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

LEONARDO VIANNA DO NASCIMENTO

Uma arquitetura multiagente para integração de fontes de contexto em cidades inteligentes

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação

Orientador: Prof. Dr. José Palazzo M. de Oliveira

Porto Alegre
2023

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Nascimento, Leonardo Vianna do

Uma arquitetura multiagente para integração de fontes de contexto em cidades inteligentes / Leonardo Vianna do Nascimento. – Porto Alegre: PPGC da UFRGS, 2023.

137 f.: il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2023. Orientador: José Palazzo M. de Oliveira.

1. Integração de fontes de dados. 2. Obtenção de Contexto. 3. Sistemas sensíveis ao contexto. 4. Sistemas multiagentes. 5. Cidades inteligentes. I. José Palazzo M. de Oliveira, . II. Título.

RESUMO

Cidades inteligentes são um dos domínios emergentes para aplicações computacionais. Atualmente, muitos sensores físicos e serviços na Web tem sido disponibilizados nestes ambientes ou relacionados a eles. Muitas dessas aplicações podem se beneficiar dos dados providos por esses sensores e serviços, juntamente com o paradigma de computação ubíqua, para fornecer melhores serviços aos cidadãos dessas cidades. Um aspecto importante desses aplicativos é como obter esses dados e entendê-los. Abordagens baseadas no contexto provaram ser sucesso na compreensão desses dados. Essas soluções se utilizam de arquiteturas de reconhecimento de contexto para inferir informações contextuais de nível mais alto. Cidades inteligentes são ambientes onde dados contextuais estão presentes em grandes quantidades e de forma distribuída. Além disso, cidades inteligentes apresentam um ambiente dinâmico onde falhas podem acontecer. Portanto, é importante que arquiteturas propostas para obtenção de contexto lidem com estas questões, além de prover mecanismos de descentralização e fusão de dados. Este trabalho apresenta uma nova arquitetura para obtenção de contexto em cidades inteligentes, a partir da integração de fontes de dados distribuídas. A arquitetura é distribuída, descentralizada e tolerante a falhas, apresentando mecanismos de fusão de dados e composição dinâmica de seleção de fontes de contexto. Ao apresentar essas cinco características simultaneamente, a arquitetura mostra-se com um diferencial em relação a outros trabalhos relacionados desenvolvidos anteriormente. A arquitetura desenvolvida utiliza o paradigma de multiagentes, que inerentemente são distribuídos e facilitam a descentralização por serem compostos por agentes autônomos. Um cenário foi utilizado para execução de diversos experimentos que demonstram que a arquitetura pode ser utilizada com sucesso para obtenção de dados de contexto de forma transparente por aplicações quaisquer.

Palavras-chave: Integração de fontes de dados. Obtenção de Contexto. Sistemas sensíveis ao contexto. Sistemas multiagentes. Cidades inteligentes.

A multi-agent architecture to integrate context data sources in smart cities

ABSTRACT

Smart cities are one of the emerging domains for computational applications. Currently, many physical sensors and Web services have been made available in these environments or related to them. Many of these applications can benefit from the data provided by these sensors and services, along with the ubiquitous computing paradigm, to provide better services to the citizens of these cities. An essential aspect of these applications is how to get and make sense of this data. Context-based approaches have proven successful in understanding these data. These solutions use context-aware architectures to infer higher-level contextual information. Smart cities are environments where contextual data is present in large amounts and in a distributed way. Furthermore, smart cities present a dynamic environment where failures can happen. Therefore, it is crucial that proposed architectures for obtaining context deal with these issues and provide mechanisms for decentralization and data fusion. This work presents a new architecture for obtaining context in smart cities by integrating distributed data sources. The architecture is distributed, decentralized, and fault-tolerant, featuring data fusion mechanisms and dynamic composition of context source selection. By presenting these five characteristics simultaneously, the architecture shows an improvement concerning other related works previously developed. The developed architecture uses the multi-agent paradigm, which is inherently distributed and facilitates decentralization by being composed of autonomous agents. A scenario was used to perform several experiments demonstrating that the architecture can obtain context data transparently by any application.

Keywords: Data source integration, Context access, Context-aware systems, Multiagent systems, Smart cities.

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
FIPA	*Foundation for Inteligent Physical Agents*
FIPA-ACL	*FIPA Agent Communication Language*
FIPA-SL	*FIPA Semantic Language*
HMM	*Hidden Markov Model*
IMU	*Inertial Measurement Unit*
IoT	*Internet of Things* (Internet das Coisas)
JADE	*Java Agent DEvelopment Framework*
JSON	*JavaScript Object Notation*
KNN	*K-Nearest Neighbors*
OSGi	*Open Services Gateway Initiative*
QoC	*Quality of Context* (Qualidade de Contexto)
SVM	*Support Vector Machines*
SWRL	*Semantic Web Rule Language*
TIC	Tecnologias de Informação e Comunicação
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Atualmente, a maior parte da população mundial vive em cidades (United Nations, 2018). Tornar esses ambientes mais eficientes e inteligentes é, portanto, de suma importância. Não é por acaso que um dos domínios emergentes para aplicações computacionais seja o de *cidades inteligentes*. Uma cidade inteligente permite o "uso de tecnologias de computação para tornar os componentes de infraestrutura crítica e serviços de uma cidade - que incluem administração, educação, saúde, segurança pública, imobiliário, transporte e serviços públicos - mais inteligentes, interconectados e eficientes" (ALAM et al., 2017). Essa tecnologia inclui o uso de inúmeros sensores de diferentes tipos, dispositivos móveis como smartphones, meios de transmissão, protocolos de rede, etc.

Iniciativas para o desenvolvimento de sistemas no âmbito de cidades inteligentes já foram propostas, incluindo uma gama extensa de serviços tais como transporte (DJAHEL et al., 2014), controle de tráfego (BARBA et al., 2012), poluição (VAKALI; ANTHOPOULOS; KRCO, 2014), coleta de lixo (PERERA et al., 2014b), saúde (HUSSAIN et al., 2015), segurança pública (GALACHE et al., 2014), água (GONZÁLEZ; DÍAZ et al., 2015), energia (YAMAMOTO et al., 2014) e gerenciamento de emergências (ASIMAKOPOULOU; BESSIS, 2011). Tais sistemas devem lidar com condições dinâmicas, pois a situação de cidades e seus usuários muda frequentemente. Portanto, muitas dessas aplicações e serviços podem fornecer melhores resultados ao utilizar informações de *contexto*.

Por exemplo, um sistema de recomendação (RESNICK et al., 1994) em cidades inteligentes pode considerar informações de contexto, além das informações do usuário e dos itens a serem recomendados. Os recursos a serem recomendados muitas vezes estão espalhados pelo ambiente e o conhecimento de informações contextuais como localização e tempo são necessárias para que a recomendação seja efetiva. Um exemplo é a recomendação de locais turísticos. Nesse cenário a informação de localização do usuário e dos locais a serem visitados, bem como o tempo disponível do usuário são algumas informações contextuais que devem ser levadas em conta para que a recomendação seja efetiva. Sistemas de recomendação cientes de contexto apresentam as características desejadas para esse tipo de tarefa (ADOMAVICIUS et al., 2005).

Outros ambientes inteligentes podem se beneficiar se solucionais baseadas em contexto construídas em cidades inteligentes. Por exemplo, um *campus* universitário é um ambiente similar ao de cidades, porém em uma dimensão menor. Muitas soluções

desenvolvidas para cidades inteligentes podem também ser aplicadas em *campi* universitários (VILLEGAS-CH; PALACIOS-PACHECO; LUJÁN-MORA, 2019; FORTES et al., 2019).

Um aspecto importante de sistemas sensíveis ao contexto diz respeito a como as informações sobre o contexto de seus usuários são obtidas e como são identificadas. Este processo é conhecido como *identificação* ou *reconhecimento de contexto*. Dados contextuais são normalmente obtidos através de sensores, instalados no próprio ambiente ou carregados pelo próprio usuário (por exemplo em um smartphone). Estes dados são processados e analisados para se obter informações de mais alto nível sobre localização do usuário, atividades (o que o usuário está fazendo no momento), tempo e etc. (VILLEGAS; MÜLLER, 2010).

Soluções em cidades inteligentes apresentam fontes de dados naturalmente distribuídas, seja através de sensores fisicamente distribuídos e/ou serviços espalhados por servidores na Web. Em virtude disso, soluções para reconhecimento de contexto devem levar em consideração a distribuição espacial dessas fontes de dados. Além disso, conceitos como *fog computing*, *edge computing* e *cloud computing* permitem que os recursos computacionais para processamento dessas dados também estejam distribuídos. Esses conceitos tem sido utilizados com frequência em soluções para cidades inteligentes (KHATOUN; ZEADALLY, 2016).

Um problema em se desenvolver aplicações sensíveis ao contexto em ambientes tão vastos e complexos como cidades inteligentes é como integrar tais fontes de contexto? Integrar fontes de contexto consiste em torná-las acessíveis a consumidores de tais informações. Por exemplo, uma determinada aplicação pode recomendar qual o meio de transporte mais eficiente para uma pessoa chegar a determinado ponto em uma cidade. Para tanto, tal sistema precisa ter acesso a informações contextuais como localização da pessoa, meios de transporte disponíveis, sua localização, estado atual do trânsito até o local desejado, etc. As fontes fornecedoras de tais informações precisam estar disponíveis para a aplicação de alguma forma.

Alguns problemas envolvidos em processos de integração como esse incluem:

- tornar os dados acessíveis, em uma estrutura tão diversa e distribuída (RAGHAVAN et al., 2020);
- dados fornecidos por fontes diferentes podem ser estruturados de formas diferentes (por exemplo, XML, JSON, CSV, etc.) e normalmente existe ausência de um padrão semântico para os termos utilizados (RAGHAVAN et al., 2020);

- mecanismos de acesso a contexto devem adaptar-se a falhas e outras mudanças ocorridas, como inclusão de novas fontes de dados (YÜRÜR et al., 2016);
- as informações de contexto são heterogêneas, porém devem existir mecanismos de padronização de forma a tornar qualquer recurso de contexto manipulável (YÜRÜR et al., 2016).

1.1 Questão de Pesquisa

É possível e viável integrar diferentes fontes de contexto em uma cidade inteligente de forma que uma aplicação possa acessar tais informações de qualquer lugar, de maneira transparente, tolerante a falhas, e que seja escalável/extensível?

1.2 Objetivos

Com base no exposto, esta tese tem como **objetivo geral** o desenvolvimento de uma arquitetura multiagente para integração de fontes de contexto em cidades inteligentes. Com isso acredita-se ser possível aproveitar características distribuídas e sociais de sistemas multiagentes para construir uma arquitetura distribuída, descentralizada, escalável, dinâmica, tolerante a falhas para processamento de contexto de usuários, permitindo que aplicações quaisquer possam ter acesso a esses dados de forma transparente. A ênfase na presença dessas cinco características simultaneamente é um diferencial deste trabalho com relação a outros similares desenvolvidos anteriormente.

Cabe ressaltar aqui que considera-se neste trabalho como fonte de contexto, qualquer dispositivo físico (hardware) ou lógico (software) que pode produzir informação considerada como contexto. Ou seja, uma fonte de contexto nada mais é do que um fornecedor de tais informações. Assim, este trabalho não está focado na integração da informação contextual em si, mas em integrar fontes de tais informações.

Os **objetivos específicos** consistem em: desenvolvimento uma arquitetura multiagente para compartilhamento de informações de contexto; elaboração de uma ontologia para contexto em cidades inteligentes; desenvolvimento de um protótipo de *framework* para integração de fontes de contexto em ambientes inteligentes baseado na arquitetura; avaliar e validar a arquitetura baseado em experimentos utilizando o protótipo.

1.3 Método de Pesquisa

Para responder à questão de pesquisa levantada e atingir os objetivos apresentados, foi necessário perpassar por um conjunto de etapas iterativas e interativas, ou seja, muitas vezes foi necessário dar um passo atrás, repensar, reimplementar, e agir novamente. Sendo assim, primeiramente, foram realizados dois mapeamentos sistemáticos da literatura, visando compreender melhor a área e identificar as oportunidades de pesquisa envolvendo cidades inteligentes e processamento de contexto. Os mapeamentos sistemáticos encontram-se publicados em Nascimento et al. (2021) e Nascimento and Oliveira (2021).

Após ampla pesquisa e estudo dos trabalhos obtidos, obteve-se um panorama das abordagens em relação ao tema proposto. Com isso, foram identificados as técnicas de reconhecimento de contexto utilizadas, padrões de arquitetura utilizados e como contexto é modelado. Desses, optou-se por aprofundar o problema de integração de fontes de contexto e cidades inteligentes, pois ainda é um ponto a ser amadurecido na literatura.

Depois de definidos o problema e o escopo de pesquisa, foram estudadas diferentes possibilidades para organização da arquitetura. A abordagem de sistemas multiagentes foi escolhida. Esses sistemas apresentam uma estrutura inerentemente distribuída e são capazes de reagir a mudanças no ambiente, lidando naturalmente com dinamismo, heterogeneidade e imprevisibilidade (JENNINGS, 2001). Outra característica interessante é a possibilidade de se decompor um problema de reconhecimento de contexto de forma escalável, em termos de organizações descentralizadas e distribuídas de agentes (DEMAZEAU; COSTA, 1996).

Também foi realizado um estudo sobre modelagem de contexto em geral. Decidiu-se seguir utilizar uma modelagem própria e a opção por ontologias foi escolhida por ser amplamente utilizada na área de contexto. Uma ontologia foi então desenvolvida para modelagem de contexto.

A partir de então partiu-se para o desenvolvimento do protótipo de *framework*. Este *framework* foi utilizado então para a implementação e execução de experimentos com o fim de avaliar e validar a arquitetura desenvolvida.

1.4 Organização do Texto

O restante deste texto está organizado como segue. O Capítulo 2 apresenta a definição de contexto utilizada nesta tese assim como diversos conceitos importantes sobre este tema e sobre cidades inteligentes. Também é apresentado como contexto é tratado nestes ambientes. O Capítulo 3 apresenta conceitos e padrões importantes sobre arquiteturas de software utilizados neste trabalho. O Capítulo 4 apresenta os trabalhos relacionados, evidenciando a oportunidade de pesquisa identificada e explorada nesta tese. O Capítulo 5 apresenta aspectos de sistemas multiagentes importantes para se compreender a arquitetura desenvolvida. O modelo de contexto e a arquitetura em si são apresentados no Capítulo 6. O Capítulo 7 descreve em detalhes as principais operações relacionadas a contexto existentes na arquitetura. O Capítulo 8 apresenta aspectos da implementação do *framework* e da avaliação experimental do trabalho. Por fim, o Capítulo 9 apresenta a conclusão, apontando para os próximos passos da pesquisa bem como pontuando os resultados obtidos e demais atividades realizadas durante o período de doutorado.

2 CONTEXTO E CIDADES INTELIGENTES

A abordagem desenvolvida neste documento está diretamente relacionada às áreas de sistemas sensíveis ao contexto e cidades inteligentes. Por esse motivo, este capítulo apresenta um breve apanhado sobre cada uma delas e como contexto é tratado em cidades inteligentes.

2.1 Definição de Contexto

Contexto é um objeto de estudo de pesquisadores em diferentes áreas da Ciência da Computação. Cada uma destas áreas apresenta uma visão diferente sobre a definição do que seja contexto. A conceituação desse termo varia grandemente quando consideramos sua aplicação em diferentes domínios (BAZIRE; BRÉZILLON, 2005). A definição mais utilizada na literatura científica é a apresentada por Abowd et al. (1999):

> **Contexto** é qualquer informação útil para se caracterizar a situação de uma entidade (uma pessoa, objeto ou lugar) que pode afetar a interação entre usuários e sistemas.

Dessa forma, contexto é visto como uma informação que caracteriza uma entidade e que pode ser usada por sistemas para algum fim, seja na solução de um problema em si ou na adaptação e melhoria da interação com seus usuários. Santos (2008) confirma essa afirmação ao enfatizar que contexto existe apenas quando relacionado a outra entidade, sendo composto por um conjunto de itens (conceitos, regras, proposições) associados a tal entidade. Tais itens podem ser considerados parte do contexto de uma entidade somente se eles são úteis para a resolução do problema sendo tratado.

Santos (2008) nomeia itens que compõe um determinado contexto como *elementos contextuais*:

> Um **elemento contextual** é qualquer porção de dados ou de informação que permite caracterizar uma entidade em um determinado domínio.

Dessa forma, o contexto da interação entre um agente e uma aplicação pode ser redefinido como um conjunto de elementos contextuais instanciados que servem para apoiar a tarefa atual. Entende-se por *agente* tanto agentes humanos quanto agentes de software.

Um sistema de computação que utiliza contexto é chamado de *sistema sensível ao contexto*. Abowd et al. (1999) também define que:

> Um **sistema sensível ao contexto** é aquele que se utiliza de contexto para fornecer informações e/ou serviços relevantes para seus usuários.

Exemplos de aplicações sensíveis ao contexto incluem:

- uma aplicação em um *smartphone* que recomenda restaurantes a um usuário baseado em sua localização;
- uma aplicação para automação residencial que desliga a TV quando ninguém é detectado na sala de estar depois de certo tempo;
- uma aplicação de cronômetro em um *smartphone* pode iniciar automaticamente quando detecta que seu usuário está correndo.

Em cada exemplo, é possível perceber o uso de informações de contexto, como comentado abaixo.

- No primeiro exemplo é possível perceber que a informação *localização* caracteriza a entidade *usuário do smartphone*. Além disso, essa informação é utilizada pela aplicação em questão para recomendar restaurantes. Portanto, a localização é um elemento contextual que compõe o contexto desse usuário.
- No segundo exemplo, o dado sobre a presença ou não de pessoas é uma característica importante da sala de estar da residência para a aplicação de automação residencial. Portanto, esse dado compõe um elemento do contexto da sala de estar.
- Por fim, no terceiro exemplo, a atividade desempenhada pelo usuário (estar correndo ou não) é uma característica importante utilizada pela aplicação de cronômetro. Dessa forma, essa atividade é um elemento contextual que faz parte do contexto do usuário da aplicação.

Informações não relevantes devem ser abstraídas do contexto de uma determinada entidade em uma aplicação. Por exemplo, a informação de que o usuário está correndo ou não pode não ser relevante para a aplicação de automação residencial utilizada como exemplo anteriormente, não sendo parte do contexto utilizado por esse sistema.

2.2 Categorização de Contexto

É comum que informações de contexto sejam classificadas de acordo com certas categorias ou tipos. Categorizar contexto permite que projetistas de aplicações computacionais identifiquem mais facilmente quais elementos contextuais serão úteis em suas aplicações.

Diversos esquemas de categorização de informações de contexto já foram pro-

postos na literatura. O trabalho de Bunningen, Feng and Apers (2005) apresenta uma interessante classificação para tais esquemas em dois tipos:

- *Categorização Operacional*, cujo objetivo é categorizar informações de contexto baseado em como elas foram adquiridas, modeladas e tratadas.
- *Categorização Conceitual*, onde informações de contexto são categorizadas com base no significado e nos relacionamentos conceituais.

Perera et al. (2014a) apresenta um esquema de categorização operacional onde informações de contexto são classificadas em dois tipos:

- *Contexto Primário*: qualquer informação obtida sem o uso de outras informações de contexto existentes e sem a aplicação de operações de fusão de dados. Exemplos: latitude obtida a partir de uma leitura em um leitor GPS, um valor de aceleração obtido a partir de um acelerômetro.
- *Contexto Secundário*: qualquer informação de contexto computada a partir de informações de contexto primário. Informações deste tipo incluem aquelas obtidas através de operações de fusão de dados, inferência ou operações de consulta a bases de dados.

Outro esquema de categorização operacional é apresentado por Bettini et al. (2010). Nesse esquema, informações de contexto foram classificadas em três níveis:

- *Baixo Nível*: dados coletados diretamente de sensores no ambiente.
- *Nível Intermediário*: informações de contexto são medidas ou inferidas de forma a ser agregada, a fim de representar uma informação com um nível mais alto de abstração. Exemplos de contexto intermediário incluem: atividades como correndo ou caminhando, derivadas de dados de aceleração de baixo nível; lugares como "em casa"ou "restaurante", derivados de coordenadas geográficas.
- *Alto Nível*: a informação de contexto é agregada com relações semânticas a fim de relaciona-la com outras informações, como tempo, localização, etc.

Há também uma gama de trabalhos que propõe diferentes esquemas de categorização conceituais. O trabalho de Schilit, Adams and Want (1994) lista três aspectos importantes do contexto de uma entidade: *onde ela está* (localização), *quem ela é* (identidade) e *quais recursos* estão nas proximidades (contexto de interesse). Abowd et al. (1999) define quatro principais categorias para contexto: *localização* (onde a entidade se encontra), *identidade* (quem é a entidade), *atividade* (que atividade está realizando) e

tempo (quando certos eventos acontecem). Por fim, Villegas and Müller (2010) identifica cinco categorias de contexto: *individual* (informações relacionadas a entidades individualmente), *localização*, *tempo*, *atividade* e *relacional* (informações referentes a relacionamentos entre entidades).

Um trabalho anterior do autor deste trabalho propôs um novo esquema de categorização conceitual, reunindo elementos já apresentados em propostas anteriores na literatura (NASCIMENTO et al., 2021). Tal esquema foi construído a fim de sistematizar pontos fortes de diferentes trabalhos anteriores. O esquema proposto é composto por seis categorias descritas na Tabela 2.1.

Tabela 2.1: Esquema de categorização conceitual proposto pelo autor deste trabalho.

Categoria	Descrição	Exemplos
Individual	Inclui o que pode ser observado sobre uma entidade.	Preferências, perfil, informações de identificação.
Localização	Especificação de onde uma entidade está situada no espaço.	Coordenada geográfica (latitude/longitude), endereço, descrição semântica do local (supermercado, loja, restaurante, escola, etc.)
Atividade	Tarefas ou ações desempenhadas pelas entidades.	Caminhando, correndo, parado, em um veículo em movimento.
Temporal	Informação sobre a duração de atividades ou quando determinados eventos ocorreram.	Duração de uma atividade, data e hora de um compromisso.
Relacional	Relacionamentos entre entidades.	Relacionamentos em uma rede social
Contexto de Interesse	Outras informações relacionadas ao ambiente ao redor da entidade que podem ser de seu interesse.	Descrição de objetos, sons, condições climáticas, recursos de interesse próximos.

