoramento Passivo*: Significa que os artefatos gerenciados e seus componentes são monitorados usando um componente de monitoração de terceiros. Este componente pode consistir em ferramentas de gerenciamento de desempenho existentes. Por exemplo, no sistema operacional Linux o comando *top* retorna a informação sobre a utilização da CPU sobre cada processador e o comando *vmstat* retorna as estatísticas de utilização da memória e da CPU;
- *Monitoramento Ativo*: Indica que há interação direta com as partes monitoradas pelo sistema autonômico, em particular com sensores fornecidos pelos artefatos monitorados. Por exemplo, modificar e adicionar código à implementação da aplicação ou do sistema operacional para que capture funções ou chamadas de sistema, o que muitas vezes pode ser, até certo ponto, automatizado.

Além disso, o processo de monitoramento pode ser contínuo ou adaptativo, o que afeta diretamente o custo de monitoramento e tempo de detecção, onde, se contínuo, coleta e processa continuamente os dados, enquanto que, se adaptativo, monitora sobre algumas características selecionadas, e caso encontre alguma anomalia, busca coletar mais dados ou alterar a forma de monitoramento (SALEIHIE; TAHVILDARI, 2009) o que Lalanda, Mccann e Diaconescu (2013) entendem como ser o processo de monitorar o próprio monitoramento.

2.4.2 Análise

A função de análise lida com a habilidade de entender o contexto atual e determinar o melhor estado para os artefatos gerenciados, provendo os mecanismos que correlacionam e modelam situações complexas (por exemplo, previsão de séries temporais e modelos de filas). Esses mecanismos permitem que o gerente autonômico aprenda sobre o ambiente e ajude a predizer situações futuras (IBM, 2005).

Para tanto há uma grande variedade de algoritmos e técnicas que podem ser usadas para se detectar comportamentos inadequados e deficiências, estabelecendo correlações, antecipando situações, diagnosticando problemas e definindo a situação mais desejável e alcançável (LALANDA; MCCANN; DIACONESCU, 2013).

Predição tipicamente monitora tendências e identifica se uma restrição ou objetivo alcançará um estado indesejável em um futuro próximo ou distante. Ela pode ser implementada, por exemplo, por uma simples análise de regressão dos históricos coletados sobre um sensor, utilizando modelos ocultos de Markov que representam os estados temporais do sistema, os quais podem ser utilizados para modelar os resultados de um plano (LALANDA; MCCANN; DIACONESCU, 2013).

No campo autonômico, Kephart e Walsh (2004) citam outras três políticas que são muito utilizadas para implementar a experiência de análise: políticas *evento-condição-ação* (ECA), políticas de função objetivo e políticas de utilidade.

Evento-condição-ação (ECA), também conhecida como política de ação, é uma forma clara e direta para expressar regras de domínio assumindo a forma: quando o **evento** ocorre e a **condição** é atendida, então executa a **ação**. Regras ECA têm sido intensamente estudadas para a gestão autonômica de sistemas distribuídos. No entanto, há uma dificuldade quando uma série de políticas é especificada, uma vez que conflitos entre as políticas podem surgir e são difíceis de detectar.

A conclusão pragmática é que as regras da ECA são muito eficazes quando se trata de um pequeno número de políticas ou quando as preocupações são ortogonais. Quando surgem muitos conflitos, outros formalismos têm que ser examinados, a fim de evitar um “pesadelo” de depuração (LALANDA; MCCANN; DIACONESCU, 2013). Mizouni et al (2014) e Garlan et al (2004) utilizam essa abordagem.

Políticas de objetivos, ao invés de especificar o que fazer no estado S, especificam um estado desejado, ou um ou mais critérios que caracterizam todo um conjunto de estados

desejáveis, o que exige replanejamento por parte do gerente autonômico e, sendo assim, necessitam de mais recursos que políticas ECA.

Apesar disso, as políticas de objetivo sofrem o problema de todos os estados serem classificados como desejáveis ou indesejáveis. Assim, quando um estado desejável não pode ser alcançado, o sistema não sabe qual dos estados indesejáveis teria um resultado mais próximo de um estado desejável (KEPHART e WALSH, 2004). Dentre os artigos avaliados, (DALPIAZ; GIORGINI; MYLOPOULOS, 2013), (FLOCH et al., 2006) e (ROSA et al., 2013) são exemplos de trabalhos que utilizam essa politica.

Políticas de utilidade são funções objetivo que expressam o valor de cada possível estado. Assim, em vez de realizar uma classificação binária entre estados desejáveis versus indesejáveis, elas atribuem um valor real de conveniência para cada estado. Por que se o caso mais desejado não for especificado com antecedência, será selecionado o estado com a maior utilidade a partir da coleção de estados possíveis.

Políticas de utilidade apresentam a especificação de comportamento mais fina e flexível do que as políticas de objetivo e ação. Por outro lado, podem exigir especialistas de políticas do domínio para especificar um conjunto multidimensional de preferências, o que pode ser difícil de obter, além de requererem o uso de modelagem e possivelmente outros algoritmos (KEPHART e WALSH, 2004). Ramirez et al (2009), Rouvoy et al (2009), além de Pascual, Pinto e Fuentes (2015) utilizam essa abordagem.

2.4.3 Planejamento

A função de planejamento cria ou seleciona procedimentos para aprovar uma alteração desejada no recurso gerenciado, podendo assumir muitas formas, variando de simples comandos a complexos fluxos de trabalho. Em outras palavras, gera o plano de mudanças apropriado, que representa o conjunto de mudanças desejadas no recurso gerenciado, e logicamente, passa o plano de mudanças para a função de execução (IBM, 2005). Além disso, um plano, segundo Lalanda, McCann e Diaconescu (2013), pode ser estático ou dinâmico:

- Um plano estático é composto por um conjunto de etapas que são realizadas quando uma condição particular ocorre. Por exemplo, em um sistema largamente distribuído, se o monitoramento detecta que um nó morreu, algumas simples etapas seriam informar o usuário e automaticamente tentar reiniciá-lo. Caso estas decisões não resultem no efeito desejado, uma decisão posterior adequada seria buscar informar o distribuidor de carga de trabalho para evitar o uso do nó.

– Um plano dinâmico e mais sofisticado para o exemplo acima seria modelar o comportamento dos componentes do artefato gerenciado e então escolher o plano (a partir de planos já existentes) gerando em tempo real pela iteração entre modelos de diferentes caminhos ou cenários e escolhendo o melhor. Para tanto, o planejador deve fazer hipóteses, até mesmo utilizar predição, sobre os efeitos das ações escalonadas nos artefatos gerenciados.

De acordo com Lee et al. (2007), com base: (1) nos problemas ou oportunidades detectadas pela função de análise; (2) das capacidades de ação disponibilizadas por um conjunto de executores dos recursos gerenciados; e (3) dos conteúdos armazenados da base de conhecimento de um gerente autonômico; o planejamento decide o que fazer, calculando as consequências de executar qualquer adaptação, para cada adaptação contemplada. Na maioria dos casos, o resultado é uma agenda de ações que a função de execução pode realizar.

2.4.4 Execução

A função de execução realiza as mudanças com base nas ações recomendadas pela função de planejamento. Ela provê todos os mecanismos para escalonar e realizar as mudanças necessárias no sistema. Uma vez que um gerente autonômico tenha gerado o plano correspondente às mudanças requisitadas, poderá modificar o estado de um ou mais recursos gerenciados. Sendo assim, a função de execução é responsável pela realização do procedimento que foi gerado pelo planejamento por meio de uma série de ações que utilizam a interface dos executores (*Effectors*) para afetar os recursos gerenciados (IBM, 2005).

De acordo com Lalanda, McCann e Diaconescu (2013), a separação das atividades de planejamento e execução é uma forma mais eficiente de lidar com sistemas dinâmicos, estocásticos ou contextos computacionais fracamente observáveis. Um plano, assim sendo, poderia especificar um conjunto de parâmetros para serem trocados, sem restrições de ordem. A atividade de execução, então, tem de determinar como e quando esses parâmetros serão alterados, respeitando as restrições de ordem. Isto é uma questão de oportunidade de sincronização quando dependências funcionais e não funcionais são consideradas. Usualmente, um parâmetro pode ser trocado somente se alguma condição for alcançada.

Por exemplo, em um ambiente ubíquo, a forma de gerenciar algumas ações pode precisar de alteração em razão da evolução do sistema. Algumas ações devem inclusive serem canceladas quando mudanças importantes ocorrerem no ambiente. Uma maneira de lidar seria utilizar uma máquina de estados finitos para orquestrar o plano de execução em ambientes

dinâmicos. Tais máquinas permitem a coordenação explícita e eficiente entre as ações corretivas, e também com operações contínuas.

Para executar tais adaptações no ambiente McKinley et al. (2004) identifica duas abordagens para implementação de adaptações em software: *por parâmetro* ou *composicional*. A *adaptação por parâmetro* é a abordagem mais simples de ser implementada, ela consiste em modificar variáveis do programa que determinam o comportamento. Contudo, esta forma de adaptação possui uma fraqueza inerente, onde ela poderia ajustar parâmetros dirigindo um aplicativo para usar uma estratégia diferente, mas não poderia adicionar novas estratégias.

Adaptação composicional é a abordagem que permite recomposição dinâmica do software durante a execução. Em outras palavras, permite a troca o algoritmo ou a estrutura dos componentes do sistema para o seu ambiente atual, os quais não seriam possíveis com ajustes de variáveis do programa ou seleção de estratégias.

2.5 Sensoriamento Urbano

Sistemas de sensoriamento em escala urbana tradicionais foram implementados por diversos governos municipais com o foco em monitoramento ambiental, tráfego de veículos e para questões de saúde e emergência em todo o mundo (CAMPBELL et al., 2006). Contudo, a rápida evolução dos recursos e capacidades de processamento de dispositivos móveis como os *smartphones*, aliada à crescente aceitação de seu uso por usuários, têm sido fonte de incentivo para diversos trabalhos relevantes sobre sensoriamento urbano centrado no usuário.

Alguns exemplos desses trabalhos utilizando somente dispositivos móveis podem ser encontrados em Miluzzo et al. (2008), Cornelius et al. (2008), Das et al. (2010), Shin et al. (2011), Brouwers e Langendoen (2012), Ra et al. (2012) e Wu, Zhu e Zhang (2013). Além desses, outros trabalhos mais abrangentes, envolvendo infraestruturas com dispositivos fixos e móveis, podem ser encontrados, por exemplo, em Campbell et al. (2006) e Sanchez et al. (2014).

Desta forma, Sensoriamento Urbano é uma área emergente que busca prover soluções para tecnologias centradas em pessoas em sistemas de larga escala como os encontrados em ambientes urbanos, os quais utilizam dispositivos de telefonia móvel como principal veículo de interação entre usuários e recursos computacionais. Nesta área, segundo Khan et al (2012), pode-se categorizar o sensoriamento urbano em torno de duas classes maiores, de acordo com a ciência e o envolvimento das pessoas que possuem os dispositivos:

1. Sensoriamento Participativo: a proposta participativa incorpora pessoas em etapas significantes de decisão no sistema de sensoriamento, tais como decidir que dados serão compartilhados e quais extensões dos mecanismos de privacidade devem ser autorizadas a afetar a fidelidade dos dados (LANE et al., 2008). Em outras palavras, a participação do usuário é diretamente envolvida com as ações de sensoriamento, como no caso de fotografar determinados locais/eventos e escolher se quer ou não compartilhar as fotografias na rede.
2. Sensoriamento Oportunista: a proposta oportunista muda o peso do suporte ao sensoriamento da aplicação do usuário para o sistema de detecção, o qual automaticamente determinará quando o dispositivo pode ser usado para atender requisições da aplicação. Neste paradigma, os usuários configuram seus dispositivos para permitir que os aplicativos sejam executados, mas não estarão cientes de quais aplicativos estão ativos em um determinado momento (CAMPBELL, 2008). Ou seja, o usuário não é diretamente envolvido nas decisões sobre o momento em que as aplicações estarão em funcionamento, pois a própria aplicação tomará as medidas necessárias para tanto. Por exemplo: enviar os dados para um servidor informando as mudanças de contexto de localização geográfica ou de temperatura corporal é uma tarefa que não necessita a intervenção direta do usuário para coletar os dados.

De acordo com Lane et al. (2008), embora soluções participativas e oportunistas possam ser consideradas complementares, deve ser salientado que uma abordagem de projeto de sensoriamento oportunista produz um sistema que suporta mais facilmente implementações de grande escala e diversidade de aplicações.

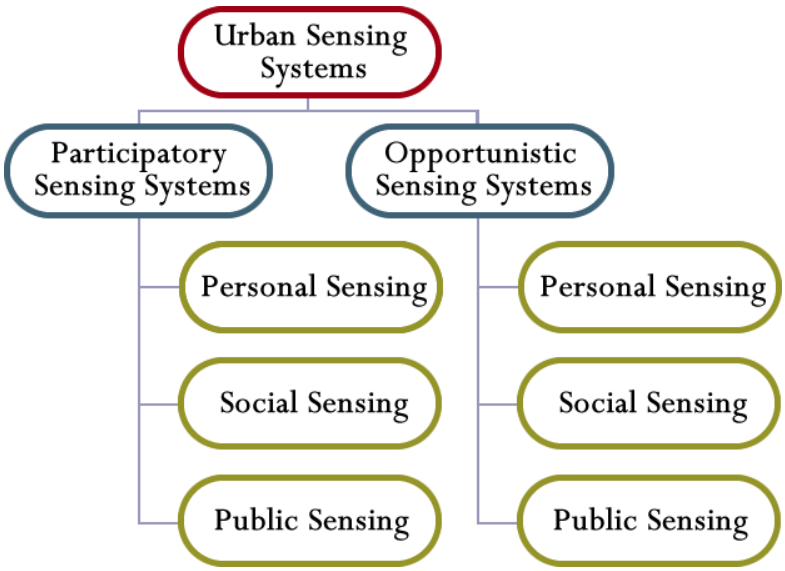
Além disso, cada uma das duas categorias de tipo de sensoriamento pode ser dividida ainda em 3 escalas diferentes de sensoriamento, representadas através da Figura 2.2, que de acordo com Campbell et al. (2008) e Lane et al. (2010) são:

1) Sensoriamento pessoal ou individual, o qual foca no monitoramento e armazenamento de informações de uso pessoal, tais como informações sobre as atividades do dia-a-dia, localização pessoal, monitoramento de saúde e de atividades físicas;

2) Sensoriamento Social ou de Grupo, no qual a informação é compartilhada dentro de grupos sociais de interesse, como amigos e comunidades;

3) Sensoriamento Público ou Comunitário, onde os dados são compartilhados entre todos para o bem público, tais como informações sobre poluição, tráfego e engarrafamento de veículos.

Figura 2.2 - Classificação de Sistemas de Sensoriamento



Fonte: Adaptado de Khan et al. (2012).

Em outras palavras, para melhor abstrair essas escalas de sensoriamento, para o Sensoriamento pessoal um simples sensor tem maior relevância para o sistema, pois o serviço que utiliza o sensor de tal dispositivo provê benefício a esse usuário e não a um conjunto maior. Caso esse sensoriamento beneficie um conjunto maior de usuários que também fazem sensoriamento, dizemos que esse sensoriamento é de escala social ou de grupo. Contudo, este provedor de serviço se limita ainda a tal conjunto específico de membros.

Por outro lado, se estes sensores tiverem seus dados compartilhados em âmbito aberto, em que os provedores de serviço beneficiem de alguma forma todos os usuários, incluindo os que não compartilharem dados, diz-se que esse sensoriamento é público ou comunitário. Desta forma, quanto maior for o público compartilhando dados, maior será a confiabilidade dos dados coletados em uma dada região, e menor será a importância de um único dispositivo de sensoriamento para o provedor do serviço.

Todas as aplicações de sensoriamento em trabalhos de Sensoriamento Urbano seguem um modelo onde há uma ou mais tarefas de sensoriamento que reportam os dados coletados de forma pura ou pré-processada para servidores. Contudo, a implantação de tais tarefas em plataformas móveis que possuem contato com o usuário pode ser classificada em duas abordagens diferentes.

Na primeira as aplicações com tarefas de sensoriamento são predefinidas de forma fixa e local na aplicação de sensoriamento, sendo no máximo escolhido pelo usuário quais delas devem executar, sobre quais sensores e em qual configuração sendo esse o caso de Miluzzo et al. (2008) e Wu, Zhu e Zhang (2013). Nestas abordagens geralmente são atribuídos a um

framework os componentes necessários para o funcionamento da aplicação. Na segunda abordagem a aplicação de sensoriamento executa tarefas de sensoriamento remotas distribuídas (*push*) ou baixadas (*pull*) através do servidor e executam essas tarefas geralmente em um modelo mestre escravo, sejam elas representadas por arquivos binários, caso de Das et al. (2010) e Theodoridis et al. (2014), ou descritas em formato de texto, como os casos de Shin et al. (2011), Brouwers e Langendoen (2012) e Ra et al. (2012). Por fim, o suporte a adaptação em sensoriamento urbano será discutido na Seção 3.2 em conjunto dos casos de autoadaptação encontrados em cada trabalho de sensoriamento urbano.

2.6 Orientação a Contexto

Uma das mais acuradas definições para contexto, de acordo com Baldauf, Dustdar e Rosenberg (2007), foi dada por Abowd et al. (1999, p. 3, tradução nossa), os quais se referem a contexto como: "*Qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto que é considerada relevante para a interação entre um usuário e uma aplicação, incluindo entre eles mesmos*".

A adaptação de contextos pode ser definida de maneira estática, em tempo de compilação, ou de maneira dinâmica, em tempo de execução. Segundo Wei e Chan (2011) criar regras para tratar adaptações gera uma carga de trabalho muito grande para os desenvolvedores, os quais sentem dificuldades em abordar todas as possibilidades em ambientes altamente dinâmicos, caso dos ambientes ubíquos, sendo assim mais adequado utilizar a adaptação de contextos em tempo de execução.

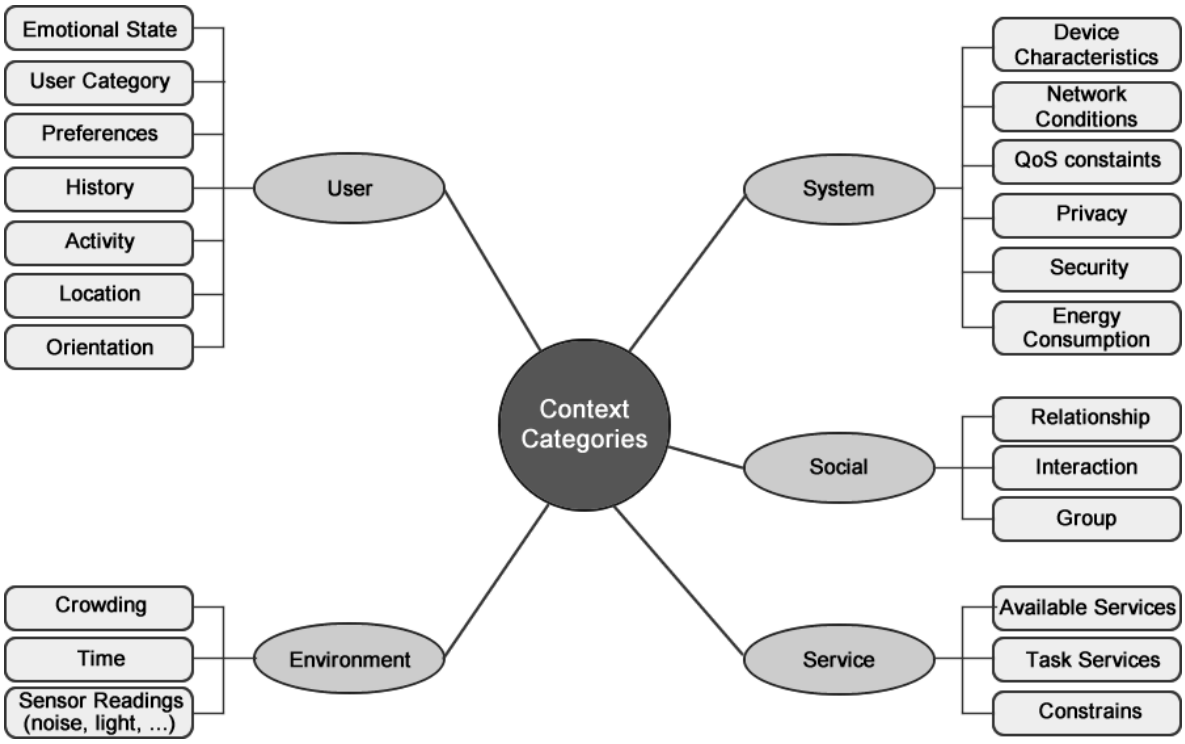
Também, o uso da adaptação baseado em contexto pode melhorar significativamente a experiência do usuário, já que cada indivíduo pode ter preferências, em lugares diferentes e em situações diferentes que podem ser utilizadas para adaptar a esses contextos individuais, contribuindo para tornar, por exemplo, a visita a um local mais agradável ou memorável com base nessas informações, como é o caso de Emmanouilidis, Koutsiamanis e Tasidou (2013).

Contextos são amplamente utilizados em diversas áreas para simbolizar informações úteis entre aplicações e usuários. No caso de Emmanouilidis, Koutsiamanis e Tasidou (2013) contextos são utilizados em dispositivos móveis no cenário de guias móveis, separando-os em cinco grandes domínios, Usuário, Ambiente, Sistema, Social e Serviço, como pode ser visto pela taxonomia de contextos presente na Figura 2.3. Cada uma das categorias possui distintas características nestes domínios.

Usuário é o centro dos serviços personalizados, necessitando ter contextos de localização e informações individuais bem definidos, como emoções e preferências; sistemas referem-se a contextos não funcionais, frequentemente relevantes para questões tecnológicas; o ambiente provê os contextos ambientais; os contextos sociais oferecem a oportunidade de serviços interagirem entre grupos de pessoas; e o serviço envolve os contextos particularmente relevantes para as funcionalidades oferecidas em guias móveis.

Para utilizar contextos é preciso definir o modelo de contextos de forma que um computador possa processar e armazenar esses dados. Strang e Linnhoff-Popien (2004) resumiram as mais relevantes abordagens de modelagem contextual com base na estrutura usada para representar esses dados, sendo elas:

Figura 2.3 - Exemplo de taxonomia de contextos em dispositivos móveis



Fonte: adaptado de Emmanouilidis, Koutsiamanis e Tasidou (2013).

- Chave-valor é mais simples para modelagem de contextos, em que utiliza pares chave-valor para descrever as capacidades de um serviço;
- Modelos de esquema de marcação usam estruturas de dados hierárquicas consistidas de marcações em tags com atributos e conteúdo;
- Modelos gráficos utilizam a linguagem de modelagem unificada (UML) para modelagem de contextos;
- Modelos orientados a objetos utilizam técnicas oriundas da orientação a objetos para modelar contextos (por exemplo, herança, encapsulamento e reutilização).

Nesta abordagem o acesso a contextos e à lógica de processamento contextual é fornecido por interfaces bem definidas;

- Modelos baseados em lógica possuem um alto grau de formalismo. Tipicamente, fatos, expressões e regras são usados pra definir um modelo contextual deste gênero. A inferência (também chamada de raciocínio) pode ser usada para derivar novos fatos com base nas regras existentes no sistema;
- Modelos baseados em ontologias representam uma descrição de conceitos e relações. Portanto, ontologias são instrumentos muito promissores para modelar informações contextuais devido à sua alta e formal expressividade, além de possibilitar a aplicação de técnicas de raciocínio sobre ontologias.

Além das formas de modelagem acima Bettini et al (2010) afirmam que é possível ainda utilizar modelos híbridos, buscando aproveitar os benefícios das diferentes abordagens, o que, segundo ele, é válido também para técnicas de raciocínio aplicadas sobre contextos.

Aplicações necessitam se adaptar às mudanças nas informações contextuais (contextos físicos, computacionais e do usuário) que são geralmente coletadas de diversas fontes que produzem diferentes qualidades de informação, muitas vezes suscetíveis a falhas. A comunidade de computação ubíqua compreende cada vez mais que o desenvolvimento de aplicações sensíveis ao contexto deve ser apoiado por adequadas modelagens de informações contextuais e técnicas de raciocínio (BETTINI et al, 2010).

2.7 Considerações Finais

Este capítulo abordou os conceitos e os paradigmas utilizados para embasamento do restante do trabalho. Cada seção possuiu um objetivo específico relacionado com o restante do trabalho. A Seção 2.1 apresentou a computação ubíqua tendo em vista ligar os demais conceitos dentro de uma área do conhecimento. A Seção 2.2 introduziu sobre a computação autonômica, além do loop de adaptação MAPE-K, escolhido para esse trabalho entre outras variações.

A Seção 2.3 apresentou as propriedades *self-** fundamentais, iniciando pela sua origem e finalizando ao elencar termos parecidos, que são frequentemente encontrados nos trabalhos sobre sistemas autoadaptativos citados no Capítulo 6 (Comparação de trabalhos relacionados). Além disso, o conceito das quatro propriedades *self-** fundamentais é utilizado para extrair os casos de adaptação encontrados no Capítulo 3, análise de requisitos.

A Seção 2.4 apresentou em maiores detalhes o loop de adaptação da computação autonômica, o MAPE-K, juntamente com diversas técnicas e características encontradas nas quatro etapas de processamento Monitoramento, Análise, Planejamento e Execução. Os conceitos explorados por essa seção implicam diretamente como o modelo proposto no Capítulo 4 foi definido e nos critérios para comparação com os trabalhos relacionados na Seção 6.2. A Seção 2.5 sobre sensoriamento urbano apresentou conceitos e características extraídas da literatura, sendo estes utilizados como critérios na Seção 3.2, ao analisar trabalhos de sensoriamento urbano.

Por fim, a Seção 2.6 apresentou conceitos e técnicas sobre orientação a contexto que serão utilizados como embasamento para definições do modelo proposto no Capítulo 4 e na Seção 5.1 sobre a implementação do protótipo.