Fonte: O Autor

É difícil conceber um esquema de categorização de contexto ideal. Uma combinação de diferentes esquemas de categorização pode permitir que os pontos fortes de cada esquema sejam complementados e pontos fracos mitigados (PERERA et al., 2014a). Por exemplo, uma posição geográfica formada por coordenadas de latitude e longitude obtida

a partir de um leitor GPS pode ser classificada como contexto primário de baixo nível, correspondente à localização de uma entidade. Outro exemplo é a ação desempenhada por uma pessoa (caminhando, parada, deitada, correndo, ...) obtida a partir de um mecanismo de inferência que utilizada dados de diferentes sensores como entrada. Nesse caso, a informação obtida pode ser categorizada como atividade e como contexto secundário de nível intermediário. Dessa forma, neste trabalho espera-se que cada informação de contexto pode ser categorizada através de categorias especificadas em diferentes esquemas e que são complementares.

Informações de contexto podem ser reunidas a fim de se obter uma *situação*. Uma situação pode ser vista como um estado composto por um conjunto de contextos que se mantém estáticos em um certo intervalo de tempo, de forma a prover alguma informação válida (GUNDERSEN, 2013; FLEISCHMANN, 2012). Qualquer mudança nesse estado gera uma nova situação. Situações fornecem uma informação de altíssimo nível sobre um usuário, como, por exemplo, *trabalhando no escritório*, *viajando para São Paulo*, *dirigindo para casa no final da tarde*.

2.3 Ciclo de Vida de Informações de Contexto

A informação de contexto, assim como qualquer conjunto de dados manipulados por aplicações de software, possui um ciclo de vida. Este ciclo de vida mostra onde os dados são gerados e onde são consumidos dentro de sistemas sensíveis ao contexto. O ciclo de vida de informações de contexto consiste de quatro fases (PERERA et al., 2014a):

- *Aquisição de contexto*, que diz respeito à obtenção de dados de contexto a partir de diferentes fontes. Estes dados podem ser obtidos diretamente de sensores de hardware, através de um *middleware* (que neste caso funciona como um intermediário entre os sensores e os consumidores de dados de contexto) ou através de um servidor de contexto (dados de contexto são obtidos de diversos outros fornecedores como bancos de dados, dados abertos, web services, etc.).
- *Modelagem de contexto*, onde o contexto obtido é representado segundo um modelo. Existem diversas técnicas já propostas na literatura científica para modelagem de contexto. As mais populares são: *modelagem chave-valor* (contexto é modelado como pares chave-valor em diferentes formatos como arquivos texto e

binários), *modelagem em esquema de marcação* (contexto é modelado na forma de *tags*, utilizando tecnologias como XML ou JSON), *modelagem gráfica* (contexto é representado com relacionamentos utilizando abordagens como UML ou modelos relacionais de bancos de dados), *modelagem baseada em objetos* (contexto é representado como conceitos em um modelo orientado a objetos), *modelagem baseada em lógica* (contexto é representado na forma de fatos, expressões e regras em estruturas e linguagens lógicas) e *modelagem baseada em ontologia* (contexto é organizado em ontologias usando tecnologias semânticas). Algumas abordagens utilizam representações híbridas de contexto, aplicando diferentes técnicas de acordo com diferentes níveis (MACHADO et al., 2018).

- *Inferência ou reconhecimento de contexto* é o processo onde novas informações de contexto são deduzidas a partir das informações existentes. Este processo pode envolver diversas etapas, como pré-processamento, fusão de dados e aplicação de técnicas de inferência. Diversas técnicas de inferência já foram propostas na literatura como técnicas de aprendizado supervisionado (árvores de decisão, redes bayesianas, redes neurais, etc.), aprendizado não supervisionado (kNN, kMeans, etc.), regras, lógica fuzzy, lógica probabilística (Dempster-Shafer, HMM, etc.) e raciocínio baseado em ontologias.
- *Distribuição de contexto*, onde as informações de contexto são disponibilizadas para aplicações consumidoras. Uma das formas de disponibilizar estas informações é através de consultas, onde os consumidores realizam requisições na forma de consultas que são utilizados por provedores de contexto para produzir os resultados. Outra forma de disponibilizar informações de contexto é através de um esquema de publicação/inscrição, onde um consumidor pode se inscrever em um provedor de contexto e obter informações periodicamente ou quando determinado evento ocorrer.

2.4 Fontes de Informação de Contexto

Existem diferentes fontes de informação que podem prover dados de contexto sobre uma determinada entidade. O trabalho de Perera et al. (2014a) faz uma análise bastante extensa sobre sensibilidade ao contexto em sistemas computacionais, identificando as seguintes fontes de informações de contexto:

- *Sensores físicos*, fontes tangíveis de contexto, compostos por dispositivos físicos de hardware, produzindo dados por si mesmos. Os dados produzidos por essas fontes correspondem a contexto primário de baixo nível. Esses dados podem ser obtidos através de comunicação direta com o sensor através de APIs (bastante usado para obtenção de dados de sensores instalados localmente, como no caso de *smartphones*) ou através de *middlewares*, que permitem acesso a dados de sensores indiretamente (o *middleware* acessa os sensores diretamente e torna disponível tais dados para agentes consumidores).
- *Sensores virtuais*, que não produzem dados de contexto necessariamente por si mesmos. Estes sensores podem obter dados de diversas fontes e disponibilizá-los como dados de um único sensor. Este tipo de sensor não possui uma presença física e muitas vezes utiliza tecnologias como *web services* para enviar e receber dados. Exemplos de tais fontes de dados incluem calendários, bases de dados, mensagens em redes sociais, etc.
- *Sensores Lógicos*, também chamados de *sensores de software*, combinam sensores físicos com sensores virtuais a fim de produzir informação mais significativa. Por exemplo, um *web service* que disponibiliza informações sobre previsão do tempo pode ser chamado de um sensor lógico. Milhares de sensores e estações meteorológicas usam sensores físicos para obter dados sobre o clima. Dados de sensores virtuais como mapas, calendários, e bancos de dados históricos também são consultados. Finalmente, informações de alto nível sobre o clima são produzidas ao se combinar dados de sensores físicos com os dados de sensores virtuais. Portanto, dados providos por esses sensores normalmente são derivados de outros fontes de dados a partir da aplicação de operações computacionais, tais como simples chamadas a *web services*, até operações mais complexas como funções matemáticas e estatísticas ou a aplicação de mecanismos de inferência.

2.5 Qualidade de Contexto

Abordagens para gerenciamento de contexto devem lidar com situações onde os dados obtidos podem ser imprecisos. Dados de sensores são passíveis de imperfeição, dadas suas limitações técnicas, disponibilidade e possíveis defeitos de funcionamento(ALSHARGABI; SIEWE; ZAHARY, 2017). Diversas outras circunstâncias podem contribuir

para a imperfeição desses dados, destacando-se:

- o ambiente onde estes sensores se encontram pode ser altamente dinâmico e acontecimentos inesperados podem afetar a leitura dos sensores;
- a alta diversidade de sensores pode levar à falta de harmonia entre leituras de diferentes sensores (leituras ambíguas);
- como sistemas abertos, sistemas ubíquos são mais suscetíveis a ataques;
- sensores capturam dados de contexto periodicamente, porém alguns eventos podem ser perdidos entre intervalos de leitura.

Além disso, quando lidamos com mecanismos de inferência de contexto intermediário ou de alto nível, estes podem apresentar maior ou menor grau de precisão e acurácia. Mais ainda, estes mecanismos são treinados utilizando um subgrupo de contextos possíveis e não são adaptados a lidar com todos os tipos de situações que possam ocorrer.

Informações de contexto imprecisas podem levar uma aplicação a falhar em atingir seus requisitos funcionais e não-funcionais. Sérios problemas podem ocorrer devido às decisões erradas tomadas de acordo com a informação incorreta de contexto fornecida por um sistema.

Considerando esse cenário, é importante que mecanismos de garantia de qualidade de informações de contexto sejam utilizados por aplicações, de forma a minimizar o impacto da imprecisão nos dados. Em virtude disso, o conceito de *Qualidade de Contexto* (*QoC - Quality of Context*) foi proposto, a fim de mensurar a qualidade de qualquer informação contextual. Uma definição mais abrangente e precisa sobre o termo é dada no artigo de Manzoor, Truong and Dustdar (2014), onde QoC *indica o grau de conformidade do contexto coletado/inferido com a situação predominante no ambiente e aos requisitos de um consumidor de contexto particular*.

Algumas métricas de QoC já foram propostas na literatura (AL-SHARGABI; SIEWE; ZAHARY, 2017; MANZOOR; TRUONG; DUSTDAR, 2014). Essas métricas são sumarizadas na Tabela 2.2. Tais métricas podem ser classificadas em dois tipos: *parâmetros objetivos* (que são independentes de requisitos da aplicação consumidora do contexto) e *parâmetros subjetivos* (dependentes de requisitos não-funcionais da aplicação).

Tabela 2.2: Principais métricas de QoC.

Métrica	Tipo	Definição
Reliability	Objetiva	Probabilidade do contexto ser verdadeiro de acordo com as limitações de alcance dos sensores
Timeliness	Objetiva/Subjetiva	Indica o grau de validade/atualização do contexto, considerando quando este foi produzido e seu tempo de validade (que pode ser definido pela própria aplicação).
Accuracy	Objetiva	Indica o quanto a informação de contexto fornecida pode estar próxima do real.
Resolution	Objetiva	Indica o quão precisa é uma informação de contexto, relacionada diretamente com sua granularidade.
Completeness	Objetiva	Indica o quão completa é a informação fornecida, ou seja, a quantidade de informação provida pelo contexto fornecido.
Security Level	Subjetiva	Indica o quanto o proprietário da informação de contexto permite que o consumidor acesse dessa informação.
Significance	Subjetiva	Indica o quanto uma informação de contexto é valiosa/significativa para uma aplicação.
Usability	Subjetiva	Indica o quanto uma informação de contexto é útil para uma aplicação, dados os seus requisitos de uso.
Confidence	Objetiva	Aplicada a contextos interpretados e situações, resultado da combinação do grau de confiança de cada elemento contextual utilizado para inferir a informação em questão.

Fonte: O Autor

2.6 Históricos de Contexto

Algumas aplicações necessitam não somente do contexto atual de uma entidade mas também de informações passadas de contexto. Dados históricos de contexto podem ser usados para identificar tendências e também para prever futuros valores relacionados a contexto. Exemplos de aplicações que utilizam histórico de contexto incluem Rosa, Barbosa and Ribeiro (2016), Aranda et al. (2021) e Martini et al. (2021).

Algumas questões surgem ao se manipular históricos de contexto (DEY; ABOWD; SALBER, 2001):

- *Armazenamento.* Um ponto fundamental para manipulação de um histórico de contexto é a possibilidade de armazenar tais informações. Sem armazenamento, tais análises históricas não podem ser efetuadas.
- *Leitura.* Dados devem ser obtidos de fontes de contexto mesmo quando não há

consumidores interessados nessas informações? Futuras aplicações podem estar interessadas nessas informações que na atualidade não são necessárias. Como conciliar esses acessos com questões como economia de energia e de recursos? Assim, definir quando informações de contexto serão lidas e armazenadas é uma questão importante relacionada ao armazenamento de dados históricos de contexto.

- *Granularidade.* Idealmente o armazenamento de contexto deveria considerar armazenar essas informações no grau mais fino de granularidade possível, a fim de garantir que qualquer futura requisição por informações de contexto seja suprida. Porém, outras questões surgem relacionadas a isso, como o esgotamento de espaços de armazenamento. Portanto, definir o nível de granularidade das informações de contexto armazenadas também é extremamente relevante.

2.7 Cidades Inteligentes

Existem diversas definições encontradas na literatura para cidades inteligentes (ALBINO; BERARDI; DANGELICO, 2015). Cada uma delas enfatiza diferentes aspectos do que constitui uma cidade inteligente. A maioria das definições está centrada nas pessoas e em que uma cidade inteligente deve proporcionar uma melhor qualidade de vida a elas. A economia também se destaca como um aspecto importante, e uma cidade inteligente deve proporcionar um desenvolvimento sustentável. A tecnologia, em especial as tecnologias de informação e comunicação (TIC), se destaca como um componente fundamental ao prover a infraestrutura necessária para melhorar os serviços oferecidos na cidade.

Existem diversas iniciativas de cidades inteligentes ao redor do mundo, com diferentes níveis de maturidade e diferentes domínios de aplicação. A maioria das iniciativas estão na Europa[1], América do Norte[2] e Ásia[3]. Exemplos de aplicações em cidades inteligentes são sistemas de distribuição de água, sistemas de distribuição de energia, monitoramento ambiental, sistemas de transporte inteligente, controle de tráfego, vigilância e saúde.

Diversas soluções baseadas em TIC já foram propostas para essas aplicações em

[1] Cidades inteligentes na Europa - http://www.smart-cities.eu

[2] 10 cidades inteligentes na América do Norte - http://www.fastcoexist.com/3021592/the-10-smartest-cities-in-north-america

[3] https://hub.beesmart.city/en/strategy/en/rising-asian-smart-cities-to-watch-in-2019-part-two#:~:text=When%20people%20think%20of%20Asian,enabled%20metropolises%20in%20the%20region

cidades inteligentes. Khatoun and Zeadally (2016) lista as principais TICs utilizadas em soluções de cidades inteligentes, dentre as quais destacam-se: *computação ubíqua*, *Internet das Coisas (IoT)*, *arquiteturas orientadas a serviço (SOA)*, *computação na nuvem* e *big data*.

Computação ubíqua é um paradigma onde computação é oferecida a usuários em qualquer lugar, a qualquer hora de uma forma transparente. Este paradigma foi predito por Mark Weiser no início dos anos 1990 (WEISER, 1991). Este novo conceito trouxe novas oportunidades de pesquisa, onde computação pode ajudar pessoas a desenvolverem suas atividades diárias sem perceber a computação envolvida nestes processos. Soluções de computação ubíqua em cidades se tornaram possíveis através da distribuição de diferentes tipos de sensores físicos. Estes sensores podem estar estaticamente instalados em lugares estratégicos de ambientes urbanos ou presentes em dispositivos móveis como *smartphones*.

Uma das tecnologias que atualmente habilitam a construção de soluções utilizando uma grande quantidade de sensores físicos em cidades inteligentes é a *Internet das Coisas* (*Internet of Things - IoT*). Soluções em IoT permitem que pessoas e coisas (sensores, atuadores, dispositivos, etc.) possam ser conectadas a qualquer momento, qualquer lugar, com qualquer coisa e qualquer um, idealmente utilizando qualquer caminho/rede e qualquer serviço (GUILLEMIN; FRIESS et al., 2009). A aplicação de IoT em soluções de cidades inteligentes é bastante difundida (TALARI et al., 2017; ZANELLA et al., 2014; ALAVI et al., 2018; ARASTEH et al., 2016; MEHMOOD et al., 2017; HAMMI et al., 2017).

Questões como heterogeneidade, interoperabilidade, segurança e o gerenciamento de uma grande quantidade de sensores são desafios enfrentados por desenvolvedores de aplicações IoT em cidades inteligentes. Diversas soluções de *midleware* já foram introduzidas na literatura científica e na indústria a fim de lidar com essas questões. Um importante requisito destes *middlewares* em IoT é possuirem uma *interface baseada em serviço* (RAZZAQUE et al., 2015). Web services são utilizados em vários *middlewares* existentes para fornecer estas interfaces. Exemplos de tais *middlewares* são *OpenIoT*[4], *OM2M*[5], *HiReply*[6], *Sentilo*[7] e *OpenMTC*[8].

A presença de milhares de sensores e dispositivos em uma cidade traz diversos de-

[4]http://www.openiot.eu/
[5]https://www.eclipse.org/om2m/
[6]https://www.reply.com/Documents/8629_img_CONR11_HI_Reply_eng.pdf
[7]https://www.sentilo.io/wordpress/
[8]https://www.openmtc.org/

safios relacionados ao gerenciamento, processamento e interpretação dos dados gerados por esses dispositivos, que apresentam um grande volume, variedade e velocidade. Abordagens de *big data* foram desenvolvidas exatamente para lidar com esse tipo de dados. Soluções para cidades inteligentes podem fazer uso de múltiplas tecnologias de hardware e software para lidar com *big data*, tais como plataformas de computação paralela e ambientes de programação.

Soluções de computação na nuvem possibilitam o acesso a recursos computacionais compartilhados, configuráveis e confiáveis, distribuídos na Web. Estes recursos podem ser rapidamente disponibilizados com um mínimo de esforço ou interação com o provedor de serviços (SÁNCHEZ-CORCUERA et al., 2019). Muitos dispositivos encontrados em soluções para cidades inteligentes, como dispositivos móveis e dispositivos IoT, possuem baixa capacidade de processamento e não podem lidar com toda a demanda de processamento exigida pelos dados gerados e também com as requisições de acesso a eles. Soluções na nuvem podem ser configuradas para receber esses dados, processá-los e disponibilizá-los para aplicações na Internet. Muitos serviços disponibilizados na nuvem também podem ser uma interessante fonte de dados (PERERA et al., 2014a). Uma alternativa para o uso exclusivo de serviços na nuvem é mesclar o uso destes com serviços na névoa (*fog computing*) e serviços na borda (*edge computing*), de forma a reduzir a necessidade de tráfego de dados para a nuvem e realizar um melhor balanceamento do uso de recursos computacionais (KHATOUN; ZEADALLY, 2016). Um exemplo de como serviços disponibilizados na névoa podem ser usados para processar dados de contexto encontra-se no trabalho de João et al. (2018). Neste trabalho, dados de sensores podem ser acessados e processados diretamente por dispositivos próximos à borda, formando uma nuvem mais próxima às fontes de dados de contexto e reduzindo a carga de tráfego em rede.

Outras fontes de dados utilizadas por aplicações em cidades inteligentes são *redes sociais* e dados obtidos por *crowdsensing* (KON; SANTANA, 2016). Algumas soluções de *crowdsourcing* também podem ser fontes de informações contextuais sobre entidades (ORREGO; BARBOSA, 2019). Outra fonte de informações contextuais em cidades inteligentes são os dados abertos (KHATOUN; ZEADALLY, 2016). Dados abertos permitem que dados sobre a cidade possam ser usados livremente, reutilizados, e distribuídos a qualquer um. Há diversas iniciativas de dados abertos relacionados a cidades, como em

Nova Iorque[9], Barcelona[10], Paris[11], Melbourne[12], São Paulo[13] e Porto Alegre[14].

2.8 Contexto em Cidades Inteligentes

Como mostrado na seção anterior, cidades inteligentes podem contar com diversas fontes de sensores físicos e virtuais para fornecimento de informações de contexto. Além destes, mecanismos de inferência podem ser usados para prover contexto de mais alto nível, o que é conhecido como *reconhecimento de contexto*. Estas fontes de contexto podem ser vistas como sensores lógicos.

Uma análise recente das abordagens utilizadas para reconhecimento de contexto em cidades inteligentes foi publicada em artigo no *journal* Springer Computing, no início de 2021 (NASCIMENTO et al., 2021). Tal análise tinha como objetivo geral traçar um panorama sobre como o reconhecimento de contexto ocorre em cidades inteligentes e foi realizada através de um mapeamento sistemático da literatura entre 2014 e 2018. Como objetivos mais específicos foram realizadas as seguintes atividades:

- identificar quais categorias de contexto são mais comuns em aplicações de reconhecimento de contexto em cidades inteligentes e quais informações são mais comumente extraídas nessas aplicações;
- avaliar o uso de técnicas de inferência de contexto e quais tipos de sensores são utilizados como fontes de dados brutos;
- avaliar qual nível de contexto é o objetivo fim de cada abordagem de reconhecimento de contexto;
- realizar um apanhado geral dos resultados obtidos e traçar oportunidades de pesquisa a partir de pontos ainda em aberto.

Com relação ao primeiro objetivo (categorias de contexto analisadas nos trabalhos), a grande maioria apresentou soluções relacionadas a reconhecimento de atividades (50,5%) e localização de usuários (35,5%). Portanto, a maioria dos sistemas sensíveis ao contexto utilizados em cidades inteligentes acaba considerando a localização de usuários e o que eles estão fazendo (atividades). Uma análise também foi realizada sobre que tipos

[9] https://opendata.cityofnewyork.us/
[10] https://opendata-ajuntament.barcelona.cat/en/open-data-bcn
[11] https://opendata.paris.fr/
[12] https://data.melbourne.vic.gov.au/
[13] http://dados.prefeitura.sp.gov.br
[14] http://datapoa.com.br/

de informações de localização e de atividades foram reconhecidas em cada trabalho. A grande maioria dos trabalhos de identificação da localização de usuários se concentrou basicamente em duas vertentes: (i) verificar se o usuário encontra-se em um ambiente fechado (*indoor*) ou aberto (*outdoor*); (ii) identificar nomes semânticos para localizações, como restaurante, escola, shopping center, etc. Quanto às atividades, ações como caminhar, correr, parado, andando em um veículo (algumas vezes identificando o tipo de veículo) foram as mais comuns.

O segundo objetivo visava identificar técnicas de inferência de contexto e tipos de sensores mais utilizados como fontes de dados brutos. Dispositivos móveis como *smartphones* foram os mais utilizados como fontes de dados (87% dos trabalhos), já que são equipados com diversos sensores. Portanto, outras fontes de dados presentes em cidades inteligentes, como sensores IoT, ainda são pouco exploradas na literatura em abordagens de reconhecimento de contexto. Para reconhecimento de atividades, sensores como acelerômetros, giroscópios e magnetômetros presentes em celulares modernos são as fontes mais comuns de dados brutos. Técnicas de aprendizagem de máquina são aplicadas nestes dados a fim de identificar atividades. Dessas técnicas, as mais utilizadas nos trabalhos analisados sobre reconhecimento de atividades destacam-se as árvores de decisão, seguidas por redes neurais e SVM. Árvores de decisão apresentam um bom desempenho em dispositivos com baixa capacidade de processamento como *smartphones*, apesar de não resultarem nas mais altas taxas de precisão e acurácia. Pouquíssimos trabalhos apresentaram alguma abordagem híbrida para reconhecimento de atividades, utilizando uma combinação de mais de uma abordagem. Isso está está de acordo com o salientado no estudo de Perera et al. (2014a). Já trabalhos relacionados ao reconhecimento de localização de usuários apresentaram o uso de receptores GPS como fonte primária de dados brutos. Outros sensores, como intensidade de sinais de Wifi e GSM e sensores de intensidade de luz também foram utilizados, principalmente em soluções onde se desejava saber se o usuário estava em um local fechado ou aberto.

Quanto ao terceiro objetivo, relacionado ao nível de contexto utilizado em cada solução, verificou-se que a grande maioria dos trabalhos identifica contexto em nível intermediário. Apenas 15% dos trabalhos apresentou uma solução relacionada a identificação de situações. Portanto este é um ponto ainda em aberto quanto ao reconhecimento de contexto em cidades inteligentes. Situações dizem respeito a informação de alto nível, e representações semânticas dessas informações ganham um papel fundamental. Ontologias e grafos de conhecimento representam uma forma interessante de representar esse

conhecimento e de se realizar inferências sobre ele utilizando mecanismos como regras SWRL, por exemplo. A limitação dessa abordagem reside em questões de performance, que em certas aplicações de cidades inteligentes é bastante relevante, principalmente naquelas que exigem resultados em tempo real.