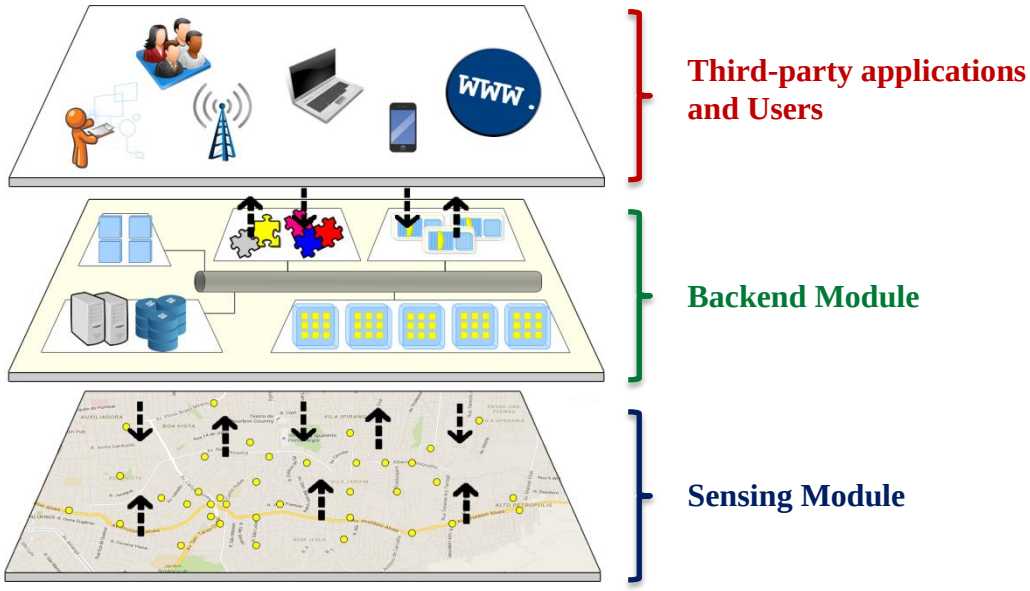
3 ANÁLISE DE REQUISITOS

Este capítulo apresenta uma visão geral da arquitetura UrboSenti, bem como trabalhos atuais de pesquisa na área de sensoriamento urbano. A partir da apresentação é feita uma análise desses trabalhos buscando requisitos e casos de adaptação segundo a visão da computação autonômica. Tais requisitos são utilizados para a construção da proposta no próximo capítulo.

3.1 Arquitetura UrboSenti

A arquitetura UrboSenti (ROLIM et al., 2014) considera o uso de diversos nodos sensores (fixos e móveis) dispersos ao longo de cidades para efetuar a coleta de dados, como ilustra a Figura 3.1. Os dados coletados são processados para posterior fornecimento de informações aos cidadãos da cidade e outros sistemas. Para tanto é utilizada uma abordagem SOA (*Services Oriented Architecture*) resultando em um baixo acoplamento entre os componentes da arquitetura, o que proporciona flexibilidade e reutilização na composição de novos serviços. Em suma, a arquitetura considera três elementos principais que compõem as cidades inteligentes: *Sensing Module*, *Backend Module* e *Third-party Applications*.

Figura 3.1 - Arquitetura geral do sistema



Fonte: adaptado de Rolim et al (2014).

As aplicações de terceiros (*Third-party Applications*) compõem o módulo mais abstrato sendo composto por aplicações, outros serviços e sistemas que se beneficiam das informações processadas pelos módulos *backend*. As informações são acessadas através de

módulos *backend* e não diretamente pelos módulos de sensoriamento, principalmente por questões de segurança e privacidade dos dados do usuário.

O módulo *backend* (*Backend Module*) interage com os módulos de sensoriamento, recebe os dados coletados, processa-os e disponibiliza as informações para os cidadãos e outros sistemas. Este módulo pode possuir diversos serviços internos que possibilitam auxiliar na análise dos dados, bem como funções mais específicas de cada aplicação. Além disso, questões sobre políticas de privacidade, a respeito de quais dados do usuário podem ser compartilhados ou não, são consideradas por esse tipo de módulo. Por vezes tornar anônimo o relato dos dados é uma medida adotada para garantir tal qualidade, como pode ser visto nos trabalhos PRISM (DAS et al., 2010), Medusa (SHIN et al., 2011) e AnonySense (RA et al., 2013).

O modulo de sensoriamento (*Sensing Modul*e) é o software executado em dispositivos móveis e sensores fixos dispersos ao longo da cidade, responsável por coletar os dados de forma oportunista e participativa. Além do sensoriamento tradicional, isto é, através de sensores de hardware e de software, é possível também realizar sensoriamento social que consiste na interação e coleta de informações de redes sociais, desde que se possua permissão e interfaces de acesso para tanto.

Os módulos de sensoriamento podem realizar mais funções do que somente coleta de dados brutos dependendo de sua aplicação e serviço. Então, dependendo da demanda, camadas adicionais de processamento podem ser consideradas, tais como identificar contextos com base nos dados coletados (BROUWERS E LANGENDOEN, 2012; WU, ZHU, ZHANG, 2013) e para executar tarefas remotas (DAS et al. 2010; RA et al., 2012). Exemplos de sensores de cada tipo contendo possíveis informações derivadas deles podem ser encontrados na Tabela 3.1.

Tabela 3.1 - Exemplo de sensores e informações derivadas.

Sensores	Origem	Informação derivada
Acelerômetro com 3 eixos	Hardware	Movimentação
Coordenadas GPS	Hardware	Localização e rastreamento ao ar livre
Lista de Processos	Software	Que aplicações e serviços estão executando
Estado da Rede	Software	Estado do tráfego de rede do dispositivo ou aplicação
Contato do usuário avalia restaurante próximo da localização atual como bom	Social	Recomendação de local ao usuário baseado na proximidade dos perfis de ambos os usuários

Fonte: Primária.

A UrboSenti propõe que cada módulo de sensoriamento possa ter os elementos descritos na Figura 3.2. No topo do módulo a camada de **Aplicações** representa as aplicações de sensoriamento que utilizam e determinam os componentes utilizados para execução. Na parte inferior do módulo, as camadas **Sistema Operacional e Hardware** representam a infraestrutura de hardware e o sistema operacional nativo utilizado para execução.

Na camada central, o módulo de sensoriamento propriamente dito se encontra. O núcleo seu é chamado **Micro-kernel**. Sua principal função é proporcionar os serviços básicos para os componentes mais externos, atendendo diferentes requisitos de cada tipo de dispositivo. Internamente, o Micro-kernel é estruturado em:

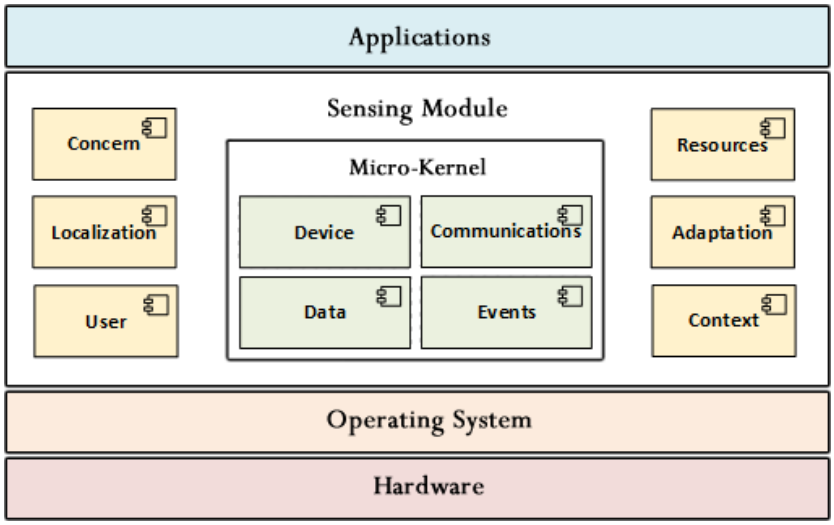
(a) *Device*: proporciona informações básicas a respeito do dispositivo (nome, endereço de rede, interfaces existentes, GPS, sensores internos, etc.);

(b) *Communications*: proporciona métodos para enviar e receber dados utilizando a estrutura de rede disponível (IEEE 802.11b/g/n - estruturado e ad-hoc, GPRS/EDGE/3G e Ethernet - 802.3 - como base para a troca de dados através de TCP ou UDP). Quando nenhuma destas infraestruturas de rede não estiver disponível, o componente fornece suporte para a troca de dados utilizando o paradigma de Redes Tolerantes ao Atraso (*DTNs - Delay-Tolerant Networks*) via interface Bluetooth, além do tratamento da desconexão;

(c) *Data*: manipula as operações de armazenar e recuperar dados;

(d) *Events*: captura eventos externos de interesse (como a mudança de posição devido à movimentação do usuário, alteração no status de uma interface do dispositivo, etc.). Os eventos detectados são disponibilizados para serem utilizados pelos outros componentes. Além disso, o próprio componente *Events* pode ser utilizado para agendar eventos temporais.

Figura 3.2 - Componentes internos do módulo de sensoriamento da arquitetura UrboSenti



Fonte: Adaptado de Rolim et al (2014).

Os demais componentes, e laranja na figura, compõem os componentes sob demanda do módulo de processamento. Estes abordados individualmente a seguir.

Localization: controla questões relacionadas a informações de geolocalização, Pontos de Interesse (*Points of Interest*) e Serviços Baseados em Localização (*Location Based Services*).

Resources: conjunto de componentes e serviços utilizados para monitorar os recursos locais e descobrir recursos existentes em outros dispositivos.

Concern: lida com questões de segurança, criptografia e privacidade das entidades envolvidas no sensoriamento.

Context: disponibiliza para as aplicações um conjunto de componentes e métodos para dar suporte à sensibilidade ao contexto (*context reasoning, context knowledge, context discovery*).

User: manipula os dados pessoais básicos do usuário, tais como preferências e perfis de redes sociais.

Adaptation: disponibiliza os componentes e serviços para proporcionar a adaptação no comportamento do dispositivo e das aplicações. O comportamento adaptativo do componente é especificado pelo mecanismo proposto de autoadaptação desenvolvido nesta dissertação utilizando o ponto de vista da computação autonômica.

Os componentes expostos conferem uma maior flexibilidade ao módulo de sensoriamento, considerando que há uma variedade de requisitos que o tornam heterogêneo em termos de plataforma de hardware e de requisitos de software para suporte à execução. Assim, esses componentes podem ser utilizados em diferentes escalas dependendo da demanda do tipo de aplicação e do tipo de dispositivo em questão, contudo em qualquer caso pelo menos os componentes do Micro-Kernel devem ser utilizados, sendo assim obrigatórios. Alguns exemplos de usos dos componentes são ilustrados abaixo, primeiramente pelo ponto de vista de aplicações e segundo por dispositivos com diferentes características de hardware.

Em aplicações robustas como as do estilo de tarefas remotas em dispositivos móveis, isto é, que recebem remotamente em tempo de execução o que devem executar, casos dos trabalhos de Das te al (2010) e de Ra et al (2012), os componentes do Micro-Kernel e os componentes sob demanda *Location*, *Concern*, *User* e *Adaptation* poderiam ser utilizados. Assim, uma vez que o acesso à localização deveria ser controlado, os perfis de usuários administrados pelo próprio middleware, as otimizações e os bloqueios de acesso a conteúdo poderiam ser feitos pelo componente *Adaptation* com suporte do componente *Concern*, além de rodarem em conjunto a um *sandbox* no topo da aplicação. Em contraste, uma aplicação

definida em tempo de desenvolvimento, como um *scanner* de pontos de acesso Wi-Fi que visa dispositivos móveis em uma cidade, somente precisaria do Micro-Kernel e do componente de localização para operar com sucesso.

Por fim, existem diferenças entre as necessidades dos dispositivos por serviços e componentes devido a questões relacionadas com o hardware e o sistema operacional. Por exemplo, um módulo de sensoriamento com conexão cabeada fixa em um poste de iluminação ou em um sensor de trava de porta somente necessitariam do Micro-Kernel para operar em conjunto com a aplicação, uma vez que a rede não é volátil e otimizações nesse caso seriam desnecessárias. Por outro lado, um carro com acesso a redes móveis e GPS precisaria, pelo menos, dos componentes do Micro-Kernel, *Localization* e *Adaptation* para operar.

3.2 Análise dos Trabalhos de Sensoriamento Urbano

Os trabalhos de sensoriamento urbano apresentados nessa seção foram utilizados como base para o levantamento de requisitos de adaptação que são empregados na presente dissertação e aplicados na arquitetura UrboSenti. Tais trabalhos foram selecionados por apresentarem as características de uma plataforma abrangente de sensoriamento resultante de um middleware, framework ou serviço proposto, de forma similar à proposta pela UrboSenti, sendo estes são expressos em cada linha da Tabela 3.2. Os critérios de comparação, dispostos nas colunas da Tabela, foram devidamente embasados na Seção 2.5 através de levantamento bibliográfico, sendo eles e suas opções resumidas a:

- Tipo de Sensoriamento Suportado (TS): Oportunista (Oport.), Participativo (Part.), ou Ambos;
- Escalas de Sensoriamento Suportadas (ES): Pessoal, Grupo e Pública, ou Todos;
- Dispositivos (DEV): Sensores Móveis (Smartphones, Veículos e similares) Sensores Fixos, ou Todos;
- Forma de implantação das tarefas de sensoriamento (DT): Local ou Remota;
- Possui algum caso de adaptação (ADPT): Sim e Não.

Pela comparação dos trabalhos na tabela supracitada é possível notar que nem todos eles possuem toda a abrangência da UrboSenti em termos de sensoriamento (TS), escala de sensoriamento (ES) e suporte a dispositivos (DEV). Quanto à abordagem de implantação de tarefas (DT), a UrboSenti optou por utilizar uma abordagem local, devido a esta possibilitar

mais opções de aplicação do que somente a implantação remota, uma vez que a própria implantação remota com sandbox pode ser feita pela aplicação. Desta forma, o suporte em nível de middleware para distribuição de tarefas remotas foi sacrificado em prol de um maior suporte a aplicações, o que não impede que trabalhos futuros adicionem um componente específico para tal funcionalidade.

Tabela 3.2 - Comparação entre trabalhos relacionados de Sensoriamento Urbano

Trabalho	**TS**	**ES**	**DEV**	**DT**	**ADPT**
MetroSense (Campbell 2006)	Oport.	Todas	Todos	Remota	Sim
CenceMe (Miluzzo 2008)	Oport.	Pessoal	Smartphone	Local	Sim
PRISM (Das 2010)	Ambos	Pública	Smartphone	Remota	Sim
AnonySense (Cornelius 2008) e (Shin 2011)	Ambos	Pública	Smartphone	Remota	Não
Pogo (Brouwers e Langedoen 2012)	Ambos	Todas	Smartphone	Remota	Sim
Medusa (Ra 2012)	Part.	Social e Público	Smartphone	Remota	Sim
MobiSens (Wu 2013)	Ambos	Todas	Smartphone	Local	Sim
SmartSantander (Sanchez 2014)	Ambos	Todas	Todos	Remota	Sim
UrboSenti (Rolim 2014)	Ambos	Todas	Todos	Local	Sim

Fonte: Primária.

O último critério apresentado na tabela, o suporte à adaptação (ADPT), exibe o valor “Sim” a todos os trabalhos que possuem algum processo autonômico de adaptação associado. Tais processos de adaptação foram extraídos dos artigos citados utilizando o ponto de vista da computação autonômica, as propriedades *self-** fundamentais (configuração, otimização, proteção e cura) e pelas semelhanças entre eles mesmos, agrupando-os em sete casos gerais de adaptação dispostos na Tabela 3.3.

Além disso, uma vez que a UrboSenti tem por objetivo suportar todos esses casos de adaptação aplicados aos módulos de sensoriamento, é necessário classificar quais adaptações são obrigatórias e onde elas devem ser processadas. Desta forma, a coluna Componente indica qual componente presente no módulo de sensoriamento da arquitetura UrboSenti participa do processo de adaptação. O papel explorado por cada componente é apresentado mais à frente.

Independente dos diversos trabalhos apresentarem a possibilidade tanto para implantação remota de tarefas quanto para a local, eles são construídos já pensando em uma estrutura de componentes obrigatória e fixa na infraestrutura básica dos dispositivos que

realizam o sensoriamento. Assim, a arquitetura de sensoriamento urbano UrboSenti, utilizada como base para o estudo, possui como diferencial adicional ser flexível, permitindo que certos componentes e funcionalidades possam ser sob demanda para atender os diferentes requisitos dos dispositivos e de suas aplicações de sensoriamento. Tal característica adiciona mais um requisito e diferencial importante para a modelagem da proposta de mecanismo de autoadaptação apresentada no próximo capítulo.

Tabela 3.3 - Casos de adaptação encontrados nos trabalhos de Sensoriamento Urbano

N	Casos de Adaptação	Trabalhos	Componentes	Propriedade
1	Autoconfiguração sem relação com o usuário	MetroSense, MobiSens e SmartSantander	Device e Adaptation	Autoconfiguração
2	Autoconfiguração a partir da decisão do usuário	MetroSene, CenceMe, PRISM, Pogo e MobiSens	User e Adaptation	Autoconfiguração
3	Identificar e tratar problemas nos nós ou serviços	MetroSense e CenceMe	Serviço no componente, Device e Adaptation	Autocura
4	Otimizações relacionadas com a rede	PRISM, Pogo, CenceMe e SmartSantander	Resources, Communication e Adaptation	Auto-otimização
5	Tratar a desconexão e a mobilidade	MetroSense, MobiSens, Pogo e SmartSantander	Communication e Adaptation	Autocura
6	Controlar o acesso a recursos no runtime do dispositivo	PRISM, Pogo e Medusa	User, Location, Resources e Adaptation	Autoproteção
7	Otimização do monitoramento dos recursos internos	CenceMe, PRISM, MobiSens e SmartSantander	User, Location, Resources e Adaptation	Auto-otimização

Fonte: Primária.

Os trabalhos de sensoriamento urbano que mais se aproximam da proposta de mecanismo de autoadaptação são o SmartSantander (Sanchez et al, 2014), o qual utiliza um gerente que pode alterar parâmetros de controle no middleware móvel, e o CenceMe (Miluzzo et al., 2008), que utiliza um mecanismo externo chamado WatchTasks, responsável por encontrar problemas nos processos, reiniciando-os caso necessário. Contudo, ambos os trabalhos não propõem soluções com a abrangência da UrboSenti.

Abaixo está listado cada um dos casos de adaptação, contendo:

- Uma breve descrição resumida de quais funcionalidades o caso engloba;
- A descrição sobre como cada um dos trabalhos relacionados trata esta adaptação;

- Ao fim, é dissertado de forma mais abstrata sobre de que forma a arquitetura UrboSenti idealmente se propõe a tratar o caso de adaptação.

1. Autoconfiguração sem relação com o usuário: engloba a autoconfiguração de elementos internos ou remotos em resposta a eventos não relacionados com o usuário. As abordagens por trabalho neste caso de adaptação são:

- MetroSense (Campbell et al., 2006): De acordo com o trabalho, as implementações de serviços devem estar cientes da assimetria que existe entre os requisitos e capacidades de cada dispositivo em relação a seus recursos e à rede. Desta forma, os serviços podem utilizar esses recursos da melhor maneira que puderem, por exemplo, alocando Jobs nos dispositivos que possuam os sensores necessários;
- MobiSens (Wu, Zhu e Zhang, 2013): Utiliza nos clientes móveis um controlador chamado Profile Pulling que dinamicamente atualiza configurações para o servidor *backend*. Desta forma, periodicamente atualiza informações sobre configurações do perfil do dispositivo, o que inclui a lista de sensores, a taxa de leitura desses sensores, estratégia, intervalo de upload para o servidor, etc;
- SmartSantander (Sanchez et al., 2014): Utiliza um processo de descoberta de recursos para detectar novos recursos, registrando-os para uso e armazenamento de descritores de recursos usando modelos padronizados. Além disso, possibilita interação entre os gerentes para troca de mensagens;
- UrboSenti: Na primeira vez que é executado o módulo de sensoriamento o componente *Device* executa um processo de descoberta dos recursos do dispositivo hospedeiro a fim de serem utilizados pelos serviços que rodam no módulo *backend* e pelos componentes do módulo de sensoriamento. Devido ao componente *Adaptation* ser sob demanda, ele não participa do processo de descoberta, entretanto, participa das mudanças de configuração em tempo de execução. Estas mudanças derivam de estados de execução dos componentes, das preferências dos usuários e da interação com o módulo *backend*.

2. Autoconfiguração a partir das decisões do usuário: compreende trabalhos que atendem questões de autoconfiguração encadeadas ou relacionadas com as ações do usuário. Por exemplo, mudanças de requisitos de privacidade e de preferências de execução do usuário podem resultar em processos de reconfiguração para manter a consistência global do sistema.

- MetroSense (Campbell et al., 2006): Cita que é delegado ao usuário a gestão do acesso de dados através das políticas de transporte, de armazenamento e de criação de dados por motivos de segurança e para detecção de problemas;
- CenceMe (Miluzzo et al., 2008): Permite por motivos de privacidade, através da interface web no portal que o usuário habilite ou desabilite o acesso aos dados de 5 modalidades de sensoriamentos suportadas em smartphones (áudio, acelerômetro, Bluetooth, fotos aleatórias e GPS). Desta forma, usuários podem escolher os dados que permitem compartilhar;
- PRISM (Das et al., 2010): Permite o usuário escolher entre três políticas para serem aplicadas no controle ao acesso a sensores: nenhum sensor, somente localização e todos os sensores;
- Pogo (Brouwers e Langendoen, 2012): Permite que o usuário tenha um controle em granularidade fina de quais informações podem ser compartilhadas para proteger sua privacidade;
- MobiSens (Wu, Zhu e Zhang, 2013): Através do mesmo controlador do item 1 acima, gerencia um perfil do cliente onde pode personalizar diferentes especificações de segurança permitindo que o sistema possa ajustar as configurações de sensoriamento de acordo com as necessidades e exigências também do usuário;
- UrboSenti: Através do componente *User*, o usuário pode escolher configurações no sistema, tais como alguns parâmetros para adaptação, limites, políticas, além de como é executado o controle de acesso e compartilhamento relacionado ao sensoriamento no componente recursos. Contudo, se o componente de adaptação estiver ativo, algumas dessas configurações podem implicar o tratamento de dependências locais ou no serviço remoto.

3. Identificar e tratar nós ou serviços com problemas: compreende trabalhos que tenham algum mecanismo de autocura para identificar e tratar problemas nos nós, ou mesmo nos serviços internos do mesmo.

- MetroSense (Campbell et al., 2006): Identifica sensores comprometidos para tentar recuperá-los, delegando essa responsabilidade a sensores de pontos de acesso, onde inclui o ser humano como parte da estratégia de detecção; usando o humano como encarregado de atualizar periodicamente o estado conhecido do sensor (por

exemplo, quando liga o PC), além da gestão do acesso de dados através de políticas de transporte, armazenamento e criação de dados;

- CenceMe (Miluzzo et al., 2008): Usa um processo em uma thread separada chamado WatchTasks para reiniciar qualquer processo que falhe;
- UrboSenti: O componente de adaptação permite que os diversos serviços e processos possam ser monitorados e restaurados para execução. Além disso, possibilita informar periodicamente o servidor sobre o estado atual de execução do nó, a fim de que o próprio servidor possa identificar problemas no nó e tomar alguma providência para tratá-lo. Para esse fim, um trabalho conjunto entre o componente de adaptação e o componente *Device* deve ser feito, pois o componente *Device* conhece e tem acesso a todos os serviços internos do módulo de sensoriamento.

4. Otimizações relacionadas com a rede: engloba trabalhos que consideram questões como o uso e a disponibilidade de rede, o uso de dados móveis, desconexão, entre outras informações para dar suporte a auto-otimização.

- PRISM (Das et al, 2010): Utiliza estratégias para upload adaptativo de relatos através da variação da taxa de upload, reduzindo a taxa de upload de acordo com a densidade de smartphones por tarefa de sensoriamento. Contudo, esse valor é definido somente no registro do dispositivo;
- Pogo (Brouwers e Langedoen, 2012): Realiza o upload dos dados coletados sempre que está carregando a bateria ou conectado a Wi-Fi. Por outro lado, quando usa dados móveis realiza esse processo ao detectar que o usuário está utilizando a aplicação de e-mail. Assim, aproveita esse tempo de conexão com o objetivo de economizar energia da bateria;
- SmartSantander (Sanchez et al., 2014): Utiliza estratégias de configuração com base em parâmetros para otimizar as operações dos protocolos de comunicação, porém não especifica quais parâmetros são;
- UrboSenti: No componente Communication, o serviço de upload realiza tal caso de adaptação considerando: 1) a estratégia de upload adaptativo encontrada em Das et al (2010), diferindo por possibilitar o envio dinâmico de taxas de upload conforme a densidade dinâmica de aplicações de sensoriamento; 2) oferece a opção de limitar ou não realizar o upload caso esteja utilizando dados móveis para

comunicação; 3) por fim, caso o componente *Resource* esteja ativo, é possível monitorar o nível de bateria e se o dispositivo está carregando ela. Nesse caso, a UrboSenti não coloca restrições de upload quando o dispositivo está carregando, em contraste, somente permitiria armazenar os dados quando a bateria está no nível baixo definido pelas configurações, deixando de realizar o upload. No mesmo componente também se pode permitir o monitoramento da quantidade de uso dos dados móveis. Tais mecanismos são configurados pelo componente de adaptação através da troca de parâmetros de controle, caso este esteja habilitado.

5. Tratar a desconexão ou mobilidade: abrange trabalhos que buscam tratar de alguma forma problemas resultantes da mobilidade relativos à volatilidade da conexão**.** O tratamento desse problemas é relacionado com a propriedade de autocura.

- MetroSense (Campbell et al., 2006): Apesar de não citar que trata a desconexão, cita que trata questões da imprevisibilidade resultante da mobilidade.
- Pogo (Brouwers e Langedoen 2012): Detecta o estado da rede usando a API do S.O. (Sistema Operacional) Android, armazenando as mensagens caso não exista conexão e enviando-as posteriormente quando há conexão.
- MobiSens (Wu, Zhu e Zhang, 2013): Agrega os dados enquanto não há conexão e faz o upload quando esta é reestabelecida;
- SmartSantander (Sanchez et al., 2014): Cita que trata a conectividade transiente, ou desconexão, dos nós móveis.
- UrboSenti: trata no componente de comunicação as questões de desconexão em dois serviços obrigatórios, um serviço de reconexão e um serviço de upload. Após todas as interfaces de comunicação disponibilizadas terem sido desconectadas, o componente de comunicação inicia o serviço de reconexão em intervalos fixos até alguma interface ser reconectada. Por outro lado, o serviço de upload verifica a restrição de envio de dados e se há conexão. Se por algum motivo não possa fazer o upload ao *backend*, armazena esses dados, enviando-os após a conexão ser reestabelecida ou a restrição removida. O componente de adaptação, quando habilitado, pode apenas fazer ajustes de parâmetros em seus elementos buscando usos melhores valores de utilidade ou para trocar configurações nesses serviços. Entre eles, para evitar o uso de armazenamento excessivo de dados por períodos muito longos de desconexão, tempos de expiração ou quantidades limite podem ser

considerados como restrições pelo componente de adaptação para apagar relatos armazenados ou restringir o upload.

6. Controlar o acesso a recursos do runtime do dispositivo. Trabalhos que suportam esse caso possibilitam o controle do acesso aos recursos do dispositivo a partir da aplicação de sensoriamento em execução no dispositivo e dos seus usuários. O objetivo principal é realizar autoproteção garantindo tanto a proteção da privacidade dos dados do usuário como controlar o acesso não autorizado aos recursos internos através de políticas de alto nível.