3 ARQUITETURAS DE SOFTWARE

O principal objetivo deste trabalho visa o desenvolvimento de uma arquitetura capaz de integrar fontes de informações contextuais e mecanismos de inferência disponíveis em uma cidade inteligente. Para o desenvolvimento dessa arquitetura, houve a necessidade de realizar um estudo a fim de se identificar características relacionadas especificamente a arquiteturas já existentes para reconhecimento de contexto em cidades inteligentes ou que possam ser adaptadas a cidades inteligentes.

Este capítulo tem o objetivo de apresentar aspectos relacionados a arquiteturas de software em geral, principalmente aqueles aplicáveis a ambientes distribuídos e em larga escala, como cidades inteligentes. Primeiramente, aspectos gerais de arquiteturas de software foram apresentados. Logo depois, uma revisão sucinta sobre os principais padrões arquiteturais é apresentada. Por fim, destaca-se aspectos específicos importantes para sistemas em cidades inteligentes, em especial: descentralização, fusão de dados, resiliência, composição dinâmica e escalabilidade.

3.1 Conceitos Importantes sobre Arquiteturas de Software

O foco deste trabalho está em uma arquitetura de software para integração de fontes de contexto. Portanto é importante se esclarecer o que se entende por arquitetura de software e quais são os aspectos de qualidade de uma boa arquitetura. Como primeiro passo, entende-se uma arquitetura de software como "*o conjunto de estruturas necessárias para raciocinar sobre o sistema, que compreende elementos de software, relações entre eles e suas propriedades*"(BASS; CLEMENTS; KAZMAN, 2012).

Quanto à avaliação de arquiteturas de software, um aspecto importante a ser considerado são seus atributos de qualidade. Um atributo de qualidade é uma propriedade mensurável ou testável de um sistema usada para indicar quão bem o sistema satisfaz as necessidades de seus interessados (BASS; CLEMENTS; KAZMAN, 2012). Os seguintes atributos de qualidade são identificados na literatura (BASS; CLEMENTS; KAZMAN, 2012; NASCIMENTO; OLIVEIRA, 2021):

- *Disponibilidade* é uma propriedade de um software relacionada à sua capacidade de estar apto a realizar suas tarefas quando necessário. Este atributo engloba confiabilidade e agrega a noção de recuperação ou resiliência (a capacidade de um módulo

de software se reparar após uma falha). Os mecanismos de resiliência e tolerância a falhas presentes em arquiteturas de gerenciamento de contexto são apresentados na Seção 3.4.

- *Interoperabilidade* diz respeito ao grau em que dois ou mais sistemas podem trocar informações significativas por meio de interfaces. As arquiteturas de gerenciamento de contexto fazem parte de sistemas sensíveis ao contexto mais extensos que consomem as informações de contexto fornecidas por elas. Assim, as interfaces fornecidas pelos subsistemas de reconhecimento de contexto são um aspecto crucial relacionado à sua capacidade de interoperabilidade. Essas interfaces estão relacionadas aos mecanismos de distribuição de contexto discutidos na Seção 2.3. Ontologias contribuem para melhorar a interoperabilidade semântica dessas interfaces, por proverem um vocabulário comum e que é extensamente usado em arquiteturas sensíveis ao contexto distribuídas (NASCIMENTO; OLIVEIRA, 2021).
- *Modificabilidade* está relacionada ao custo e risco de mudanças nos sistemas de software. É um aspecto crucial das arquiteturas de gerenciamento de contexto em cidades inteligentes, pois esses ambientes são muito dinâmicos. Tais sistemas devem se adaptar às mudanças no ambiente, como o acréscimo de novas capacidades de sensoriamento e a necessidade de novos mecanismos de raciocínio para lidar com esses dados. Em ambientes dinâmicos e de larga escala como cidades inteligentes, novas fontes de dados e informações contextuais podem ser anexadas ao sistema a qualquer momento. Portanto, arquiteturas utilizadas para gerenciamento de contexto devem ser escaláveis e permitir que tais mudanças sejam configuradas dinamicamente. Tal aspecto é discutido em mais detalhes na Seção 3.5.
- *Desempenho* está relacionado à capacidade de um sistema de software de atender aos requisitos de tempo. Os dados derivados das fontes de dados urbanos são caracterizados pela heterogeneidade e normalmente são de grande escala (MOUSTAKA; VAKALI; ANTHOPOULOS, 2018). Mecanismos de inferência distribuídos são frequentemente usados para processar esse alto volume de dados e atingir os requisitos de tempo (NUAIMI et al., 2015). No entanto, o tempo de processamento não é o único aspecto relacionado ao desempenho em soluções de gerenciamento de contexto em cidades inteligentes. A latência é outro aspecto crucial desses sistemas e está relacionada à latência das infraestruturas de comunicação da rede. Arquiteturas descentralizadas podem reduzir a latência fornecendo nós de processamento próximos a dispositivos que adquirem dados (SÁNCHEZ-CORCUERA et al., 2019). O

uso de arquiteturas descentralizadas é detalhado na Seção 3.3.

- *Segurança* é a capacidade de um sistema de software de proteger dados e informações de acesso não autorizado. Mecanismos específicos de segurança estão fora do escopo deste trabalho.
- *Proteção (safety)* é a capacidade do software de evitar entrar em estados que causem ou levem a danos, ferimentos ou perda de vida aos atores no ambiente do software. As soluções de gerenciamento de contexto não são usadas diretamente pelos usuários e não atuam no sistema. Eles são responsáveis por obter dados de seus ambientes e realizar inferências sobre esses dados. No entanto, os aplicações consumidoras de contexto podem usar esses dados para decidir quais ações executar. Nessa situação, os subsistemas de gerenciamento de contexto em cidades inteligentes fundem dados de diferentes fontes, combinando-os a fim de produzir informações mais precisas, mais completas e mais confiáveis que não poderiam ser alcançadas por meio de um única fonte (PERERA et al., 2014a). Questões relacionadas à fusão de dados são detalhadas na Seção 3.6.

Outros atributos de qualidade relacionados à arquiteturas de software incluem usabilidade e ética. Estes atributos também estão fora do escopo deste trabalho, pois arquiteturas de gerenciamento de contexto não costumam ser manipuladas diretamente por usuários finais.

3.2 Padrões Arquiteturais

Outro aspecto importante relacionado às arquiteturas de software são os padrões. Padrões arquitetônicos são formas de capturar estruturas de projeto comprovadamente boas, para que possam ser reutilizadas (BASS; CLEMENTS; KAZMAN, 2012). Na literatura encontramos diversos padrões arquiteturais (BASS; CLEMENTS; KAZMAN, 2012; RODA et al., 2018; NASCIMENTO; OLIVEIRA, 2021) onde destacam-se os seguintes:

- *Multicamadas*, onde é possível se separar responsabilidades do sistema em diferentes unidades chamadas *camadas*, que podem ser desenvolvidas e mantidas de forma independente. Cada camada fornece um conjunto coeso de serviços. As camadas interagem umas com as outras em uma relação de ordem estrita. Cada camada, normalmente, pode utilizar apenas serviços da camada diretamente abaixo.

- *Intermediário (broker)*, responsável por intermediar o acesso a serviços distribuídos. Quando um cliente necessita de um serviço, envia uma consulta ao intermediador via uma interface disponível para isso. O intermediador, por sua vez, encaminha a requisição recebida ao serviço adequado, que a processa. O resultado é encaminhado pelo serviço ao intermediador, que o repassa do cliente. O cliente, portanto, desconhece a identidade, localização e características dos servidores.
- *Cliente-servidor*, onde clientes requisitam serviços ou recursos fornecidos por servidores. Cada servidor pode fornecer um conjunto de recursos e/ou serviços. Alguns componentes podem atuar tanto como clientes e servidores. Arquiteturas desse tipo podem conter um único servidor central ou diversos servidores distribuídos.
- *Peer-to-peer*, onde existem um conjunto de componentes distribuídos sem papéis definidos. Todos podem iniciar e receber requisições. Cada componente pode interagir com qualquer outro. Novos componentes devem conectar à rede e descobrir outros componentes com quem pode interagir. Esses componentes podem ser removidos da rede, sem causar grandes impactos, resultando em uma grande escalabilidade para o sistema. Arquiteturas desse tipo são inerentemente distribuídas e facilmente descentralizáveis (não há um nó central que controla toda a comunicação do sistema).
- *Orientada a serviços*, onde um número de serviços é oferecido por provedores de serviço e é consumido por consumidores. Cada componente pode desempenhar ambos os papéis (fornecedor e consumidor). Servidores e consumidores são normalmente implantados de forma independente, podendo pertencer a diferentes sistemas e organizações. Cada serviço deve possuir uma interface clara, onde são especificadas, entre outras coisas, entradas necessárias e saídas fornecidas.
- *Baseada em componentes*, onde a estrutura de execução do sistema é organizada em agrupamentos lógicos de componentes. Os critérios de agrupamento podem ser baseados no tipo de componentes, ambiente físico de execução, etc.
- *Multiagente*, onde o processamento de dados e informações é realizado através da interação de agentes autônomos. Os agentes possuem conhecimento total ou parcial sobre o sistema e a interação entre eles ocorre através de protocolos de comunicação bem definidos. Agentes podem possuir mecanismos sofisticados de inteligência artificial para tomada de decisão. Também existe a possibilidade de ações em conjunto, onde agentes podem cooperar para a solução de um problema. Estratégias de

negociação podem ser usadas na ocorrência de conflitos entre agentes.

3.3 Arquiteturas Descentralizadas

Cidades inteligentes são ambientes em larga escala, com uma grande quantidade de sensores gerando grandes quantidades de dados brutos de contexto a serem processados. Portanto, uma arquitetura descentralizada para este processamento poderia ser mais interessante do que uma arquitetura centralizada, onde gargalos podem comprometer a performance do sistema. Analisamos nesta seção possibilidades de descentralização para cada padrão arquitetural apresentado na Seção 3.2.

Arquiteturas utilizando o padrão *broker* (intermediário) possuem um ponto central de acesso (o intermediador em si) e tem uma estrutura inerentemente centralizada. Portanto, o uso de tal padrão inviabiliza um sistema descentralizado.

Arquiteturas *peer-to-peer* são inerentemente descentralizadas. Nenhum componente central de controle ou acesso é exigido. Conexões entre nós podem ser estabelecidas através de varreduras ou buscas na rede. Quando um nó necessita de recursos ou serviços, envia requisições a nós conectados a ele. Essa requisição pode ser propagada às conexões desses outros nós em cascata, até um certo limite.

Arquiteturas multiagente e cliente-servidor também podem ser descentralizadas. Nesse caso, basta se evitar um ponto central, incorporando em tais sistemas princípios de arquiteturas *peer-to-peer*.

Arquiteturas orientadas a serviços também podem ser descentralizadas. Provedores de serviços são inerentemente distribuídos. Para facilitar a busca por provedores que fornecem determinados serviços podem ser usados serviços especiais de registro. Tais registros podem ser um ponto de centralização. Entretanto, trabalhos na literatura propõe registros descentralizados, baseados principalmente em arquiteturas *peer-to-peer* e multiagente (HEIDARI; NAVIMIPOUR, 2021).

Arquiteturas multicamadas e baseadas em componentes podem ser descentralizadas combinando-as com arquiteturas *peer-to-peer*, orientadas a serviços ou multiagente. Nessas soluções híbridas, cada camada ou agrupamento de componentes pode conter um conjunto descentralizado de nós, agentes ou serviços que podem utilizar serviços providos por componentes descentralizados em camadas inferiores. Portanto, a organização interna de cada camada ou agrupamento passa a ser descentralizada.

3.4 Resiliência e Tolerância a Falhas

Tolerância a falhas é a capacidade de um sistema de manter em execução suas funções previstas, independentemente da ocorrência de falhas (KUMARI; KAUR, 2021). Neste trabalho utilizam-se os conceitos de tolerância a falhas e resiliência como sinônimos.

Mecanismos de tolerância a falhas são importantes em sistemas de software pois trazem garantias relacionadas a confiabilidade, desempenho e disponibilidade de serviços. Alguns desses mecanismos, identificados na literatura, podem ser caracterizados em dois grupos: *reativos* e *pró-ativos*.

Abordagens reativas de tolerância a falhas são executadas, como o próprio nome diz, em reação a ocorrência de uma falha. Portanto, seu objetivo é mitigar a influência de uma falha no sistema. Mecanismos de tolerância a falhas reativos citados na literatura (KUMARI; KAUR, 2021) são listados e descritos abaixo.

- *Checkpointing*: o estado do componente é salvo periodicamente. Em caso de falha, o componente é reiniciado a partir do último estado registrado.
- *Migração de trabalho*: se um componente não pode completar sua execução em algum dispositivo físico específico em virtude de algum problema, é migrado para outro dispositivo.
- *Replicação*: múltiplas cópias de componentes são realizadas e implantadas em diferentes localizações. Em caso de falha, uma tarefa pode continuar a ser executada através das réplicas de um componente.
- *S-Guard*: baseado em recuperação de *rollback*, isto é, desfazer e/ou descartar as alterações realizadas durante determinada transação.
- *Repetição*: um componente que falha tem sua execução repetida, até ser finalizado com sucesso.
- *Reenvio de solicitação*: a mesma solicitação é enviada novamente para o mesmo local ou para outro, a fim de que a execução do processo que falhou seja repetida.
- *Workflow de resgate*: fluxos alternativos de ação permitem ao sistema continuar funcionando após a ocorrência da falha, até que não exista alternativas sem que a falha seja tratada de outra forma.

Técnicas pró-ativas de tolerância a falhas tentam predizer falhas pró-ativamente, substituindo componentes suspeitos por outros. Uma visão geral sobre tais técnicas é

mostrada abaixo.

- *Rejuvenescimento de Software*: o sistema é reiniciado periodicamente e inicia a partir de um novo estado a cada vez.
- *Migração Preemptiva*: nesta abordagem, componentes são observados e analisados constantemente. Em caso de o estado atual ser considerado perigoso, o componente em questão é migrado para outra localização.
- *Autocura*: baseado na técnica de divisão e conquista, divide uma grande tarefa em partes menores, executadas de forma distribuída. Caso ocorra alguma falha em alguma dessas instâncias, esta pode ser resolvida automaticamente pelo sistema.
- *Balanceamento de Carga*: nesta abordagem um balanceamento de carga de memória e CPU é realizado quando excede um certo limite. A carga extra é transferida para outra CPU onde o limite não é excedido.

3.5 Escalabilidade e Fontes de Dados

Como já observado na seção 3.1, um dos pontos a ter atenção em sistemas de gerenciamento de contexto modernos é a capacidade de agregar novas fontes de contexto na arquitetura, preferencialmente em um esquema *plug-and-play*. Esse é um requisito interessante para sistemas em cidades inteligentes, onde a quantidade dessas fontes pode aumentar frequentemente.

Estes novos componentes devem ser agregados/removidos ao sistema sem alterar os componentes existentes e prejudicar a execução do sistema. Exemplos de arquiteturas reais que utilizam esse tipo de estratégia são o padrão OSGi (*Open Services Gateway Initiative*)[1] e o *middleware* FIWARE[2].

O padrão *broker* suporta naturalmente a extensão das capacidades de um sistema, ao tornar transparente a adição/remoção de componentes gerenciados pelo intermediador. Arquiteturas *peer-to-peer* também permitem a adição/remoção de nós de forma independente dos nós existentes. Cada novo nó agregado ao sistema pode se conectar a outros através de uma varredura na rede. Essas novas conexões permitem que tal nó possa ser "encontrado"por nós existentes. Uma abordagem similar também pode ser aplicada em sistemas multiagentes. Protocolos de comunicação como o *Contract Net* permitem que

[1] <https://www.osgi.org/>
[2] <https://www.fiware.org/>

agentes possam encontrar e negociar serviços com novos agentes agregados ao sistema, assim como novos agentes podem fazer o mesmo com agentes já existentes (WOOLDRIDGE, 2009).

3.6 Fusão Dados e Informações Contextuais

O avanço de soluções em cidades inteligentes e IoT disponibiliza uma gama de fontes de dados contextuais. Estas fontes de dados podem fornecer dados redundantes ou até complementares em relação umas às outras. Mecanismos de fusão de dados podem ser usados a fim de se combinar dados provenientes dessas fontes. Com isso, é possível aumentar a confiabilidade desses dados e extrair conhecimento de mais alto nível.

Lau et al. (2019) resume os objetivos da aplicação de fusão de dados em quatro tópicos:

- *Correção de dados problemáticos*, quando as fontes de dados apresentam problemas relacionados à qualidade dos dados produzidos, tais como inconsistência, imperfeição, disparidade, etc.
- *Aumento da confiabilidade de dados*, quando dados são coletados em um ambiente com presença de ruídos que podem interferir em sua qualidade. Nesse caso, a presença de fontes de dados redundantes e a fusão desses dados pode ajudar a minimizar o impacto negativo de tais ruídos.
- *Extração de informação de mais alto nível*. Mecanismos de aprendizagem de máquina podem ser usados para combinar informações de fontes de dados heterogêneas e gerar informação de contexto de mais alto nível. Mecanismos de reconhecimento de contexto entram nesta categoria. Por exemplo, um mecanismo de aprendizagem de máquina, como uma rede neural artificial, pode ser usado para interpretar a atividade realizada pelo usuário de um *smartphone* através da combinação de dados de um acelerômetro e um giroscópio presentes no equipamento.
- *Aumento da completude dos dados*. Diferentes fontes de dados podem prover diferentes perspectivas sobre a observação de um fenômeno. Portanto, tais informações são complementares e a fusão delas pode levar a um quadro mais completo sobre o contexto em questão.

Os métodos de fusão de dados estão diretamente relacionados ao nível dos dados em questão (NWEKE et al., 2019). O nível mais baixo, ou *nível de dados*, utiliza mé-

todos mais tradicionais para fusão de dados a fim de tratar dados brutos provenientes de fontes como sensores físicos. Muitas tarefas de fusão nesse nível também são conhecidas como abordagens *Data In Data Out (DAI-DAO)*, onde múltiplos dados brutos de entrada são fundidos a fim de produzir um dado bruto de saída com maior confiabilidade (DASARATHY, 1997). Métodos comuns para fusão de dados de baixo nível incluem média ponderada (onde diferentes fontes de dados recebem pesos diferentes), métodos probabilísticos (como filtros de Kalman, máxima verossimilhança e redes Bayesianas) e métodos baseados em evidência, como a Teoria de Dempster-Shafer. Estes métodos exigem um conhecimento prévio sobre o sistema em questão, como por exemplo associar pesos a fontes de dados específicas, modelos probabilísticos e/ou de incertezas do sistema.

Em um nível intermediário, também conhecido como nível de características, estas são fundidas a fim de obter outras características. Tais características podem incluir dados como médias, desvios padrão, formas, texturas, etc. Essas informações representam contextos ainda sem um alto valor semântico, porém derivadas a partir de informações de contexto bruto. A fusão de tais informações pode utilizar métodos matemáticos, estatísticos, ou mesmo abordagens de aprendizagem de máquina.

É possível também realizar a fusão de informações de contexto de mais alto nível semântico e de abstração. Neste caso, múltiplas fontes de contexto de alto nível estão presentes e apresentam informações conflitantes ou complementares. No caso de conflitos, métodos de votação podem ser usados para se selecionar a informação contextual a ser utilizada. Estes métodos de seleção podem ser apoiados, inclusive, por funções de suporte que definem valores de *score* para valores de contexto fornecidos por determinadas fontes. Modelos de aprendizagem de máquina também podem ser treinados para realizar a fusão dessas informações de contexto.

Parâmetros de QoC podem ser usados como base para seleção de valores de contexto mais apropriados a determinada situação (MANZOOR; TRUONG; DUSTDAR, 2009). Por exemplo, uma aplicação que priorize informações de contexto atualizadas, poderia utilizar o parâmetro *Timeliness* para selecionar aquelas mais recentes. Estes parâmetros também podem ser usados como apoio em métodos de fusão. Por exemplo, o trabalho de Derrick, Renfa and Yongheng (2016) utiliza QoC no cálculo da evidência para o processo de fusão de dados de contexto utilizando a Teoria de Dempster-Shafer.

3.7 Composição Dinâmica

Outro aspecto importante a ser levado em conta nessas arquiteturas inclui a capacidade de composição dinâmica. Em ambientes em larga escala, como cidades inteligentes e redes IoT, é impossível identificar ou planejar a interação exata de componentes do sistema durante a fase de desenvolvimento, ainda mais se tal sistema for escalável e extensível (Seção 3.5).

Essa característica diz respeito à capacidade da arquitetura de entender os requisitos e demandas de uma determinada situação, organizando e estruturando seus componentes internos de acordo a elas. Em um sistema de gerenciamento de contexto, isso envolve a obtenção de dados de sensores, modelos de inferência, operadores de fusão de dados, bases de conhecimento, etc. Nesse caso, um sistema de gerenciamento de contexto deveria ser capaz de arranjar uma composição de um subconjunto de tais componentes a fim de prover informações sobre algum contexto específico (PERERA et al., 2014a).

Composição dinâmica é um elemento importante em sistemas onde contexto secundário é obtido. Neste caso, agentes de inferência de contexto obtém instâncias de elementos contextuais de outras fontes de contexto, que por sua vez podem necessitar de instâncias providas por outras fontes, e assim por diante, formando um encadeamento. O uso de composição dinâmica permite que esta composição de fontes de contexto em cadeia mude ao longo e possa aproveitar novas fontes agregadas ao sistema ou se adaptar à remoção de fontes anteriormente acessíveis.

4 TRABALHOS RELACIONADOS

Este capítulo descreve a análise de arquiteturas para gerenciamento de contexto propostas na literatura. Primeiramente, uma revisão mais abrangente sobre arquiteturas sensíveis ao contexto é apresentada. Posteriormente, mostra-se os resultados de um mapeamento sistemático da literatura que destaca arquiteturas distribuídas para gerenciamento de contexto aplicáveis a cidades inteligentes. Os principais trabalhos encontrados são resumidamente descritos. Por fim, uma breve discussão sobre os resultados e trabalhos identificados é apresentada.

4.1 Aspectos Gerais de Arquiteturas Sensíveis ao Contexto

Primeiramente, buscou-se encontrar na literatura revisões, mapeamentos sistemáticos ou *surveys* que apresentassem uma análise de trabalhos relacionados à temática desta tese. Uma análise bastante extensa sobre arquiteturas para sistemas sensíveis ao contexto em geral encontra-se no trabalho de Roda et al. (2018). Esse trabalho apresenta um mapeamento sistemático onde um dos objetivos era analisar arquiteturas existentes para sistemas sensíveis ao contexto aplicadas a diferentes domínios. Ao todo, 463 trabalhos publicados entre 1999 e 2017 foram analisados após um processo de busca e filtragem.

De acordo com o trabalho de Roda et al. (2018), observou-se um crescimento grande no número de trabalhos relacionados ao desenvolvimento de arquiteturas sensíveis ao contexto até 2007. Esse patamar tem-se mantido estável desde então. As arquiteturas identificadas foram aplicadas principalmente nos domínios de ambientes inteligentes (incluindo casas inteligentes, hospitais e escritórios) e cuidados de saúde.

Os tipos de arquiteturas mais utilizados nos trabalhos analisados foram: multicamadas, baseadas em componentes, orientadas a serviço e multiagente. Arquiteturas multicamadas são comumente utilizadas pois apresentam uma forma simples de separar responsabilidades, onde cada camada deve prover uma funcionalidade diferente. Os meios mais utilizados em cada camada para prover tais funcionalidades incluem combinações com outros modelos arquitetônicos, como sistemas multiagentes, serviços e componentes.

4.2 Mapeamento Sistemático da Literatura

As arquiteturas analisadas no trabalho de Roda et al. (2018) incluem soluções não aplicáveis a ambientes distribuídos e em larga escala, como cidades inteligentes. Além disso, nenhuma análise específica sobre características arquiteturais destacadas no capítulo anterior (descentralização, resiliência, composição dinâmica, fusão de dados e escalabilidade) foi considerada. Portanto, houve a necessidade de se realizar uma nova revisão da literatura, a partir de um mapeamento sistemático, voltado especificamente para avaliação dessas questões relacionadas a arquiteturas de gerenciamento de contexto em cidades inteligentes. Tal mapeamento encontra-se publicado em Nascimento and Oliveira (2021) e um breve resumo do trabalho é descrito à seguir.

O mapeamento realizado seguiu as mesmas diretrizes propostas por Petersen, Vakkalanka and Kuzniarz (2015). A pesquisa foi realizada em quatro mecanismos de busca de artigos (ACM Digital Library, IEEE Xplore, Scopus, Springer Link) utilizando a seguinte *string* de busca: *(architecture OR design OR framework OR component OR infrastructure OR specification OR middleware) AND ("context-aware"OR "situation-aware"OR "context-sensitive"OR"environment-directed") AND (user OR human) AND ("city"OR "smart environment"OR "urban"OR ubiquit* OR pervasive).*

Como resultado da busca, 1977 artigos foram obtidos. A filtragem dos resultados ocorreu em duas etapas. A primeira etapa consistiu na análise de títulos e resumos dos artigos. Os artigos selecionados passaram, então, por uma segunda etapa de filtragem, onde foram consideradas outras seções do texto, como introdução, conclusão e partes onde a solução era descrita em mais detalhes. Os seguintes critérios foram considerados em ambas as etapas de filtragem:

- estudos científicos publicados em conferências, workshops ou revistas relacionadas à ciência da computação (critério de inclusão);
- estudos publicados antes de 2020 (critério de inclusão);
- estudos publicados em inglês (critério de inclusão);
- artigos com quatro ou mais páginas (critério de exclusão);
- estudos primários (critério de exclusão);
- artigos que não sejam duplicatas de outros estudos (critério de exclusão);
- artigos que não apresentem uma publicação mais antiga dos mesmos autores sobre a mesma abordagem (critério de exclusão);

- estudos que claramente apresentem uma arquitetura para reconhecimento de contexto de usuários com um processo de reconhecimento descentralizado ou que processem dados oriundos de sensores distribuídos (critério de exclusão);
- estudo não apresenta uma proposta aplicada a cidades inteligentes ou que seja aplicável a cidades inteligentes, ou seja, não é restrita a algum outro domínio específico (critério de exclusão).

Após a primeira filtragem, 682 artigos foram obtidos. Destes, 87 foram selecionados após a aplicação da segunda filtragem.

O mapeamento apresentado analisou trabalhos publicados até o ano de 2019. A fim de analisar trabalhos similares publicados posteriormente, uma análise da literatura similar foi realizada em trabalhos publicados a partir de 2020. Um total adicional de 22 trabalhos foram selecionados, de acordo com os critérios de inclusão e exclusão apresentados anteriormente. Isso elevou o total de trabalhos selecionados para 109.

Uma leitura na íntegra de cada um destes 109 artigos foi realizada a fim de identificar padrões arquiteturais utilizados, como o contexto foi modelado e como questões de descentralização, resiliência, escalabilidade, fusão e composição dinâmica foram tratadas. Os resultados encontrados são descritos nas seções à seguir.

4.3 Uso de Padrões Arquiteturais

Para fornecer uma visão geral dos padrões de arquitetura de software usados para construir subsistemas de gerenciamento de contexto, as seções dos artigos que descrevem a arquitetura de cada abordagem foram analisadas e extraiu-se o padrão de arquitetura usado por cada uma. A abordagem baseada em componentes é a mais comum, seguida por multiagente, multicamadas, orientada a serviços, *broker* e cliente-servidor.

As arquiteturas baseadas em componentes geralmente possuem módulos para obter dados de sensores, módulos para agregar/fundir esses dados e módulos para realizar raciocínio e distribuição de contexto. Sensores distribuídos e heterogêneos podem ser conectados a eles de maneira *plug-and-play*. No entanto, a maioria das arquiteturas baseadas em componentes apresenta módulos centralizados que agregam informações de contexto de diversas fontes e fornecem interfaces para aplicações interessadas nesses dados. Algumas exceções incluem os seguintes trabalhos:

- Dey and Mankoff (2005) descreve uma arquitetura composta por três tipos de com-

ponentes: *widgets*, *interpretadores* e *mediadores*. Pode haver várias instâncias de cada componente, e um arranjo distribuído das instâncias é possível. *Widgets* encapsulam o acesso a fontes de dados de contexto, que podem ser sensores ou outros componentes da arquitetura, e armazenam as informações obtidas. Os interpretadores aplicam abordagens de inferência aos dados de contexto, e os mediadores são responsáveis por lidar com questões de ambiguidade.

- Malandrino et al. (2010) propõe um framework que utiliza uma arquitetura de gerenciamento de perfil distribuído. Três módulos são responsáveis pelo gerenciamento de perfil: módulo de gerenciamento de perfil de usuário, módulo de gerenciamento de perfil de operador e módulo de gerenciamento de provedor de serviços. Esses módulos enviam dados de perfil para um módulo provedor de contexto centralizado que agrega esses dados.
- Ranganathan and Campbell (2003) apresenta uma arquitetura distribuída baseada em dois tipos de componentes: *fornecedores de contexto* e *sintetizadores de contexto*. Provedores de contexto distribuído são fontes de dados de contexto de baixo nível. Os sintetizadores de contexto geram informações de contexto de alto nível a partir do contexto obtido por um ou mais provedores de contexto. Os aplicativos podem obter dados de contexto de provedores, sintetizadores ou ambos.
- Efstratiou et al. (2001) apresenta uma arquitetura onde serviços fornecem dados de contexto. Um componente de descoberta de contexto pode detectar serviços de contexto disponíveis e acessá-los. Um módulo de reconhecimento de contexto é composto por diversos agentes, que encapsulam mecanismos de inferência de contexto. Esses agentes podem ser agregados à plataforma para incorporar novos recursos de reconhecimento de contexto de alto nível ao sistema.
- Ryu et al. (2007) mostra uma arquitetura onde as fontes de contexto estão distribuídas em diversos dispositivos móveis e vestíveis. *Widgets* de contexto são as fontes de dados implantadas em cada dispositivo. Um processo distribuído agrega esses dados, onde cada dispositivo desenvolve uma agregação local e um coordenador agrega os dados de cada dispositivo. O coordenador é responsável por disponibilizar as informações de contexto para as aplicações.
- Gaire et al. (2020) apresenta uma arquitetura com componentes distribuídos onde componentes coletores de dados obtém contexto de fontes de dados. Os coletores repassam os dados coletados para um serviço distribuído na nuvem, que os agrega e processa, utilizando tecnologia como Apache Kafka e Apache Spark.

Em geral, as arquiteturas multiagentes analisadas podem ser agrupadas nas seguintes categorias:

- *Equipes* composta por um número fixo de agentes com capacidade de raciocínio. Cada agente desempenha um papel diferente no processo de reconhecimento de contexto. Agentes específicos podem ser responsáveis pela coleta de dados, enquanto outros podem executar processos de inferência, fusão, armazenamento e distribuição de contexto. O processo como um todo ocorre através da execução de tarefas por parte dos agentes e da troca de conhecimento entre eles. A maioria das arquiteturas multiagentes analisadas são organizadas dessa maneira.
- Agentes múltiplos e distribuídos são organizados em uma *hierarquia*. Agentes no topo recebem solicitações de aplicações. Não há necessariamente um ponto central de acesso para solicitações de aplicações (podem existir diversos agentes nos níveis superiores). Esses agentes solicitam as informações de contexto requisitadas pelas aplicações a agentes de níveis inferiores. Esses agentes podem obter dados de fontes de contexto e processá-los, fornecendo as informações de contexto solicitadas por agentes de níveis superiores.
- *Sociedade* de agentes onde cada um tem uma visão parcial das informações de contexto. Cada agente possui mecanismos de inferência e pode trocar conhecimento de contexto com outros agentes. Portanto, o conhecimento do contexto é distribuído por todos os agentes. Existem regras e leis sociais que definem como os agentes podem se comunicar e trocar conhecimento. Assim, uma requisição por certa informação de contexto pode gerar um processamento distribuído, exigindo a interação de diversos agentes entre si, trocando conhecimentos e colaborando para gerar a informação solicitada.

A arquitetura em camadas é uma arquitetura em que uma camada diferente executa cada etapa de reconhecimento de contexto. Essas camadas podem ser implantadas em um servidor centralizado ou distribuídas em vários dispositivos. As camadas comuns encontradas na maioria dos trabalhos são:

- *Camada de detecção*, *Camada de percepção*, ou *Camada de aquisição*, que visa obter dados de sensores e outras fontes de contexto bruto. Esta camada pode apresentar uma estrutura distribuída utilizando, por exemplo, serviços ou uma organização multiagente.
- *Camada de agregação* que é responsável por agregar e fundir dados de várias fontes

de dados. Depois de receber os dados de contexto de diversas fontes, a camada de agregação de contexto limpa, pré-processa e aplica técnicas de fusão de dados para fornecer dados de contexto precisos e confiáveis para serem processados posteriormente.

- *Camada de inferência* responsável por aplicar técnicas de inferência para reconhecer contextos de alto nível. Este processo pode ser dividido em duas ou mais camadas e pode ser distribuído e executado em paralelo.

Em arquiteturas baseadas em serviços, os serviços encapsulam a aquisição e a inferência de contexto. Foram identificados quatro tipos de organizações de serviços entre os trabalhos analisados:

- Os *Web services* distribuídos realizam a aquisição de contexto de sensores e outras fontes de dados. *Web services* centralizados processam esses dados e reconhecem o contexto de alto nível.
- *Web services* distribuídos são organizados em camadas lógicas. Normalmente, serviços responsáveis por obter dados brutos de sensores estão na camada mais baixa, enquanto os serviços que podem reconhecer o contexto de alto nível estão nas camadas mais altas. Não existe um serviço centralizado responsável por reconhecer o contexto. Bernardos, Madrazo and Casar (2010) apresenta um caso particular onde os serviços estão localizados em nós móveis que podem trocar informações de contexto.
- *Rede de serviços provedores de contexto*: esses serviços podem entrar em contato uns com os outros para trocar dados de contexto. Os serviços devem ser registrados em um *broker* ou registro de serviços. Esse *broker* ou registro possibilita que os serviços sejam encontrados e identificados.

As arquiteturas do tipo *broker* são caracterizadas por um único ponto de acesso que intermedia a comunicação entre aplicativos e provedores de contexto. Os provedores de contexto são modelados como *plug-ins* que podem obter dados brutos de sensores ou ter recursos de inferência de contexto. *Plug-ins* são diferentes de agentes e serviços porque não possuem autonomia e não são distribuídos.

As arquiteturas cliente-servidor analisadas utilizam clientes implantados em dispositivos móveis para acessar serviços de inferência de contexto e armazenamento em servidores. Essas requisições por serviços normalmente contém dados brutos coletados nesses dispositivos. Servidores na nuvem são a abordagem mais comum. Esses servido-

res podem usar o poder de processamento de alta capacidade da nuvem para processar dados de contexto brutos.

A análise dos trabalhos existentes mostra uma organização lógica dos componentes arquitetônicos de acordo com o ciclo de vida de contexto, onde componentes de aquisição e componentes de raciocínio fora organizados logicamente em um conjunto de camadas ou módulos diferentes. Este tipo de organização permite uma clara separação de funcionalidades entre os diferentes componentes.

Os componentes de aquisição geralmente são organizados como *plug-ins*, para que os sistemas de reconhecimento de contexto possam adicionar facilmente novos recursos de detecção. Como dito anteriormente, componentes distribuídos, *Web services* e agentes são as implementações mais comuns de componentes de aquisição.

Poucos trabalhos apresentam uma organização distribuída de componentes de inferência. A maioria das arquiteturas fornece módulos de inferência centralizados que agregam dados de componentes de aquisição distribuídos para inferir contexto de alto nível. Sistemas orientados a serviços e multiagentes são os paradigmas mais comuns na implementação de mecanismos de raciocínio distribuído fornecidos pelos trabalhos selecionados. Esses paradigmas são naturalmente distribuídos e podem fornecer um mecanismo de processamento descentralizado. Registros fornecem um mecanismos de *páginas amarelas*, que permite a agentes ou serviços registrarem suas capacidades, fornecendo uma maneira de localizar esses recursos dentro de um sistema distribuído. Esses mecanismos de seleção também podem fornecer meios para selecionar serviços ou agentes baseados em parâmetros de qualidade (HUEBSCHER; MCCANN, 2006).

Portanto, uma combinação de padrões arquiteturais pode fornecer a necessária separação de interesses e distribuição de processamento. Exemplos de tais arquiteturas são:

- arquiteturas em camadas que organizam logicamente um conjunto de serviços distribuídos que fornecem recursos de aquisição em infraestrutura de IoT e mecanismos de inferência de contextos de alto nível;
- arquiteturas baseadas em componentes em que agentes realizam inferências sobre dados de contexto de baixo nível fornecidos por *Web services*, a partir de middlewares para captação de dados brutos em cidades inteligentes.

4.4 Suporte a Descentralização, Resiliência, Composição Dinâmica, Fusão de Dados e Escalabilidade

Uma análise do suporte das arquiteturas selecionadas a descentralização, resiliência, composição dinâmica, fusão de dados e escalabilidade também foi realizada. Para isso, verificou-se as seções de cada artigo que descreviam cada solução proposta. Uma descrição geral desta análise, por cada característica, é mostrada abaixo.

- *Escalabilidade*: é uma característica geral das arquiteturas analisadas, presente em cerca de 84 das 109 analisadas. Novas fontes de contexto são tratadas, em geral, como *plug-ins*. Esses *plug-ins* incluem não só novas fontes de dados de baixo nível (geralmente sensores físicos), mas também novos recursos de inferência de alto nível. *Web services*, sistemas multiagentes e componentes *plug-and-play* são as principais tecnologias usadas para implementar *plug-ins*. Tais componentes podem se registrar automaticamente em algum serviço de registro ou serem registrados manualmente por um usuário.
- *Descentralização*: cerca de 34% das 109 arquiteturas analisadas apresentaram um processo descentralizado de gerenciamento de contexto. Em geral, essas arquiteturas usam um dos seguintes arranjos de componentes: em camadas, *peer-to-peer*, baseado em registro, ou multiagente. Uma organização em camadas contém componentes distribuídos, logicamente organizados em uma camada superior, responsáveis por receber solicitações de aplicações. Esses componentes solicitam informações de contexto de componentes distribuídos em camadas inferiores. Normalmente, essas arquiteturas usam uma organização de duas camadas, onde a camada mais baixa obtém dados de sensores e a camada mais alta processa esses dados para obter informações de contexto de alto nível. Em organizações *peer-to-peer*, os nós distribuídos obtêm dados de sensores locais e os processam para obter informações de contexto. Os nós podem compartilhar seus dados para melhorar seu conhecimento sobre o contexto. As arquiteturas baseadas em registro usam um conjunto de serviços de registro (que são distribuídos), onde os provedores de contexto podem se inscrever. Aplicações acessam esses registros para encontrar provedores de contexto. Organizações multiagente descentralizadas funcionam de forma similar às organizações *peer-to-peer*, porém agentes autônomos estão distribuídos e são responsáveis por processar requisições de aplicações. Esses agentes compartilham conhecimento sobre o contexto do ambiente e podem realizar inferências sobre ele.

- *Composição Dinâmica*: cerca de 27% dos trabalhos apresentam uma solução que utiliza composição dinâmica. A abordagem mais comum para composição dinâmica é selecionar provedores de contexto e mecanismos de inferência a partir de uma consulta ou requisitos de contexto especificados por uma aplicação. Essa consulta é disponibilizada através de registros de provedores de contexto ou através de mecanismos de *multicasting*. As capacidades informadas por cada provedor de contexto são cruciais neste processo de seleção. É possível melhorar o processo de seleção usando parâmetros de QoC, de forma a selecionar provedores que forneçam contexto de melhor qualidade. Outra abordagem consiste em mecanismos de auto-organização, onde nós ou agentes podem anunciar suas capacidades, descobrir outros nós e usar um mecanismo de contrato para organizar cadeias de fornecimento e consumo de informações de contexto. Esse agentes podem trocar conhecimentos sobre o contexto e colaborar na obtenção do resultado requisitado por uma aplicação. Outras abordagens para composição incluem o uso de regras para selecionar planos de processamento de contexto específicos ao ocorrerem determinados eventos.
- *Fusão de dados*: cerca de 13% das arquiteturas estudadas apresentaram algum mecanismo de fusão durante o processo de reconhecimento de contexto, porém poucas especificaram como esse mecanismo funciona. Políticas baseadas em QoC foram usadas em alguns trabalhos, através de índices de confiabilidade/confiança para selecionar informações de maior qualidade e resolver conflitos em dados obtidos de diferentes provedores de contexto. Os critérios para definição do que seja uma boa qualidade dependem de políticas específicas (SYLLA et al., 2019; HEGDE; KURNIAWAN; RAO, 2009; INDULSKA et al., 2003a). Métodos baseados em evidência, como a teoria de Dempster-Shafer, também foram usados para fusão de dados de contexto de mais baixo nível. Outros métodos utilizados incluem regras e mecanismos de aprendizagem de máquina.
- *Resiliência*: apenas 8% dos estudos apresentaram algum mecanismo de tolerância a falhas. A abordagem mais utilizada apresenta uma arquitetura onde provedores de contexto podem substituir outros equivalentes em caso de falha. Nesse caso, existem outros componentes diferentes que fornecem informação similar ao componente que falha, de modo que podem substitui-lo, num processo similar a uma replicação. Outra abordagem usa o compartilhamento de informações de contexto entre nós distribuídos ou agentes que tornam a arquitetura adaptável em caso de

falha de um nó (nesse caso, replicação de dados).

4.5 Análise de Trabalhos Selecionados

Alguns dos trabalhos obtidos na revisão da literatura apresentaram pelo menos quatro das cinco características analisadas (descentralização, resiliência, fusão de dados, escalabilidade / extensibilidade, composição dinâmica). Por se aproximarem mais das características desejadas para a arquitetura desenvolvida neste trabalho, foram analisados em mais detalhes abaixo.

O trabalho de Geihs and Wagner (2012) apresenta uma proposta de arquitetura de um middleware para gerenciamento de contexto projetado para ser usado dentro de um projeto maior chamado MUSIC *(Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environments*). Na solução apresentada, novos sensores e mecanismos de inferência podem ser agregados à plataforma como *plug-ins*. Novos sensores podem ser acrescentados de forma *on-thef-fly*, à medida que são descobertos. Um mecanismo automatizado ativa e desativa *plug-ins* de acordo com as necessidades da aplicação. Estes *plug-ins* podem estar disponíveis em repositórios e serem facilmente reutilizados (portanto a arquitetura suporta composição dinâmica). A arquitetura permite que esses *plug-ins* possam ser organizados em nós distribuídos. Esses nós podem trocar informações de contexto entre si. *Plug-ins* são tratados como serviços e selecionados segundo um processo de negociação baseado em QoS. A arquitetura não possui um mecanismo muito claro de tolerância a falhas (a distribuição de informação de contexto em nós distribuídos pode garantir algum grau de resiliência), e também não oferece nenhum claro mecanismo de fusão de dados ou validação de informações de contexto.

Sebbak et al. (2010) apresenta uma arquitetura multiagente integrada a uma camada mais baixa, composta por serviços responsáveis por obter dados de sensores. Quatro tipos de agentes são usados: o agente de serviço de contexto, o agente de agregação de contexto, o agente de conhecimento de contexto e o agente de inferência de contexto. Esses agentes usam o agente de diretório para registrar serviços e pesquisar um provedor de serviço de contexto específico. O agente de serviço de contexto pode adquirir contextos de serviços OSGi ou diretamente de sensores ubíquos localizados no ambiente. Os serviços de contexto são usados para fornecer a outros agentes conhecimento contextual, como localização do usuário ou clima em um local específico. O agente de agregação é dedicado à fusão de conhecimento contextual coletado de vários agentes de

serviço de contexto. Quando algum provedor de contexto fica indisponível, é possível ao agente de agregação substituir o serviço indisponível por outro serviço semelhante, sem suspender seu comportamento. O agente de conhecimento de contexto armazena a ontologia de contexto e os contextos passados para uso posterior. A arquitetura apresentada é descentralizada, resiliente, sensores podem ser agregados e dados fundidos para formar informação de contexto. Porém, a arquitetura não suporta composição dinâmica.

Huebscher and McCann (2006) apresentam um middleware organizado em quatro camadas. A camada inferior consiste em sensores físicos em uma infraestrutura IoT. A próxima camada consiste em provedores de contexto (*Context Providers* - CPs) que agregam e interpretam os dados brutos de um mais sensores. Quando um CP entra na rede, ele se anuncia a um serviço de diretório (DS). No primeiro contato, o CP informa ao DS o tipo de contexto que ele pode fornecer e anexa os valores de atributos de QoC que descrevem esta provisão. No entanto, CPs são ocultos de um aplicativo, introduzindo uma camada extra de abstração, o serviço de contexto (CS). Os serviços de contexto recuperam informações de contexto dos CPs em nome dos aplicativos. Esta camada de abstração é útil porque, se um CP diferente for melhor para um aplicativo do que o atualmente usado, o CS pode se adaptar de forma autônoma, mudando para a melhor alternativa sem ter que envolver o aplicativo. A qualidade de um provedor de contexto é avaliada através de uma função de utilidade, informada pelas aplicações. Outra característica útil é que em caso de falha de um CP, ele pode ser substituído por outro que esteja relacionado ao mesmo CS. Um serviço de contexto é responsável por lidar com uma categoria de contexto específica. Em suma, o middleware proposto permite certa descentralização (vários serviços de diretório podem ser executados em paralelo), assim como resiliência e composição dinâmica. Um interessante mecanismo de avaliação de qualidade dos serviços de contexto também é apresentado, o que não foi visto nas soluções apresentadas anteriormente. Porém, nenhum claro mecanismo de fusão de dados é apresentado na arquitetura.