- PRISM (Das et al, 2010): Permite ao usuário escolher entre três políticas para serem aplicadas no controle ao acesso de sensores: Nenhum sensor, Somente localização e todos os sensores. Quando uma aplicação for enviada ao dispositivo móvel e o sensor que ela necessita estiver bloqueado, é feita uma requisição ao usuário para pedir permissão de execução.
- Pogo (Brouwers e Langedoen 2012): Permite que o usuário aplique configurações de granularidade fina que definem quais informações de sensores podem ser compartilhadas, com o objetivo de proteger a privacidade do usuário.
- Medusa (Ra et al., 2012): Permite que o usuário limite o uso dos recursos, tais como: limitar o quanto uma tarefa pode enviar de dados, limites de uso da CPU e granularidades entre tempos de leitura.
- UrboSenti: Permite que o usuário através do componente *User* atribua configurações e limites para o módulo de sensoriamento (especialmente relacionados com os componentes *Location* e *Resources*, utilizados para coleta de dados de sensores) assim como os demais trabalhos. Contudo, para garantir a completa atualização no módulo de sensoriamento, ou caso uma dessas configurações afete algo no servidor *backend*, o componente de adaptação deve estar ativo para identificar dependências e tomar as devidas ações. Nesses casos todos os eventos são síncronos e podem fazer o uso de *timeout* para definir um limite para tempo de resposta. Um evento síncrono é um evento bloqueante, requerendo uma resposta para continuar a executar.

7. Otimização do monitoramento dos recursos internos. Indica que o trabalho monitora e auto-otimiza o uso dos recursos internos com o objetivo principal de fazer o uso eficiente de bateria e evitar a exaustão da leitura dos recursos. Este monitoramento pode

resultar, por exemplo, na interrupção de algum processo, na otimização de parâmetros ou na identificação de melhores estratégias a fim de tornar o software mais eficiente no uso de energia. Este critério não realiza otimização de rede, responsável especificamente pelo item 4, abordado anteriormente, porém pode usar os estados da rede para tomada de decisão;

- CenceMe (Miluzzo et al., 2008): busca encontrar um equilíbrio entre tempo de *delay* da leitura dos dados com base na latência para compartilhar os dados no servidor. O valor de latência é considerado, mas não otimizado neste caso de adaptação;
- PRISM (Das et al., 2010): monitoramento de recursos como energia e uso de banda para influenciar tomadas de decisão sobre interromper ou não a execução da aplicação. Além disso, identifica através de predição se o usuário está parado no mesmo local, diminuindo a necessidade de atualizar a localização, tendo como consequência a redução do consumo de energia;
- MobiSens (Wu, Zhu e Zhang, 2013): Utiliza localização e aplica técnicas de aprendizagem para reduzir o consumo de bateria. Além disso, possibilita a customização pelo perfil do dispositivo e o controle da taxa de amostragem de alguns sensores.
- SmartSantander (Sanchez et al., 2014): Utiliza estratégias de configuração com base em parâmetros para otimizar as operações de componentes. Assim, através de requisitos impostos pela aplicação componentes e funções podem ser reconfigurados, por exemplo, ajustar a taxa de amostragem dos dados dos sensores dinamicamente.
- UrboSenti: Realiza tais otimizações através de ações que podem trocar parâmetros ou instanciar objetos novamente nos diversos componentes conhecidos pelo componente de adaptação. Este critério depende dos componentes habilitados do módulo de sensoriamento, tais como o componente *Resources* que permite o ajuste da taxa de amostragem dos sensores em tempo real a fim de consumir menos bateria ou ainda parar o processo de coleta de dados através de configurações prévias. O componente *Location*, de forma semelhante, permite configurar a taxa de atualização automática em conjunto com técnicas de predição para encontrar o melhor momento de atualizar a localização, considerando tempo, espaço e dados de movimentação do acelerômetro.

3.3 Resumo dos Requisitos Abordados

A partir da análise dos trabalhos acima relacionados, e da UrboSenti descrita na Seção 3.1, os seguintes requisitos para a proposta de um mecanismo de adaptação para sistemas de sensoriamento urbano foram elencados:

- O mecanismo de autoadaptação em um dispositivo de sensoriamento urbano deve ser flexível ao ponto que permita ter suporte às adaptações abordadas não obrigatórias (itens 7, 6, 4, 3, 2 e 1), utilizando diversas combinações de componentes, uma vez que muitos deles são sob demanda, o que inclui o próprio componente de adaptação. Desta forma, os requisitos da aplicação de sensoriamento e do dispositivo serão respeitados - como os exemplificados na Seção 3.1 sobre a arquitetura UrboSenti;
- Cada mecanismo de adaptação deve ser autônomo e funcionar de maneira externa aos elementos obrigatórios, como requisito da arquitetura UrboSenti - como destacado no Capítulo 1 de introdução Tal característica é compatível com o modelo da Computação Autonômica;
- O mecanismo pode interagir com outros mecanismos para dar suporte a adaptações, como no caso dos itens 1 (auto-configuração) e 3 (identificar problemas nos nós através da troca de mensagens);
- Suporte a eventos síncronos (item 6) e assíncronos (demais itens);
- Possibilidade de suporte dos sete objetivos de adaptação abordados.

4 MODELO PROPOSTO

O presente capítulo apresenta o mecanismo de autoadaptação proposto para sensoriamento urbano com base no modelo da computação autonômica. Este mecanismo é incorporado ao componente de adaptação do módulo de sensoriamento da arquitetura UrboSenti sendo a relação entre ambos explorada em maiores detalhes também neste capítulo. Desta forma, para que o mecanismo seja capaz de atender os requisitos destacados no Capítulo 3 e o loop de adaptação MAPE-K, apresentado na Seção 2.2, os seguintes temas serão abordados e justificados:

- A Visão Lógica do Componente de Adaptação, de modo a proporcionar uma visão de alto nível sobre o funcionamento conceitual interno do mecanismo com base no modelo da Computação Autonômica (Seção 4.1);
- A descrição das formas de adaptação utilizadas: por componentes internos da UrboSenti (Seção 4.2) e por interação entre mecanismos de adaptação contidos em cada módulo (Seção 4.3);
- O protocolo de interação que define as diretivas de comunicação para permitir aos mecanismos de adaptação interagir entre si, adaptando-se ao ambiente externo (Seção 4.3.1);
- Um modelo de mundo que proporcione as informações sobre a estrutura do ambiente observável para suporte a adaptação dinâmica e para o processo de descoberta das informações do sistema (Seção 4.4);
- O algoritmo utilizado para autogerenciar os componentes internos, provendo adaptação em tempo de execução, considerando as interações, os contextos e a visão lógica do componente de adaptação (Seção 4.5).

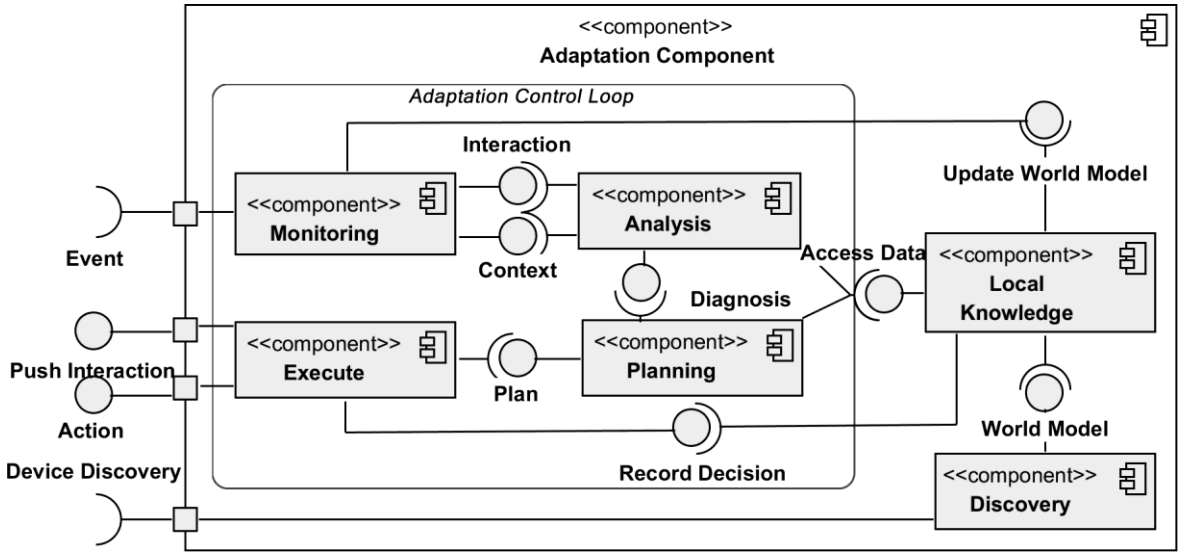
4.1 Visão Lógica do Componente de Adaptação

O modelo conceitual do mecanismo de adaptação toma por base o ciclo proposto pela IBM em 2001, que de acordo com Nallur e Bahsoon (2013) é o modelo mais utilizado pelos trabalhos autoadaptativos, o qual é composto por um ciclo monitorar-analisar-planejar-executar com *feedback* (denominado, MAPE-K). Tal ciclo expresso pela Figura 4.1 no campo *Adaptation Control Loop*, executado no componente de adaptação da UrboSenti. Contudo, para que o componente de adaptação esteja pronto para execução, uma etapa de descoberta de

informações deve ser realizada. Essa etapa consiste na descoberta de informações através da interface de descoberta (*DeviceDiscovery*) do Componente *Device* do módulo de sensoriamento, apresentada em ambas Figura 4.1 e Figura 4.2.

Tal processo é executado no componente *Discovery*, o qual irá coletar todas as informações sobre a estrutura do ambiente e salvá-las no componente *Local Knowledge* através da interface *World Model*. Somente após a geração deste modelo de mundo, as interfaces providas para acesso aos dados do ambiente (*Access Data*), o processo de atualizar o modelo de mundo (*Update World Model*) e o processo para armazenar resultado da execução das ações (*Record Decision*) podem ser utilizados.

Figura 4.1 - Diagrama conceitual do mecanismo de autoadaptação no Compontente de Adaptação



Fonte: Primária.

A partir do componente *Device* são descobertos quais componentes estão habilitados para funcionamento, quais entidades e seus respectivos estados devem ser monitorados, quais serviços, mecanismos de adaptação ou tipos de mecanismos existem e quais interações podem ser realizadas (maiores detalhes nas próximas seções). Além disso, também é indicado o descritor de que eventos e ações estes componentes possuem suporte, tal estrutura de conhecimento é tratada em maiores detalhes na Seção 4.4.

Uma vez que a etapa de descoberta for executada, o processo de adaptação pode ser executado. Esse processo inicia pelo componente *Monitoring*, o qual recebe eventos tanto síncronos como assíncronos pela interface *Event*, atualiza o modelo de mundo e classifica o tipo de evento entre dois: contextos internos (*Context*) gerados pelos componentes do módulo de sensoriamento ou um contexto de interação (*Interaction*) enviado por outro mecanismo de adaptação. Esta separação é necessária, pois o conteúdo interações é interpretado pelo próprio mecanismo de adaptação, portanto as informações devem ser extraídas e utilizadas pelos

componentes posteriores. Depois de identificado esses eventos, eles são passados para o componente *Analysis*.

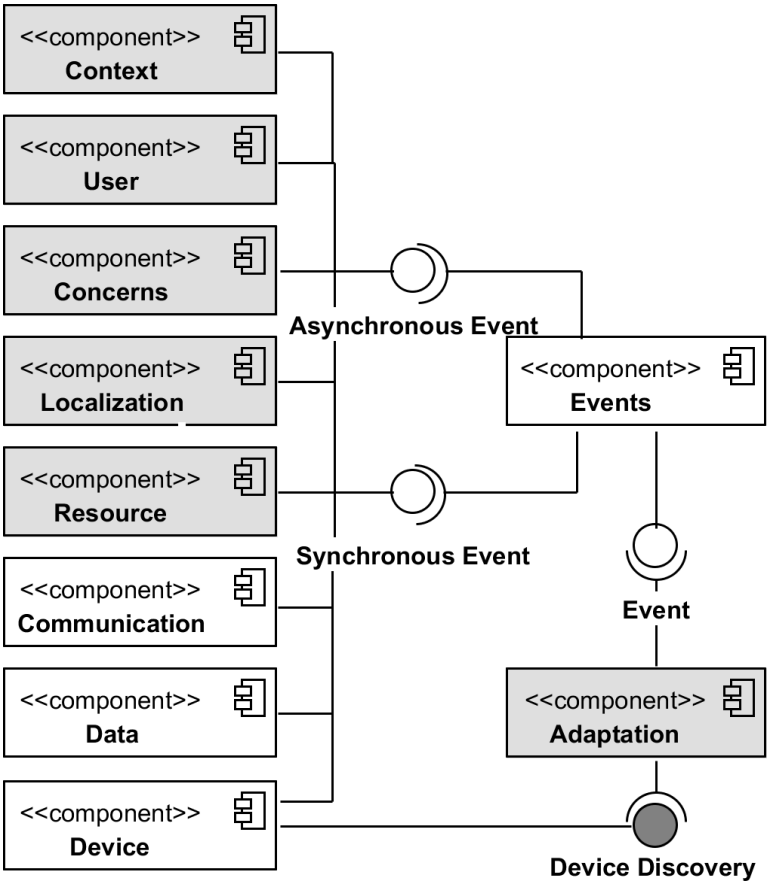
Analysis realiza o processo de diagnóstico identificando se as condições do novo contexto exigem uma ou mais mudanças. Para tanto, ele utiliza um modelo de diagnóstico definido pelo desenvolvedor, o qual pode acessar dados do componente *Knowledge* para aplicar técnicas (por exemplo, funções ECA, objetivo e de utilidade ou predição) para tratamento das interações e contextos recebidos pela interface, antes de passar o diagnóstico para o componente *Planning*. Tanto o componente *Planning* quanto o componente *Analysis* podem utilizar registros anteriores para suporte à tomada de decisão através do componente *Knowledge*. Cabe ressaltar que os modelos de diagnóstico e planejamento podem ser melhor visualizados nas pseudo regras utilizadas para representar os casos de autoadaptação implementados para avaliação, ambos encontrados na Seção 5.1.2 do próximo Capítulo.

Planning realiza o planejamento de quais ações devem ser tomadas, além de determinar as prioridades e resolver dependências, e, em seguida encaminhar o plano para o componente de execução. As possíveis dependências que podem ser encontradas são relativas a necessidades de interação com outros módulos, caso o valor deva ser alterado também em outros componentes ou entidades. Como uma interação com outro módulo pode demorar, o *Planning* pode definir limites para expiração do plano ou validações entre etapas do plano. Além disso, é possível nesta etapa definir uma sequência de ações necessárias para que o *Execute* aplique. *Execute* põe em prática o plano, encaminhando as devidas ações (*Action*) componentes internos do módulo de sensoriamento além de enviar mensagens pela interface *Push Interaction* para outros mecanismos de adaptação que devem interagir. Cabe ressaltar que, as *Push Interactions* são abstrações para ações aplicadas no componente de comunicação para envio da mensagem.

4.2 Processo de Adaptação nos Componentes Internos

O mecanismo de adaptação (Componente *Adaptation*) é responsável por fazer o processo de adaptação no módulo de sensoriamento da arquitetura UrboSenti. Contudo, para a arquitetura UrboSenti, ele também é um dos componentes “sob demanda”, assim como os componentes *User*, *Concerns*, *Localization*, *Resource* e *Context*, expressos com a cor cinza na Figura 4.2. Em contraste, para fins de representação do funcionamento do módulo, serão tratados nessa seção em conjunto os componentes do Micro-Kernel (*Device*, *Data*, *Communication* e *Events*) com a cor branca.

Figura 4.2 - Diagrama demonstrando a interface para geração de eventos e descoberta



Fonte: Primária.

Em suma, o componente de adaptação pode interagir com todos os componentes da arquitetura dependendo da necessidade do dispositivo que o está usando, exercendo a função de monitorar diversos aspectos que podem envolver um processo de adaptação ou reconfiguração em tempo de execução em nível de sistema. Por exemplo, um microcontrolador acoplado a um sensor que hipoteticamente monitora a temperatura de um ambiente, somente necessita dos componentes básicos para seu funcionamento. Em contraste, em um smartphone a aplicação necessitaria também do componente de usuário para gerenciar preferências de localização para dar suporte à atualização de localização e de adaptação para tratar questões voláteis e decorrentes da mobilidade.

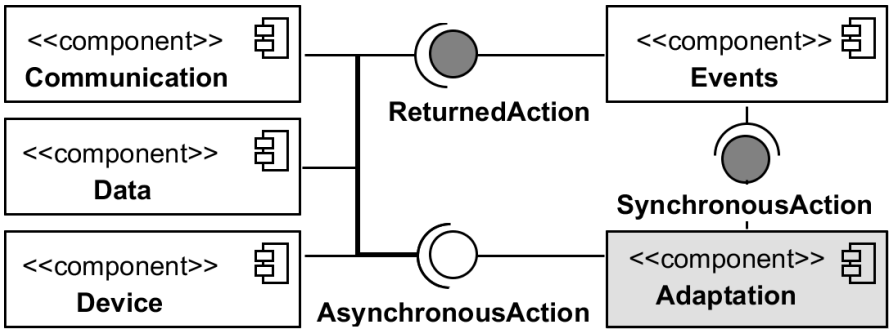
Para iniciar o processo de adaptação, cada componente monitorado deve possuir um monitor para identificar e gerar eventos, que são enviados para o componente *Events*. Tais eventos podem ser tanto síncronos (*Synchronous Event*) ou bloqueantes, como assíncronos (*Asynchronous Event*), que não são bloqueantes. Ambos os eventos são reencaminhados para o componente *Adaptation* através da interface *Event*. Contudo, caso este componente não tenha sido configurado ou desabilitado os eventos de sistema serão encaminhados normalmente para o tratador de eventos da aplicação.

O uso de eventos contextuais em mecanismos de adaptação como interface é uma prática comum em trabalhos de sistemas autoadaptativos como os encontrados em Floch et al (2006), Ayala, Amor e Fuentes (2013) e Dalpiaz, Giorgini e Muloupoylos (2013) e em trabalhos sobre sensoriamento urbano como o de Brouwers e Langendoen (2012), por serem facilmente escaláveis e representáveis. Além disso, o uso dessas técnicas aliadas a uma modelagem baseada em componentes permite o baixo acoplamento entre os módulos, fato que colabora com a flexibilidade da arquitetura, permitindo fácil adaptação para casos que tenham de utilizar desde somente o Micro-Kernel, até a totalidade dos componentes.

Contudo, antes de ser iniciado o processo de adaptação, é necessário um processo de descoberta de conhecimento. A descoberta inicia com a interface *DeviceDiscovery* (Preto) exibida na Figura 4.2, a qual tem por objetivo prover as informações de quais componentes estão ativos, quais devem ser monitorados e em que contextos, além da informação sobre o estado atual de cada um desses contextos.

Após a fase inicial, o mecanismo torna-se apto a receber eventos e tomar decisões sobre quais ações deve enviar aos componentes. Essas ações podem ser síncronas (cinza) ou assíncronas (branco), representadas na Figura 4.3. Para as ações assíncronas, ou não bloqueantes, como identificar um novo usuário ou a desconexão de um método de comunicação, o ajuste é feito diretamente nos componentes selecionados.

Figura 4.3 - Aplicação de ações nos componentes pelo componente de adaptação



Fonte: Primária.

Em outras palavras, essas ações são usadas quando os eventos gerados não necessitam de uma resposta imediata. Por exemplo, o evento informando desconexão de um método de conexão A não necessita intervenção, pois se sabe que o componente tentará se conectar automaticamente usando outro método B diretamente sem ser bloqueado. Por outro lado, através desses eventos pode ser identificado que o método B tornou-se, através dos registros, preferível ao método A, e então sua ordem de prioridade para escolha de conexão pode ser alterada (ou reconfigurada) pelo mecanismo de adaptação.

Em contraste, os eventos síncronos, ou bloqueantes, necessitam de uma resposta síncrona para continuar a executar a função, por exemplo, em casos em que o acesso a um recurso é restrito a um usuário e em casos que é importante interpor a chamada por questões de segurança. Desta forma, depois de tomada a decisão pelo mecanismo de adaptação a ação é enviada ao componente de eventos e em seguida retornada para o componente que iniciou o evento para efetuar o ajuste. O uso desta modelagem se justifica pelo fato dos componentes poderem ter diferentes requisitos de eventos e ações.

Para ilustrar, um exemplo de ciclo completo de adaptação síncrona seria: uma dada tarefa de sensoriamento remoto é enviada a um smartphone e faz a requisição do uso de dados de sensores, como o GPS e acelerômetro. Então o monitor identifica o evento “tentativa de acesso a sensor” e envia-o para o componente de eventos que o reencaminha para o componente de adaptação. Desta forma, o diagnóstico e a tomada de decisão são feitos levando em conta critérios tais como preferências do usuário, restrições do dispositivo e condições impostas por contextos de segurança, liberando o acesso ou não ao sensor naquele momento. Uma vez que a decisão é tomada, a ação resultante é enviada ao componente de evento que a reencaminha ao componente de origem do evento para tratamento da decisão.

Por fim, para que os eventos e ações sejam interpretados corretamente pelos componentes do módulo de sensoriamento, cada componente deve possuir internamente um ComponentManager. O ComponentManager de cada componente, para diferenciar dos demais, é denominado segundo seu nome mais “Manager”, desta forma ao se referir ao ComponentManager de *Location*, este seria LocationManager, para *Adaptation*, AdaptationManager, para *Events*, EventManager e assim por diante.

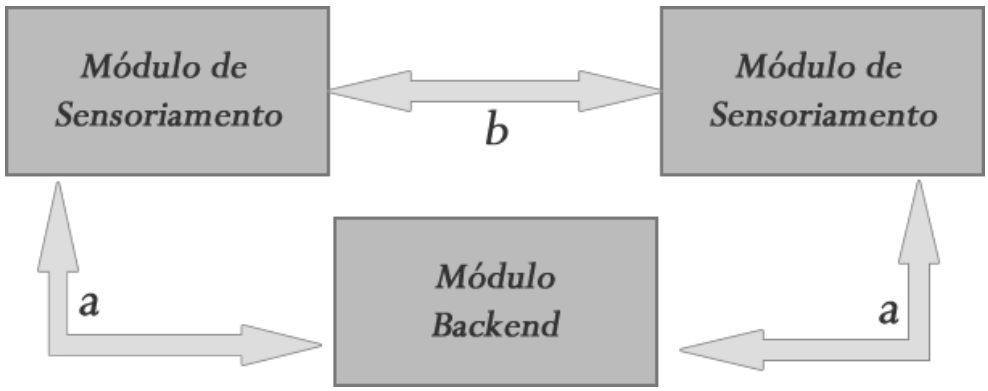
Cabe ressaltar que os eventos gerados pelos ComponentManagers podem ser feitos de duas formas, por um monitor estático ou por um monitor dinâmico Aqueles monitores que realizam monitoramento estático geram os eventos devido a causas fixas, tal como a geração de uma exceção sempre que um erro de função for detectado ou em intervalos de monitoramento imutáveis, por exemplo, os dados de consumo de memória em intervalos fixos de 2 minutos. Por outro lado, um monitor adaptativo permite que o próprio ato monitorar seja monitorado e parâmetros de monitoramento possam ser modificados visando uma melhor otimização, como no caso do LocationManager, que possui uma entidade que monitora a localização do dispositivo gerando um evento de nova localização toda vez que um novo dado de localização for identificado. Contudo, a sua frequência de atualização pode ser alterada por uma ação para fins de economia de energia. Outras entidades que realizam o monitoramento adaptativo seriam os sensores do dispositivo encontrados no ResoucesManager e a entidade

de agendamento de eventos no EventManager, utilizada para encadear tarefas agendadas para um próximo intervalo dado.

4.3 Processo de Adaptação por Interação entre Módulos

Além de realizar adaptações devido a eventos internos dos componentes de adaptação, um mecanismo de adaptação pode realizar adaptações resultadas de eventos causados por troca de mensagens com outras entidades de software. Esses eventos são denominado nesta seção como interação. As interações podem ser feitas de duas formas, como pode ser visto na Figura 4.4. Primeiramente, representado pela letra **a**, uma interação pode ocorrer entre um módulo de sensoriamento contendo um componente de adaptação ativado e uma entidade qualquer de software que conheça o protocolo de interação para poder interpretar e responder as mensagens corretamente. Desta forma, essa interação pode ocorrer mesmo com um módulo de sensoriamento, expresso como exemplo representado na mesma Figura.

Figura 4.4 - Exemplos de processos de interação entre módulos



Fonte: Primária.

Um exemplo de interação factível entre o módulo de sensoriamento com o módulo *backend* seria a função otimização de upload utilizada por Das et al. (2010), o qual compõe o Caso 3 – otimizar questões relacionadas com a rede – apresentado no capítulo passado. Essa função tem o objetivo de otimizar o período de upload de relatos coletados pelos nós sensores ao servidor, buscando melhorar a escalabilidade com uma grande quantidade de nós em largamente distribuídos de forma dinâmica. Em resumo o servidor identifica se a quantidade de nós que está fazendo upload está aumentando ou diminuindo e, para evitar maiores gargalos ou tempos de espera longos desnecessários, calcula uma nova probabilidade de realizar o upload com base nos números de nós e lhes envia a informação. Contudo, originalmente este função inicia com uma taxa de upload atribuída na inicialização do registro

do nó de sensoriamento, mas que pode ser facilmente migrada para aceitar novas taxas de upload em tempo de execução.

Por segundo, as interações podem ser feitas entre módulos de sensoriamento, representado pela letra **b** na Figura Figura 4.4, desde que ambos possuam o seu componente de adaptação habilitado. Apesar de nenhum dos trabalhos relacionados de sensoriamento urbano terem explorado tal interação em nível de sistema, uma vez que o protocolo de interação pode ser usado com outras entidades de softwares de terceiros todos os componentes de adaptação podem interpretar um ao outro, este tratado em maiores detalhes na Seção 4.3.1.

Uma interação que seria útil em um ambiente entre diversos módulos de sensoriamento seria o caso de utilizar os dados de monitoramento de recursos remotos que sejam mais vantajosos fazer utilizando os sensores do próprio dispositivo. Por exemplo, diversos passageiros estão viajando de ônibus com wi-fi de uma cidade para outra, onde cada passageiro utiliza um smartphone com um módulo de sensoriamento e o próprio ônibus possui um módulo de sensoriamento também em execução. Nesse cenário, todos os módulos de sensoriamento utilizam dados de localização por GPS, contudo, o uso contínuo GPS consome uma quantidade considerável de energia, o que seria ruim para os smartphones, pois, normalmente, não há locais no ônibus para carregar todos os celulares. Apesar disso, o ônibus não teria problemas em executar o GPS, pois além da bateria do ônibus possuir maior capacidade energética, o próprio ônibus aproveita o movimento para carregar continuamente a bateria. Desta forma, os dados de localização do ônibus serão utilizados de forma compartilhada com todos os smartphones, os quais poderiam eventualmente checar se o ônibus anda está em alcance, utilizando o GPS do dispositivo caso ambos tenham se separado.