O trabalho de Alvarez-Napagao et al. (2014) propõe uma arquitetura que utiliza dispositivos móveis inteligentes (como *smartphones*) para tornar possível o conceito de *human as a sensor* (pessoas como sensores). Assim, o trabalho proposto tem como objetivos: (i) melhorar o conhecimento obtido através de fontes de dados; (ii) detectar situações inesperadas em cidades que podem afetar grandes grupos de pessoas; (iii) disponibilizar serviços a usuários que podem explorar o conhecimento gerado (como sistemas de recomendação, por exemplo). Para isso, uma arquitetura chamada *SUPERHUB* foi proposta. Essa arquitetura possui um interpretador semântico responsável por prover conhecimento

de contexto através de tuplas RDF. Internamente, o interpretador semântico é formado por diversos agentes. Agentes rastreadores atribuem a si próprios uma API ou *web service* como fonte de dados de contexto e gerenciam a recepção de dados deles. Os agentes de trabalho são responsáveis por agregar dados dos rastreadores e produzir triplas RDF com conhecimento de contexto em alto nível. Esses processos de agregação podem ser incluídos ou removidos em tempo de execução. Todos os agentes podem estar distribuídos. Um agente que falha pode ser substituído por outro agente através de um *pool* de agentes gerenciado pelo interpretador semântico. Portanto, a arquitetura descrita possui certo nível de descentralização (apesar da informação de contexto ser finalmente centralizada em um interpretador semântico) e resiliência. Sensores podem ser agregados através de novos agentes rastreadores. Mecanismos de fusão de dados podem ser incluídos em agentes de trabalho, apesar da arquitetura não prever nada específico. Além disso, nenhum mecanismo de composição dinâmica é previsto.

O trabalho de Wei et al. (2020) descreve uma arquitetura de um sistema de resposta a desastres naturais. A arquitetura, em geral, é dividida em camadas. A camada mais baixa, de geração de dados, é dedicada à captura de dados a partir de fontes que podem ser: drones, sensores IoT, câmeras, *smartphones*, etc. A camada de comunicação de dados é responsável por estabelecer links de comunicação entre dispositivos e permitir a transferência de dados entre eles. A camada de processamento de dados é responsável por processar dados brutos de contexto de baixo nível e gerar informações de mais alto nível. Nesta camada, uma rede de drones é utilizada para coletar e processar informações de contexto. Requisições de contexto serão recebidas pelo drone mais próximo, que pode consultar informações de seus próprios sensores, sensores externos ou outros drones e dispositivos. Assim, uma rede *peer-to-peer*, descentralizada, de drones é utilizada para processar contexto. Novas fontes de dados podem ser encontradas e agregadas ao sistema. Um módulo de armazenamento, em cada drone, é responsável por agregar e fundir dados de contexto. A composição dinâmica ocorre pela proximidade física de dispositivos (dispositivos próximos fornecem as informações de contexto necessárias). Nenhum mecanismo de tolerância a falhas é especificado na arquitetura.

Um *middleware* multi-domínio para processamento de contexto foi proposto no trabalho de Vahdat-Nejad (2022). A arquitetura proposta possui duas camadas distribuídas. A primeira camada, H-CAMID, contém componentes instalados em dispositivos com maior capacidade de processamento, como servidores. Já a segunda camada, L-CAMID, contém componentes implantados em dispositivos com capacidade de processa-

mento mais limitada. Provedores de contexto estão presentes em dispositivos na camada L-CAMID, e podem ser simples componentes que obtém dados de fontes de contexto, como componentes mais complexos, que realizam inferências sobre essas informações. Esses provedores de contexto se registram na camada H-CAMID, que mantém um registro de todos eles e de suas localicações. Os servidores são replicados como mecanismo de tolerância a falhas. Esses servidores são organizados em um esquema mestre-escravo. Um servidor mestre é consultado periodicamente pelos escravos, a fim de informar sobre atualizações. Atualizações em escravos são propagadas ao mestre. Caso o servidor mestre falhe, um dos escravos assume seu lugar. A camada H-CAMID também realiza uma composição dinâmica limitada, selecionando provedores de contexto de acordo com solicitações recebidas da camada L-CAMID. A camada L-CAMID também pode conter aplicações consumidoras, que requisitam dados de contexto da camada H-CAMID. Não há nenhum mecanismo previsto para fusão de dados.

A tabela 4.1 mostra um resumo da comparação dos trabalhos mencionados nesta seção. É possível observar que nenhuma das arquiteturas identificadas na literatura apresenta uma proposta ao mesmo tempo descentralizada, resiliente, extensível, com suporte a fusão de dados e composição dinâmica.

Tabela 4.1: Comparação de trabalhos relacionados.

Referência	Descentr.	Resil.	Extens.	Fusão	Comp. Din.
Geihs and Wagner (2012)	Sim	Lim.	Sim	-	Sim
Sebbak et al. (2010)	Sim	Sim	Sim	Sim	-
Huebscher and McCann (2006)	Sim	Sim	Sim	-	Sim
Alvarez-Napagao et al. (2014)	Sim	Sim	Sim	Lim.	-
Wei et al. (2020)	Sim	-	Sim	Sim	Sim
Vahdat-Nejad (2022)	Sim	Sim	Sim	-	Sim

Fonte: O Autor

4.6 Conclusões do Mapeamento da Literatura

Após a etapa de análise da literatura realizada neste trabalho, é possível concluir que:

- *Arquiteturas descentralizadas, extensíveis e tolerantes a falhas*. Diversas arquiteturas distribuídas foram propostas na literatura para gerenciamento de contexto que são adequadas para ambientes como cidades inteligentes. As cidades inteligentes

contêm um grande número de fontes de dados de contexto distribuídas. As arquiteturas de gerenciamento de contexto em cidades inteligentes devem ser extensíveis. Elas devem permitir a fácil adição de novos sensores e recursos de inferência de contexto. Elas também devem tirar proveito de novos paradigmas, como névoa e computação de borda para descentralizar o processamento de contexto. Além disso, essas arquiteturas podem usar mecanismos de tolerância a falhas para lidar com falhas em nós e se tornar resilientes. No entanto, apenas cinco das arquiteturas analisadas são, ao mesmo tempo, descentralizadas, extensíveis e tolerantes a falhas. Portanto, há uma lacuna de pesquisa no desenvolvimento de arquiteturas de reconhecimento de contexto em cidades inteligentes que incorporem esses recursos em uma abordagem única.

- *Fusão de contexto*. A fusão de dados de contexto é uma abordagem interessante em arquiteturas desse tipo, quando várias fontes fornecem dados sobre a mesma propriedade de entidade. Existem diversas técnicas para fundir esses dados para melhorar a confiabilidade das informações de contexto resultantes. No entanto, apenas 13% dos artigos analisados utilizam fusão de dados ou informações no processamento de contexto. Será necessário mais esforço para explorar técnicas de fusão de dados e informações em reconhecimento de contexto aplicadas a cidades inteligentes para atingir um bom nível de maturidade.

Este trabalho traz uma nova contribuição ao trazer uma arquitetura que é, ao mesmo tempo, descentralizada, resiliente, escalável e que suporta fusão de dados e composição dinâmica.A descentralização é realizada através da utilização de um modelo multiagente distribuído sem um controlador central. A escolha de uma arquitetura multiagente se deve ao fato de proporcionar autonomia a cada agente bem como ser um modelo distribuído por natureza. Cada agente possui capacidades de tomada de decisão e ação. Agentes com capacidades redundantes podem substituir outros agentes em caso de falha. Informações desses agentes redundantes podem ser fundidas, com auxílio, inclusive, de parâmetros de QoC. Novos agentes podem ser adicionados ao modelo para fornecer novas capacidades de sensoriamento e inferência, sem interferir nos agentes já em funcionamento.

5 SISTEMAS MULTIAGENTES

A arquitetura desenvolvida neste trabalho utiliza um padrão multiagente. Este capítulo apresenta uma visão geral sobre sistemas multiagentes, focando especialmente em conceitos e abordagens utilizados em capítulos posteriores.

5.1 Agentes

Segundo Wooldridge (WOOLDRIDGE, 2009), um agente é um sistema computacional que está situado em algum ambiente e é capaz de agir de forma independente em nome de seu usuário ou proprietário. Um agente possui autonomia para perceber por si mesmo como agir para atingir seus objetivos, sem a necessidade de ser explicitamente orientado sobre o que fazer a todo momento. Portanto, com base nestas definições, é possível definir as seguintes características gerais de agentes:

- *Autonomia*, onde agentes devem tomar decisões sem a necessidade de interação humana, com base em mecanismos de tomada de decisão, tais como regras ou planos.
- *Ser situado em um ambiente*, onde agentes percebem o estado do ambiente através de sensores e agem nesses ambientes através de atuadores.
- *Interação* com outros agentes, envolvendo mecanismos de comunicação, negociação, coordenação e cooperação.

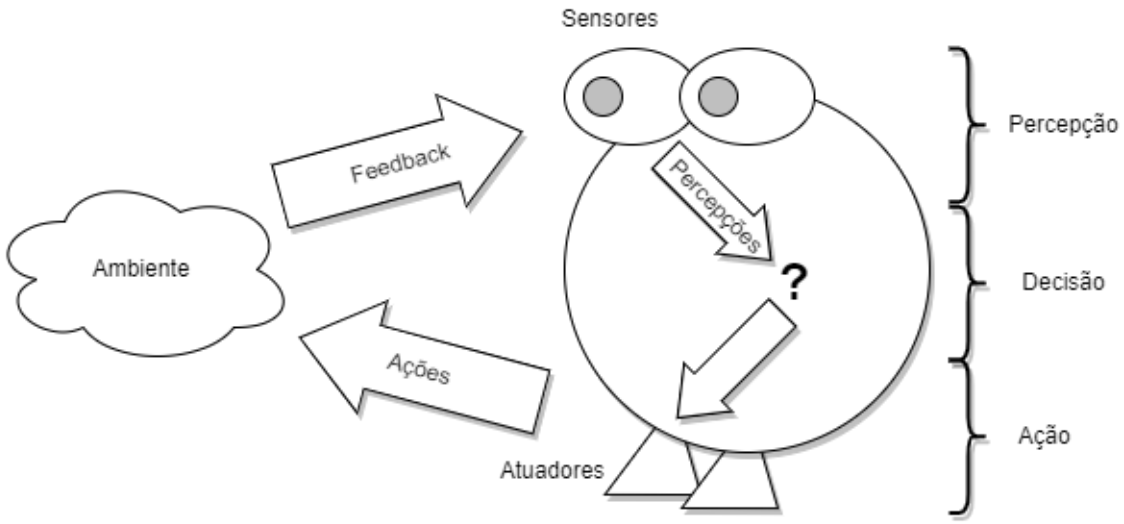
Uma visão abstrata geral de um agente é mostrada na Figura 5.1. O agente é capaz de receber *feedbacks* do ambiente onde se encontra inserido através de mecanismos de percepção (sensores). A partir de mecanismos de tomada de decisão, o agente pode escolher executar ações que podem alterar o estado do ambiente através de seus atuadores.

5.2 Percepção, Ações e Ambientes

A capacidade de percepção do agente e a complexidade de suas ações depende das propriedades do ambiente. Ambientes podem ser classificados de acordo com as seguintes propriedades (RUSSELL; NORVIG, 1995; WOOLDRIDGE, 2009):

- *Totalmente ou parcialmente observáveis* (também chamados de acessíveis ou inacessíveis). Um ambiente totalmente observável ocorre quando um agente possui a

Figura 5.1: Visão abstrata geral de um agente.



Fonte: Wooldridge (2009)

capacidade de obter informações completas sobre o estado atual do ambiente. Já em ambientes parcialmente observáveis, agentes não podem obter uma visão completa sobre o estado do ambiente. Normalmente, cada agente possui uma visão parcial diferente dos demais.

- *Determinístico ou não determinístico*. Em um ambiente determinístico, agentes podem prever o estado do ambiente após a execução de uma determinada ação, a partir do estado atual conhecido por ele. Já em ambientes não determinísticos, os resultados de ações de agentes não podem ser completamente previstos, pois uma mesma ação aplicada em um mesmo estado do ambiente pode resultar em resultados diferentes.
- *Dinâmico ou estático*. Em ambientes dinâmicos, o estado pode mudar enquanto um agente está em processo de decisão. Ou seja, a decisão tomada pelo agente pode falhar em virtude de mudanças imprevistas e podem ser revistas. Essas mudanças podem ocorrer, por exemplo, como resultado de ações executadas por outros agentes. Em ambientes estáticos, alterações no ambiente ocorrem somente por ações executadas pelo agente.
- *Discreto ou contínuo*. Um ambiente é discreto se há um número de ações a percepções que é finito, fixo.

Uma cidade inteligente envolve um ambiente parcialmente observável (um agente

individual não tem a possibilidade de conhecer o contexto de todos as entidades presentes no ambiente), não determinístico (nem sempre a ação executada por um agente produzirá o mesmo resultado, pois o próprio ambiente pode mudar de maneiras imprevistas), dinâmico (o ambiente pode mudar durante a tomada de decisão do agente, resultado em falhas na execução de ações) e contínuo (não há como fixar um número de percepções finito). Ambientes desse tipo (parcialmente observáveis, não determinísticos, dinâmicos e contínuos) representam os mais desafiadores para a implementação de agentes.

5.3 Decisão

Agentes, como entidades autônomas, tem a capacidade de tomada de decisão, ou seja, escolher por conta própria quais ações deve executar. Para tanto, o agente deve realizar algum processo de *raciocínio*. Tal processo pode se dividir em duas etapas (WOOLDRIDGE, 2009):

- *Raciocínio teórico*, direcionado a crenças, onde um conjunto de crenças do agente pode levar a inferência de outras crenças, mas sem gerar ações. Por exemplo, se um agente conhece que todo o homem é mortal e que Sócrates é um homem, pode deduzir a nova crença de que Sócrates é mortal.
- *Raciocínio prático*, onde o foco é na obtenção de um conjunto de ações a serem realizadas pelo agente. Esse processo, por sua vez, pode ser subdividido em duas etapas: *deliberação* (onde se decide qual estado se deseja atingir, ou seja, decidir quais serão as intenções ou objetivos do agente) e *planejamento* (decidir quais ações devem ser executadas para se atingir a intenção ou objetivo escolhido). O resultado do processo de planejamento é sempre um *plano*, contendo um conjunto de ações a serem realizadas.

Os processos de deliberação e planejamento podem estar limitados devido a restrições existentes. Essas restrições podem ser de cunho temporal (como no caso de sistemas de tempo real, por exemplo) quanto relacionadas a recursos computacionais (limites de memória do dispositivo, capacidade de processamento, etc.). Isso limita a quantidade de processamento que pode ser realizada ao se tomar decisões em agentes e implica que o mesmo deve ter algum grau de controle sobre o processo de raciocínio, e fim de garantir que este seja executado dentro das restrições existentes quanto a tempo e recursos. Existem situações onde o agente não pode realizar a escolha ótima, e deve se limitar a alguma

opção sub-ótima.

Um *planejador* é um sistema ou componente responsável por elaborar planos. O processo de planejamento executado por ele toma como entrada representações de:

- um *objetivo ou intenção*, algo que o agente deseja realizar ou um estado que o agente deseja atingir;
- o *estado atual do ambiente*, de acordo com as crenças do agente;
- as *ações disponíveis para o agente*.

Tanto o objetivo quanto o estado do ambiente podem ser representados de diferentes formas, como proposições lógicas, indivíduos em uma ontologia, etc. Ações podem ser vistas como tuplas, onde são especificadas as seguintes informações:

- *nome ou identificação* da ação (argumentos de entrada também podem ser especificados, quando houver);
- *pré-condições*, uma lista de fatos que devem ser verdadeiros para que a ação seja executada;
- *pós-condições ou efeitos* da ação, especificando o que deve mudar após a execução da ação.

O planejador é responsável, portanto, por elaborar uma sequência de ações de forma que o objetivo seja atingido, a partir do estado atual do ambiente. Tal plano deve ser organizado de tal forma que as pré-condições da primeira ação executada correspondam a um subconjunto do estado atual do ambiente, e as pós-condições ou efeitos da última ação executada garantam que o objetivo desejado seja uma parte do estado final do ambiente.

Para que tal plano seja obtido, planejadores fazem uso de alguma *estratégia de planejamento*. Uma estratégia simples pode envolver uma coleção de planos previamente montados e inseridas nos agentes pelos projetistas. Nesse caso, a tarefa de planejamento se resume a realizar uma busca pela plano cujo objetivo equivale ao objetivo desejado e cujas pré-condições sejam satisfeitas pelo estado atual do ambiente. Porém, estratégias mais complexas podem ser empregadas, como uma busca em espaço de estados (tanto em progressão quanto regressão) e o uso de heurísticas (NORVIG; RUSSELL, 2013).

5.4 Sistemas Multiagentes

Um sistema multiagente é aquele composto por um certo número de agentes, com a capacidade de interagir uns com os outros, com o objetivo de resolver um conjunto de tarefas. Cada agente pode possuir seu próprio objetivo. Esse objetivo pode ser, inclusive, distinto dos objetivos dos demais agentes. Esse sistema de agentes pode ser estruturado de diferentes formas (HORLING; LESSER, 2004):

- *Hierarquia*, onde agentes são organizados na forma de uma árvore, onde aqueles que se encontram em níveis superiores tem algum nível de autoridade sobre os que se encontram nos níveis inferiores. Geralmente a hierarquia é obtida a partir da decomposição de tarefas, onde agentes no topo da hierarquia possuem uma visão mais completa do problema, enquanto agentes em níveis mais baixos possuem uma visão mais parcial do todo, relativa ao subproblema que devem resolver.
- *Holarquia*, onde agentes estão agrupados em estruturas chamadas *holons*. Cada *holon* é composto por subpartes que são outros *holons* e assim por diante. *Holons* podem se relacionar com outros *holons* e possuir certo grau de autonomia na tomada de decisões. Uma organização hierárquica pode existir entre *holons* de forma que problemas possam ser divididos em problemas menores resolvidos por *holons* individuais.
- *Coalizões*. Cada coalizão é um agrupamento temporário de agentes, criada para um determinado propósito, em um ambiente de agentes cooperativos. Uma vez formada, uma coalizão pode ser vista pelo sistema como uma entidade única. Apesar de não existir uma estrutura definida dentro de uma coalizão, esta pode apresentar subestruturas internas e os agentes componentes devem definir alguma forma de coordenação a fim de atingir os objetivos estipulados. Agentes são motivados a participar de uma coalizão por considerarem que seu ganho final será maior do que realizando a mesma tarefa isoladamente.
- *Equipes*, onde agentes se organizam em agrupamentos similares a coalizões, porém o objetivo é maximizar o ganho total da equipe, e não necessariamente de cada agente individual participante.
- *Congregações*, que apresentam uma estrutura similar a coalizões e equipes, onde agentes se organizam em grupos. Porém, existem duas diferenças cruciais com relação aos tipos de grupos apresentados anteriormente. Primeiro, não há um objetivo

específico para a construção de uma congregação (nenhuma tarefa específica a ser realizada). Segundo, congregações são grupos de longo prazo. A motivação para agentes formarem congregações está nas capacidades do grupo: congregações reúnem agentes com capacidades complementares. Assim, é possível reduzir o espaço de busca na procura por agentes complementares durante a realização de alguma tarefa, já que a interação entre agentes ocorre dentro de uma mesma congregação.

- *Sociedades*, que são grandes grupos de agentes reunidos a longo prazo, onde existem limites impostos ao comportamento dos agentes, comumente chamados de *leis sociais*, *normas* ou *convenções*. Sociedades são grupos abertos, onde agentes podem entrar e sair a qualquer momento.
- *Federações*, onde os agentes participantes de um grupo delegam parte de sua autonomia a um delegado ou intermediador. Esse intermediador é responsável por gerenciar interações do grupo com entes externos e é comumente chamado também de facilitador, mediador ou agregador.
- *Mercado*, onde certos agentes compradores podem requisitar ou realizar ofertas por um conjunto de itens comuns, recursos compartilhados, tarefas, serviços ou bens. Alguns agentes ofertam itens para serem vendidos. Outros agentes agem como vendedores ou leiloeiros, recebendo e processando ofertas. Essa organização cria um sistema produtor-consumidor altamente competitivo.
- *Matriz*, onde um agente pode estar subordinado a diferentes supervisores em diferentes contextos.
- *Organizações Compostas*, onde diferentes tipos de organizações de agentes podem ser combinadas.

5.5 Comunicação

Em um ambiente multiagente é necessário que os mesmos interajam de alguma forma. Essas interações ocorrem tipicamente através da troca de mensagens (WOOLDRIDGE, 2009). Essa troca de mensagens ocorre normalmente através de Linguagens de Comunicação entre Agentes (*Agent Communication Languagens - ACLs*), tais como *KQML (Knowledge Query and Manipulation Language)*, *KIF (Knowledge Interchange Format)* e *FIPA-ACL*. A comunicação em sistemas multiagentes também pode ocorrer de forma implícita (sem o uso de mensagens), através de mecanismos baseados em *Teoria*

dos Jogos.

No caso de comunicação explícita, linguagens de comunicação entre agentes costumam definir uma estrutura de "envelope"para mensagens, sem se preocupar tanto com seu conteúdo (definido por outras linguagens). Toda mensagem tem uma *performativa* (que pode ser vista como a classe ou tipo da mensagem) e um número de *parâmetros* (pares atributo-valor que definem propriedades da mensagem). Algumas performativas presentes na linguagem FIPA-ACL são mostradas na Tabela 5.1. Parâmetros comuns em mensagens incluem seu conteúdo, remetente, destinatário, linguagem usada na representação do conteúdo e ontologia usada para representação semântica de termos.

Tabela 5.1: Algumas performativas presentes na FIPA-ACL.

Performativa	Descrição
request	Permite a um agente requisitar a outro que execute alguma ação.
subscribe	O remetente deseja ser notificado quando algum evento ocorrer.
inform	O remetente informa algo ao destinatário.
agree	O remetente concorda em executar o solicitado.
refuse	O remetente recusa executar o solicitado.
not-understood	A mensagem recebida não foi compreendida.