Contudo, cabe ressaltar que independente da forma de interação, acessar os dados dos relatos dos usuários compartilhando-os com software de terceiros sem a permissão explicita do usuário pode causar uma falha grave de segurança, além de ser uma quebra nas regras de negócio do sensoriamento urbano, pois os relatos deveriam ser enviados unicamente para o módulo *backend*. Tal restrição deve ser usada, de acordo com Brouwers e Langendoen (2012), Ra et al. (2012) e Shin et al. (2011), devido a questões de segurança e privacidade relativos aos dados do usuário, que não devem ser violados sem consentimento.

4.3.1 Protocolo de Interação

Nos eventos de interação da arquitetura UrboSenti, o conteúdo da mensagem é encapsulado em formato XML e é interpretado antes do evento ser enfim processado. Esse

conteúdo, em destaque no Quadro 4.1, é encapsulado inicialmente pela *tag* `<content>`, a qual possui como primeiro elemento a *tag* `<interactionId>`, responsável por conter o identificador da interação descrita no modelo de mundo. O teor desta interação é por fim expresso dentro da *tag* `<message>` que contém primeiramente o formato do conteúdo na *tag* `<language>`, a qual é utilizado para interpretar o conteúdo presente no elemento `<content>`. Os conteúdos utilizam os parâmetros da interação descritos no modelo de mundo da Seção 4.4, os quais são expressos nas mensagens em formato chave-valor.

Assim como os eventos contextuais gerados pelos componentes internos, os eventos de interação provem o conjunto de primitivas de conhecimento comum entre os mecanismos de autoadaptação de um sistema. Tais primitivas permitem que ambos os mecanismos saibam interpretar o significado de cada mensagem utilizado para implementar o modelo de interação. Cabe ressaltar que o formato de mensagem utilizado neste modelo, no Quadro 4.1, é simplificado para atender os requisitos elencados, sendo fora do escopo atual deste trabalho o uso de outras linguagens ou formatos para representar o seu conteúdo.

Quadro 4.1 - Exemplo de conteúdo de mensagem

```
<content>
     <interactionId>1</interactionId>
     <message>
          <language>json</language>
          <content>{"chave":"valor"}</content>
     </message>
</content>
```

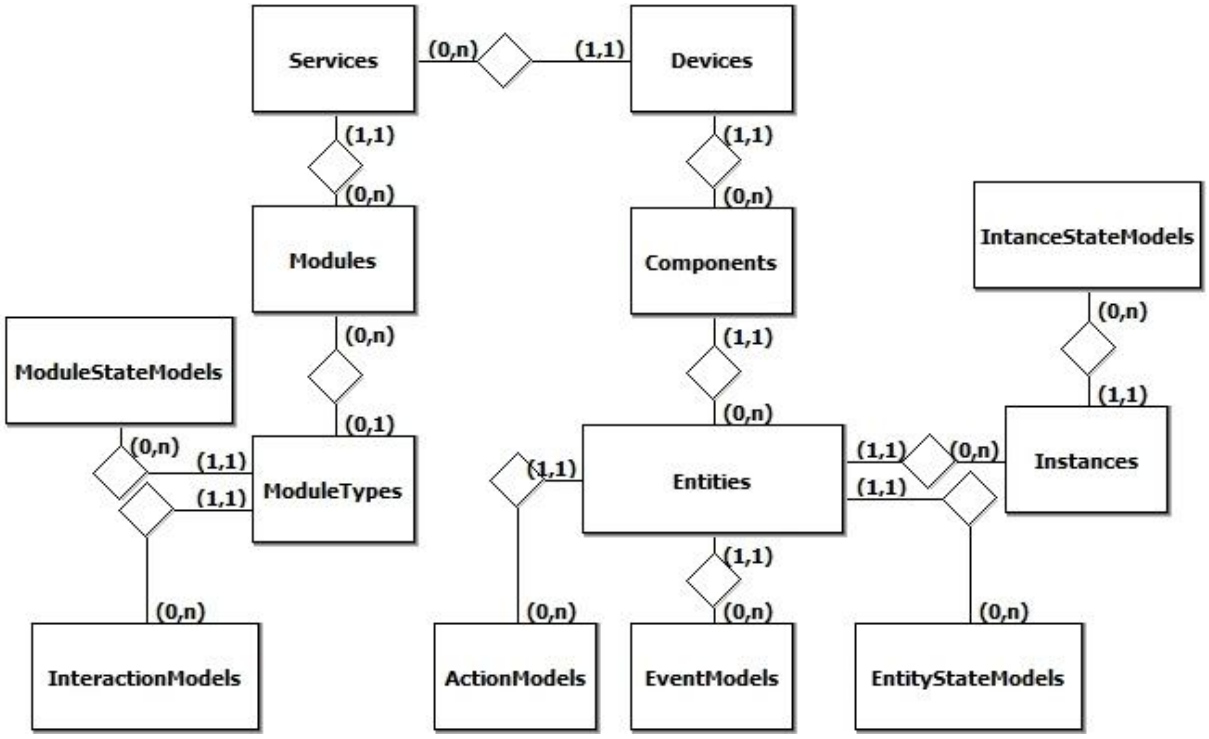
4.4 Modelo de Mundo

O modelo de mundo representa a modelagem do domínio, ou ambiente, que o mecanismo de adaptação irá monitorar e atuar. Esta modelagem de mundo representa toda a informação da estrutura do ambiente e seus dados transitórios, sendo acessada e atualizada através do componente *Local Knowledge*. Contudo, para que esse modelo de mundo esteja disponível para o uso, ele deve ser introduzido no mecanismo de adaptação através de um processo de descoberta de conhecimento no dispositivo, referente à interface *Discovery* encontrada no diagrama conceitual apresentado no início do capítulo pela Figura 4.1.

A partir do processo de descoberta realizado, o mecanismo de adaptação conhecerá o ambiente observável, sua estrutura e a forma como é possível interagir com ele. Esta estrutura é representada de forma simplificada no modelo entidade relacionamento (MER) na Figura

4.5. No MER apresentado é possível visualizar que o ambiente é primariamente formado por um dispositivo (entidade *Devices*) composto por componentes (*Components*) e serviços (*Services*). As entidades *Components* representam os *Component Managerns* componentes do módulo de sensoriamento da UrboSenti. Cada componente pode possuir diversas entidades (*Entities*), as quais são utilizadas para separar conceitualmente os diversos conjuntos de funções monitoradas e ações a serem aplicadas. Uma entidade pode ser uma função, um objeto ou serviço (autônomo) presente no ambiente interno do dispositivo, os quais podem possuir estados, descritos pela entidade *EntityStateModels*. Estes são alterados com base em eventos descritos pela entidade *EventModels* e as ações, que podem alterar tais parâmetros e requerer a execução de funções, descritas pela entidade *ActionModels*.

Figura 4.5 - Modelo de Mundo em formato relacional



Fonte: Primária.

Além disso, todo registro em *Entities* pode possuir instâncias (*Instances*) que são descritores de estados particulares, com as mesmas características dos descritores de estados *EntityStateModels* declarados para a entidade, denominados de estados de instância (*InstanceStateModels*). Para melhor ilustrar, instâncias podem ser, por exemplo, tipos diferentes de comunicação em uma entidade do tipo interface que possuem os mesmos modelos de estados e utilizam os mesmos eventos e ações para acesso. O mesmo seria para novos usuários representados em uma entidade *Entities*, onde cada usuário seria uma nova instância para o modelo de mundo. Desta forma, o uso de instâncias se justifica pelo fato de

ser desnecessário criar uma nova entidade para cada instância, uma vez que cada instância da mesma entidade possui os mesmos eventos, ações e estados, diferindo somente no conteúdo.

Em adicional, cada entidade do MER utilizada para interação com o ambiente (*ActionModels*, *EventModels* e *InteractionModels*) pode ter nenhum ou diversos parâmetros que descrevem o valor que possuem como opcional ou obrigatório. Estes parâmetros podem ser relacionados a um modelo de estado (*EntityStateModels* e *ModuleStateModels*) indicando as mudanças de valores deles, além disso, os parâmetros, em tempo de execução, são identificados em formato rótulo-valor. Tanto os modelos de estado como os parâmetros citados possuem tipos de dados e suas alterações são salvas conservando o valor e o tempo anterior para serem utilizados pela análise do componente de adaptação.

Para melhor expressar uma modelagem com eventos internos: em dado um dispositivo móvel com o módulo de sensoriamento da UrboSenti existe um componente chamado *Communication*, o qual possui a entidade do tipo função denominado de "função de reconexão". Tal função possui um estado que representa o intervalo utilizado entre as tentativas de conexão e dois eventos, um para sucesso de reconexão em uma interface e outro para indicar falha de tentativa de reconexão. Como todos os estados anteriores de tentativas de reconexão são salvos, uma análise desses dados, após agregar uma quantidade relevante de dados, pode encontrar intervalos de reconexão que seriam mais eficientes em um dado período de tempo. Desta forma, seria possível definir um intervalo de tentativa de reconexão mais longo em áreas que geralmente não há conexão com o objetivo de evitar o uso de recursos do dispositivo móvel. Para tanto uma ação de alteração de tal valor seria necessária.

Esse intervalo de reconexão, se considerar como sendo em milissegundos, seria representado por um estado com tipo de dado inteiro. Além disso, a ação para alterar o intervalo usaria um parâmetro também inteiro relacionado com o estado de intervalo de reconexão, além de ter uma chave chamada "*interval*". Tal descrição, com base no MER citado anteriormente, pode ser abstraída da seguinte estrutura em um dispositivo móvel:

- Entidade *Components*: Communication;
    - Entidade *Entities*: Função de reconexão;
        - Entidade *EntityStateModels*: Intervalo de reconexão:
            - Tipo de dado: Inteiro.
        - Entidade *ActionModels*: Alterar intervalo de reconexão:
            - Rótulo: interval;
            - Tipo de dado: Inteiro;

- Entidade *EventModels* Relacionada: Intervalo de reconexão.
- Entidade *EventModels*: Falha ao tentar reconectar;
- Entidade *EventModels*: Sucesso ao tentar reconectar.

A entidade *Services* representa todos os serviços remotos com os quais o módulo de sensoriamento pode interagir para realizar adaptações. Sempre que um serviço permite eventos de interação, este possui um ou mais módulos (*Modules*), cada um especificado por um tipo de módulo (*ModuleTypes*). Cada registro em *ModuleTypes* define quais interações esses módulos podem realizar tanto de saída como de entrada, bem como os estados observáveis destes, de forma semelhante aos componentes. Essas interações e estados são representados pelas respectivas entidades *InteractionModels* e *ModuleStateModels*.

Por fim, o resultado das interações são mensagens compostas de parâmetros representados por rótulo-valor, tempo da geração da interação, origem e destino. Um exemplo de interação seria: o módulo A*i* propõe um novo valor 15 ao estado *x* do módulo A*j*, contudo A*j* considera 15 um valor muito baixo comparado com o estado atual que é 30. Este então envia uma contraproposta a A*i* no valor de 22, a qual o módulo A*i* aceita. Assim, quatro mensagens foram trocadas para decidir o valor para adaptação. Tal descrição utilizando o MER, apresentado anteriormente, pode ser abstraída da seguinte forma a partir do módulo A*i*:

- Entidade *ModuleTypes*: Módulo *Backend*;
  - Entidade *ModuleStateModels*: Valor:
    - Tipo de dado: Inteiro.
  - Entidade *InteractionModels*: Propor valor:
    - Rótulo: value;
    - Tipo de dado: Inteiro;
    - Sentido: Saída;
    - Entidade *ModuleStateModels* Relacionada: Valor.
  - Entidade *InteractionModels*: Receber proposta de valor:
    - Rótulo: value;
    - Tipo de dado: Inteiro;
    - Sentido: Entrada;
    - Entidade *ModuleStateModels* Relacionada: Valor.

4.5 Algoritmo de Adaptação

Após a execução do processo de descoberta, o componente de adaptação está apto a iniciar a execução do algoritmo de adaptação expresso pelo ciclo de controle encontrado no Algoritmo 1. Este algoritmo é resultado da interpretação do modelo MAPE-K apresentado anteriormente na Seção 4.1, a qual é referente ao conjunto de componentes que compõem o *Adaptation Control Loop* da visão lógica do componente de adaptação. O componente Monitoring é representado pela função UpdateWorldModel (linha 8 e 9), o componente Analysis é representado pela função Analysis (linha 10), o componente de Planning é representado pela função Planning (linha 11) e o componente Execute é representado pelas funções executadas entre a linha 12 e a 15. Todos esses componente executam em um ciclo infinito.

ALGORITIMO 1: ADAPTATION CONTROL LOOP

```
1.  Diagnosis d;
2.  Plan p;
3.  Result r;
4.  DiagnosisModel dm ← dm0;        /* dm0 are the rules of analysis */
5.  PlanningModel pm ← pm0;         /* pm0 are the known plans  */
6.  LocalKnowledge lk ← lk0;        /* lk0 is the knowledge about the environment */
7.  while true do
8.      get next Event en;
9.      UPDATEWORLDMODEL(en, lk);
10.     d ← ANALYSIS(en, dm, lk);
11.     p ← PLANNING(d, dm, pm, lk);
12.     if (NEEDTODOSOMETHING(p)) do
13.         r ← EXECUTE (p);                    /* Execute Process */
14.         RECORDDECISION (r, en, p, lk);      /* Update Knowledge */
15.     end if
16. end while
```

Fonte: Primária.

As três primeiras variáveis apresentadas no algoritmo (linhas 1 a 3), Diagnosis ***d***, Plan ***p*** e Result ***r***, representam, respectivamente: o resultado da etapa de análise; o resultado da fase de planejamento; e o resultado do processo de execução. Por outro lado, as variáveis declaradas nas linhas 4 e 5, DiagnosisModel ***dm*** e PlanningModel ***pm***, são definidos atualmente em tempo de execução. As respectivas funções delas no algoritmo são: representar as regras de diagnóstico definidas para execução na fase de análise e representar os planos conhecidos pelo componente de adaptação para a fase de planejamento.

A variável LocalKnowledge (declarada na linha 6) representa todo o conhecimento descoberto pelo componente de adaptação durante o processo de descoberta, além de todos os

estados alterados em tempo de execução, o que inclui tanto os valores atuais como passados. Este é sempre utilizado nos algoritmos desta seção para representar que a base de dados está sendo acessada para leitura ou escrita. Após as declarações das variáveis, na linha 7, o ciclo de adaptação é iniciado ao receber pelo processo de monitoramento um evento ***en*.** Este evento é submetido a seguinte sequência de passos da linha 9 até a 15:

- Na linha 9, o evento ***en*** e o conhecimento local ***lk*** são utilizados para atualizar o modelo de mundo. Este representado através do Algoritmo 2;
- Na linha 10, através de ***en***, ***lk*** e o modelo de diagnóstico ***dm***, o processo de análise - detalhado no Algoritmo 3 - é executado tendo como resultado um diagnóstico ***d***;
- Na linha 11, ***d***, ***lk***, ***dm*** e o modelo de planejamento ***pm*** são submetidos a um planejamento, resultando em um plano ***p***. Este detalhado pelo Algoritmo 4;
- Na linha 12, o plano resultante em ***p*** é analisado verificando se há necessidade de executar tarefas, caso necessite, dois processos são executados:
    - Na linha 13: ***p*** é passado para o processo de execução, tendo o resultado da aplicação das ações armazenado em ***r***. Este processo é apresentado no Algoritmo 5;
    - Por fim, na linha 14, é atualizado no conhecimento local ***lk*** o resultado de ***r*** em virtude do evento ***en*** e do plano ***p***. Maiores detalhes são encontrados no Algoritmo 6.

O Algoritmo 2 trata da função de atualização do modelo de mundo, que representa o conhecimento do ambiente. Inicialmente a variável ***p*** (linha 1) é definida para representar os parâmetros de cada ação executada no ambiente. Em sequência, na linha 3, é identificada a origem do evento que pode ser de outro módulo ou de um componente interno. Se módulo, é extraída a mensagem de ***en*** utilizando descrição do modelo de interação em ***lk*** atribuindo o conteúdo nos parâmetros do próprio ***en***. Na linha 5, é analisada a necessidade de armazenar o evento através da descrição do evento no ***lk***.

Para cada parâmetro em ***en*** atribuído em ***p*** (na linha 7) o ciclo da linha 8 a 16 é executado. Inicialmente, na linha 8 é feita a verificação quanto à existência de relação a algum modelo de estado e, em caso afirmativo, este estado é atualizado em ***lk***. Um exemplo deste caso seria o nível de bateria atual, que está relacionado diretamente ao estado monitorado de nível de bateria. Por fim, se o estado foi atualizado é feito o processo de atualizar o conteúdo referente à fonte do evento, caso exista uma relação, entre um estado de instância (linha 9), um estado monitorado de outro módulo (linha 12), ou um estado de entidade (linha 14).

ALGORITIMO 2: UPDATE WORLD MODEL

```
    UPDATEWORLDMODEL(en, lk)
1.  Parameter p;
2.  begin function
3.      if (ISINTERACTIONEVENT (en)) do
4.          en ← EXTRACTINTERACTIONMESSAGE(en, lk);
5.      if (ISNECESSARYSTORE(en, lk)) do
6.          STOREEVENT(en, lk);
7.      foreach p from en.parameters do
8.          if (ISRELATEDSOMESTATE(p, lk)) do
9.              if (isFROMINSTANCE(en, lk)) do
10.                 STOREINSTANCESTATEVALUE(p, lk);
11.             else if (isFROMMODULE(en, lk)) do
12.                 STOREMODULESTATEVALUE(p, lk);
13.             else
14.                 STOREENTITYSTATEVALUE(p, lk);
15.             end if
16.         end if
17.     end for
18. end function
```

Fonte: Primária.

O Algoritmo 3 executa o processo de análise do evento. Esse processo de análise, como citado anteriormente, é feito através de regras genéricas que podem ser substituídas por técnicas específicas (ECA, utilidade, objetivo ou predição), onde todas estão representadas dentro do modelo de diagnóstico ***dm***, passado por parâmetro. Para tanto, as seguintes variáveis são definidas (linhas 1 a 5): ***d*** é o diagnóstico resultante da análise, o qual será retornado para o ciclo de controle; ***interaction*** é o modelo de interação que descreve a estrutura de interação do evento de interação; ***context*** é o modelo de evento que descreve a estrutura do evento contextual recebido; ***data*** é a variável auxiliar para conter os dados utilizados no processamento das regras e ***rule*** é a variável auxiliar para uso das regras. Cabe ressaltar que as variáveis ***interaction*** e ***context*** são referentes às interfaces com os mesmos nomes entre os componentes Monitoring e Analysis presentes no diagrama conceitual da Figura 4.1.

Primeiramente, é verificada a origem do evento que pode ser de outro módulo (linhas 8 a 12) ou de um componente interno (linhas 15 a 20). Se a origem for de um módulo, os seguintes passos são aplicados:

- Na linha 9, é retornado do conhecimento local ***lk***, o modelo de interação que representa a descrição completa do evento de interação em ***interaction***.

- Na linha 10, é filtrado entre todas as regras no modelo de diagnóstico ***dm*** as que correspondem ao modelo de interação identificado em ***interaction.*** Cada uma dessas regras é atribuída para a variável ***rule*** em cada iteração do laço;
- Para cada ***rule*** é retornado os dados em ***lk*** necessários para processamento das condições das regras (linha 10) e, na linha 11, verificada se a regra necessita de uma adaptação com base nas informações descobertas. Caso as condições da regra sejam alcançadas, estas implicarão em uma ou mais mudanças, assim, na linha 12, essas mudanças são adicionadas no diagnostico ***d*** para serem acessadas pelo processo de planejamento;

ALGORITIMO 3: ANALYSIS PROCESS

```
    ANALYSIS(Event en, DiagnosisModel dm, LocalKnowledge lk)
1.  Diagnosis d;
2.  InteractionModel interaction;
3.  EventModel context;
4.  Rule rule;
5.  GeneralData data;
6.  begin function
7.      if (ISINTERACTIONEVENT(en)) do
8.          interaction ← GETINTERACTIONMODEL(en, lk);
9.          foreach rule ← FILTEREDRULES (dm.rules, interaction) do
10.             data ← GETREQUIREDDATA(rule, lk);
11.             if (REQUIRESADAPTATION(interaction, rule.conditions, data, en)) do
12.                 d.change += rule.changes;
13.         end for
14.     else
15.         context ← GETEVENTMODEL(en, lk);
16.         foreach rule ← FILTEREDRULES(dm.rules, context) do
17.             data ← GETREQUIREDDATA(rule, lk);
18.             if (REQUIRESADAPTATION(context, rule.conditions, data, en)) do
19.                 d.change += rule.changes;
20.         end for
21.     end if
22.     return d;
23. end function
```

Fonte: Primária.

Caso a origem do evento ***en*** seja um componente interno um processo similar ao descrito acima é aplicado. A principal diferença consiste que ao invés de retornar um modelo de interação, como na linha 8, é retorna do um modelo de evento em ***context*** para descrever a estrutura do evento. Desta forma, o modelo ***context*** retornado da função da linha 15 é utilizado das linhas 16 até 20. Ao fim do processamento de todas as regras, o diagnóstico ***d*** é retornado na linha 22.

O processo de planejamento de mudanças no componente autogerenciado é expresso de forma abstrata pelo Algoritmo 4. Os parâmetros da função são o diagnostico ***d***, o modelo de planejamento implementado em ***pm***¸ e o conhecimento expresso pela base de dados é dado por ***lk***. Inicialmente, são definidas as variáveis: ***p*** que é o plano a ser retornado para o algoritmo 1 e ***change*** que é a variável temporária referente a cada mudança que será processada.

Na linha 4 é verificado se diagnóstico ***d*** exige alguma mudança no ambiente, checando se ele está vazio. Caso não seja necessária uma mudança, atribui ao plano ***p*** o valor usado para indicar que não é necessário realizar nenhuma ação e o retorna (linhas 5 e 6). Caso contrário, cada mudança contida no diagnóstico ***d*** é atribuída em ***change*** (na linha 8) para ser utilizada no processo entre as linhas 9 a 13.

Na linha 9 é identificado que tipo de ação a mudança irá resultar entre uma ação estática ou uma dinâmica. Se estática, é adicionado diretamente o plano de execução de ações na estrutura que contém os planos selecionados, conforme a linha 10. Se dinâmica (linha 12) é escolhido o melhor plano de execução de ação dado o contexto. Ao fim é retornado o plano com todos os planos de ação necessários para executar as mudanças no ambiente.

ALGORITIMO 4: PLANNING CHANGES

```
    PLANNING(Diagnosis d, DiagnosisModel dm, PlanModel pm, LocalKnowledge lk)
1.  Plan p;
2.  Change change;
3.  begin function
4.      if (ISEMPTY(d.changes)) do
5.          p ← DONOTHING();
6.          return p;
7.      end if
8.      foreach change from d.changes do
9.          if(ISSTATICACTION(change, pm)) do
10.             p.actionPlans += GETSTATICEXECUTIONPLAN(change, lk, pm);
11.         else
12.             p.actionPlans  += GETDYNAMICEXECUTIONPLAN(pm, change, lk, dm);
13.         end if else
14.     end for
15.     return p;
16. end function
```

Fonte: Primária.

O algoritmo de execução é exposto no Algoritmo 5. Primeiramente, são declaradas as variáveis (linhas 1 a 3): ***action*** representa a ação a ser executada, ***r*** contém o resultado de todas as ações e ***actionPlan*** contém a sequência de ações de um plano a serem aplicadas. Na lina 5, são extraídos os planos de ação a partir do plano ***p***, os quais são filas contendo ações relacionadas com o plano.

ALGORITIMO 5: EXECUTION

```
    EXECUTE (p)
1.  Action action;
2.  Result r;
3.  ActionPlan actionPlan;
4.  begin function
5.      foreach actionPlan from p.actionPlans do
6.          foreach action from p.actions do
7.              if (TARGET(action, module)) do
8.                  action ← MAKEINTERACTIONMESSAGE(action, lk);
9.                  r += PUSHINTERACTION(action, action.targetAddress);
10.             else
11.                 r+= APPLYACTION(action, action.componentManager);
12.             end if
13.         end for
14.     end for
15.     return r;
16. end function
```

Fonte: Primária.

Desta forma, para cada ação ***action***, de um plano de ação, é executado o processo da linha 7 a 12. Se esta ação tiver como destino um módulo, através do teste na linha 7, então a ação é tratada como uma interação, sendo atribuído em parâmetros de ***action***: a mensagem gerada no formato suportado, a informação do endereço de destino e o tipo de ação (linha 8). Após, na linha 9, a ação resultante é enviada para o serviço de upload responsável por enviar a mensagem desse módulo para o seu servidor utilizando o endereço descrito pela ação. Contudo, se a ação estiver especificada como do tipo síncrona, esta é enviada diretamente ao módulo de destino, dispensando o intermédio do serviço de upload.

Por outro lado, caso o destino desta ação ***action*** seja um componente interno, ela é enviada ao ComponentManager respectivo que tratará da execução (linha 11). Cabe ressaltar que caso a ação seja assíncrona, ela é enviada diretamente para o ComponentManager que executará a adaptação. Se síncrona, ela é enviada para o componente Event que retornará como resultado a ação para o processo que executou o evento dentro do intervalo de *timeout* estabelecido. Após a execução de ambas as funções (linhas 9 e 11), o resultado de cada ação é atribuído em ***r***, o qual pode ser: um sucesso na execução da função ou uma falha (por *timeout*, se síncrona, ou por falha geral de execução em caso contrário), se por algum motivo não tiver sido possível executar com sucesso.

Por fim, o Algoritmo 6 demonstra como o resultado das decisões é armazenado no conhecimento local. Inicialmente a variável ***parameter*** (linha 1) é definida para representar os parâmetros de cada ação executada no ambiente. Para cada ação executada pelo plano (linha

3), é salvo o resultado na linha 4, composto pelo resultado retornado da aplicação da ação ***r***, o plano utilizado ***p***, o evento ***en***, a referência do conhecimento local ***lk*** e ação executada ***action***.

Após, se o resultado ***r*** retornado na aplicação da ação for bem sucedido, cada ação é submetida a um teste verificando quais parâmetros da ação são relacionados a um estado. Para cada parâmetro em ***parameter*** que tenha sido bem sucedido em alterar o estado relacionado, seu valor é atualizado no modelo de mundo. Da linha 7 até 14, um processo semelhante ao Algoritmo 2, update world model, é executado, onde o alvo da ação é testado entre estado de instância, módulo ou entidade, persistindo o novo valor bem sucedido no modelo de mundo.