Fonte: O Autor

Para representação de conteúdo de mensagens, existem linguagens específicas como a linguagem *KIF*(*Knowledge Interchange Format*) e *FIPA SL*(*FIPA Semantic Language*). Além disso, ontologias podem ser usadas para especificar a semântica de termos usados no conteúdo dessas mensagens.

Um exemplo de mensagem em FIPA-ACL é mostrado abaixo:

```
(inform
    :sender agent1
    :receiver agent2
    :content (price good1 100)
    :language sl
    :ontology ex-auction
)
```

Nessa mensagem, a performativa *inform* foi utilizada, onde o agente *agent1* deseja informar o agente *agent2* sobre algo (especificado no conteúdo da mensagem, parâmetro *content*). O conteúdo foi especificado utilizando a linguagem FIPA SL e a ontologia usada para especificar a semântica dos termos no conteúdo da mensagem foi especificada no parâmetro *ontology*.

5.6 Outras Interações Entre Agentes

A fim de interagir com sucesso, estes agentes necessitam de habilidades para cooperar e coordenar seus objetivos e intenções com outros agentes, de modo similar a como pessoas cooperam e coordenam com outras pessoas no dia-a-dia. Coordenar significa gerenciar dependências entre atividades desempenhadas por agentes diferentes (uso de recursos compartilhados, restrições relacionadas a pré-requisitos, sincronização, etc). Cooperação relaciona-se ao caso de agentes combinarem suas capacidades a fim de realizarem alguma tarefa em conjunto.

5.7 Agentes e Ontologias

Se dois ou mais agentes desejam se comunicar com relação a algum domínio, então é necessário que concordem sobre a terminologia que utilizam para descrevê-lo (WOOLDRIDGE, 2009). Por exemplo, se um agente recebe a medida de distância em quilômetros da localização de uma estação meteorológica até a cidade de Porto Alegre de outro agente, é necessário que ambos concordem sobre o que os termos *quilômetro* e *Porto Alegre* significam.

A especificação de um conjunto de termos com o objetivo de prover uma base comum de entendimento é chamado de *ontologia*. Este não é um termo utilizado exclusivamente na área de sistemas multiagentes, sendo utilizado principalmente em *web semântica*.

Uma ontologia é definida por Studer, Benjamins and Fensel (1998) como *uma especificação explícita e formal de conceptualização compartilhada.* Uma *conceptualização*, por sua vez, é definida como *um modelo abstrato de algum fenômeno no mundo onde se identificou conceitos relevantes de tal fenômeno.* Uma ontologia deve ser formal, de forma que seja compreensível por uma máquina.

Três componentes muito importantes em ontologias são *classes*, *indivíduos* e *propriedades*. Uma classe é uma coleção de termos com propriedades similares. Por exemplo, com relação à informações de contexto, poderíamos conter classes como *Elemento Contextual*, *Atividade* e *Localização*. Já um indivíduo ou instância compõe um termo específico em uma ou mais classes. Por exemplo, uma instância de elemento contextual seria um indivíduo da classe *Elemento Contextual*, enquanto o termo *Andando* poderia ser um indivíduo tanto das classes *Atividade* quanto *Elemento Contextual*. Propriedades per-

mitem que sejam definidas características relacionadas à classes e indivíduos, bem como relacionamentos destes com outras classes e/ou indivíduos presentes na ontologia ou em outras ontologias. Por exemplo, um indivíduo da classe Atividade pode possuir uma propriedade chamada *ocorreuEm* que indica o instante de tempo em que a ação ocorreu. Por outro lado, a classe Atividade pode possuir uma propriedade *subclasseDe* que a relaciona a classe *ElementoContextual*.

Wooldridge (2009) também define uma *base de conhecimento* como formada por uma ontologia juntamente com um conjunto de instâncias de classes definidas nessa ontologia. Assim, um agente, por exemplo, pode possuir seu conhecimento sobre o mundo organizado na forma de uma base de conhecimento baseada em uma ou mais ontologias.

Por fim, ontologias podem ser classificadas em diversos níveis, dependendo de seu papel em aplicações:

- *Ontologias de alto nível*, que se preocupam em definir termos mais gerais, geralmente utilizadas como base para construção de ontologias mais específicas;
- *Ontologias de domínio*, que definem termos específicos de um determinado domínio de aplicação (geralmente com base em ontologias de alto nível), como por exemplo, um domínio médico (que seria utilizada por um certo número de aplicações na área médica);
- *Ontologias de aplicação*, geralmente definida sobre uma ontologia de domínio, define conceitos usados por uma aplicação específica, sendo tipicamente de relevância apenas apenas dentro dessa aplicação.

6 ARQUITETURA PARA INTEGRAÇÃO DE CONTEXTO

Este trabalho envolveu a elaboração de uma arquitetura multiagente voltado à integração de fontes de contexto em cidades inteligentes. Neste capítulo, faz-se uma descrição geral da arquitetura e seus componentes. No capítulo seguinte serão abordados em mais detalhes os processos envolvidos nas diferentes operações suportadas pela arquitetura.

6.1 Definição Formal de Contexto Usada no Trabalho

Neste trabalho, considera-se a definição de contexto apresentada na Seção 2.1, onde definiu-se contexto como qualquer informação que possa caracterizar uma entidade e que é composto por um conjunto de elementos contextuais instanciados. Neste trabalho, considera-se que cada elemento contextual instanciado é uma tupla γ, definida como:

$$\gamma = (e, v, Q, t, C) \tag{6.1}$$

onde e é o elemento contextual associado à instância, v é o valor desse elemento contextual, Q é um conjunto de valores de QoC associados à instância, t o instante de tempo em que a informação foi gerada, e C um conjunto de categorias de contexto associadas à instância.

Cada elemento contextual e é definido formalmente como uma tupla

$$e = (\beta, \alpha) \tag{6.2}$$

onde β é a entidade caracterizada pelo elemento contextual e α corresponde ao aspecto da entidade relacionado ao elemento contextual. Exemplos de aspectos seriam aceleração, latitude de uma posição geográfica e a temperatura ambiente. Como exemplos, poderia-se definir um elemento contextual $(Fulano, Latitude)$, correspondente à latitude da posição geográfica de $Fulano$. Outro elemento contextual poderia ser $(PortoAlegre, Temperatura)$, representando a temperatura na cidade de Porto Alegre.

Cada elemento no conjunto de valores de qualidade Q, por sua vez, pode ser definido como uma tupla q:

$$q = (p, v) \tag{6.3}$$

onde p é um parâmetro de QoC quaisquer e v é o valor associado.

Dessa forma, é possível definir exemplos de instâncias de elementos contextuais. Por exemplo, uma instância que define o valor $Andando$ para o aspecto $Atividade$ da pessoa $Fulano$ obtido no instante de tempo t_1, poderia ser definido como:

$$((Fulano, Atividade), Andando, \{(Acuracia, 0, 89)\}, t_1, \{Secundario\}) \quad (6.4)$$

onde se pode perceber a definição de um valor de QoC (0,89 para acurácia) e da categoria $Secundario$ para a instância (o contexto é do tipo secundário, obtido a partir de outros contextos, como definido na Seção 2.2).

6.2 Modelagem Ontológica do Contexto

Entidades, aspectos e elementos contextuais representam termos cuja semântica precisa ser claramente definida, de forma que os agentes que compõe a arquitetura possam trocar informações com base em um vocabulário comum (ver Seção 5.7). Para tanto, uma ontologia de alto nível deve ser usada de forma a definir termos mais genéricos do modelo, compartilhados por qualquer domínio de aplicação. Tal ontologia pode ser estendida por aplicações para definição de termos e indivíduos específicos de um domínio.

Dessa forma, a ontologia a ser usada neste trabalho deve:

- ser capaz de representar os conceitos definidos na seção 6.1;
- ser suficientemente genérica para possibilitar sua extensão para diferentes domínios dentro de cidades inteligentes;
- possibilitar a modelagem de diferentes tipos de entidades possíveis em cidades inteligentes;
- possibilitar a associação de diferentes categorias a informações contextuais.

6.2.1 Ontologias de Alto Nível Existentes para Representação de Contexto

Um primeiro passo na direção da definição de uma ontologia a ser usada neste trabalho foi a análise de ontologias existentes de alto nível, para representação de contexto. Algumas dessas ontologias já foram definidas na literatura e são analisadas a seguir.

A ontologia *CONtext ONtology* (CONON) permite que informações contextuais sejam modeladas a partir de quatro conceitos base relacionados com entidades computacionais, localização, pessoas e atividades (WANG et al., 2004). Portanto, a ontologia

CONON, apesar de definir conceitos de alto nível relacionados a contexto, limita a representação de tais informações às categorias de localização, atividade, individual e representação do contexto de interesse computacional. Além disso, não é previsto a associação de informações temporais às informações de contexto.

A ontologia *CoBrA-ONT* (*Context Broker Architecture Ontology*) foi desenvolvida com o propósito de definir termos usados na comunicação entre agentes em uma arquitetura para gerenciamento de informações de contexto (CHEN; FININ; JOSHI, 2003). Porém, os conceitos de alto nível definidos limitam-se às categorias de localização, individual e atividade. Esta ontologia também não é suficientemente genérica para ser aplicada em diferentes domínios, já que é voltada para uso em aplicações de salas de reuniões inteligentes.

A ontologia *CoOL* (*Context Ontology Language*) modela contexto em três níveis: *aspecto*, *escala* e *informação de contexto* (STRANG; LINNHOFF-POPIEN; FRANK, 2003). Esta ontologia foi desenvolvida no contexto de um sistema distribuído de descoberta e execução de serviços na Web, porém seus conceitos podem ser adaptados em outros domínios.

SOUPA (*Standard Ontology for Ubiquitous and Pervasive Applications*) foi projetada para dar suporte a aplicações pervasivas (CHEN et al., 2004). Consiste em duas ontologias: *SOUPA Core* (vocabulário genérico comum a diferentes aplicações) e *SOUPA Extension* (suporte a diferentes domínios de aplicações). A ontologia *Core* define grupos de conceitos que descrevem pessoas, agentes, ações, políticas de segurança e privacidade, tempo, espaço e eventos. Apesar da ontologia possuir um conjunto interessante de conceitos para uso na modelagem de contexto em aplicações multiagentes, carece da possibilidade de representação de entidades que não sejam pessoas, lugares ou agentes (como objetos físicos ou virtuais, por exemplo), assim como contexto de interesse.

A ontologia utilizada no projeto CoDAMoS (*Context-Driven Adaptation of Mobile Services*) apresenta quatro conceitos principais de alto nível: usuário, ambiente, plataforma e serviço (PREUVENEERS et al., 2004). Possibilita sua extensão para diversos subdomínios, mesmo em cidades inteligentes. Entretanto, a representação de atividades que vão além do uso de aplicações em dispositivos móveis é bastante limitada.

A ontologia CACOnt (*Context-Aware Computing Ontology*) permite representar conceitos relacionados a usuários, dispositivos, serviços, localização e ambiente (XU et al., 2013). É possível representar diversas categorias diferentes de contexto, porém sem a associação de informação temporal, o que dificulta a implementação de histórico de

contextos.

Uma ontologia em três níveis é utilizada pela 3LConOnt (*Three-Level Context Ontology*): um nível superior (conceitos mais genéricos), um nível intermediário e um nível inferior (ontologias de domínio). O nível superior define basicamente dois conceitos: informação de contexto e entidade. Informações de contexto foram subdivididas em sete subclasses: Atividade, Localização, Estados, Papel, Ambiente, Perfil e Tempo (CABRERA; FRANCH; MARCO, 2019). Dessa forma, diversas categorias de contexto conceitual podem ser representadas nesta ontologia.

Por fim, a ontologia CAMeOnto (*Context Awareness Meta Ontology*) foi definida em dois níveis: um primeiro nível onde conceitos de alto nível foram definidos, e um segundo nível para ontologias específicas de domínio (AGUILAR; JEREZ; RODRÍGUEZ, 2018). Seis conceitos chave foram definidos no primeiro nível: Usuário, Atividade, Tempo, Dispositivo, Serviços e Localização .

Como apresentado, existem diversas ontologias de alto nível que se propõe a modelar conceitos relacionados a contexto. De forma geral, essas ontologias trazem soluções interessantes, porém não são suficientemente ricas para representar aspectos de contexto necessários a este trabalho. Em primeiro lugar, algumas dessas ontologias, mesmo se propondo como sendo de alto nível, foram modeladas com ambientes específicos em mente (como salas inteligentes), o que limitam sua aplicabilidade para representar contexto em cidades inteligentes. Em segundo lugar, carecem de conceitos para modelagem de informações importantes em ambientes como cidades inteligentes, tais como contextos de interesse e entidades. Em terceiro lugar, informações contextuais podem ser classificadas combinando-se categorias operacionais e contextuais, o que não é suportado pelas ontologias analisadas.

6.2.2 Ontologia para Representação de Contexto Definida neste Trabalho

Considerando os requisitos para a ontologia a ser usada neste trabalho e as limitações das ontologias existentes, decidiu-se elaborar uma nova ontologia para modelagem de contexto. Para o desenvolvimento desse modelo utilizou-se a metodologia proposta por Noy, McGuinness et al. (2001) chamada de *Ontology Development 101*. A metodologia é dividida em sete passos principais.

No primeiro passo deve-se determinar o escopo e o domínio da ontologia. O objetivo desta ontologia é descrever conceitos utilizados por agentes para representação de

elementos contextuais e suas instâncias. Assim sendo, esta ontologia deve prover uma representação de mais alto nível de contexto, deixando a representação de conceitos mais específicos e suas relações para ontologias de domínio derivadas. Portanto, não há um domínio de aplicação específico, mas esta ontologia visa ser extensível a qualquer subdomínio dentro de cidades inteligentes e ambientes inteligentes. O escopo da ontologia engloba os conceitos e termos a serem utilizados e compartilhados entre agentes com relação a elementos contextuais e suas instâncias. A ontologia também deve suportar a representação de histórico de contextos.

O segundo passo da metodologia considera a reutilização de outras ontologias. Como já analisado na seção 6.2.1, nenhuma ontologia de alto nível pré-existente específica foi utilizada neste trabalho. Porém, estas ontologias podem ser utilizadas para representação de conceitos específicos de contexto em subdomínios. Além disso, outras ontologias também foram consideradas neste trabalho para reutilização, como a ontologia *DOLCE+DNS Ultralite* (DUL)[1] para representação de entidades em geral, a ontologia *Time*[2] para representação de conceitos temporais, e a ontologia *QUDT*[3] para especificação de quantidades e unidades de medida.

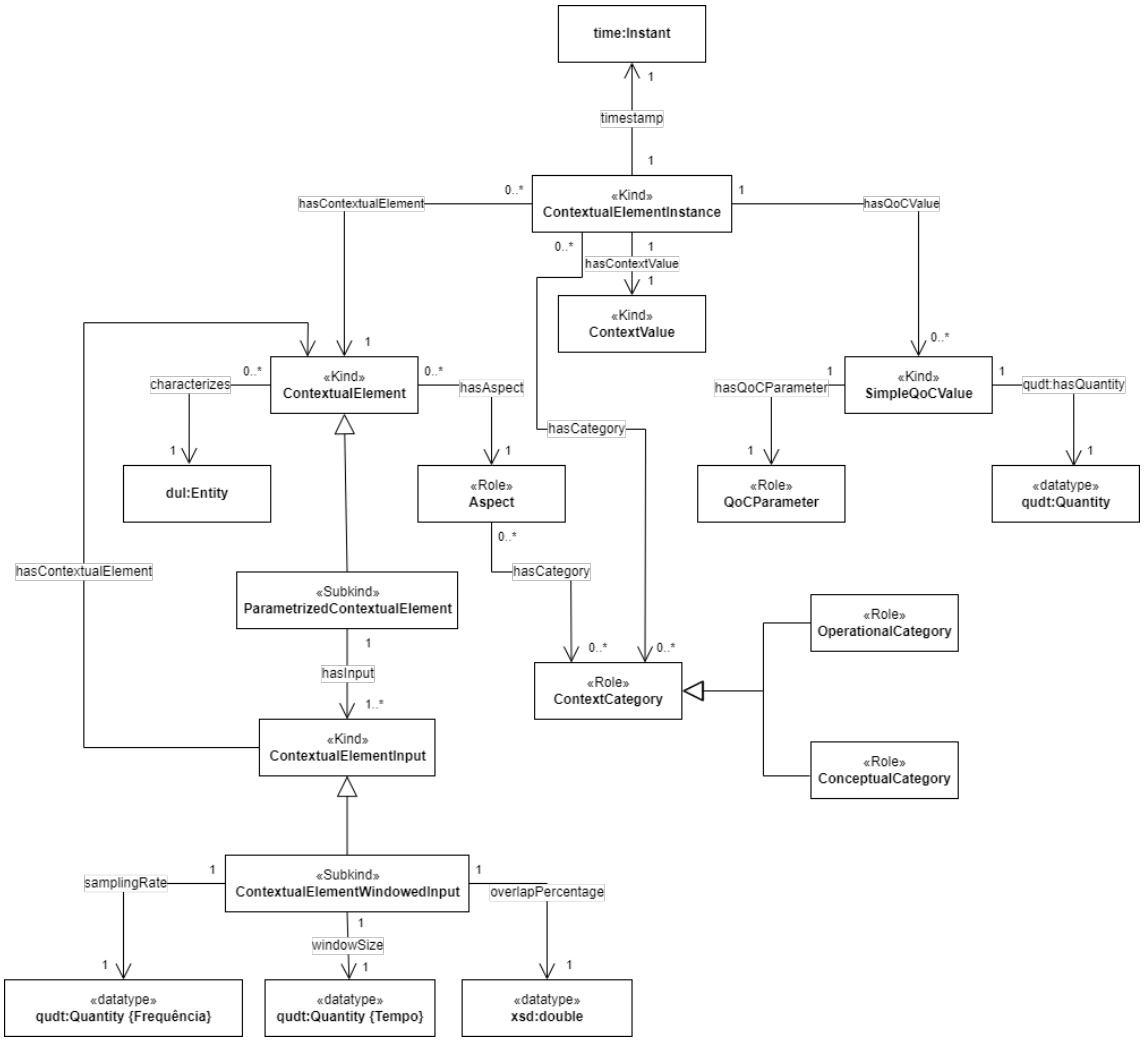
O terceiro passo está relacionado à enumeração de termos importantes relacionados ao domínio de aplicação da ontologia. Através das definições já mencionadas neste trabalho foi possível identificar os seguintes termos: *elemento contextual*, *instância de elemento contextual*, *entidade*, *valor de contexto*, *instante de tempo*, *aspecto*, *qualidade de contexto*, *parâmetro de qualidade de contexto*, *valor de qualidade de contexto*, *categoria de contexto*.

Os passos quatro, cinco e seis estão relacionados, respectivamente, à definição da hierarquia e relacionamentos entre classes, definição de propriedades e restrições. Após a aplicação desses passos, obteve-se o conjunto de classes mostrado na Figura 6.1.

O principal classe nesta ontologia é *ContextualElementInstance*, que representa uma instância de elemento contextual. Cada uma destas instâncias está associada a um determinado instante de tempo (representado por um indivíduo da classe *Instant* definida na ontologia *Time*), que indica quando a instância foi obtida. Além disso, toda instância possui um elemento contextual associado a ela, um conjunto opcional de valores de QoC relacionados e também um conjunto opcional de categorias de contexto associadas. Este último permite relacionar categorias operacionais e conceituais a instâncias de elementos

[1] <http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite>
[2] <https://www.w3.org/TR/owl-time/>
[3] <https://qudt.org/>

Figura 6.1: Ontologia para representação de contexto definida neste trabalho.



Fonte: O Autor

contextuais. Por exemplo, é possível definir que uma instância de elemento contextual relacionada ao aspecto *latitude* de uma pessoa, obtida a partir de um receptor GPS, está relacionada às categorias *localização* (categoria conceitual) e *contexto primário* (categoria operacional). Uma informação de *latitude* obtida a partir do processamento de um sinal GSM de uma torre de telefonia celular poderia ser classificada com as categorias *localização* e *contexto secundário*.

Dados de QoC relacionados a instâncias de elementos contextuais possuem um parâmetro (uma instância da classe *QoCParameter*) e um valor (uma instância da classe *Quantity*, definida na ontologia QUDT). Uma instância da classe *Quantity* possui um valor e uma unidade associadas, além de facilitar a conversão de valores entre unidades diferentes.

A classe *ContextValue* permite representar um valor qualquer associado a uma instância de elemento contextual. Esses valores podem ser definidos de forma mais específica em ontologias de domínio derivadas. Possibilidades incluem, por exemplo, valores obtidos a partir da observação de sensores (como por exemplo, instâncias da classe *Observation* definida na ontologia *SOSA - Sensor, Observation, Sample, and Actuator*[4]) ou valores de mais alto nível representados na forma de outras ontologias de contexto específicas de domínio.

Elementos contextuais são representados através da classe *ContextualElement*. Cada elemento contextual está relacionado a uma entidade (representada como uma instância da classe *Entity* definida na ontologia *DUL*) e a um aspecto (instância da classe *Aspect*). Cada aspecto pode estar diretamente relacionado a categorias de contexto. Por exemplo, coordenadas de latitude e longitude sempre estarão associadas a uma categoria de *localização*. Categorias relacionadas a um aspecto serão automaticamente associadas a todas as instâncias de elementos contextuais relacionadas a esse aspecto.

Alguns elementos contextuais derivados podem ser caracterizados também por outros elementos contextuais de entrada. Um exemplo de situação como essa seria a média do componente X da aceleração de um determinado *smartphone*. Tal elemento contextual está associado a uma entidade (smartphone) e a um aspecto (média), mas também é caracterizado por um outro elemento contextual de entrada (componente X da aceleração do smartphone). Tais elementos contextuais podem ser representados na ontologia como instâncias da classe *ParametrizedContextualElement*, que devem estar associadas a pelo menos uma entrada representada por uma instância da classe *ContextualElementInput*. É possível que uma determinada entrada seja um conjunto de valores resultado de uma janela de observações. A classe *ContextualElementWindowedInput* permite representar a entrada de conjuntos de valores através da especificação de uma frequência de amostragem, tamanho (em unidades de tempo) e percentual de sobreposição.

6.3 Visão Geral da Arquitetura Multiagente Desenvolvida

Este trabalho tem como principal objetivo fornecer uma arquitetura multiagente capaz de responder requisições de aplicações a elementos contextuais relacionados dentro de uma cidade inteligente. Nessa arquitetura, cada agente pode acessar alguma fonte de contexto (sensores físicos, virtuais, lógicos) e/ou prover algum mecanismo de infe-

[4]<https://www.w3.org/TR/vocab-ssn/>

rência capaz de gerar alguma informação de contexto de mais alto nível (ver Seção 2.4). Dessa forma, diferentes fontes de contexto podem ser integradas, de forma a fornecer as informações solicitadas.