ALGORITIMO 6: RECORD DECISION

```
    recordDecision (Result r, Event en, Plan p, LocalKnowledge lk);
1.  Parameter parameter;
2.  begin function
3.      foreach action from p.actions do
4.          storeActionResult(r, p, en, lk, action);
5.          foreach parameter from action.parameters do
6.              if (isRelatedSomeState(parameter, lk) & successResulted(r)) do
7.                  if (isFromInstance(action, lk)) do
8.                      storeInstanceStateValue(parameter, lk);
9.                  else
10.                     if (isFromModule(action, lk)) do
11.                         storeModuleStateValue(parameter, lk);
12.                     else
13.                         storeEntityStateValue(parameter, lk);
14.                 end if
15.             end if
16.         end for
17.     end for
18. end function
```

Fonte: Primária.

5 IMPLEMENTAÇÃO E AVALIAÇÃO

O presente capítulo tem por objetivo apresentar a implementação e a avaliação efetuadas para validação preliminar da proposta. O cenário de aplicação utiliza por base a UrboSenti, destacando quais componentes do módulo de sensoriamento e do módulo *backend* foram implementados. Na sequência são apresentadas as avaliações e os experimentos executados, contendo uma discussão dos resultados encontrados. Em suma, o principal foco desta avaliação é verificar o impacto que a inclusão da infraestrutura ao adicionar a propriedade autonômica no módulo de sensoriamento.

5.1 Implementação

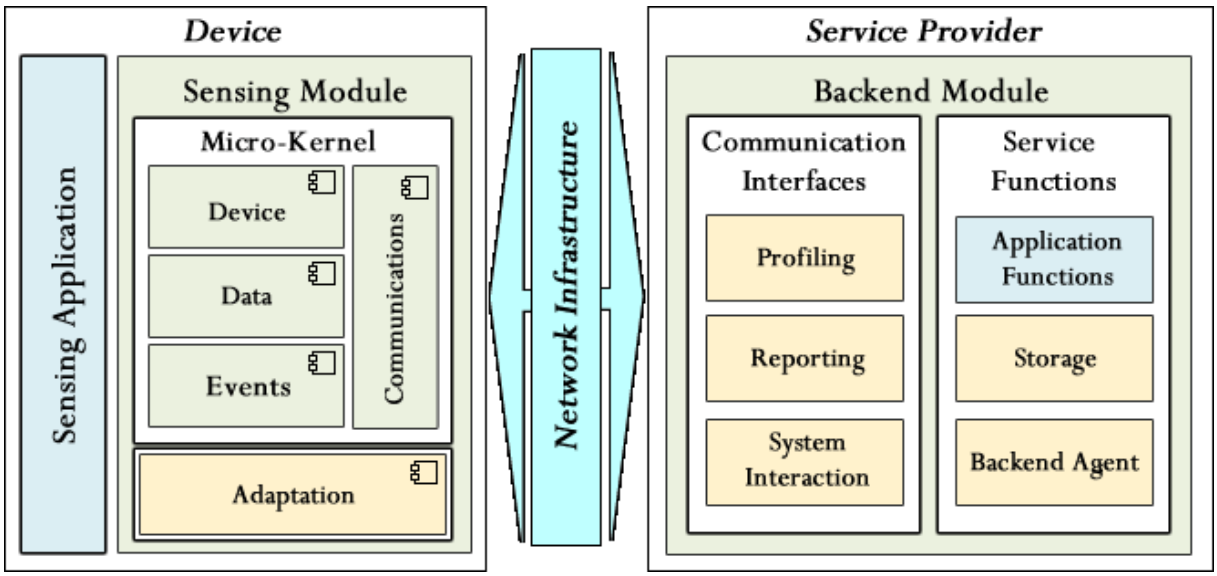
A arquitetura UrboSenti possui flexibilidade para permitir o uso dos componentes que somente são necessários para o funcionamento da aplicação de sensoriamento. Desta forma, nesta seção são explicados quais elementos da UrboSenti foram desenvolvidos para avaliação, com o principal objetivo de validar a proposta do mecanismo de autoadaptação.

O cenário proposto foi implementado para dispositivos de telefonia móvel com suporte ao sistema operacional Android e para desktop com Java SE (Standard Edition) com o objetivo de dar suporte a mais de um tipo de dispositivo. Estas implementações necessitaram somente do Micro-Kernel em conjunto com o componente sob demanda de adaptação do módulo de sensoriamento para validação da proposta. O desenvolvimento de outros componentes e a portabilidade deles para outros dispositivos foi condicionado para trabalhos futuros, uma vez que o desenvolvimento de tais elementos não era necessário para esta validação preliminar, além do fato de que consumiriam um longo tempo de desenvolvimento. Os módulos e componentes implementados são melhor ilustrados pela Figura 5.1.

Primeiramente, a figura apresenta o cenário proposto para avaliação, com base na arquitetura UrboSenti. A arquitetura é composta por dois grandes módulos: *backend* e de sensoriamento. No topo do módulo de sensoriamento é executada uma aplicação de sensoriamento simulada que faz uso das funcionalidades básicas dos componentes Micro-Kernel e de adaptação (*Adaptation* na Figura). Para o módulo *backend* foram desenvolvidas somente as interfaces providas pelo serviço para as operações básicas de sistema e aplicação, como interfaces para registro do módulo de sensoriamento, armazenamento de relatos e interação com os mecanismos de autoadaptação do sistema. A interface entre os módulos é

dada por uma infraestrutura de rede e troca de mensagens. Tais detalhes são abordados nas próximas subseções.

Figura 5.1 - Elementos implementados da arquitetura UrboSenti



Fonte: Primária.

5.1.1 Módulo de Sensoriamento

O módulo de sensoriamento é responsável por fornecer diferentes componentes e serviços com objetivo de prover suporte à aplicação de sensoriamento. Contudo, para a validação do mecanismo de adaptação somente os componentes do Micro-Kernel, *Device*, *Data*, *Events* e *Communication* necessitaram ser implementados. Cada um dos componentes do Micro-Kernel é responsável por executar as seguintes funções, na implementação atual:

- *Device*: O componente *Device* é o gerente do módulo de sensoriamento responsável por prover as informações básicas sobre o dispositivo (componentes habilitados, modelo de dispositivo, serviços ativos, capacidade de bateria, etc.). Desta forma, através dele os serviços podem ser acessados e executados, e os demais componentes sob demanda podem ser instanciados e habilitados. No caso aqui apresentado somente o componente de adaptação foi habilitado. Além disso, são executadas nesse componente as operações básicas para registro do módulo de sensoriamento no servidor, a configuração inicial de todos os componentes e as interfaces de comunicação, além do próprio processo de descoberta do componente de adaptação;

- *Data*: Responsável por realizar as operações para armazenar e recuperar dados. Entre essas operações está a criação do banco de dados relacional local utilizado por todo o módulo de sensoriamento para guardar configurações atuais e passadas, mensagens, além das informações referentes às operações para acesso ao *Knowledge* do componente de adaptação. A ferramenta para armazenamento utilizada foi a SQLite;
- *Events*: captura eventos de interesse, gerados pelos componentes, disponibilizando-os para serem utilizados por outros componentes que os requeiram. Os eventos de interesse do sistema (como desconexão ou troca de interface de comunicação) são encaminhados diretamente para o componente de adaptação que poderá tratar devidamente esses eventos. Para esse encaminhamento ser possível, cada tratador de eventos deve implementar uma interface utilizada para receber os eventos e assinar o recebimento dos eventos no componente *Events* através do padrão de projeto *Observer*, possibilitando uma operação por *Publish/Subscribe*. Para a avaliação realiza, somente foi necessário implementar eventos assíncronos, contudo o suporte a eventos síncronos já é suportado. Além disso, este componente proporciona o agendamento de eventos dinâmicos temporais necessários para a realização de gatilhos que são executados em dado tempo ou intervalo específico;
- *Communication*: proporciona os métodos para enviar e receber dados utilizando a estrutura de rede disponível para a aplicação e aos demais componentes. Na implementação atual somente tem suporte as interfaces de comunicação cabeada, Wi-Fi e de redes móveis (Por exemplo, 2G, 3G e 4G) sobre os métodos de comunicação *Socket* e *HTTP*. Além disso, oferece tratamento de desconexão que inclui um serviço de reconexão e armazenamento temporário dos relatos que não puderam ser enviados devido à desconexão. Para que estes relatos sejam enviados depois de retomada a comunicação, um segundo serviço, denominado serviço de *upload*, é responsável for fazer o *upload* desses relatos para o servidor *backend*.

O componente de adaptação, então, é o único componente sob demanda escolhido para ser implementado para avaliação nesse capítulo. Ele toma por base a representação conceitual introduzida na Seção 4.1 do capítulo anterior. A implementação dos elementos internos que compõem este modelo de mecanismo é abordada a seguir.

O processo de descoberta utilizado pelo mecanismo de adaptação para descobrir o modelo de mundo e que elementos podem ser manipulados é chamado *Discovery*, sendo

executado pelo componente *Device*. O componente *Knowledge* é representado pelo componente *Data* do módulo de sensoriamento, sendo este responsável por possibilitar o acesso às informações e a manipulação das mesmas. De forma semelhante, o componente *Communication* realiza o papel de enviar e receber as mensagens de interação com outros mecanismos de adaptação, gerando um evento de interação quando as recebe e enviando a mensagem de interação quando especificado por uma ação.

Por fim, a função central para tomada de decisão de forma autonômica é realizada pelo Loop de Adaptação presente no componente de adaptação da arquitetura UrboSenti. Tal loop é executado em um fluxo de execução próprio acessando os eventos por uma fila com acesso sincronizado para exclusão mútua. No processo de adaptação toma por base o algoritmo apresentado na Seção 4.5. O processo de monitoramento utilizado pelo loop é passivo, tal qual descrito no capítulo anterior, fazendo uso do componente *Data* para acesso à base de dados (*Knowledge* do modelo MAPE-K). Na implementação atual os modelos de diagnóstico e planejamento são implementados na forma de classes, os quais possuem uma implementação padrão descrita nesta dissertação na Seção 5.1.2. Tais classes podem ser substituídos por objetos instanciados por classes equivalentes os quais foram implementadas pelo programador do sistema de sensoriamento para adicionar novas regras e planos durante a etapa de configuração do middleware.

O modelo de análise utilizado nesta implementação toma por base o modelo de regras *Evento-Condição-Ação (ECA)* realizadas em sua forma mais simples, através de *if-else*, uma vez que os casos expostos na próxima seção não exigiram uma técnica mais robusta, pois o foco da avaliação é o suporte a autoadaptação para sensoriamento urbano e seu impacto na infraestrutura. Para o processo de planejamento foram utilizados somente planos estáticos devido à simplicidade dos casos implementados. Cabe ressaltar que o estudo da aplicação em outros casos de técnicas mais robustas para análise e planejamento dinâmico está aberto para trabalhos futuros, os quais adicionarão mais uso dos recursos que somente as técnicas atualmente utilizadas. O processo de execução funciona tal qual apresentado no algoritmo 5 do capítulo anterior, com exceção das funções de aplicação das ações que são de acesso sincronizado, isto é, com exclusão mútua devido ao caráter concorrente da implementação.

5.1.2 Casos de Adaptação

Para fins de avaliação da proposta somente os componentes do Micro-Kernel (*Device*, *Communication*, *Data* e *Event*) e o componente sob demanda *Adaptation* foram

implementados, devido a maior parte dos componentes sob demanda necessitar de mais tempo de pesquisa para serem desenvolvidas, além de estarem fora do escopo desta dissertação. Desta forma, entre os sete casos de adaptação elencados na Seção 3.2, onde foi feita a análise de trabalhos de sensoriamento urbano, os seguintes casos foram implementados para avaliação, sendo abordados nas próximas seções:

- (1) Autoconfiguração sem a participação do usuário;
- (3) Identificar e tratar problemas nos módulos ou serviços;
- (4) Otimização do upload de relatos;
- (5) Tratamento da mobilidade e desconexão.

Assim, os demais casos: (2) Autoadaptação com a interação do usuário, (6) Controlar o acesso a recursos no *runtime* do dispositivo e (7) Otimização do monitoramento dos recursos internos não foram implementados por exigirem o uso de componentes sob demanda que não foram ainda implementados na atual versão da UrboSenti. Apesar disso, tal fato não invalida a avaliação realizada, uma vez que a infraestrutura para aplicação das ações e monitoramento não serão modificados quando os componentes e casos de adaptação restantes forem implementados.

5.1.2.1 Caso 1: Identificar recursos heterogêneos e mudanças de requisitos no ambiente

Este é o caso de autoadaptação mais simples dentre os 7 casos identificados. Este caso é responsável por envolver a tarefa de autoconfiguração que busca atualizar o servidor e outros componentes sobre as questões heterogêneas do dispositivo. Tais questões envolvem mudanças nas configurações e requisitos das aplicações, tais como: capacidade de energia, comunicação, disponibilidade para execução de tarefas e sensores disponíveis.

Duas etapas são operadas neste caso, onde a primeira é relativa ao processo de descoberta feito fora do componente *Adaptation*, por ser uma característica obrigatória, e a segunda etapa executada no próprio processo de autoadaptação. O processo de descoberta foi implementado no componente *Device* do módulo de sensoriamento, com o objetivo de realizar a descoberta dos recursos e configurações do dispositivo hospedeiro. Tais informações descobertas são utilizadas pelos serviços que rodam no módulo *backend* e pelos próprios componentes do módulo de sensoriamento.

Desta forma, o componente *Adaptation* participa das mudanças de configuração em tempo de execução. Tais mudanças poder ser relacionadas aos estados de execução dos

componentes, as preferências dos usuários e as interações com o módulo *backend*. Contudo, como o componente *User* não está habilitado, as mudanças de políticas em tempo de execução feitas pelo usuário não foram implementadas. As regras de autoconfiguração implementadas foram:

- Evento: Serviços do módulo de sensoriamento foram ativados. Ações:
    - Se política dinâmica de upload de relatos estiver ativa: envia interação ao servidor pedindo a assinatura de taxas de upload (usada pelo caso 4);
    - Se política de armazenamento de mensagens for adaptativa: inicia contagem para varrer mensagens expiradas;
    - Se política de relatar funcionamento do módulo de sensoriamento estiver ativa: inicia intervalo para geração do próximo relatório (usado pelo caso 3);
    - Inicia tarefa de varredura de erros em serviços usando intervalo fixo (usada pelo caso 3);
- Evento: Novo endereço de entrada. Ações:
    - Interação informando o *backend* sobre o novo endereço de entrada;
- Interação: Assinatura da taxa de upload bem sucedida. Ações:
    - Altera estado interno indicando que a assinatura de taxa de upload do serviço de upload foi confirmada. Isso implica que as taxas de upload enviadas pelo serviço assinado serão aceitas. Vide caso na Seção 5.1.2.3.

5.1.2.2 Caso 3: Identificar e Tratar Nós ou Serviços com Problemas;

O mecanismo de adaptação permite que os diversos serviços e processos possam ser restaurados para execução, além de, no caso especificado, informar periodicamente ao servidor sobre o estado atual de execução do módulo. Este relato sobre os estados permite que o próprio servidor *backend* possa identificar problemas no módulo de sensoriamento e tome alguma providência para recuperá-lo, apesar desta tarefa no servidor estar fora do escopo deste trabalho. Para tanto, um trabalho conjunto entre o componente de adaptação e o componente *Device* deve ser feito, pois o componente *Device* conhece e tem acesso a todos os serviços internos do módulo de sensoriamento. As seguintes operações foram implementadas com base nos eventos gerados pelos componentes do Micro-Kernel:

- Evento: Erro grave de inicialização do nó. Ação:

    - Avisa o *backend* usando uma interação contendo a exceção gerada durante a inicialização e informando que este módulo de sensoriamento não pode ser iniciado;
- Evento: Varredura de erros em serviço:
    - Considerando o serviço de upload: (1) se ele possui mensagens para enviar; (2) possui conexão; (3) não enviou a mensagem dentro do intervalo limite para envio de mensagens calculado pelo: *timeout* da interface de comunicação utilizada mais o *valor de tolerância* em milissegundos; e (4) a interface de comunicação com conexão tem permissão pela política de uso dos dados móveis para fazer o upload de relatos. Caso as condições sejam atendidas as seguintes ações serão aplicadas:
        - Acorda o serviço;
        - Salva um aviso no relatório de funcionamento;
    - Considerando serviço de upload: (1) se ele possui mensagens para enviar; (2) possui conexão; (3) não enviou a mensagem dentro do limite estabelecido (*timeout* + *limite*); (4) a interface de comunicação com conexão tem permissão pela política de uso dos dados móveis para fazer o upload de relatos; e (5) o serviço já foi acordado na última varredura de erros. Ação:
        - Reinicia o fluxo de execução;
        - Salva no relatório de funcionamento o erro encontrado no serviço;
    - Se (1) o módulo de sensoriamento estiver completamente desconectado; e (2) o serviço de reconexão geral não responder dentro do intervalo limite definido pelo: *timeout* da interface de comunicação tentando reconectar mais o *limite* de espera do intervalo de reconexão. Ação:
        - Acorda o serviço de reconexão geral;
        - Salva no relatório de funcionamento um aviso;
    - Se: (1) o módulo de sensoriamento estiver completamente desconectado; (2) o serviço de reconexão geral não responder dentro do intervalo limite (*timeout* + *limite*); e (3) o serviço geral de reconexão já foi acordado na última varredura. Ação:
        - Reinicia o fluxo de execução do serviço de reconexão geral;
        - Salva no relatório de funcionamento o erro no serviço;
- Evento: Intervalo para geração do relatório de funcionamento atingido:
    - Ação: Gerar relatório de funcionamento;

- Evento: Relatório de funcionamento foi gerado
    - Ação: Interação com o *backend* informando funcionamento;
- Evento: Erro no loop de adaptação;
    - Ação: Salva no relatório de funcionamento o erro encontrado no *loop*.

5.1.2.3 Caso 4: Otimizar o Upload de Relatos

As informações coletadas da aplicação de sensoriamento são enviadas na forma de relato para o servidor. Contudo, questões sobre disponibilidade de rede, de uso de dados móveis e de desconexão devem ser consideradas pelos elementos autônomos no software. Assim, no componente Communication, o serviço de upload realiza tal papel considerando: a estratégia de upload adaptativo encontrada em Das et al (2010), diferindo por possibilitar o envio dinâmico de taxas de upload conforme a densidade de aplicações de sensoriamento em tempo de execução, apresentado no último exemplo da Seção 4.3, e por oferecer a opção de controlar a frequência de upload ao utilizar dados móveis, podendo até impedir o upload.

Como o componente *Resources* não foi implementado, o monitoramento de bateria e de uso de dados móveis não foi considerado para estas regras. Além disso, como o próprio serviço de upload detecta se a interface usa dados móveis e decide por si só se deve fazer o upload ou não do relato armazenado, somente a seguinte regra foi implementada:

- Interação: Alterar taxa de upload recebida do módulo *backend*:
    - Condição: Se (1) a política dinâmica de upload de relatos for adaptativa e (2) a assinatura com o módulo *backend* foi informada como bem sucedida pela interação do Caso 1. Ação:
        - Altera o estado “taxa máxima de upload” do serviço de upload que corresponde ao endereço do módulo *backend* que enviou o novo valor de taxa máxima de upload.

5.1.2.4 Caso 5: Tratar a Desconexão ou Mobilidade

No componente de comunicação as questões de desconexão são tratadas em dois serviços obrigatórios, um serviço de reconexão e um serviço de upload. Após todas as interfaces de comunicação disponibilizadas terem sido desconectadas, inicia-se o serviço de reconexão em intervalos fixos até alguma interface ser reconectada. Por outro lado, o serviço de upload verifica a restrição de envio de dados e se há conexão, caso ele por algum motivo

não possa fazer o upload ao *backend*, armazena esses dados, enviando-os após a conexão ser reestabelecida ou a restrição removida.

O componente de adaptação enquanto gerente autonômico, quando habilitado, apenas pode fazer ajustes de parâmetros em seus elementos buscando configurações ótimas ou para trocar configurações nesses serviços. Entre eles, para evitar o uso de armazenamento excessivo de dados por períodos muito longos de desconexão, tempos de expiração ou quantidade limite são considerados como restrições pelo componente de adaptação para apagar relatos armazenados ou restringir o upload.

Além disso, entre as interfaces de comunicação há preferências de reconexão, onde se conectado a cabos, prevalece a IEEE 802.11, que prevalece em relação ao uso de dados móveis (2G, 3G, 4G,...) que, por sua vez, prevalece em relação ao uso de DTN por *bluetooth*. Assim, um serviço auxiliar de reconexão faz testes periódicos nas interfaces de comunicação disponíveis com maior prevalência do que a atual e, se conectar, utiliza a prevalente. Desta forma, as seguintes regras foram implementadas:

- Evento: Desconexão completa do módulo de sensoriamento:
    - Se serviço auxiliar ativo;
        - Ação: faz serviço auxiliar dormir;
- Evento: Desconexão de interface:
    - Se interface desconectada for mais relevante que a atual. Ações:
        - Remove do serviço de reconexão auxiliar o uso da interface conectada;
        - Adiciona a nova interface desconectada no serviço de reconexão auxiliar;
        - Acorda o serviço de reconexão auxiliar.
- Evento: Interface reconectada:
    - Se interface reconectada for mais relevante que a interface sendo atualmente utilizada, ou ainda, se não havia conexão geral. Ações:
        - Interface reconectada torna-se a atual;
        - Acorda serviços de upload.
    - Se serviço de reconexão geral ainda ativo. Ação:
        - Faz serviço de reconexão geral dormir;
- Evento: Gatilho temporal para varredura de mensagens expiradas foi acionado:
    - Se política de armazenamento for adaptativa. Ações:
        - Apagar mensagens expiradas e acima da quantidade limite;

    - Agendar novo gatilho para verificação de mensagens expiradas.
- Nota: Como os experimentos apresentados neste capítulo utilizaram somente uma interface de comunicação, um teste de funcionamento teve de ser feito separado para o serviço de reconexão auxiliar utilizando mais de uma interface de comunicação. Desta forma, foi possível constatar que os eventos, condições e ações especificadas nesta seção realmente funcionam.

5.1.3 Aplicações

Para validação do componente de adaptação do módulo de sensoriamento foi escolhida uma aplicação para simulação em ambiente controlado utilizando o Java desktop, a qual também será utilizada para o ambiente em Android. Desta forma, para realizar a simulação da aplicação, foi escolhido o conjunto de dados[2] coletado por Kwapisz, Weiss e Moore (2011) através de uma aplicação real de sensoriamento executada em *smartphones*. Tal aplicação foi utilizada pelos autores para monitorar as informações oriundas de acelerômetros com o objetivo de reconhecer padrões de movimentação. No total foram coletadas 38.209.772 amostras de diversos usuários em um arquivo de 2,11 GB de dados não tratados (*raw*).

O conjunto de dados citado anteriormente foi utilizado como uma aplicação simulada com o objetivo de fornecer uma entrada padrão de eventos nos experimentos aplicados nas implementações de módulo de sensoriamento utilizadas para avaliação do componente de adaptação. Através de tal entrada padronizada, é possível identificar e comparar o desempenho do componente de adaptação em relação ao uso dos recursos de armazenamento, memória, CPU e rede, tanto na implementação para Java SE, Seção 5.2.1.1, quanto na implementação para Android, Seção 5.2.1.2.

5.1.4 Módulo Backend

O módulo de sensoriamento implementado para avaliação da proposta considera duas partes principais, as interfaces de comunicação e as funções do serviço. As interfaces de comunicação utilizam *webservices*, com modelo *send-reply*, para chamadas de serviço oriundas dos clientes. Atualmente, estas chamadas de serviços são limitadas: (1) ao registro do módulo de sensoriamento no servidor (*profiling*), a qual retorna o identificador único

[2] Acesso ao 1ª conjunto de dados: http://www.cis.fordham.edu/wisdm/dataset.php

gerado pelo servidor; (2) ao envio e armazenamento dos relatos (*reporting*) dos módulos de sensoriamento; e (3) a uma interface responsável pela interação entre os módulos no sentido do módulo de sensoriamento para o módulo *backend* (*system interaction*).

Além da interface de entrada, há também a possibilidade de enviar mensagens de interação do módulo *backend* para o módulo de sensoriamento via *socket* ou HTTP. Estes processos de comunicação são representados na Figura 5.1 através do bloco *system interaction*. Uma possibilidade de trabalho futuro será o uso de bibliotecas de *Push* desenvolvidas por terceiro, como a GCM (Google Cloud Messaging) para Android.

Ainda na Figura 5.1, o campo funções de serviço (*Service Functions*) representa as funções de tratamento das chamadas de serviço feitas pelo módulo de sensoriamento. Dentro dela, o componente *Application Functions* realiza o tratamento das chamadas de serviços *profiling* e *reporting*. O *storage* representa o SGBD (Sistema de Gerenciamento de Banco de Dados) utilizado para armazenamento de informações, sendo adotada a ferramenta PostgreSQL e a linguagem de consulta estruturada SQL (Structured Query Language).

Por fim, o módulo *backend* é responsável por processar as mensagens de interação oriundas do módulo de sensoriamento, tomar a decisão com base na interação e respondê-la através do serviço de *push* ou *socket* para os módulos alvos. A implementação deste processo foi feita utilizando um fluxo de execução separado do principal que serve para receber as chamadas de serviço. Sempre que uma nova mensagem é recebida, um evento de interação é gerado e adicionado na fila do módulo, que caso esteja dormindo é acordado. Quando este não possui mais eventos de interação para processar, fica em espera.

5.1.5 Ferramentas e Tecnologias Utilizadas

As principais ferramentas e linguagens utilizadas para a construção de ambos os módulos foram:

- PostgreSQL 9.4: utilizado no *backend* para gerenciamento e consulta de dados;
- SQLite 3.8.11.2: banco de dados utilizado para gerenciamento e consulta de dados feitos pelos módulos de sensoriamento. Devido à biblioteca SQLite do Android ser incompatível com a do Java SE, duas implementações do acesso às bases de dados do módulo de sensoriamento tiveram de ser feitas;
- SQL: utilizado para consultas no banco de dados de ambos os módulos;

- XML: utilizado como script para descrever o módulo de sensoriamento na inicialização; também foi utilizado como formato o conteúdo das mensages;
- Java SE (Java Standard Edition): Linguagem de programação utilizada tanto para Android quanto para o Java;
- Tomcat 8.0: container de web utilizado para hospedar o módulo *backend*;
- JAX-RS 2.0: biblioteca para suporte à criação de webservices REST (Representational State Transfer).

5.2 Avaliações

Esta seção possui como objetivo avaliar a infraestrutura proposta e implementada em duas partes. A primeira parte, referente à Seção 5.1.2, analisa o uso de recursos feito pelo uso do componente de adaptação em comparação ao Micro-Kernel e em dois dispositivos reais, desktop e smartphone. A segunda é desenvolvida na Seção 5.2.2, onde experimentos para avaliação do tempo de processamento de eventos também são executados nos dois dispositivos reais abordados na primeira parte.

5.2.1 Avaliação sobre o Uso de Recursos

Para avaliar o uso de recursos do componente de adaptação do módulo de sensoriamento com e sem componente de adaptação, o conjunto de dados apresentado na Seção 5.1.3 foi separado por usuários, sendo um usuário escolhido arbitrariamente para ser usado como base da simulação da aplicação. Os dados coletados consistem de dados de acelerômetros lidos a cada 50 milissegundos, estes agregados em cada 10 segundos e reportados ao módulo *backend*. Com isso, a entrada de dados da aplicação será uniforme em todos os dispositivos, diminuindo a variância causada entre as aplicações uma vez que o foco da avaliação consiste na infraestrutura de autoadaptação adicionada no módulo de sensoriamento.

Os experimentos das duas subseções foram executados utilizando as implementações de módulo de sensoriamento Android e Java SE em ambiente controlado. O ambiente controlado é composto por um computador dedicado para ser o módulo *backend* e dois outros computadores responsáveis por executar a aplicação simulada sobre o módulo de

sensoriamento. Contudo, a rede 100 Mb utilizada para simulação era compartilhada pelo laboratório de pesquisa. A configuração individual dos computadores é:

- *Módulo Backend*: Sistema Operacional Ubuntu 12.10 (GNU/Linux 3.5.0-27-generic i686); Processador Intel(R) Pentium(R) 4 CPU 3.00GHz; 2 GB de memória RAM; 100 GB de espaço para armazenamento;
- *Dispositivo de sensoriamento desktop 1 (acrônimo: PC 01)*: Sistema operacional Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-28-generic x86_64); Processador Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz; 2 GB de memória RAM disponível DDR2; 350 GB de espaço de armazenamento em HD SATA Maxtor DiamondMax 21 7.200 rpm;
- *Dispositivo de sensoriamento desktop 2 (acrônimo: PC 02)*: Sistema operacional Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-28-generic x86_64); Processador Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz; 2 GB de memória RAM disponível DDR2; 250 GB de espaço de armazenamento em HD SATA Western Digital Caviar SE WD2500JS 7.200 rpm.
- *Dispositivo de sensoriamento móvel (acrônimo: Android)*: Motorola E2 XT1506 com Android Lollipop 5.0.2, contando com um Processador Qualcomm Snapdragon 200 MSM8210 / ARM Cortex-A7 com 1.2 GHz de clock Quad Core e 32bits; 1 GB (Giga Bits) de memória RAM LPDDR2 disponível (936,056 Kbytes utilizável segundo o arquivo /proc/meminfo) e 8 GB de memória interna de armazenamento

Na Seção 5.2.1.1 são apresentados os resultados dos experimentos usando os dispositivos PC 01 e PC 02, e na Seção 5.2.1.2 são apresentados os resultados com experimento usando o dispositivo Android. Esses experimentos foram repetidos 4 em cada dispositivo vezes em cada dispositivo, duas vezes com o componente de adaptação habilitado e duas vezes com o componente de adaptação desabilitado. Ambos os resultados foram coletados usando ferramentas disponibilizadas para monitoramento do sistema operacional nativo de cada dispositivo, sendo os dados brutos coletados processados utilizando a linguagem R com um intervalo de confiança de 95%. Além disso, os resultados em formato bruto, os scripts utilizados para coletar e processar os resultados, bem como o código fonte das implementações para experimentação podem ser encontrados em: <https://github.com/gaborges/urbosenti-sensing-module-experiments>.

5.2.1.1 Implementação Java SE em Ambiente Desktop

Foram executados quatro experimentos em cada computador, dos quais dois foram executados com o componente de adaptação desabilitado enquanto os outros dois foram executados com o componente habilitado. Esses experimentos foram feitos em duas etapas, na primeira etapa os computados PC 01 e PC 02 foram comparados entre si para verificar se há muita variância no uso de recursos entre dispositivos com o objetivo de diminuir a probabilidade de que um deles apresente valores diferenciados devido a problemas de hardware. Na segunda etapa foram comparados os valores coletados com adaptação e sem adaptação. Cada experimento foi executado por 48 horas (172800 segundos).

O resultado dos dados processados pela linguagem R com e sem adaptação podem ser encontrados na Tabela 5.1. Cabe ressaltar que *b* indica valor em bytes, além disso, a linha "Total com adapt." e "Total sem adapt." indicam os valores absolutos das médias entre os experimentos de PC1 e PC2. A diferença absoluta, ou diferença, representa os valores médios totais com e sem adaptação. Por outro lado, a diferença relativa é um produto das somas das médias com e sem adaptação, que considera o valor sem adaptação como 100.

Em outras palavras, a Formula 5.1 exibe o cálculo onde o valor Sem adaptação é considerado 100% tendo como resultado o Valor Relativo com adaptação. Contudo, o valor exibido nas tabelas posteriores é referente à diferença relativa entre o Valor relativo com adaptação e o valor 100%. Tal cálculo é expresso pela Fórmula 5.2. Este método é utilizado em todas as tabelas posteriores para representar o valor relativo de diferença.

Fórmula 5.1: Cálculo do valor relativo usando o valor sem adaptação como 100%

$$Valor\ relativo\ com\ adaptação = \frac{Valor\ absoluto\ com\ Adaptação}{Valor\ absoluto\ sem\ Adaptação} * 100$$

Fórmula 5.2: Cálculo da diferença apresentado nas tabelas

$$Diferença\ Relativa = Valor\ relativo\ com\ adaptação - 100$$

Pela tabela é possível notar que o componente de adaptação necessita fazer o uso de mais recursos computacionais para manter as funcionalidades adicionais sob demanda inicialmente por 1,57% e chegando a diferir em 28,23% de pico em alguns casos após 48h. O desvio padrão da memória de forma semelhante indica maior variância em comparação ao sem o componente *Adaptation*. Apesar destes valores, em geral, serem usos adicionais

moderados dos recursos, para um computador desktop (considerando que um desktop possui maior disponibilidade de recursos), o espaço ocupado pela base de dados gerada pela UrboSenti aumentou significativamente em 48h em cerca de 72,3 vezes em comparação ao tamanho médio final de 117.760 bytes, sem o componente de adaptação.

Tabela 5.1 - Uso de recursos por computador com e sem o componente de adaptação

Identificação			*Memória RAM (b)*				*Tamanho final base de dados (b)*	*Uso de CPU (%)*		
PC	Exp.	Adapt.	Mínimo	Médio	Máximo	Desvio Padrão		Médio	Max.	Desvio Padrão
1	1	Sim	36.735	369.794	430.677	36.735	8.627.200	3,2411	88	7,2395
1	2	Sim	37.334	366.320	428.018	37.334	8.634.368	3,2675	85	7,2805
1	3	Não	35.599	306.848	312.420	6.132	128.000	2,4215	93	6,7650
1	4	Não	50.126	299.984	304.441	5.587	107.520	2,4215	80	6,7262
2	1	Sim	36.213	371.092	435.792	40.357	8.634.368	3,3035	69	7,1522
2	2	Sim	38.259	375.487	437.634	39.319	8.634.368	3,3279	88	7,2085
2	3	Não	33.758	311.953	317.944	7.489	128.000	2,4212	86	6,6785
2	4	Não	36.622	303.043	308.328	5.814	107.520	2,4234	87	6,7308
Total com adapt.			39.641	370.674	433.031	-	8.632.576	3,2850	82,5	-
Total sem adapt.			39.027	305.457	310.784	-	117.760	2,4219	86,5	-
Diferença Absoluta			614	65.217	122.247	-	8.514.816	0,8630	-4	-
Diferença (%)			1,57%	17,59%	28,23%	-	7230,65%	35,64%	-4,62%	-

Fonte: Primária.

Tal fato em pouco tempo acarretaria em problemas de desempenho para buscas na base de dados, além do uso desnecessário do espaço de armazenamento, o qual é vital especialmente para sistemas com limitações de espaço e de bateria como é o caso dos *smartphones*. Estes dados extras são decorrentes dos registros passados de eventos, ações e estados que são armazenados e não apagados durante a execução. Muitos desses dados não precisam ser mantidos e poderiam ser apagados. Contudo, é necessário especificar quais dados e em que quantidade cada elemento pode armazená-las, além de ser definida a forma como esses dados extras devem ser apagados. O estudo de tal funcionalidade está em aberto para trabalho futuro.

Ainda considerando os dados da tabela, o uso de CPU foi semelhante nos dois computadores exibindo uma diferença relativa nos valores médios de 35,64%, que mesmo com esse aumento, o impacto na CPU foi em geral baixo, tendo como médios máximos de 3,3279%. Por fim, como nota adicional, o uso de rede não foi adicionado à tabela, pois não houve diferença significativa entre os módulos com e sem o componente de adaptação, uma vez que o único uso de rede decorrente de tal componente era o relato de erros para o servidor *backend* a cada 12 horas.

5.2.1.2 Implementação Android em Ambiente Móvel

A aplicação apresentada na seção anterior também foi utilizada para avaliar o desempenho do módulo de sensoriamento da UrboSenti com e sem componente de adaptação em dispositivo móvel com o sistema operacional Android instalado. Para tanto o módulo de sensoriamento em Java SE teve de ser migrado para a plataforma Android, o que incluiu a aplicação simulada utilizada para fazer a análise do módulo de sensoriamento Java SE. Nessa migração todas as classes que faziam manipulação da base de dados e criação da mesma, além de classes específicas para descoberta dos recursos do sistema operacional e comunicação, tiveram de ser reimplementadas. Tal implementação está compartilhada em repositório pelo link: <https://github.com/gaborges/UrboSentiExperimentalMobileSensingModule>.

Para aplicação dos experimentos, foi utilizado o dispositivo smartphone MotoE2 com android e o PC 01, ambos citados anteriormente. Somente o PC 01 foi selecionado pois não houve significante variância entre os dois computadores, logo ele possuem um comportamento semelhante. Com esta configuração foram executados quatro experimentos de 14h - dois com o componente de adaptação ativado e dois com ele desativado - com o objetivo de avaliar o uso de memória, CPU e quantidade de armazenamento feito pelo banco de dados.

Os comandos para acesso às medidas de tais recursos foram feitos pelo terminal *shell* disponibilizado pelo ADB (Android Debug Bridge). Para verificar o uso de memória foi utilizado o comando *dumpsys meminfo [nome_aplicação]*[3] para uso de CPU o comando *dumpsys cpuinfo [nome_aplicação]*[4] e para conferir o tamanho do banco de dados o comando *ls -l [caminho]*[5]. Adicionalmente, os arquivos */stat/[pid]/stat* e */proc/stat* também foram copiados. Contudo, estes não foram necessários para a análise apresentada, uma vez que a ferramenta *dumpsys* já apresenta as estatísticas de uso com base nos mesmos arquivos. Estes arquivos são disponibilizados juntamente com os resultados e os códigos fonte do aplicativo no github, apresentados anteriormente.

Tais comandos foram executados a cada 5 segundos até o final da aplicação, inicialmente com o objetivo de executar em 12 horas. Todavia, o uso dos comandos com o *adb shell "comando"* possui um overhead para copiar o conteúdo dos comandos, assim, tempos irregulares foram registrados. Desta forma, para igualar os tempos de comparação, foi

[3] http://developer.android.com/intl/pt-br/tools/debugging/debugging-memory.html
[4] https://source.android.com/devices/tech/debug/dumpsys.html
[5] http://linux.die.net/man/1/ls

escolhido o valor 14h, por ser o menor valor de tempo entre os experimentos em Android. Os registros sobressalentes foram retirados da avaliação do experimento.

Um dos problemas esperados desse experimento era o consumo elevado de memória, que chegou a picos de até 437.634 bytes conforme apresentado na tabela da seção anterior. Desta forma, consultou-se a documentação do Android em busca de boas práticas para melhorar o consumo de memória[6]. De acordo com ela, é recomendada a substituição de Classes do Java SE mais ineficientes em relação ao uso de memória (por exemplo, HashMap por SparceArray[7] e a biblioteca DOM, utilizada para gerar os XMLs para comunicação por SAX ou mesmo a versão em JSON). Além disso, é recomendado sempre que possível fechar todas as conexões e limpar listas, para facilitar o trabalho do Garbage Collector.

Uma solução testada foi a adição do método *clear()* nos HashMaps que transportam o conteúdo dos eventos após serem processados pelo loop de adaptação e pelo tratador de eventos da aplicação, tanto na implementação Android como na Java SE. A partir disso, foram executados novos experimentos no computador 1 usando a implementação em Java SE também com 14h, comparando com os resultados antigos, antes destes serem comparados com os resultados em Android. Assim, a Tabela 5.2 exibe os resultados do novo experimento onde a identificação “Normal” contém os dados extraídos de 14h no experimento sem a modificação e o identificador “Modificado” contém os dados extraídos no novo experimento. O formato segue o padrão da tabela apresentada na seção anterior referente ao uso de CPU, da memória RAM e do tamanho do banco de dados.

Como resultado, houve uma redução no crescimento do uso de memória RAM ao utilizar a modificação. O uso médio e máximo respectivamente foram por volta de 19,60% e de 17,32% maior para o experimento sem a modificação em comparação ao componente de adaptação habilitado, e 20,58% e 20,13% respectivamente, com o mesmo desabilitado. Apesar disso, o consumo de CPU, o crescimento do banco de dados e o uso de memória inicial continuaram semelhantes.

Os resultados da comparação do experimento com a modificação em Java SE juntamente com os resultados em Android podem ser encontrados na Tabela 5.3 com o objetivo de discutir as diferenças sobre o uso de recurso de ambas as plataformas. Nesta tabela os experimentos em Java são identificados como *PC* e os experimentos em Android são identificados pelo rótulo *Android*. Além disso, os dados comparados exibidos utilizaram os

[6] https://source.android.com/devices/tech/config/low-ram.html
[7] http://developer.android.com/intl/pt-br/reference/android/util/SparseArray.html

valores médios calculados dos dados brutos, onde Sim-Não é a diferença absoluta e Sim/Não é a diferença relativa.

Como resultado, é possível notar que ambos os experimentos apresentaram valores semelhantes em relação ao uso máximo de memória. Além disso, embora o desvio padrão do uso de memória seja maior para o *Android* do que para o *PC*, houve pouca variância entre os resultados de desvio padrão relacionados ao uso de memória nos resultados brutos, não apresentando considerável diferença quando o componente *Adaptation* foi habilitado, considerando os experimentos realizados.

Tabela 5.2 - Comparação Java SE com e sem a modificação do código fonte

Identificação		*Memória RAM (b)*				*Tamanho final data base (b)*	*Uso de CPU (%)*		
PC 1	Adapt.	Mínimo	Media	Máximo	Desvio Padrão		Médio	Max	Desvio Padrão
Normal	Sim	34.986	324.064	344.542	16.393	2.549.760	2,61	88	6,8776
Normal	Sim	49.103	320.510	340.245	16.832	2.553.856	2,62	85	6,7294
Normal	Não	35.599	302.032	304.236	9.119	107.520	2,42	93	6,8381
Normal	Não	39.487	295.834	298.917	8.525	107.520	2,42	80	6,8040
Modificado	Sim	41.533	269.891	292.983	17.244	2.552.832	2,60	68	6,7794
Modificado	Sim	41.124	269.038	290.733	16.832	2.552.832	2,61	88	6,8776
Modificado	Não	40.714	249.196	253.087	7.329	107.520	2,41	56	6,6604
Modificado	Não	42.965	246.610	248.995	7.329	107.520	2,42	80	6,8040
Comparação Normal vs Modificado									
Diferença com adapt.		716	52.823	50.535	-	-	-	8,5	-
Diferença sem adapt.		-4.296	51.030	50.535	-	-	-	18,5	-
Diferença com adapt (%)		1,73%	19,60%	17,32%	-	-	-	10,90%	-
Diferença sem adapt (%)		-1,22%	20,58%	20,13%	-	-	-	27,21%	-

Fonte: Primária.

Tabela 5.3 - Uso de recursos no smartphone com e sem o componente de adaptação

Identificação		*Memória RAM (b)*				*Tamanho final data base (b)*	*Uso de CPU (%)*		
Exp.	Adapt.	Mínimo	Media	Máximo	Desvio Padrão		Médio	Max	Desvio Padrão
Android	Sim	19.365	139.205	270.294	63.004	2.584.576	38,13	94	13,0566
Android	Sim	19.666	139.763	294.443	64.294	2.629.632	39,73	83	9,4891
Android	Não	19.194	145.633	260.394	60.692	233.472	35,23	71	7,5418
Android	Não	17.949	136.003	265.817	64.883	233.472	37,60	76	7,5230
PC	Sim	41.533	269.891	292.983	17.244	2.552.832	2,60	68	6,7794
PC	Sim	41.124	269.038	290.733	16.832	2.552.832	2,61	88	6,8776
PC	Não	40.714	249.196	253.087	7.329	107.520	2,41	56	6,6604
PC	Não	42.965	246.610	248.995	7.329	107.520	2,42	80	6,8040
Dados Comparados									
PC Sim-Não		-511	21.561	40.817	-	2.445.312	0,2	10	0,10
PC Sim/Não (%)		-1,22%	8,70%	16,26%	-	2274,29%	7.31	12,82	1,41
Android Sim-Não		944	-1.334	19.263	-	2.373.632	2,51	15,00	3,79
Android Sim/Não (%)		5,08%	-0,95%	7,32%	-	1016,67%	6,90	20,41	50,71

Fonte: Primária.

Além disso, o uso de CPU em percentagem para o *PC* e o *Android* teve grande diferença na comparação entre as plataformas, sendo que o uso médio de CPU para o *PC* foi, no pior caso, 2,61% enquanto para o *Android* foi, com o Componente *Adaptation* habilitado,

de 39,73%. Por outro lado, o mesmo componente teve pouca influência relativa sobre o uso de CPU, o qual foi por volta de 2% maior para o experimento *Android*, e de 0.2% para o *PC*.

O maior uso de processamento pelo dispositivo Android deve-se, possivelmente, ao fato do uso de memória ser mais limitado em Android, assim sendo, o *Garbage Collector* (GC) acaba por ser executado com maior frequência, tendo como consequência um maior uso do processamento no dispositivo, além de implicar um maior desvio padrão na análise de memória, como apresentado anteriormente. Tal efeito pode ser melhor visto utilizando a ferramenta *adb*, a qual foi utilizada para coletar alguns logs em tempo de execução, sendo que estes podem ser encontrados entre os dados coletados. Contudo, são necessários mais estudos para identificar de forma quantitativa as fontes de uso de CPU.

Apesar disso, dadas às informações apresentadas, é possível concluir que a fonte da maior parte desse processamento origina-se do Micro-Kernel, sendo que a ativação do Componente *Adaptation* teve pouca influência relativa no uso de memória e no uso de CPU. Mesmo assim, um uso de processamento por volta de 36% de CPU é indesejável e possivelmente utilizaria mais bateria do usuário, e, portanto deve ser considerado como trabalho futuro para melhorias na implementação da UrboSenti. Entre estas melhorias, a substituição, quando possível, de classes Java por classes nativas do Android seria relevante, como no caso da substituição da classe Timer[8] pela ScheduledThreadPoolExecutor[9], recomendada para novas implementações pela documentação exposta na nota do rodapé.

Por fim, o banco de dados SQLite nativo de Android, para todos os experimentos, inicia com o valor de 233.472 (duzentos e trinta e três mil quatrocentos de setenta e dois) bytes, após a criação do banco de dados, sendo pouco maior que o registrado pela biblioteca em Java SE de 70.656 bytes em termos absolutos. Também, pelos dados apresentados na coluna *Tamanho final*, é possível notar que o banco de dados SQLite do Android com o componente de adaptação ativado cresce de maneira proporcional à biblioteca utilizada Java SE (cerca de 2% maior para o Android que o tamanho em Java SE). Contudo, como mencionado na seção anterior, o estudo deste crescimento do banco de dados está destinado a trabalhos futuros.

[8] http://developer.android.com/intl/pt-br/reference/java/util/Timer.html
[9] http://developer.android.com/intl/pt-br/reference/java/util/concurrent/ScheduledThreadPoolExecutor.html

5.2.2 Tempo de Processamento da Infraestrutura do Mecanismo de Autodaptação Proposto

O objetivo geral desta seção é verificar a capacidade da infraestrutura do mecanismo de autoadaptação de processar a resposta a cada evento assíncrono gerado nas implementações feitas do Componente de Adaptação da UrboSenti. Para tanto, o tempo de resposta é utilizado como principal métrica de desempenho. As avaliações efetuadas foram: impacto que a quantidade de eventos por experimento executado, Seção 5.2.2.1; o tempo de processamento para tratar eventos gerados pelos componentes da UrboSenti, Seção 5.2.2.2; e, por fim, o tempo de processamento para tratar interações feitas com outro módulo, Seção 5.2.2.3.

Foi criado um componente extra para experimentação, chamado *TestManager*, o qual contém os parâmetros, ações, interações e eventos para realizar os experimentos. Além disso, cabe ressaltar que o módulo *backend* será utilizado somente para registro dos módulos de sensoriamento e que as interações foram testadas entre diversos módulos de sensoriamento. As comunicações foram feitas por via do método HTTP tanto para envio de mensagens como para recebimento. O código utilizado para experimentação pode ser encontrado no GitHub acessando o link: https://github.com/gaborges/urbosenti-sensing-module-experiments.

5.2.2.1 Impacto da Quantidade de Eventos no Tempo de Processamento

Primeiramente, é necessário identificar o impacto que a quantidade de eventos possui por execução tendo em vistas encontrar a quantidade de eventos para ser utilizada nos demais experimentos. Desta forma, quatro experimentos foram executados utilizando diferentes quantidades de eventos no computador 1 com Java SE, o qual foi apresentado na Seção 5.2.1.1. O resultado de cada um dos experimentos foi agregado pela quantidade de eventos e processados pela linguagem R. Os dados processados estão dispostos na Tabela 5.4, onde a primeira coluna indica a quantidade de *eventos* executados, a segunda coluna apresenta o *tempo médio total* para executar todos os eventos em segundos (s), a terceira coluna apresenta quatro *tempos* em milissegundos (ms) resultados da análise do R.

Em primeira instância, é possível notar na tabela que o tempo máximo e o mínimo não exibem significante variância entre si em relação à quantidade de eventos, com a do caso contendo 10000 eventos. Entre os experimentos de 10000 eventos um deles apresentou um período de tempo com diversos valores acima da média, o que implicou também em um

desvio padrão maior, possivelmente causado por algum processo concorrente dos recursos do sistema.

Tabela 5.4 - Impacto da quantidade de eventos por experimento

Eventos	*Tempo Médio Total (s)*	*Tempos (ms)*			
		Médio	*Desvio Padrão*	*Mínimo*	*Máximo*
200	71	355,123	115,67294	300	1969
500	175	351,815	79,96410	300	1922
1000	349	349,297	60,62657	300	1942
2000	698	349,252	52,64595	300	1933
3000	1044	348,201	48,46381	295	1910
4000	1396	349,040	47,28536	292	1982
5000	1742	348,573	45,86361	281	1931
6000	2092	348,711	44,71815	278	1980
7000	2444	349,180	44,44496	278	1904
8000	2812	351,600	44,15833	279	1944
9000	3147	349,668	43,47788	278	1955
10000	3718	371,839	129,13207	280	4183

Fonte: Primária.

Por outro lado, em relação ao tempo médio, na linha dos 200 eventos e 500 eventos um valor maior é encontrado, o qual tende a ter um valor aproximado a partir dos 1000 eventos, com exceção dos 10000 eventos que, pelo motivo discutido acima, apresentou um valor superior. Além disso, é possível observar claramente que o desvio padrão tende a diminuir conforme o número de eventos aumenta, entretanto, por outro lado, o tempo médio total de processamento em segundos destes eventos aumenta consideravelmente.

Para os experimentos posteriores foi escolhida a quantidade de 1000 eventos por apresentar um tempo médio de processamento semelhante aos experimentos com quantidade maior de eventos. Além disso, apesar de apresentar um desvio padrão um pouco maior que os experimentos com mais eventos, o tempo médio total de processamento pouco abaixo de 6 minutos compensa dada a quantidade de eventos executados nas próximas seções.

5.2.2.2 Tempo de Processamento para Tratar Eventos Internos

A avaliação do tempo de processamento do mecanismo de adaptação para tratar os eventos gerados pelos componentes internos pode ser encontrada na Tabela 5.5. No total foram executados 11 experimentos cada um com 1000 eventos, e os resultados foram processados utilizando a ferramenta R. O mesmo procedimento foi repetido em um segundo computador com a mesma configuração de hardware, com exceção do HD. Cada evento possuía somente uma ação a ser aplicada, sem condições e nem regras.

Tabela 5.5 - Avaliação de desempenho por eventos em dois computadores

Parâmetros			Tempos (ms)			
Eventos	PC	Execução	Médio	Desvio Padrão	Mínimo	Máximo
1000	1	1	349,410	65,03732	301	2097
1000	1	2	349,882	49,05481	308	1427
1000	1	3	350,500	69,39711	308	2168
1000	1	4	351,393	53,37847	309	1520
1000	1	5	350,890	74,65772	302	2340
1000	1	6	351,041	49,40435	301	1460
1000	1	7	349,563	67,64486	309	2139
1000	1	8	351,933	53,25736	309	1423
1000	1	9	351,830	68,13771	309	2087
1000	1	10	351,838	56,09847	308	1468
1000	1	11	353,372	53,16685	308	1429
1000	2	1	607,865	84,78637	522	3140
1000	2	2	614,262	68,16982	547	2597
1000	2	3	607,193	80,54090	504	3016
1000	2	4	615,064	67,40576	546	2585
1000	2	5	620,450	106,23198	537	3062
1000	2	6	615,238	67,96934	547	2603
1000	2	7	612,554	80,57394	522	3016
1000	2	8	619,546	68,68984	546	2621
1000	2	9	615,177	82,15013	530	3059
1000	2	10	614,509	67,51841	546	2586
1000	2	11	615,323	70,31066	546	2654

Fonte: Primária.

O tempo de processamento de cada evento é feito com base na diferença de tempo entre cada execução de ação (em outras palavras, *tempo de processamento[n] = Tempo da ação[n] - Tempo da ação[n-1]*), em milissegundos (ms). Desta forma, quando uma ação termina de executar, o ciclo de adaptação recomeça e seleciona imediatamente o próximo evento, sendo esse o tempo mais aproximado para calcular o intervalo de processamento do ciclo de adaptação. O único tempo de evento considerado para o cálculo é o tempo de criação do primeiro evento, o qual demonstra o maior tempo utilizado em decorrência das consultas iniciais no banco de dados, que após são feitas em memória.

Os resultados individuais considerando os resultados agregados de todos os experimentos indicam que para o computador 1 o tempo médio foi de 351,059 ms, com um desvio padrão de 60,540 ms, e respectivamente, tempo mínimo e máximo para processamento dos eventos de 301 ms e 2340 ms. O segundo computador obteve um tempo médio de 614,289 ms, um desvio padrão de 77,656 ms, e respectivamente o tempo mínimo e máximo das leituras de 504 ms e de 3140 ms. Os valores máximos são referentes ao primeiro evento executado pelo loop de adaptação, como explicado anteriormente. Por outro lado, apesar da

diferença dos tempos médios entre o computador 1 e do 2, as medidas de variâncias apesentam valores proporcionais e semelhantes.