Estes agentes estão inseridos em um ambiente composto por outros agentes e fontes de dados (ver Seção 5.2). Este ambiente é parcialmente observável, pois cada agente tem uma visão limitada sobre os agentes e fontes de dados disponíveis. Para cada agente o ambiente é determinístico, pois cada ação executada por um agente sempre gerará um mesmo resultado (as ações executadas pelos agentes envolvem basicamente consultas). Por fim, o ambiente também é dinâmico, pois o estado deste pode mudar de formas inesperadas o que pode ocasionar falhas (conexões de rede podem ser quebradas, sensores podem apresentar mal-funcionamento, problemas de hardware podem ocorrer, etc.). Um exemplo seria o caso de uma fonte de dados inesperadamente se tornar indisponível por causa de uma falha em um sensor ou conexão de rede.

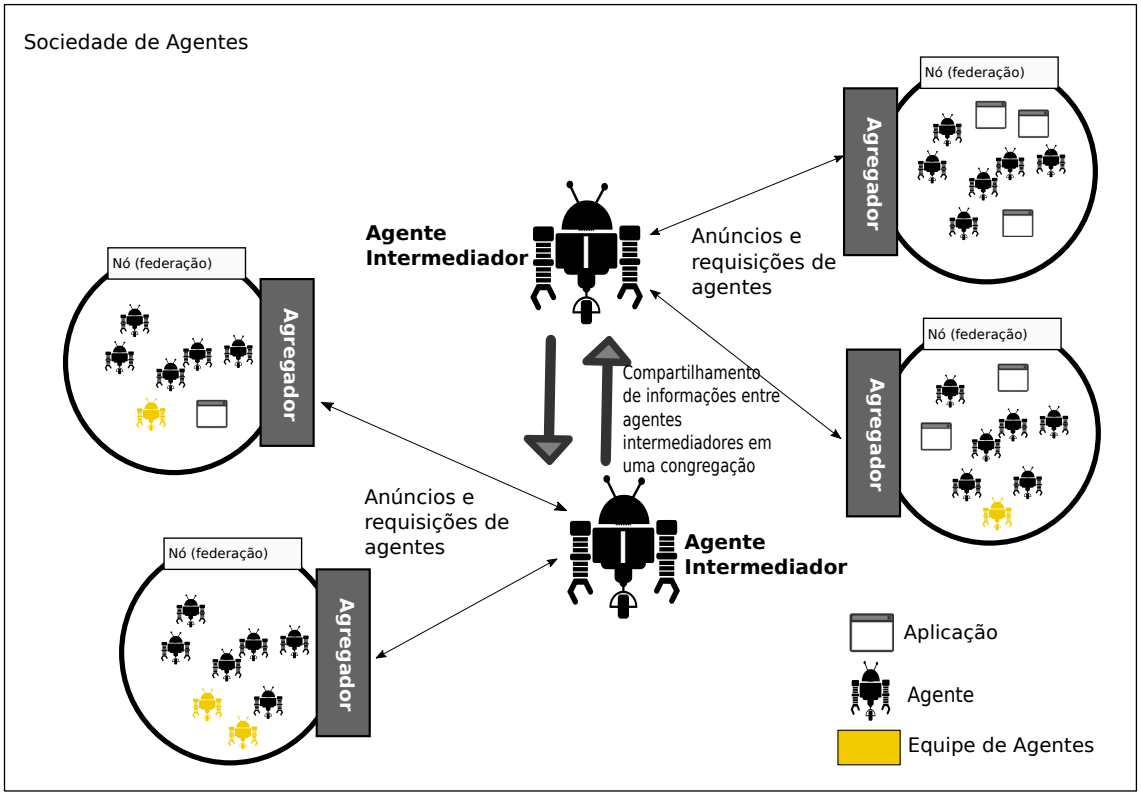
Uma visão geral sobre a arquitetura desenvolvida é mostrada na Figura 6.2. Cada fonte de contexto está associada a um ou mais agentes. Os agentes no modelo estão distribuídos uma organização composta (ver Seção 5.4). Primeiramente, todo a arquitetura pode ser vista como uma grande sociedade, onde agentes podem entrar e sair. Agentes podem entrar na sociedade fornecendo acesso a novos sensores e/ou a novos mecanismos de inferência. Estes agentes podem sair, caso suas capacidades não sejam mais necessárias. Esta caraterística da arquitetura permite que novas fontes de contexto possam ser adicionadas de uma forma *plug-and-play*, promovendo a escalabilidade (ver Seção 3.5).

Em uma sociedade de agentes, um conjunto de convenções sociais deve existir. Na arquitetura desenvolvida neste trabalho, o seguinte conjunto de leis sociais foi definido:

- todo novo agente que ingressa na sociedade deve anunciar suas capacidades (quais informações de contexto pode fornecer);
- a comunicação entre os agentes ocorre através de troca de mensagens seguindo o padrão definido em alguma linguagem de comunicação (ver Seção 5.5);
- agentes são cooperativos por natureza.

Os agentes na arquitetura são organizados em conjuntos lógicos chamados *nós*. Um agente pode fazer parte de apenas um nó. Cada nó funciona como uma federação, onde um agente chamado *agregador* faz a função de facilitador e gerencia toda a informação que entra e/ou sai do nó. A troca de informações entre nós é realizada através da presença de agentes intermediadores. Esses agentes estão organizados em uma congrega-

Figura 6.2: Visão geral da arquitetura desenvolvido.



Fonte: O Autor

ção na forma de uma rede *peer-to-peer*, de forma a conectar nós sem a necessidade de uma entidade centralizadora. Equipes de agentes podem ser formadas, mesmo entre agentes de nós diferentes, a fim de que possam cooperar para fornecer os elementos contextuais solicitados por uma aplicação.

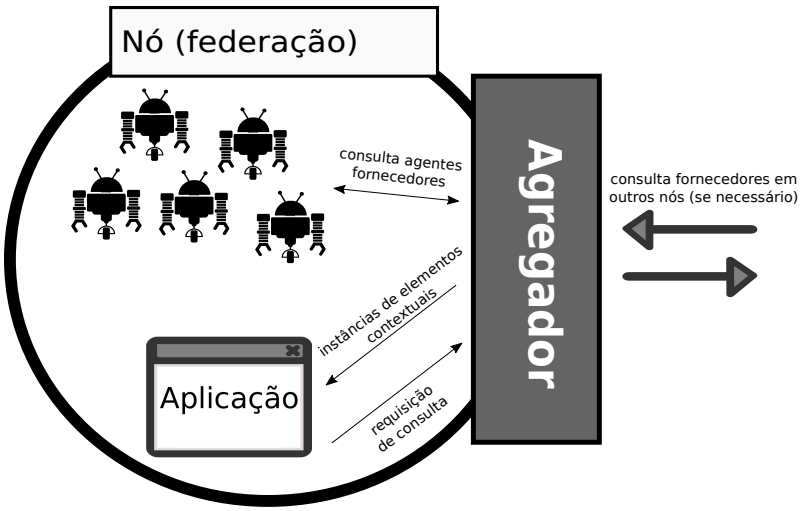
Aplicações também funcionam como participantes da arquitetura. Entende-se por aplicação, neste trabalho, qualquer entidade de software diferente de um agente, que funcione como um consumidor de contexto gerenciado pela arquitetura. Cada aplicação deve se inserir dentro de um nó e acessar informações de contexto providas a partir dele.

6.4 Aquisição de Contexto pelas Aplicações

Cada aplicação pode adquirir instâncias de elementos contextuais da arquitetura de duas formas: através de *consultas* ou através de um processo de *inscrição/notificação*. Primeiramente, será apresentada uma visão geral sobre o processo de consulta, ilustrado na Figura 6.3. Este processo, segue a sequência de passos descrita a seguir.

1. A aplicação envia ao agente agregador do nó ao qual está relacionada a requisição de consulta por elementos contextuais.
2. O agregador realiza uma busca por agentes locais que possam fornecer os elementos contextuais solicitados. Caso algum elemento contextual não possa ser fornecido por agentes locais, uma busca junto aos intermediadores é executada a fim de identificar nós remotos que possam fornecer essa informação.
3. O agregador consulta os agentes selecionados. Um processo de fusão é realizado, caso exista mais de um agente que forneça um mesmo elemento contextual. Esse processo de fusão pode ocorrer através de uma seleção de instâncias utilizando parâmetros de QoC (ver Seção 2.5) ou algum dos processos conhecidos de fusão descritos na Seção 3.6.
4. As instâncias de elementos contextuais obtidas pelo agregador são enviadas à aplicação.

Figura 6.3: Visão geral da aquisição de contexto por consulta.

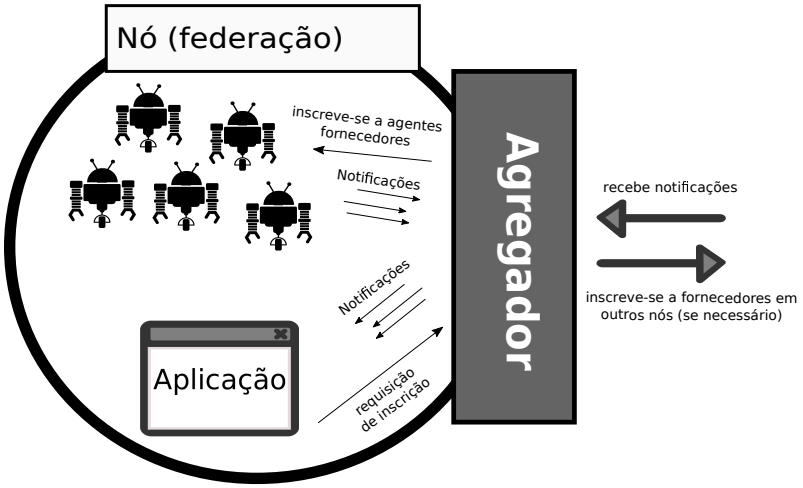


Fonte: O Autor

Uma visão geral sobre o processo de inscrição/notificação é mostrada na Figura 6.4. O processo começa com a aplicação enviando uma requisição de inscrição ao agregador do nó ao qual está associada, onde demonstra o interesse em receber atualizações periódicas sobre determinados elementos contextuais. O agregador realizará as inscrições necessárias junto a agentes provedores de contexto locais e de outros nós (se necessário). Estes agentes enviarão notificações periódicas contendo novas instâncias de elementos contextuais, que serão encaminhadas pelo agregador à aplicação requisitante. Assim como na consulta, operações de fusão de dados podem ser aplicadas em casos

em que existam mais de um fornecedor enviando notificações sobre um mesmo elemento contextual. Por fim, uma aplicação pode cancelar uma inscrição a qualquer momento. Nesse caso, as notificações deixam de ser enviadas. Para receber os dados novamente, a aplicação deve realizar uma nova inscrição.

Figura 6.4: Visão geral da aquisição de contexto por inscrição/notificação.



Fonte: O Autor

Tanto uma requisição de consulta, quanto de inscrição, contém um conjunto finito de itens $CR = i_1, i_2, ..., i_N$. Cada item i é uma tupla, tal que:

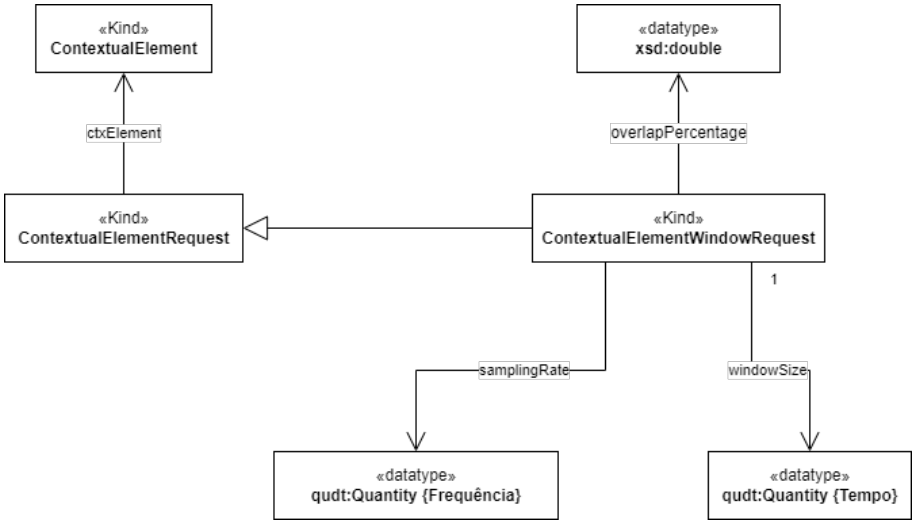
$$i = (rd, fs) \tag{6.5}$$

onde rd contém os dados da requisição e fs especifica a estratégia de fusão de dados a ser utilizada pelo agregador com relação a este item.

O principal dado especificado em rd é a especificação do elemento contextual desejado, uma instância da classe *ContextualElement* presente na ontologia (Seção 6.2.2). Dados extras podem ser necessários, em situações específicas. Por exemplo, uma aplicação pode estar interessada em uma janela de observação de um determinado elemento contextual. Essa janela contém instâncias desse elemento capturadas em um determinado intervalo de tempo e com uma determinada frequência de amostragem. A obtenção de janelas consecutivas também pode exigir a especificação de um percentual de sobreposição. Portanto, o componente rd, pode ser ampliado para situações específicas. Assim, decidiu-se ampliar a ontologia deste trabalho, acrescentando uma classe chamada *ContextualElementRequest*, mostrada na Figura 6.5.

Dessa forma, uma requisição por um elemento contextual simples poderia ser re-

Figura 6.5: Classes na ontologia para representação de dados relacionados à requisição de elementos contextuais.



Fonte: O Autor

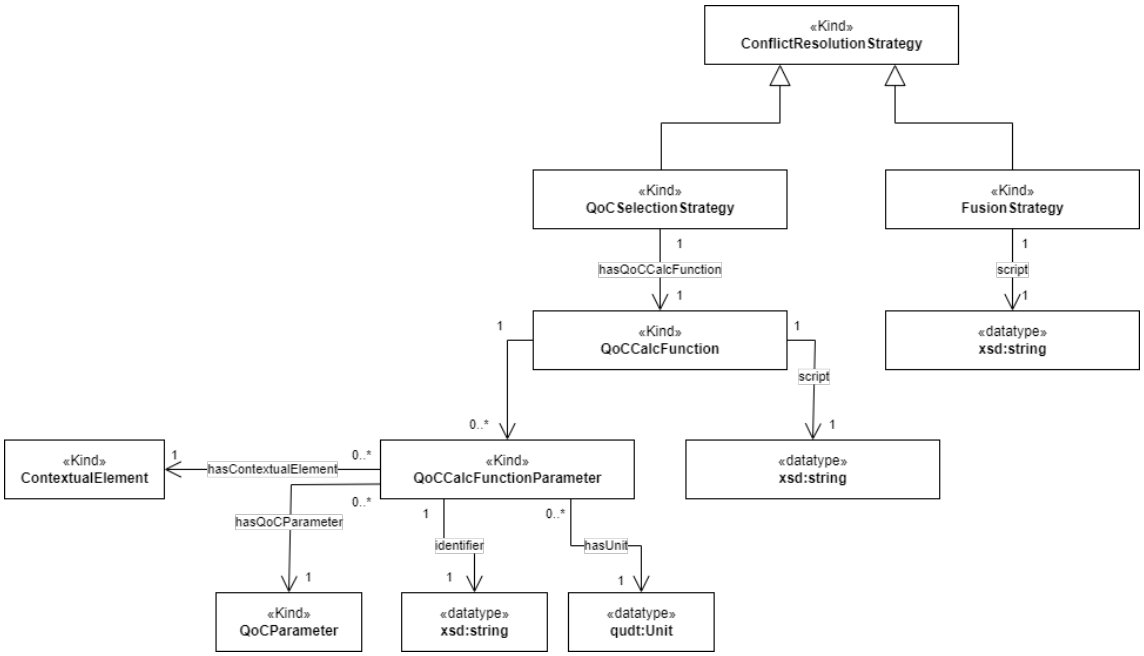
presentada por uma instância da classe *ContextualElementRequest*. Por outro lado, uma requisição de uma janela de observação poderia ser representada por uma instância da classe *ContextualElementWindowRequest*. Estruturas diferentes para requisições de elementos contextuais que exijam outros tipos de dados podem ser representadas a partir de subclasses das classes já definidas.

A estratégia de fusão de dados fs especificada pela aplicação na requisição será a estratégia utilizada pelo agregador para fundir os dados recebidos dos agentes para cada elemento contextual requisitado. Duas estratégias foram definidas na arquitetura: uma estratégia de seleção de instância baseada em QoC (a instância selecionada é aquela que apresenta o melhor valor de uma função de utilidade calculado com base nos valores de QoC presentes nas próprias instâncias) e outra estratégia baseada em algum método de fusão (uma nova instância é gerada resultado da fusão das instâncias obtidas). Essas estratégias podem ser representadas a partir, por exemplo, de *scripts* escritos em alguma linguagem geral ou específica.

As estratégias descritas também foram modeladas na forma de conceitos na ontologia como descrito na Figura 6.6. Estratégias de seleção por QoC são modeladas através de instâncias da classe *QoCSelectionStrategy*, enquanto estratégias de fusão são modeladas através da classe *FusionStrategy*. Estratégias de fusão possuem um *script* associado a elas que especifica como a ação de fusão deve ocorrer. Já estratégias de fusão por QoC possuem uma função de cálculo associada (a função de utilidade), que gera um valor nu-

mérico utilizado para verificar qual a melhor instância a ser selecionada. Essa função também possui um *script*, que define como o cálculo ocorre. Parâmetros também podem estar relacionados a essa função, que por sua vez estão associados a um elemento contextual, um parâmetro de QoC, uma unidade (utilizada para conversão de valores de QoC de instâncias para uma unidade comum) e também um identificador (utilizado para identificar o parâmetro dentro do *script*). Assim, para cada instância conflituosa a função de utilidade é executada, passando-se os valores de QoC relacionados a essa instância e que são esperados como parâmetros para a função. Cada valor de QoC é convertido para a unidade esperada no parâmetro, caso seja necessário.

Figura 6.6: Classes na ontologia para representação de estratégias de resolução de conflitos (fusão e seleção).



Fonte: O Autor

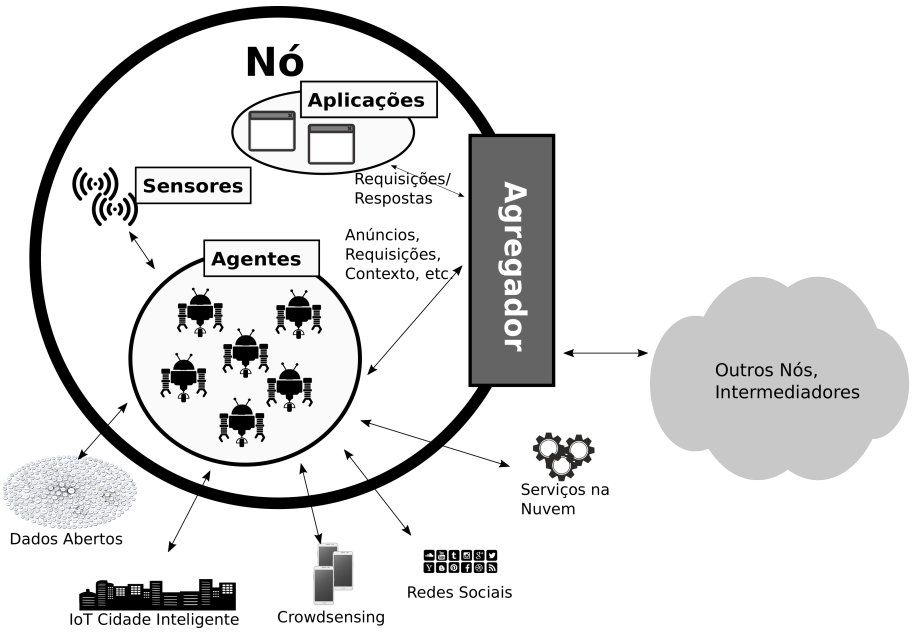
6.5 Nós

Como visto anteriormente, nós são basicamente *containers* para agentes responsáveis por obter certas informações de contexto a partir de fontes externas. Essas fontes externas podem ser qualquer fonte de dados de contexto esperada em uma cidade inteligente e descritas na Seção 2.7.

Uma visão geral de um nó é mostrada na figura 6.7. Um nó pode ser consultado

por uma aplicação ou por outros nós através de intermediadores. Um agente agregador desempenha um papel central em um nó ao receber essas requisições e respondê-las. A cada requisição recebida, agentes específicos devem ser acionados para gerar essas informações. Afim de obter essas informações, agentes podem ter acesso a diferentes fontes de dados conectadas ao nó.

Figura 6.7: Esquema geral de um nó.



Fonte: O Autor

Nós podem ser acrescentados ao sistema a qualquer momento, em tempo de execução, seja por desenvolvedores ou administradores de sistemas. Ao entrar em funcionamento, um nó deve se anunciar à intermediadores mais próximos. Nesse anúncio devem constar quais informações de contexto podem ser fornecidas pelo nó a serviços externos e o nível de qualidade destas informações. Esse anúncio é descrito em detalhes na seção 7.2. Os nós também podem enviar atualizações desses dados aos intermediares próximos quando, por exemplo, um novo agente é acrescentado ao nó.

Novos agentes podem ser incluídos em um nó a qualquer momento, pode desenvolvedores ou administradores de sistemas. Agentes podem ser incluídos com capacidades de sensoriamento através de novos sensores, ou com novas capacidades de inferência sobre dados existentes. Dessa forma, cada agente é capaz de obter alguma informação de contexto que se encaixa em alguma categoria e em algum nível. Alguns agentes podem captar informações de baixo nível (contexto bruto) diretamente de sensores, enquanto outros podem realizar inferências sobre esses dados e produzir contexto de mais alto nível.

Ou seja, alguns agentes podem obter contexto primário, enquanto outros obtém contexto secundário que pode depender, inclusive, de instâncias de elementos contextuais fornecidas por outros agentes. Quando isso acontece, agentes podem formar *equipes temporárias*, para que cooperem fornecendo dados necessários uns aos outros, seja para responder a consultas ou para gerar notificações.

Cada agente deve anunciar a si mesmo ao agregador. Assim, o agregador é capaz de registrar todos os agentes presentes em um nó, como também suas capacidades. Assim, ao ser consultado por alguma aplicação ou intermediador sobre a oferta de informações de contexto, uma resposta rápida pode ser dada. Para maiores detalhes sobre este processo, ver a seção 7.1.

Informações de contexto solicitadas pela aplicação podem não ser fornecidas por nenhum agente local. Nesse caso, agregadores iniciam uma busca dessas informações em outros nós próximos e intermediadores (seções 7.3 e 7.4).