Os experimentos de eventos também foram executados no smartphone descrito na Seção 5.2.1.2, utilizando a implementação em Android com a máquina virtual Art 5.0.1. Os resultados analisados pela Linguagem R referentes ao tempo de processamento das 11 repetições com 1000 eventos foram: tempo médio aproximado de 47,773 ms; desvio padrão aproximado de 8,637; tempo mínimo de 36 ms e tempo máximo de 300 ms. Tais tempos de processamento apresentaram valores bastante menores que os encontrados nos experimentos utilizando os computadores, que no melhor caso é de 7 (sete) vezes maior que a média. Os principais fatores para esse melhor desempenho nos resultados de tempo do Android em comparação com os Desktops foram inicialmente especulados como sendo o uso da máquina virtual Art e o processador ARM quad core. Essa suposição foi confirmada através da argumentação a seguir.

De acordo com OH et al (2012), existem diferenças entre a máquina virtual em Java (JVM) e a máquina virtual Dalvik (DVM) em Android, onde a JVM é frequentemente melhor em desempenho de processamento. Por outro lado, os experimentos aplicados por Yadav e Bhadoria (2015) usando o substituto do DVM, a Art, apresentaram desempenho, em geral, pouco superior. A máquina virtual Art foi adicionada como opcional na versão 4.4.2 do Android e é substituta nativa da Dalvik a partir da versão 5.0 ou superior. De acordo com a documentação do Android[10], a Art é mais eficiente que a versão anterior devido a melhorias no funcionamento do *Garbage Collector* e na compilação via *Ahead-of-time* que passa a ser feita em tempo de instalação no dispositivo ao invés de realizar uma compilação gerando *bytecodes* para serem interpretados pela máquina virtual local, o qual é o método utilizado tanto pela Dalvik como pelo Java SE.

Como os experimentos feitos por Yadav e Bhadoria (2015) mostraram que o Art é pouco mais eficiente que a DVM, além de OH et al (2012) mostrarem que a JVM é superior que a DVM, não seria possível que a diferença nos valores de tempo do processamento encontrados nos experimentos seja causada somente pelo uso de diferentes máquinas virtuais. Assim, foi considerada a hipótese de que o processador utilizado pelo desktop não tem suporte a três conjuntos de instruções (SSE4.1, SSE4.2 e POPCNT), os quais são utilizados para emular o Android em alto desempenho em Linux. Tais conjuntos foram exibidos na seguinte mensagem ao inicializar uma máquina virtual Android: "*WARNING: Host CPU is*

[10] http://source.android.com/devices/tech/dalvik/index.html

missing the following feature(s) required for x86_64 emulation: SSE4.1 SSE4.2 POPCNT Hardware-accelerated emulation may not work properly!".

Desta forma, um novo experimento foi aplicado em um computador mais recente com um processador i7-4500U 1.80 GHz, 8GB de memória RAM DDR3 que possui suporte à tais conjuntos de instruções para aceleração de emulação com o objetivo de identificar o impacto em diferentes máquinas virtuais. Os resultados podem ser encontrados no Apêndice A, os quais são referentes ao experimento de eventos internos, com as mesmas configurações dos experimentos anteriores. Em suma, foram aplicados experimentos em uma máquina virtual com Linux com a implementação Java da UrboSenti, duas máquinas virtuais em Android 4.4.2, uma com Dalvik e a outra com o Art, e uma última máquina virtual emulando a mesma versão do smartphone utilizado nos experimentos, o Android 5.0.1 com Art.

Como resultado, verificou-se que, para o experimento avaliado, o Art na versão 5.0.1 é por volta de 1,93 vezes mais rápido que o Java em uma máquina virtual, 2 vezes mais rápido que o Dalvik e 1,16 vezes mais que o Art versão 4.4.2. Isto corrobora que os tempos mais lentos nos experimentos em ambiente desktop foram majoritariamente influenciados por questões relacionadas ao hardware utilizado e, em sequência, pela escolha da máquina virtual que executa o módulo de sensoriamento.

5.2.2.3 Tempo de Processamento para Tratar Interações com Outro Módulo

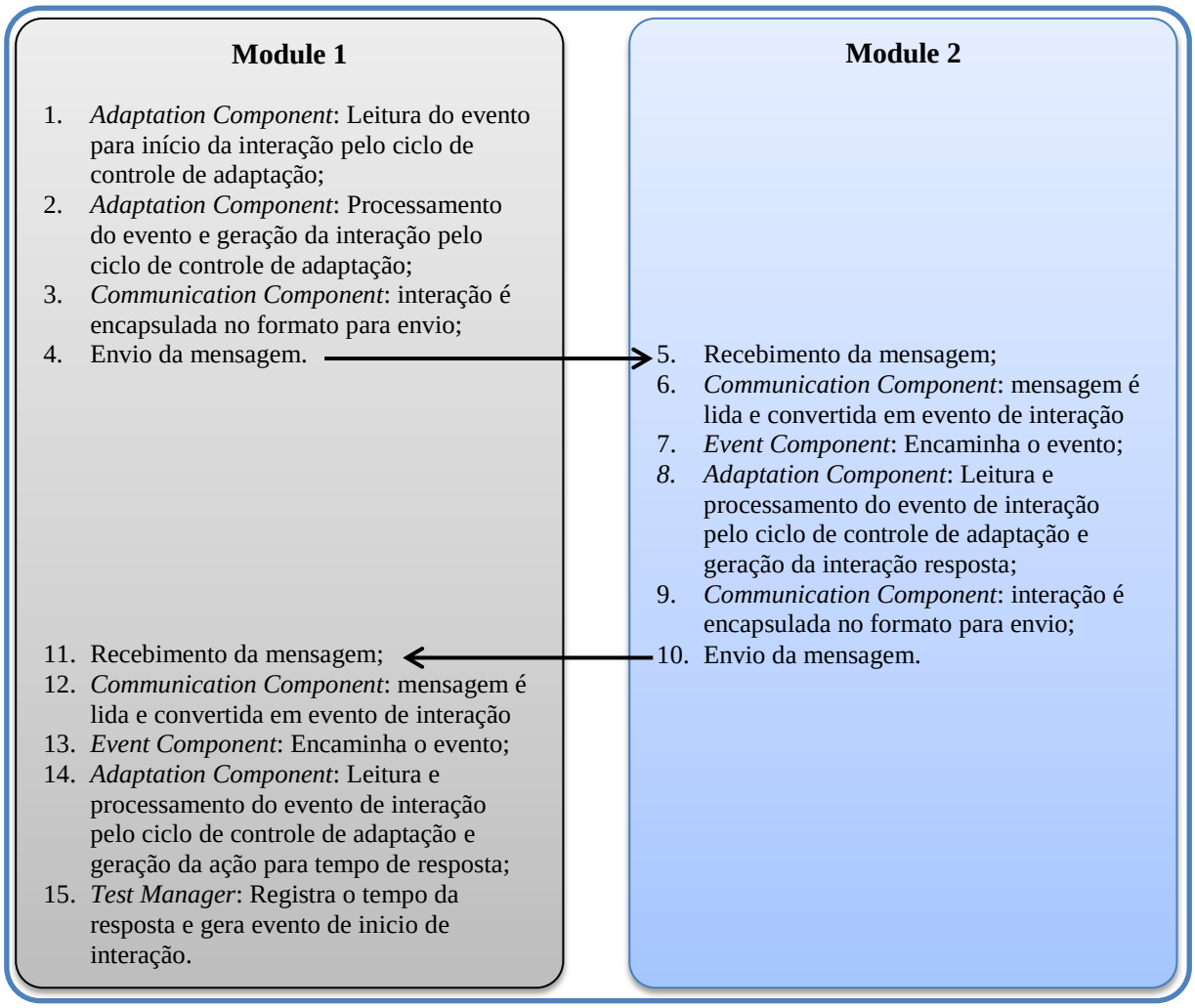
O impacto na quantidade de interações realizadas com um único módulo é analisado nesta seção. O experimento realizado consistiu em dois módulos com diferentes computadores onde um módulo envia uma interação para o outro módulo que a responde. Os dispositivos PC 01 e PC 02 foram escolhidos para tanto. O tempo entre cada interação é dado pelo tempo de percepção do evento que encadeia a interação com o outro módulo até a aplicação da ação como resposta da interação retornada para o módulo que começou a interação. A soma de todos os fatores do processo pode ser encontrada, em maiores detalhes, na Figura 5.2.

Nos experimentos de interação, assim como no experimento dos eventos, foram considerados os resultados de 1000 interações. Tal experimento foi repetido 10 vezes em cada computador com um módulo de sensoriamento, onde os resultados processados por experimento pela linguagem R podem ser encontrados na Tabela 5.6.

Uma vez que os computadores eram similares em configuração, com exceção do HD, e compartilhavam a rede entre outros computadores, a diferença entre o tempo de

processamento entre os dois computadores foi resultante principalmente de dois fatores: a velocidade de leitura e escrita nos HDs e a latência de rede. Os resultados individuais considerando os resultados agregados de todos os experimentos indicam que para o computador 1 o tempo médio foi 1477,040 ms, com um desvio padrão de 371,301 ms, e respectivamente, tempo mínimo e máximo para processamento dos eventos de 1291 ms e 16509 ms. O segundo computador possuiu um tempo médio aproximado de 2484,175 ms, um desvio padrão aproximado de 119,171 ms, e respectivamente o tempo mínimo e máximo das leituras de 2296 ms e 5519 ms.

Figura 5.2 - Processo utilizado para medir o tempo de cada interação



Fonte: Primária.

Devido à grande diferença de tempo de resposta nos eventos do primeiro computador, os experimentos desse computador foram repetidos em um horário onde houve pouca possibilidade do uso da rede por outros usuários. Estes resultaram nos tempos: médio 1412,168 ms; desvio padrão 79,261 ms; mínimo 1292 ms; e máximo 3102 ms. Tais

resultados corroboram interpretação que a variância nos resultados anteriores era resultado do uso compartilhado da rede.

Tabela 5.6 - Avaliação de desempenho das interações com outro módulo

Parâmetros			Tempos (ms)			
Interações	PC	Execução	Médio	Desvio Padrão	Mínimo	Máximo
1000	1	1	1494,207	338,63394	1337	8593
1000	1	2	1471,301	189,56502	1294	4341
1000	1	3	1519,514	472,74851	1312	8606
1000	1	4	1495,644	578,18317	1291	16329
1000	1	5	1483,388	578,96009	1328	16509
1000	1	6	1468,242	244,08624	1295	4639
1000	1	7	1456,618	288,81837	1303	8479
1000	1	8	1461,016	272,09500	1293	8363
1000	1	9	1452,360	204,53547	1311	4440
1000	1	10	1468,113	271,06029	1291	4508
1000	2	1	2494,074	216,08960	2313	5519
1000	2	2	2523,055	153,80967	2312	5394
1000	2	3	2500,445	117,59950	2320	4371
1000	2	4	2471,154	98,62222	2320	4077
1000	2	5	2470,294	84,82228	2329	4403
1000	2	6	2470,360	86,09039	2329	4293
1000	2	7	2472,680	87,58590	2304	4513
1000	2	8	2471,554	85,50417	2320	4387
1000	2	9	2480,740	87,37423	2323	4490
1000	2	10	2487,401	90,94948	2296	4326

Fonte: Primária.

Os resultados comparados da primeira tomada de experimentos com a segunda indicam que a rede, durante o experimento do computador 1, estava sendo utilizada por outros computadores da infraestrutura de rede compartilhada. Para melhor ilustrar o impacto da latência no tempo do processo foram executados experimentos para medi-la usando o programa ping por 15 min (54000 pacotes) em cada computador. Os dados gerados por esse experimento nos dois computadores foram agregados e processados pelo R, tendo como resultados: tempo médio de 275,917 ms; tempo máximo de 12754 ms; tempo mínimo de 130 ms; e desvio padrão 45,816 de ms. Como há duas etapas de comunicação, para fazer esse processo de interação, o tempo médio teria o dobro de seu valor adicionado no tempo da interação (cerca de 550 ms).

O mesmo experimento foi executado posteriormente com a implementação em Android utilizando comunicação Wi-Fi via um roteador D-Link DIR-615 Wireless N 300Mbps. Dez experimentos com 1000 interações foram executados do PC 01 ao Smartphone Android e na direção contrária. Os resultados gerais do Android para o PC 01 foram: tempo médio de 827,680 ms, tempo mínimo de 521 ms, tempo máximo de 5943 e desvio padrão de

102,890 ms. Os resultados gerais do PC 01 para o Android foram: tempo médio de 1548,674 ms, tempo mínimo de 1319 ms, tempo máximo de 6204 ms e desvio padrão de 351,243 ms.

Para analisar o impacto do uso de Wi-Fi na comunicação também foi executado o comando *ping –c 900 <IP>* para análise de latência da rede em ambos os dispositivos. Os resultados de tempo do Android para o PC 01 foram: médio 17,105 ms, mínimo 1,33 ms, máximo 607 ms e desvio padrão 40,634 ms. Do PC 01 para o Android teve como resultados: médio 35,665 ms, mínimo 1,16 ms, máximo 793 ms e desvio padrão 96,167 ms. Apesar de não ser uma variação significativa o tempo maior de resposta no sentido do PC 01 ao Android indica que a rede cabeada é capaz de responder mais rapidamente que a Wi-Fi no cenário analisado. Importante destacar que nenhum deles teve perda de pacote durante a execução do experimento, o qual durou por volta de 15 minutos (900 segundos).

5.3 Considerações Finais e Discussão sobre Limitações

A arquitetura UrboSenti possui flexibilidade para permitir o uso dos componentes que somente são necessários, utilizando componentes opcionais e um Micro-Kernel com funções básicas. Entre tais opcionais, o componente de adaptação é responsável por processar as adaptações de forma autonômica, tomando decisões. Dentre os casos elencados nos trabalhos relacionados, os seguintes casos/objetivos de adaptação foram implementados para avaliação considerando a combinação de componentes elencados:

- Autoconfiguração sem relação com o usuário;
- Identificar e tratar problemas nos módulos e serviços;
- Otimizar questões relacionadas com a rede; e
- Tratar a desconexão e mobilidade.

Tais casos de adaptação foram testados tanto em ambiente desktop como em ambiente móvel com Android comprovando a capacidade da UrboSenti de suportar diferentes plataformas de dispositivos. Além disso, foram executados experimentos para avaliação do mecanismo de adaptação, que revelaram um desempenho razoável devido principalmente pela diferença da arquitetura entre os computadores e a diferença entre as máquinas virtuais.

Entre os resultados dos experimentos, verificou-se também que todos os eventos, ações e estados passados são salvos no banco de dados e não são excluídos. Tal fato resulta em um tamanho de banco de dados que pode prejudicar, conforme cresce, o desempenho na busca de dados, além de ocupar um espaço desnecessário em um ambiente limitado de

capacidade de armazenamento como o dos *smartphones*. Assim, como trabalho futuro será necessário ter um elemento para limitar a quantidade de dados monitorados armazenados, o qual poderia ser utilizado por uma tarefa de limpeza ao excluir registros.

Os experimentos a cerca do uso de recursos revelaram que o uso do componente de adaptação, tanto para o ambiente em desktop como o móvel, adiciona pouco uso dos recursos ao ser comparado com o total. Desta forma, o principal responsável pelo uso de tais recursos foi à implementação atual do Micro-Kernel, especialmente quanto ao uso de memória para ambos os ambiente e CPU para o ambiente móvel. Contudo, esses usos tendem a crescer à medida que os demais casos de adaptação forem implementados e outras aplicações testadas, o que é uma grande preocupação, principalmente para o ambiente móvel, pois o uso excessivo de CPU drena rapidamente a limitada disponibilidade de energia das baterias.

Quanto às limitações, o mecanismo proposto não conseguiu atender todos os casos de autoadaptação identificados, bem como não explorou em profundidade as técnicas das etapas de análise e planejamento, nem como elas deveriam ser modeladas para atender os casos de adaptação. Isso de seve ao fato que adicionar uma infraestrutura para monitoramento e tomada de decisão autonômica no software de sensoriamento não é uma tarefa fácil, visto que é necessário um esforço inicial maior para modelar todo o ambiente, pois uma visão ampla do ambiente é necessária para que o mecanismo de adaptação possa tomas as devidas decisões.

Por segundo, apesar dos resultados referentes à avaliação do uso de recursos serem interessantes, outras avaliações seriam mais relevantes para um mecanismo de adaptação, como o caso de uma avaliação com programadores e experimentos sobre adaptabilidade. Os experimentos com programadores poderiam ser utilizados para descobrir se o mecanismo proposto realmente facilita o trabalho de programação em uma aplicação de tal tipo. Por outro lado, experimentos sobre adaptabilidade seriam interessantes, contudo necessitariam de uma definição mais concreta das etapas de análise e planejamento para serem comparadas com outras abordagens, ou mesmo com trabalhos de sensoriamento urbano.

6 ANÁLISE DE TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos relacionados sobre sistemas e frameworks autoadaptativos. Foram selecionados trabalhos com características similares à proposta enunciada (tais como componentes de gerenciamento externo e tomada de decisão em loop), cada qual abordado individualmente por ordem cronológica na próxima seção. Ao final é apresentada uma tabela comparando características entre eles.

6.1 Sistemas Autoadaptativos

Rainbow (GARLAN et al., 2004) é um framework que usa uma arquitetura de software composta por uma infraestrutura reutilizável que busca dar suporte à autoadaptação em sistemas de software. Esta infraestrutura implementa um mecanismo externo autoadaptativo com as capacidades de monitorar, detectar, decidir e atuar, com base no próprio conhecimento de adaptação que permite especificar explicitamente estratégias de adaptação para vários problemas de sistema, inclusive para sistemas embarcados e móveis. Para validar a proposta os autores realizaram alguns estudos de caso teóricos, além de terem implementado e avaliado um protótipo.

MADAM (FLOCH et al., 2006) propõe o modelo de um middleware e de ferramentas de suporte para facilitar o desenvolvimento de aplicações adaptativas com capacidade de mobilidade. A principal contribuição do projeto consiste em um middleware que proporciona adaptação dinâmica em aplicações baseadas em componentes, permitindo que esses componentes de middleware genéricos raciocinem sobre o controle de adaptação. Além disso, o trabalho propõe uma metodologia de desenvolvimento orientada a modelo com base em modelos de adaptação abstratos e transformação do modelo em código.

Em (ROUVOY et al., 2009) é proposta uma extensão de MADAM, o MUSIC, o qual utiliza uma estrutura de planejamento baseada em componentes que otimizam a utilidade global das aplicações quando condições relacionadas a contextos ocorrem. Ambos fazem decisões de forma dinâmica baseadas nas preferências dos usuários e QoS, sendo operados automaticamente para otimizar a utilidade do aplicativo em um dado contexto de execução.

Em Chun et al (2013) é proposto um sistema autoadaptativo baseado em agentes para dispositivos inteligentes, o qual utiliza um esquema de ciclo de controle via *feedback* para diminuir as condições de incerteza. Em conjunto com a teoria de controle da engenharia os

autores utilizam um mecanismo de ajuste utilizando o modelo MAPE-K, o qual é executado pelo agente.

O MalacaTiny é um sistema proposto por Ayala, Amor e Fuentes (2012) para ambientes assistidos baseado em agentes incorporando tarefas de autoconfiguração. Tal trabalho busca através da abordagem multiagente tratar a heterogeneidade dos dispositivos e de suas comunicações, além de ter autonomia para assistir o ambiente e dar apoio aos usuários, sua principal função. Além disso, um loop de autoconfiguração em cada agente é utilizado para que consigam se adaptar e interagir com outros agentes visando configurações mais adequadas e redução do tempo de resposta.

Os autores avaliam um estudo de caso onde idosos têm suas atividades monitoradas através do uso de *smartphones* HTC-Desire com sistema operacional Android e sensores SunSPOT. Desta forma, os idosos são monitorados por diversos aspectos a fim de coletar contextos que posteriormente são analisados e considerados pelo ciclo de autoconfiguração

É proposto por Nallur e Bahsoon (2012) um mecanismo de autoadaptação para aplicações orientadas a serviços em nuvem. O trabalho faz uso de heurísticas baseadas em mercado (MBH) com o objetivo de modificar seus parâmetros de qualidade de serviço (QoS) dependendo da demanda. O resultado do processamento de QoS dos serviços é utilizado para, se necessário, escolher outro serviço automaticamente. Apesar de ser um mecanismo muito útil, ele não considera cenários com mobilidade e tolerância a atrasos.

Rosa et al. (2013) propõem um framework que possibilita descrever os comportamentos através de políticas de alto nível. Tais políticas estabelecem os objetivos para os indicadores de desempenho a fim de realizar adaptações baseadas em componente. A técnica utilizada para determinar e avaliar o modelo de adaptação é composta por duas etapas, uma off-line e outra on-line.

Na etapa off-line as regras são geradas especificando adaptações de componentes que podem ajudar a alcançar as metas estabelecidas quando uma determinada mudança no contexto de execução ocorrer. Na etapa online, quando uma alteração ocorre às regras correspondentes são avaliadas para escolher que adaptações devem ser executadas. Apesar desta técnica ser muito útil para casos onde há muitas ações e objetivos a serem avaliados, em casos simples de adaptação não haveria necessidade de tamanho esforço, já que simples regras ECA seriam o suficiente. Além disso, este trabalho não explora a interação com outros componentes de adaptação para alcançar melhores resultados, o que é um diferencial do mecanismo proposto nesta dissertação.

No trabalho apresentado por Dalpiaz, Giorgini e Mylopoulos (2013) é proposta uma arquitetura autoconfigurável para lidar com sistemas sócio-técnicos (STS) que, em suma, consistem da interação entre seres humanos, organizações e sistemas técnicos. Tais elementos, segundo os autores, possuem características heterogêneas, e apresentam mudanças constantes no ambiente operacional de forma inesperada, além de serem fracamente controláveis.

Para tratar tais condições, um ciclo Monitorar-Diagnosticar-Reconciliar-Compensar foi utilizado para *monitorar* de forma contínua os comportamentos do ator (pelo autor, termo representa o tipo de usuário ou entidade que interage) e as mudanças de contexto, bem como, para realizar o *diagnóstico* de falhas e de baixo desempenho. Após, nas etapas de *Reconciliação* e *Compensação*, é verificado se o comportamento monitorado é compatível com os objetivos do ator, encontra uma solução para resolver o problema e realiza uma compensação sobre as ações para conciliar o comportamento real e o desejado.

Além de apresentar a arquitetura conceitual e os algoritmos utilizados, o trabalho de Dalpiaz, Giorgini e Mylopoulos (2013) demonstra resultados a partir da implementação de um protótipo em três diferentes cenários que envolveram diagnóstico de falhas e compensação em casa inteligente, a fim de demonstrar a capacidade da arquitetura de detectar e reagir (autoconfigurando-se). As diversas falhas que ameaçam as exigências dos atores englobam: plano para falhas de dependência, plano para tempo limite de execução e plano para tempo limite de execução do objetivo. Contudo não engloba como responder a falhas dos dispositivos, nem se preocupa com questões de tolerância a atraso via comunicação, desconexão e otimização do upload de dados coletados, como é o caso em sistemas de sensoriamento urbano.

Mizouni et al (2014) apresenta um framework para aplicações móveis autoadaptativas conscientes do contexto. O trabalho foca em utilizar técnicas de SPL (Software Product Line) e modelagem com base em recursos para delimitar as adaptações. Um caso de estudo é sobre uma aplicação de mobile *healthcare* onde diversas características são monitoradas e classificadas em prioridades, as quais servem de requisitos para a tomada de decisão. As decisões são tomadas com base nos recursos disponíveis, estes determinando se características menos críticas podem ser desabilitadas para manter as funções críticas funcionando por maior tempo. Tal propósito não foi abordado pela UrboSenti nos estágios preliminares, apesar de utilizar contextos de baixo nível (os eventos monitorados) para tomada de decisão, a UrboSenti ainda não suportada a consciência de contexto de nível mais alto para fazer uso de representação de contexto e inferência, uma vez que o componente de contexto não foi ainda

projetado para a UrboSenti. Um exemplo disso é o caso considerado por esse trabalho sobre a prioridade e a subutilização de recursos em tempo de execução, o qual utiliza contextos de alto nível. Por outro lado, este trabalho não considera que outros dispositivos móveis possam ser utilizados para realizar interações e conseguir melhores resultados de interações, uso de serviços de terceiros, ou mesmo, utilizar comunicações oportunistas.

Uma abordagem de mecanismo de adaptação para gerar as configurações de reconfiguração em tempo de execução com o foco em sistemas pervasivos móveis foi proposta por Pascual, Pinto e Fuentes (2015). Para tanto, na etapa de análise do modelo MAPE-K , é verificado se o critério (ex.: menor consumo de bateria) foi satisfeito pela nova configuração, senão, é gerado outro plano. Os planos são gerados utilizando algoritmos genéticos onde as populações são definidas pelas configurações possíveis a serem aplicadas. Apesar de ter o foco em ambientes móveis não apresenta questões que indiquem que não possa ser utilizado em outros ambientes com limitações de recursos.

Esta abordagem é promissora, porém este mecanismo somente possibilita tomadas de decisão utilizando funções de utilidade e saídas discretas por dois estados, 0 ou 1, os quais não são aplicáveis a todos os tipos de casos, já que muitas vezes requerem tomada de decisões sem métrica de desempenho, simplesmente resolvidas por regras. Por exemplo, uma configuração de preferência do usuário é trocada implicando o desligamento de uma dada funcionalidade, ou ainda, uma interface troca seu IP de entrada, o qual é enviado a um servidor dada uma regra de autoconfiguração. Apesar disso, um mecanismo de adaptação poderia usar as duas abordagens simultaneamente para tratar de pontos em que cada uma é melhor utilizada. Além disso, o mecanismo de adaptação não considera que a interação entre outros mecanismos possa ocorrer.

6.2 Comparação

Os trabalhos abordados na seção anterior e o mecanismo de autoadaptação proposto nesta dissertação são comparados nesta seção através das seguintes características em comum sobre: realizar adaptação de forma externa dos artefatos gerenciáveis, usarem uma modelagem com base em componentes e tomarem decisão por um ciclo de controle.

Tal comparação é expressa na Tabela 6.1, onde cada linha representa um trabalho relacionado, este especificado na primeira coluna, e com os critérios de comparação dispostos nas demais colunas. Cada critério é detalhado a seguir, onde os valores encontrados na tabela são destacados em negrito para melhor visualização.

Tabela 6.1 - Comparação dos trabalhos relacionados autoadaptativos

Abordagem	*Ciclo*	*Monitoramento*	*Análise*	*Planejamento*	*Mecanismo de Otimização*	*Execução*	*Interação*	*Mobilidade*
Garlan et al (2004)	MDDA	Ambos	ECA	Estático	-	Parâmetros e componentes	Não cita	Não cita
Rouvoy el al (2009)	Modelo Personalizado	Passivo	Utilidade e predição	Dinâmico	Busca configuração com maior utilidade	Parâmetros e componentes	Sim	Parcial
Chun et al (2011)	MAPE-K	Passivo	Utilidade	Dinâmico	Estratégias por busca em árvore	Parâmetros e componentes	Não cita	Não cita
Ayala, Amor e Fuentes (2012)	Modelo Personalizado	Passivo	Objetivos	Estático	-	Parâmetros	Sim	Parcial
Nallur e Bahsoon (2012)	Modelo Personalizado	Ambos	Utilidade	Estático	-	Parâmetros	Sim	Não cita
Rosa et al (2013)	Modelo Personalizado	Passivo	Objetivo KPI	Estático	-	Parâmetros	Não	Não cita
Dalpiaz, Giorgini e Mylopoulos (2013)	MDRC	Passivo	Objetivos	Dinâmico	Busca com heurísticas	Parâmetros e componentes	Sim	Não cita
Mizouni et al (2014)	Modelo Personalizado	Ativo	ECA	Estático	-	Parâmetros	Não cita	Sim
Pascual, Pinto e Fuentes (2015)	MAPE-K	Passivo	Utilidade	Dinâmico	Algoritmo Genético	Parâmetros e componentes	Não cita	Parcial
Mecanismo proposto	MAPE-K	Passivo	ECA	Estático	-	Parâmetros e componentes	Sim	Sim

Fonte: Primária.

Ciclo: A coluna indica qual modelo de ciclo de controle de adaptação é utilizado. A maioria dos trabalhos utiliza **modelos personalizados** de ciclo, isto é, específicos para o caso a ser resolvido. Em seguida o **MAPE-K** (Monitorar, Analisar, Planejar, Executar e Conhecimento) é o segundo mais utilizado, presente em três trabalhos. Por fim, algumas variantes do modelo anterior são utilizadas por dois trabalhos, o **MDRC** (Monitorar, Decidir, Recompensar e Conciliar) e o **MDDA** (Monitorar, Diagnosticar, Decidir e Atuar). Apesar das diferenças de modelo, todos possuem ao menos uma etapa de monitoramento, uma tomada de decisão e a aplicação da decisão (execução), além de uma etapa para verificar se o problema foi resolvido, tratada aqui como sendo o planejamento.

Monitoramento: Indica a forma de monitoramento para dar início à tomada de decisão. Em sua maioria, os trabalhos utilizam o monitoramento **passivo**, o qual recebe eventos gerados por outros software ou componentes. Somente o trabalho de Mizouni et al (2014) utiliza como única abordagem o monitoramento **ativo**, o qual consulta os artefatos gerenciados por si próprio. Além desses, dois trabalhos possibilitavam **ambas** as abordagens.

Análise: Refere-se à técnica para análise ou tomada de decisão, entre funções **ECA** (Evento, Condição e Ação), **utilidade**, **objetivo** e **predição**. Quatro trabalhos utilizam funções de utilidade, três utilizam funções objetivo, três utilizam funções ECA e um utiliza predição para auxiliar na tomada de decisão. O trabalho proposto utiliza a mais simples, ECA, devido aos 4 casos utilizados necessitarem de tomada de decisões simples e com poucas possibilidades de execução e planos. Apesar disso, para futuros casos de adaptação o uso de outra técnica pode ser estudado, ou mesmo o uso de técnicas em conjunto, como no caso de Rouvoy et al (2009), o qual utiliza funções de utilidade e predição.

Planejamento: Indica qual forma de planejamento o trabalho utiliza, entre **estático** e **dinâmico**. Em resumo, com base nos conceitos abordados no Capítulo 2, a forma de planejamento estático utiliza um planejamento com etapas pré-determinadas, estas frequentemente determinadas pela etapa de análise. Por outro lado, o planejamento dinâmico indica que tais etapas podem ser classificadas ou geradas dinamicamente em tempo de execução, sendo especificadas pelo próximo critério.

Mecanismo de otimização: Refere-se ao mecanismo de otimização utilizado pelo trabalho para gerar ou classificar as alternativas de tomada de decisão em tempo de execução. Somente quatro dos trabalhos apresentam suporte a este mecanismo, sendo que o mecanismo proposto pela UrboSenti ainda não suporta um caso de adaptação com planejamento dinâmico, o qual será abordado em trabalhos futuros. Tal planejamento dinâmico seria útil

para determinar configurações próximas do ótimo para o monitoramento adaptativo e preditivo de recursos como localização, movimentação via acelerômetro e desconexão.

Execução: Indica a técnica de implementação das ações a serem aplicadas no ambiente. Todos os trabalhos abordados possibilitam realizar ações, isto é, adaptações no ambiente através da troca de **parâmetros** de controle. Em adicional, seis trabalhos também possibilitam executar adaptações por **componentes**.

Interação: O trabalho aborda interação entre mecanismos de adaptação, sendo: **Não cita**; **Sim** e **não**. Cinco dos trabalhos suportam a interação com outro mecanismo de adaptação, quatro não citam e um cita que não suporta.

Mobilidade: Aponta se o trabalho considera o suporte a *smartphones* na implantação ou em algum caso teórico. Os trabalhos que afirmam que suportam mobilidade e tratamento de desconexão são assinalados com **Sim**; os trabalhos que somente citam que possuem suporte à mobilidade, mas não citam que tratam desconexão, são assinalados como **Parcial**; por fim, os trabalhos que não citam, ou explicitamente afirmam que não suportam, são tratados como **Não cita**. A maioria dos trabalhos apresentados não cita o suporte a dispositivos móveis ou cita apenas de forma parcial. O único trabalho que trata mobilidade e desconexão além do Mecanismo de Autodaptação proposto neste trabalho é o de Mizouni et al (2014).

6.3 Considerações Finais

Em comparação aos trabalhos relacionados elencados, o mecanismo proposto se destaca por possibilitar tanto interações com outros gerentes de adaptação quanto tratamento do caso de mobilidade e desconexão. Em adicional, o cenário de sensoriamento urbano é outro diferencial em relação aos demais, que apesar de poderem ser aplicados no mesmo ambiente, não o foram. Contudo, o mecanismo de adaptação proposto ainda tem muitos pontos em aberto que podem ser melhorados de acordo, tais como:

- Suporte a planejamento dinâmico e um mecanismo de otimização; e
- Estudo do uso de outras técnicas de análise, mais adequadas a casos com gerações de configurações dinâmicas.

Além dessas questões, diversos trabalhos optaram por utilizar a abordagem com base em agentes inteligentes. Tal abordagem está fora do escopo de trabalho desta dissertação, sendo o estudo do impacto, vantagens, desvantagens e oportunidades dela em relação ao modelo clássico MAPE-K um possível trabalho futuro em aberto.

7 CONCLUSÃO

A presente dissertação apresentou um mecanismo de autoadaptação para sensoriamento urbano com base no modelo proposto pela computação autonômica. Tal proposta tomou por base a pesquisa de literatura sobre sistemas de sensoriamento urbano, os quais revelaram diversos casos de autoadaptação extraídos com base nas propriedades self-* fundamentais. Contudo, na análise de tais casos foi verificado que os trabalhos identificados propõem as formas de tratamento, em geral, diretamente nos componentes de negócio, ou, quando externos, considerando poucos contextos de outros componentes. Tais questões e casos serviram de base para propor o uso do modelo MAPE-K no componente de adaptação da arquitetura UrboSenti visando tratá-las a partir dos princípios da Computação Autonômica.

Além disso, foram executadas as seguintes etapas para realizar os objetivos específicos: (1) os principais casos de adaptação que ocorrem em sistemas de sensoriamento urbano foram identificados, sendo no total, sete casos; (2) foi proposto um modelo conceitual para o mecanismo que atendeu tais requisitos; (3) a implementação de um protótipo que possa rodar em diferentes dispositivos foi parcialmente alcançada, uma vez que foram feitas duas implementações da UrboSenti, uma para Java SE e outra para Android, devido a algumas particularidades das bibliotecas disponíveis para cada plataforma; e, por fim, (4) experimentos foram aplicados na implementação do mecanismo proposto a fim de analisar suas capacidades em ambas as plataformas.

Para validação preliminar do mecanismo proposto, o Micro-Kernel e o componente de adaptação da arquitetura UrboSenti foram implementados em duas plataformas, Java SE e Android, conseguindo abranger 4 dos 7 casos de autoadaptação identificados no levantamento bibliográfico. Além disso, experimentos executados buscando verificar a capacidade do mecanismo de processar os eventos e as interações de forma quantitativa revelaram tempos de processamento satisfatórios. Porém, foi identificado durante estes experimentos que há bastante influência da arquitetura de hardware e da escolha da máquina virtual nas diferenças do tempo de processamento entre os resultados.

Ainda, experimentos executados com uma aplicação simulada foram utilizados para avaliar o impacto do componente de adaptação no uso de recursos, primeiro na plataforma Java SE e depois em Android. A implementação em ambas as plataformas exibiram um consumo de memória, CPU e espaço de armazenamento que precisam ser melhorados, através da revisão dos métodos e da escolha de classes utilizadas na implementação, além de um serviço de limpeza para o banco de dados.

Em especial para a implementação em Android, o uso de CPU exibiu valores mais expressivos em comparação ao experimento usando Java SE, desta forma é necessário mais experimentos para identificar as fontes de consumo dos recursos na UrboSenti. Uma das possíveis implicações seria a substituição de métodos e classes Java SE por nativas da plataforma Android. Mesmo assim, apesar de haver uso considerável dos recursos nas duas plataformas, o impacto neste consumo encontrado nos experimentos, utilizando o componente de adaptação habilitado em comparação aos com ele desabilitado, não exibiu grandes mudanças. Desta forma, a implementação da UrboSenti é a responsável por usar a maior parte dos recursos.

Na sequência, a literatura de sistemas autoadaptativos foi comparada com o mecanismo proposto, o que revelou pontos fortes, como o suporte completo à mobilidade e desconexão, e fracos, os quais estão incluídos para serem explorados em trabalhos futuros. Estes pontos em aberto são:

- Implementação dos demais casos de estudo;
- Modelagem formal do componente de conhecimento;
- Estudo de uma forma de remover os registros anteriores de ações, eventos e estados anteriores especificados pelo conhecimento inicial;
- Estudo do uso de *feedbacks* de ações que foram mal sucedidas. Estas poderiam ser utilizadas para análise de dependência de alguma outra ação sendo executada (que também falharia), geração de eventos para agendar uma nova tentativa, ou mesmo a restauração do último estado consistente de todas as ações executadas que foram inconsistentes na execução atual de ações;
- Estudo da implementação para melhor uso de memória no ambiente móvel, evitando consumos desnecessários;
- Suporte a planejamento dinâmico e a um mecanismo de otimização;
- Estudo do uso de outras técnicas de análise, mais adequadas a casos com gerações de configurações dinâmicas;
- Portar para outras plataformas de sistema operacional;
- Estudo da abordagem com base em agentes para autoconfiguração em comparação com o modelo convencional MAPE-K.

REFERÊNCIAS

ABDELZAHER, Tarek et al. Mobiscopes for human spaces. **Center for Embedded Network Sensing**, 2007.

ABOWD, Gregory D. et al. Towards a better understanding of context and context-awareness. In: **Handheld and ubiquitous computing**. Springer Berlin Heidelberg, p. 304-307, 1999.

AUGUSTIN, Iara et al. Isam, a software architecture for adaptive and distributed mobile applications. In: **Computers and Communications, 2002. Proceedings. ISCC 2002. Seventh International Symposium on**. IEEE, p. 333-338, 2002.

AYALA, Inmaculada; AMOR, Mercedes; FUENTES, Lidia. Self-configuring agents for ambient assisted living applications. **Personal and Ubiquitous Computing**, p. 1-11, 2012.

BALDAUF, Matthias; DUSTDAR, Schahram; ROSENBERG, Florian. A survey on context-aware systems. **International Journal of Ad Hoc and Ubiquitous Computing**, v. 2, n. 4, p. 263-277, 2007.

BETTINI, Claudio et al. A survey of context modelling and reasoning techniques. **Pervasive and Mobile Computing**, v. 6, n. 2, p. 161-180, 2010.

BROUWERS, Niels; LANGENDOEN, Koen. Pogo, a middleware for mobile phone sensing. In: **Proceedings of the 13th International Middleware Conference**. Springer-Verlag New York, Inc., 2012. p. 21-40.

BURKE, Jeffrey A. et al. Participatory sensing. **Center for Embedded Network Sensing**, 2006.

CAMPBELL, Andrew T. et al. People-centric urban sensing. In: **Proceedings of the 2nd annual international workshop on Wireless internet.** ACM, 2006. p. 18.

CAMPBELL, Andrew T. et al. The rise of people-centric sensing. **Internet Computing, IEEE**, v. 12, n. 4, p. 12-21, 2008.

CHUN, Ingeol et al. An agent-based self-adaptation architecture for implementing smart devices in smart space. **Telecommunication Systems**, p. 1-12, 2011.

CORNELIUS, Cory et al. Anonysense: privacy-aware people-centric sensing. In: **Proceedings of the 6th international conference on Mobile systems, applications, and services**. ACM, 2008. p. 211-224

CUFF, Dana; HANSEN, Mark; KANG, Jerry. **Urban sensing: out of the woods**. Communications of the ACM. v. 51, n. 3, p. 24-33, 2008.

DA COSTA, Cristiano Andre; YAMIN, Adenauer C.; GEYER, Claudio Fernando Resin. Toward a general software infrastructure for ubiquitous computing. **IEEE Pervasive Computing**, v. 7, n. 1, p. 64-73, 2008.

DALPIAZ, Fabiano; GIORGINI, Paolo; MYLOPOULOS, John. Adaptive socio-technical systems: a requirements-based approach. **Requirements engineering**, v. 18, p. 1-24, 2013.

DAS, Tathagata et al. PRISM: platform for remote sensing using smartphones. In: **Proceedings of the 8th international conference on Mobile systems, applications, and services**. ACM, 2010. p. 63-76.

DOBSON, Simon et al. A survey of autonomic communications. **ACM Transactions on Autonomous and Adaptive Systems (TAAS)**, v. 1, n. 2, p. 223-259, 2006.

DOBSON, Simon et al. Fulfilling the vision of autonomic computing. **Computer**, v. 43, n. 1, p. 35-41, 2010.

EMMANOUILIDIS, Christos; KOUTSIAMANIS, Remous-Aris; TASIDOU, Aimilia. Mobile guides: taxonomy of architectures, context awareness, technologies and applications. **Journal of Network and Computer Applications**, v. 36, n. 1, p. 103-125, 2013.

FLOCH, Jacqueline et al. Using architecture models for runtime adaptability. **Software, IEEE**, v. 23, n. 2, p. 62-70, 2006.

GANEK, Alan G.; CORBI, Thomas A. The dawning of the autonomic computing era**. IBM systems Journal**, v. 42, n. 1, p. 5-18, 2003.

GARLAN, David et al. Rainbow: Architecture-based self-adaptation with reusable infrastructure. **IEEE Computer**, v. 37, n. 10, p. 46-54, 2004.

HUEBSCHER, Markus C.; MCCANN, Julie A. A survey of autonomic computing—degrees, models, and applications. **ACM Computing Surveys (CSUR)**, v. 40, n. 3, p. 7, 2008

IBM, Corporation. An Architectural Blueprint for Autonomic Computing. **IBM White paper**. Ed. 3rd June, 2005.

KHAN, Wazir et al. Mobile phone sensing systems: A survey. **IEEE Communications Surveys & Tutorials**. V. 15, n. 1, p. 402-427, 2012.

KRAMER, Jeff; MAGEE, Jeff. Self-managed systems: an architectural challenge. In: **Future of Software Engineering, 2007. FOSE'07**. IEEE, 2007. p. 259-268.

KRUPITZER, Christian et al. A survey on engineering approaches for self-adaptive systems. **Pervasive and Mobile Computing**, v. 17, p. 184-206, 2015.

KEPHART, Jeffrey O.; CHESS, David M. The vision of autonomic computing. **Computer,** v. 36, n. 1, p. 41-50, 2003.

KEPHART, Jeffrey O.; WALSH, William E. An artificial intelligence perspective on autonomic computing policies. In: **Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on.** IEEE. p. 3-12, 2004.

KWAPISZ, Jennifer R.; WEISS, Gary M.; MOORE, Samuel A. Activity recognition using cell phone accelerometers. **ACM SigKDD Explorations Newsletter**, n. 2, p. 74-82, 2011.

LADDAGA, Robert; ROBERTSON, Paul; SHROBE, Howard E. Self-adaptive software. **Proposer Information Pamphlet BAA**, n. 98-12, 1997.

LALANDA, Philippe; MCCANN, Julie A.; DIACONESCU, Ada. **Autonomic Computing: Principles, Design and Implementation**. Springer Science & Business Media, 2013.

LANE, Nicholas D. et al. Urban sensing systems: opportunistic or participatory?. In: **Proceedings of the 9th workshop on Mobile computing systems and applications**. ACM, p. 11-16, 2008.

LANE, Nicholas D. et al. A survey of mobile phone sensing. **Communications Magazine, IEEE**, v. 48, n. 9, p. 140-150, 2010.

LEE, Kevin et al. Workflow adaptation as an autonomic computing problem. In: **Proceedings of the 2nd workshop on Workflows in support of large-scale science**. ACM. p. 29-34, 2007.

MARQUESAN, Clarissa C. **On the investigation of the joint use of self-* properties and peer-to-peer for network management**. 2010 PhD Thesis in Federal University of Rio Grande do Sul. – Informatics Institute, UFRGS, Porto Alegre .

MCKINLEY, Philip K. et al. Composing adaptive software. **Computer**, v. 37, n. 7, p. 56-64, 2004.

MILUZZO, Emiliano et al. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In: **Proceedings of the 6th ACM conference on Embedded network sensor systems**. ACM, 2008. p. 337-350.

MIZOUNI, Rabeb et al. A framework for context-aware self-adaptive mobile applications SPL. **Expert Systems with applications**, v. 41, n. 16, p. 7549-7564, 2014.

NALLUR, Vivek; BAHSOON, Rami. A Decentralized Self-Adaptation Mechanism for Service-Based Applications in the Cloud. **IEEE Transactions on Software Engineering** 2012.

OH, Hyeong-Seok et al. Evaluation of Android Dalvik virtual machine. In: **Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems**. ACM, 2012. p. 115-124.

OREIZY, Peyman et al. An architecture-based approach to self-adaptive software. **Intelligent Systems and Their Applications, IEEE,** v. 14, n. 3, p. 54-62, 1999

PASCUAL, Gustavo G.; PINTO, Mónica; FUENTES, Lidia. Self-adaptation of mobile systems driven by the Common Variability Language. **Future Generation Computer Systems**, v. 47, p. 127-144, 2015.

RA, Moo-Ryong et al. Medusa: A programming framework for crowd-sensing applications. In: **Proceedings of the 10th international conference on Mobile systems, applications, and services**. ACM, 2012. p. 337-350.

RAMIREZ, Andres J. et al. Applying genetic algorithms to decision making in autonomic computing systems. In: **Proceedings of the 6th international conference on Autonomic computing**. ACM, 2009. p. 97-106.

ROLIM, Carlos et al. Towards a Ubiquitous Service-Oriented Architecture for Urban Sensing. In: **The First International Workshop on Social Computing and Urban Intelligence and Smarter Lives (SMARTLIFE) in The Sixth ASE International Conference on Social Computing (SocialCom)**, 2014.

ROUVOY, Romain et al. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In: **Software engineering for self-adaptive systems**. Springer Berlin Heidelberg, 2009. p. 164-182.

ROSA, Liliana et al. Self-management of adaptable component-based applications. **IEEE Transactions on Software Engineering**, 2013.

SALEIHIE, Mazeiar; TAHVILDARI, Ladan. Self-Adaptative Software: Landscape and Research Challenges. **ACM Transactions on Autonomous and Adaptive Systems (TAAS)**, v. 4, n. 2, p. 14, 2009.

SALIM, Flora; HAQUE, Usman. Urban computing in the wild: A survey on large scale participation and citizen engagement with ubiquitous computing, cyber physical Systems, and internet of Things. **International Journal of Human-Computer Studies**, v. 81, p. 31-48, 2015.

SANCHEZ, Luis et al. SmartSantander: IoT experimentation over a smart city testbed. **Computer Networks**, v. 61, p. 217-238, 2014.

SATYANARAYANAN, Mahadev. Pervasive computing: Vision and challenges. **Personal Communications, IEEE**, v. 8, n. 4, p. 10-17, 2001.

SCHAFFERS, Hans et al. Smart cities and the future internet: Towards cooperation frameworks for open innovation. **Springer Berlin Heidelberg**, 2011.

SCHÖNER, Gregor; DOSE, Michael. A dynamical systems approach to task-level system integration used to plan and control autonomous vehicle motion. **Robotics and Autonomous Systems**, v. 10, n. 4, p. 253-267, 1992.

SHIN, Minho et al. AnonySense: A system for anonymous opportunistic sensing. **Pervasive and Mobile Computing**, v. 7, n. 1, p. 16-30, 2011.

STERRITT, Roy. Autonomic computing. **Innovations in systems and software engineering**, v. 1, n. 1, p. 79-88, 2005.

STRANG, Thomas; LINNHOFF-POPIEN, Claudia. A context modeling survey. In: **Workshop Proceedings. First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004**. 2004.

TESAURO, Gerald et al. A multi-agent systems approach to autonomic computing. In: **Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems**. IEEE Computer Society, v. 1, p. 464-471, 2004.

YADAV, Radhakishan; BHADORIA, Robin Singh. Performance Analysis for Android Runtime Environment. In: **Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on**. IEEE, 2015. p. 1076-1079.

WEI, Edwin JY; CHAN, Alvin TS. CAMPUS: A middleware for automated context-aware adaptation decision making at run time. **Pervasive and Mobile Computing**, 2011.

WEISER, Mark. The computer for the 21st century. **Scientific american**, v. 265, n. 3, p. 94-104, 1991.

WEYNS, Danny; MALEK, Sam; ANDERSSON, Jesper. FORMS: Unifying reference model for formal specification of distributed self-adaptive systems. **ACM Transactions on Autonomous and Adaptive Systems (TAAS),** v. 7, n. 1, p. 8, 2012.

WU, Pang; ZHU, Jiang; ZHANG, Joy Ying. Mobisens: A versatile mobile sensing platform for real-world applications. **Mobile Networks and Applications**, v. 18, n. 1, p. 60-80, 2013.

APÊNDICE A: COMPARAÇÃO MÁQUINAS VIRTUAIS JAVA E ANDROID

Este apêndice destina-se a comparar máquinas virtuais Java, Dalvid e Art em computador com suporte ao conjunto de instruções SSE4.1/4.2. A computador utilizado contém um processador i7-4500U 1.80 GHz, 8GB de memória RAM DDR3 1333MHz e com o sistema operacional Windows 8.1. Cada experimento de eventos internos utilizou um módulo de sensoriamento da UrboSenti e a mesma configuração encontrada na Seção 5.2.2.2, que, recapitulando, são 1000 eventos em 11 execuções. Idealmente esses experimentos poderiam ser aplicados em um dispositivo real para maior precisão e confiabilidade dos resultados, contudo como objetivo do experimento era verificar somente a proporção do tempo de processamento entre a JVM, DVM e o Art, o *overhead* da emulação/virtualização não influenciaria significativamente os resultados. Desta forma, experimentos com dispositivos reais estão em aberto para trabalhos futuros.

Os resultados totais do processamento de todos os experimentos podem ser encontrados na Tabela A e as configurações de máquina virtual utilizadas por experimento podem ser encontradas na seguinte lista:

- JVM 1.7: Foi executado em uma máquina virtual Virtual Box 5.0.4 contendo um Sistema operacional Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-28-generic x86_64) limitado ao uso de 2GB de memória e a um único processador.
- DVM 4.4.2: executado utilizando o emulador disponibilizado pelo SDK do Android. A imagem utilizada foi a Google Intel x86 com Android 4.4.2 e o dispositivo emulado foi a configuração padrão de Nexus 7 (2012) com 1GB de RAM disponível.
- Art 4.4.2: foi executado com a mesma configuração do DVM 4.4.2, apenas modificando o *runtime* para Art.
- Art 5.0.1: foi executado com a mesma configuração do DVM 4.4.2, contudo usando o Art e a imagem Intel Android x86 5.0.1.

A Tabela A apresenta o resumo dos experimentos realizados onde a primeira coluna representa o identificador e a coluna tempos o resultado do processamento dos eventos em milissegundos. Além disso, a coluna "***identificador***" apresenta em VM o resultado calculado dos raw e o cálculo relativo derivado das colunas VM.

O principal objetivo destes experimentos era verificar se há muita diferença entre a JVM e o Art, usados na dissertação. Como pode ser visto o Art 5.0.1 foi 1,93 vezes mais

rápido que o JVM, sendo a diferença de desempenho na Seção 5.2.2.2 causada majoritariamente por causa do hardware. Além disso, a tabela mostra também que a implementação 5.0.1 é mais eficiente que a Dalvik e o próprio Art da versão 4.4.2 do Android.

Tabela A – Comparação do desempenho das máquinas virtuais

Identificador	***Tempos (ms)***			
VM	*Médio*	*Desvio Padrão*	*Mínimo*	*Máximo*
JVM 1.7	27.053	35.2123	12	2077
DVM 4.4.2	28.225	14.9522	16	501
Art 4.4.2	16.283	9.2330	9	346
Art 5.0.1	13.948	6.4224	8	251
Cálculos	*Médio*	*Desvio Padrão*	*Mínimo*	*Máximo*
(DVM 4.4.2 / JVM 1.7) - 100	4,33%	-	33,33%	-75,88%
(DVM 4.4.2 / Art 4.4.2) - 100	73,34%	-	77,78%	44,80%
(JVM 1.7 / Art 4.4.2) - 100	66,15%	-	33,33%	500,29%
(Art 4.4.2 / Art 5.0.1) - 100	16,74%	-	12,50%	37,85%
(JVM 1.7 / Art 5.0.1) - 100	93,96%	-	50,00%	727,49%

Fonte: Primária.

Nota: Além da máquina virtual Android x86 5.0.1 (32 bits) utilizada anteriormente, foi feito um experimento utilizando uma máquina virtual Android x86_64 5.0.1 (64 bits). Contudo, a versão 64 bits apresentou um defeito na precisão dos valores inteiros obtidos a partir do comando Sytem.currentMillis(), onde o último valor representando a unidade dos milissegundos apresentou o valor 0 (zero) em todas as chamadas. Desta forma, este foi descartado da comparação devido à precisão para comparação ser menor, porém, este foi o que apresentou o melhor desempenho, sendo por volta de 10 ms por evento, totalizando em um tempo total de execução por experimento de pouco mais de 10 segundos, em contraste com o Art 5.0.1 x86 que exibiu um tempo médio total por volta de 14 segundos.

Por fim, todos os arquivos *raw* extraídos diretamente do experimento, os arquivos processados e os scripts em R para processamento estatístico podem ser encontrados no mesmo link passado no capítulo de experimento, sendo ele: https://github.com/gaborges/urbosenti-sensing-module-experiments/tree/master/experimentos.