6.6 Agregadores

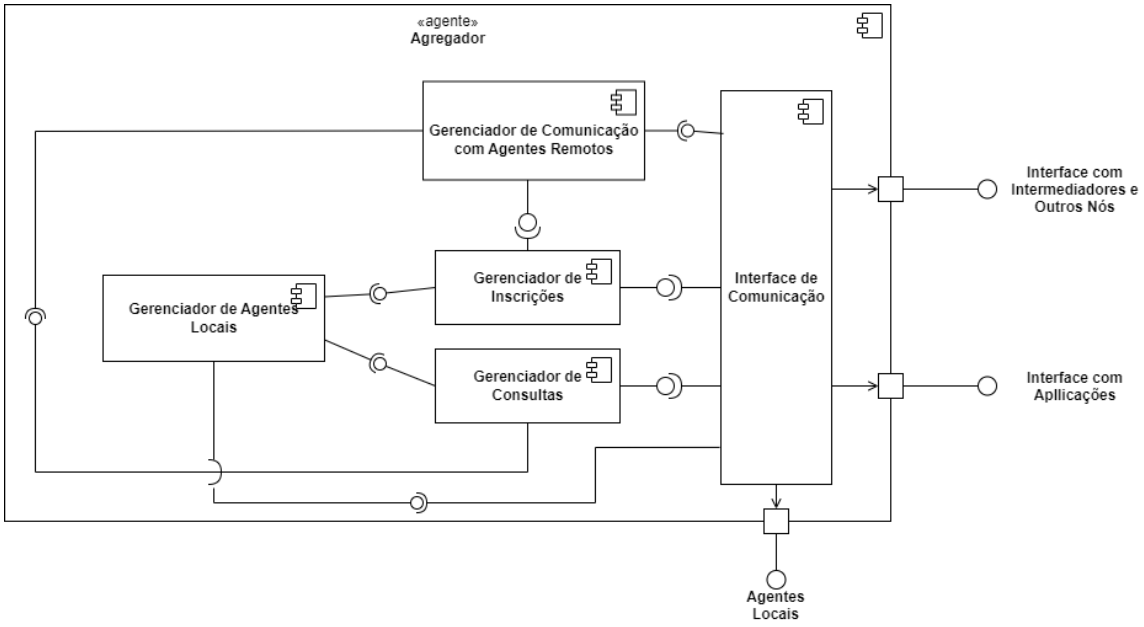
Cada nó possui um único agente agregador que, como já descrito, tem um papel central e possui as seguintes responsabilidades:

- receber e responder consultas, inscrições e cancelamentos enviados por aplicações e/ou outros agentes;
- enviar notificações de alterações percebidas em informações contextuais a aplicações inscritas;
- selecionar agentes locais que possam responder às requisições enviadas por aplicações;
- receber anúncios de novos agentes e efetuar outras tarefas de gerenciamento de agentes locais;
- realizar anúncio em intermediadores próximos;
- contactar intermediadores para obter informações de contexto inexistentes no nó local ou que possam ser de maior qualidade;

A fim de alcançar estes objetivos, o agente agregador conta com a estrutura mostrada na Figura 6.8. Todo agregador é internamente organizado em um conjunto de componentes. A *interface de comunicação* é responsável por receber solicitações provenientes de aplicações, intermediadores, outros agregadores e agentes locais, enviar respostas a so-

licitações e realizar consultas a intermediadores e outros nós. Ela também é responsável por efetuar o anúncio do nó junto a intermediadores próximos e receber anúncios de agentes locais. O *gerenciador de consultas* é responsável por receber consultas e processar suas respostas, consultando agentes específicos através do gerenciador de agentes locais e o gerenciador de comunicação com agentes remotos (se necessário). O *gerenciador de inscrições* processa inscrições ou cancelamentos recebidas pela interface de comunicação. O *gerenciador de agentes locais* tem a responsabilidade de selecionar agentes que respondam às solicitações recebidas, receber anúncios de agentes e efetuar outras tarefas de gerenciamento. Para tanto, esse componente mantém um registro de todos os agentes locais e suas capacidades (quais elementos contextuais cada um pode prover). Por fim, o *gerenciador de comunicação com agentes remotos* é responsável pela identificação de intermediadores próximos e o acesso, quando necessário, a agregadores remotos. Existe um registro local de intermediadores identificados, bem como um *cache* de agregadores remotos já contactados anteriormente.

Figura 6.8: Estrutura interna de um agregador.



Fonte: O Autor

Um agregador portanto é responsável por diversos processos envolvendo diversos componentes internos diferentes. Nas próximo capítulo serão abordados em mais detalhes os passos envolvidos em cada processo e as funções desempenhadas por cada componente.

6.7 Agentes Fornecedores

Agentes fornecedores são os atores responsáveis por obter instâncias de elementos contextuais, seja diretamente através de uma fonte de contexto, ou através de mecanismos de inferência. Estão organizados na forma de uma sociedade, onde cada um possui capacidades distintas. Novos agentes podem ingressar nessa sociedade a qualquer momento. Ao ingressar na sociedade, um agente deve anunciar suas capacidades ao agregador local, ficando seu registro a cargo do componente *gerenciador de agentes locais* desse agregador.

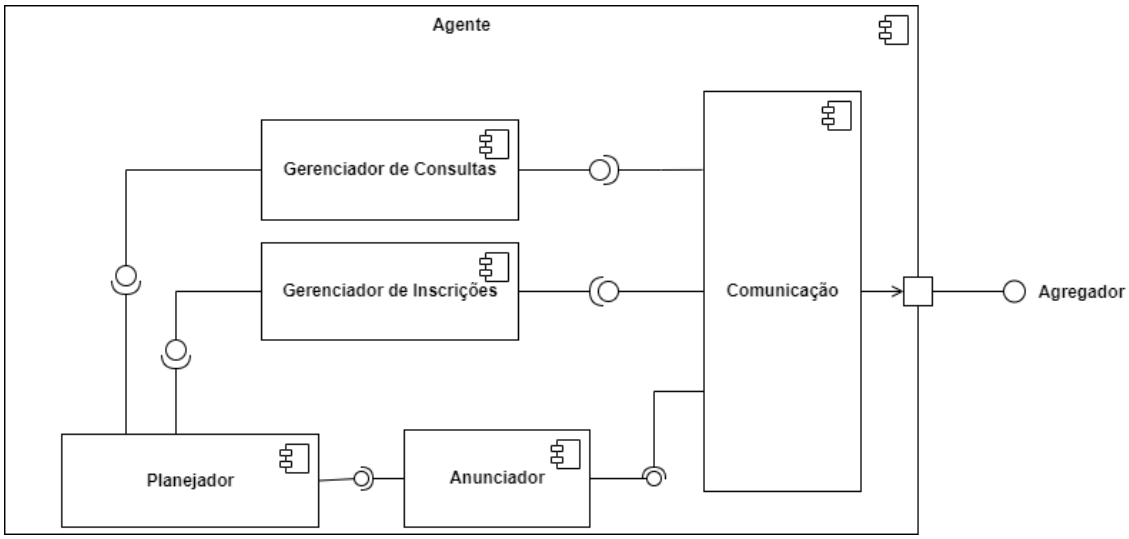
Cada agente está organizado em uma arquitetura baseada em componentes. Essa arquitetura é mostrada na Figura 6.9.

- *Comunicação*, responsável por gerenciar o recebimento e envio de mensagens. Toda mensagem recebida chegará por este componente e toda mensagem será enviada através dele.
- *Gerenciador de Consultas*, responsável por receber e processar requisições de consultas recebidas pelo agente. Processar essas requisições envolve requisitar ao planejador que execute um plano apropriado para obter o resultado esperado.
- *Gerenciador de Inscrições*, responsável por receber e processas requisições de inscrições recebidas pelo agente, assim como enviar notificações ao agregador relacionadas a essas inscrições. Notificações também geram requisições ao planejador para a execução de planos apropriados.
- *Anunciador*, responsável por enviar um anúncio das capacidades do agente ao agregador local. As capacidades do agente dizem respeito aos elementos contextuais que esse agente é capaz de prover.
- *Planejador*, que obtém planos a serem executados pelo agente para responder a consultas e inscrições.

6.7.1 Planos

Ações de agentes são guiadas por planos. Um agente pode possuir um ou mais planos, cada um associado a um elemento contextual diferente. Planos são executados por um agente em duas situações:

Figura 6.9: Estrutura interna de um agente.



Fonte: O Autor

- Quando o agente recebe uma mensagem solicitando uma consulta a um elemento contextual. O componente de comunicação recebe a mensagem e repassa a solicitação ao módulo gerenciador de consultas. Este, por sua vez, procura por um plano associado ao elemento contextual solicitado e executa o plano.
- Há necessidade de gerar uma informação de contexto para enviar uma notificação relacionada a uma inscrição previamente recebida. Nesse caso, o componente de gerenciamento de inscrições detecta que há a necessidade de uma nova notificação e executa o plano associado ao elemento contextual solicitado na inscrição anterior repassada ao agente.

Planos são especificados seguindo uma estrutura do tipo *pipe-and-filter* proposta no trabalho de Sabagh and Al-Yasiri (2015). Um plano é formado por um conjunto de processos modelados como *filtros* ou *módulos*, enquanto o fluxo de informação entre os módulos é modelada na forma de *canos ou conexões* (Figura 6.10). Módulos apresentam uma interface bem definida, um conjunto de elementos contextuais que necessitam como entrada (um conjunto de dependências) e podem ser reutilizados pelos agentes em diferentes planos, o que permite uma alta reusabilidade.

A arquitetura acima foi escolhida para planos por possibilitar que os mesmos sejam facilmente construídos e alterados através do uso de blocos funcionais. Neste trabalho, três diferentes tipos de módulos podem ser construídos e usados em planos:

- *Getter*: responsáveis pela aquisição de informações de contexto necessárias para

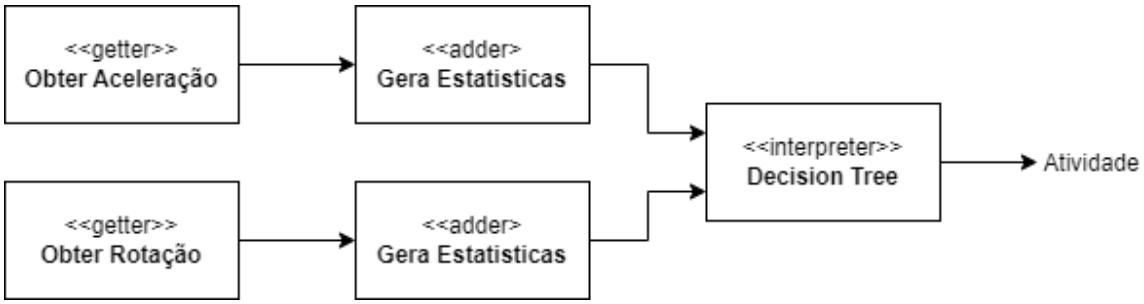
Figura 6.10: Esquema geral da arquitetura *pipe-and-filter*.

Fonte: O Autor

o plano. Estes módulos podem extrair dados diretamente de fontes de dados ou através de outros agentes (neste último caso, ver seção 7.3).

- *Adder*: responsáveis por agregar informações de contexto na forma de outras informações. Por exemplo, é comum que dados de alguns sensores, como acelerômetros ou giroscópios, sejam agregados utilizando coeficientes ou medidas estatísticas, como médias, desvio padrão, etc. Outras formas de agregação de dados também podem ser executadas, como fusão de dados heterogêneos de baixo nível.
- *Interpreter*: onde mecanismos de inferência devem ser implementados. Estes blocos são capazes de gerar contexto em um nível mais alto.

Figura 6.11: Exemplo de plano para obtenção da atividade de um usuário.



Fonte: O Autor

A Figura 6.11 mostra um exemplo de plano. Os módulos *Obtém Aceleração* e *Obtém Rotação* obtém dados, respectivamente, de um acelerômetro e um giroscópio em um *smartphone*. Estes dados são obtidos na forma de janelas de leitura, em que diversos valores são lidos em um intervalo de tempo. O módulo *Gera Estatísticas* calcula médias, desvios padrões, e outras medidas estatísticas nos resultados de ambos os módulos anteriores. Finalmente, o módulo *Decision Tree* é responsável por inferir a atividade de um usuário a partir de uma árvore de decisão.

É possível perceber neste exemplo que a arquitetura de planos é bastante genérica.

É possível desenvolver módulos para diferentes tipos de situações e aplicá-los a diferentes planos.

A função do componente planejador em um agente é, portanto, a partir da requisição de um elemento contextual, elaborar um plano a partir dos módulos disponíveis no agente. Segundo mostrado na Seção 5.3, para gerar um plano, um planejador deve receber um objetivo, uma listagem de ações disponíveis, e o estado atual do ambiente. O planejador de agentes locais utiliza como objetivo o elemento contextual requisitado. As ações consideradas são os módulos presentes no agente. O estado do ambiente não é considerado no planejamento. Cada módulo é visto como uma ação possível, que deve especificar argumentos (entradas), pré-condições e pós-condições. Cada módulo possui uma identificação e um possível conjunto de entradas (para módulos que geram contexto secundário). Cada entrada é uma requisição por um elemento contextual. Módulos não possuem pré-condições e devem especificar um resultado (um elemento contextual).

O processo de planejamento executado em um agente fornecedor é baseado em uma busca em regressão, onde se inicia pela busca de um módulo que forneça como resultado o elemento contextual informado na requisição. Caso este módulo possua entradas, a busca prossegue, procurando por outros módulos que possam prover essas entradas. A busca termina quando todas as entradas que podem ser providas por módulos locais foram satisfeitas. Caso existam entradas que não correspondam a nenhum módulo local, o agente envia uma requisição ao agregador por tais informações. Cada um desses elementos contextuais é associado a um *módulo temporário*, que recebe as instâncias enviadas pelo agregador e as fornece ao plano (ver seções 7.3, 7.4 e 7.5). A execução do plano começa com os módulos que não possuem entradas e continua com os módulos que já possuem todas as entradas satisfeitas, até chegar ao módulo que fornece o elemento contextual requisitado.

6.8 Intermediadores

Intermediadores são agentes responsáveis por interligar nós. Basicamente, as atividades desempenhadas por intermediadores são:

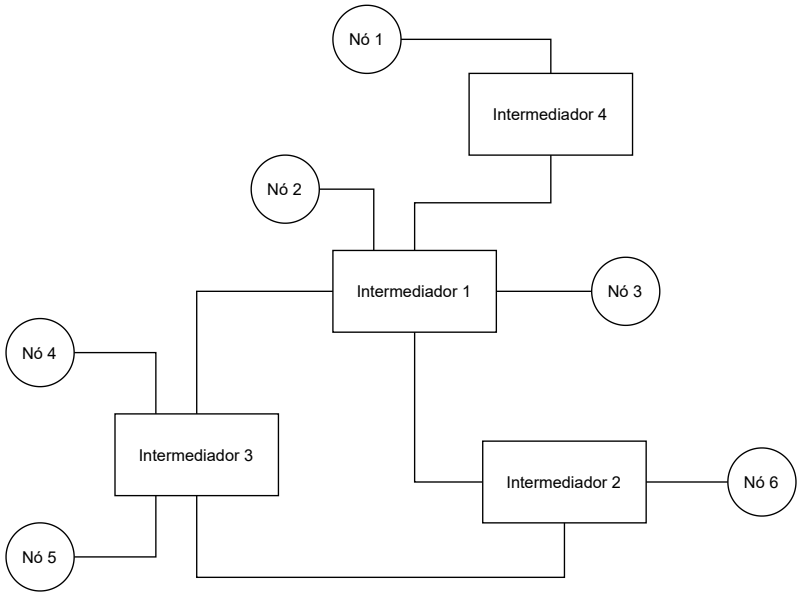
- receber anúncios e atualizações de nós, enviados por agregadores;
- receber solicitações de agregadores por agentes e enviar uma proposta;
- manter contato com outros intermediadores, formando uma rede de contatos;

- enviar/receber solicitações para/de outros intermediadores.

Intermediadores basicamente mantém uma base de conhecimento interna responsável por armazenar o conjunto de nós conectado a ele, suas capacidades, e sua rede de contatos. Assim, ao ser consultado por um agregador, o que o intermediador faz é acessar sua base de conhecimento e enviar um conjunto contendo agregadores de nós que possam responder aos requisitos enviados.

A rede de contatos é um ponto crucial para que nós remotos possam ser visíveis uns aos outros. Essa rede forma um arquitetura *peer-to-peer* entre intermediadores, permitindo que estes possam ser facilmente distribuídos e sem a necessidade de um nó central de controle. Uma rede de contatos deve se manter em constante atualização, para que novos agentes possam ser visíveis na rede o mais rápido possível. Um exemplo de rede de intermediadores é mostrada na Figura 6.12.

Figura 6.12: Exemplo de rede de contatos entre intermediadores.



Fonte: O Autor

No exemplo mostrado há nós diferentes conectados a intermediadores diferentes. Pode-se perceber alguns contatos entre intermediadores. Por exemplo, a rede de contatos do intermediador 1 é formada pelos intermediadores 2, 3 e 4. Por outro lado, a rede de contatos do intermediador 4 é formada apenas pelo intermediador 1. É possível de verificar que essa organização em rede permite que cada nó possa, em algum momento, utilizar ou fornecer informações contextuais de qualquer outro nó.

6.9 Reflexões sobre a Arquitetura

O Capítulo 3 descreve aspectos arquiteturais importantes na construção de sistemas de software. Os elementos descritos neste capítulo podem ser adaptados a uma arquitetura de software distribuída e descentralizada. Os agentes componentes da arquitetura podem ser distribuídos em equipamentos diferentes, presentes em diferentes camadas na Internet (borda, névoa e nuvem). Isso é importante, a fim de se alcançar um processamento distribuído e balanceado (ver Seção 2.7). Por exemplo, agentes fornecedores e agregadores podem estar implantados em dispositivos móveis na borda como *smartphones*, enquanto agregadores de nós maiores e intermediadores podem estar disponíveis em servidores na névoa e nuvem. Agentes fornecedores com mecanismos de inferência mais pesados podem estar também localizados em nós na nuvem. Um nó pode reunir agentes em dispositivos diferentes que estejam próximos em uma rede.

Aplicações podem acessar agregadores mais próximos. Por exemplo, aplicações em *smartphones* podem acessar um agregador que esteja disponível no mesmo dispositivo ou em outro dispositivo no mesmo nó.

Segundo os padrões mostrados na Seção 3.2, a arquitetura de um nó utiliza uma arquitetura híbrida, onde o padrão multiagente é mesclado com o padrão *broker*, onde o agente agregador funciona como facilitador. Já as conexões entre intermediadores usam um padrão de rede *peer-to-peer*, mesclado com um padrão multiagente.

O uso do padrão *broker* nos nós permite a escalabilidade (Seção 3.5) local (adição de novas fontes de dados através de novos agentes registrados em agregadores). Já o uso do padrão *peer-to-peer* nos intermediadores permite a adição de novos nós e intermediadores, e que os mesmos possam ser identificados por outros intermediadores já presentes na rede.

Composição dinâmica (ver Seção 3.7) é alcançada na arquitetura através do uso da elaboração de novos planos nos agentes fornecedores a cada requisição e pela seleção de agentes fornecedores a cada requisição pelo agregador (há mudança de acordo com a requisição realizada e os agentes disponíveis). Dessa forma, diferentes agentes podem ser selecionados a cada requisição, já que entre uma requisição e outra novos agentes podem ser incluídos em um nó ou removidos deste. Cada agente elabora um plano para cada requisição recebida. Intermediadores são consultados periodicamente, o que pode resultar em diferentes nós sendo consultados, já que entre consultas diferentes, novos nós podem ser incluídos ou removidos da rede.

A resiliência (ver Seção 3.4) é alcançada, pelo menos em parte, através da disponibilidade de diferentes agentes que fornecem o mesmo elemento contextual. Isso resulta em uma espécie de replicação, onde uma fonte supre a falta de outra em caso de falha, pois todos os agentes disponíveis para determinado elemento contextual são contactados em cada requisição. Também podemos entender que existe um processo de autocura, pois o processamento de uma requisição é dividido entre agentes distribuídos e falhas podem ser resolvidas automaticamente.

6.10 Segurança e Privacidade

Em uma arquitetura de gerenciamento de contexto distribuída, questões relacionadas à segurança e privacidade ganham importância. Informações sobre o usuário podem circular entre nós distribuídos. Dados armazenados em nós também devem ter sua segurança garantida contra acessos indevidos. Aplicações devem ser devidamente registradas para que usuários mal intencionados não tenham acesso a dados pessoais sobre outros usuários.

A revisão bibliográfica sobre arquiteturas de reconhecimento de contexto descrita no Capítulo 4 identificou que pouquíssimos trabalhos abordam questões de segurança. Esses poucos trabalhos utilizam principalmente controle de acesso a dados baseados em políticas de privacidade geralmente definidas por regras (PU et al., 2010; MACIEL et al., 2014; INDULSKA et al., 2003a), autenticação de usuários e aplicações (FERREIRA; KOSTAKOS; DEY, 2015; MACIEL et al., 2014; INDULSKA et al., 2003b; CASSENS; SCHMITT; HERCZEG, 2013) e o uso de conexões de rede seguras para transferência de dados entre diferentes módulos.

A implementação de aspectos de segurança e privacidade na arquitetura apresentada neste trabalho inclui um trabalho mais amplo de investigação e está fora do escopo da tese proposta. Porém é possível identificar que aspectos como o uso de protocolos seguros de transferência de dados poderiam ser incorporados à implementações da arquitetura sem alterações significativas em sua estrutura. Além disso, a arquitetura proposta privilegia o uso de agentes locais, a fim de minimizar o tráfego de informações pessoais entre nós.

7 OPERAÇÕES ENVOLVIDAS NA ARQUITETURA

No capítulo anterior, a arquitetura desenvolvida neste trabalho foi descrita de forma geral, dando destaque à descrição de seus componentes. Neste capítulo encontra-se a descrição das principais operações que fazem parte da arquitetura e descritas brevemente no capítulo anterior: anúncio de agentes fornecedores em nós, anúncio de nós, consultas, inscrições e notificações.

7.1 Anúncio de Agentes Locais

Todo agente deve anunciar suas capacidades ao agregador quando ingressar na sociedade. Um anúncio é realizado na forma de uma mensagem enviada pelo agente ao agregador. O conteúdo desta mensagem é estruturado na forma de um conjunto C_a composto pelas N capacidades do agente a, como mostrado abaixo:

$$C_a = \{c_{a1}, c_{a2}, ..., c_{aN}\} \tag{7.1}$$

onde c_{ai} representa a i-ésima capacidade do agente a. Cada capacidade individual c_{ai}, por sua vez, é representada pela especificação de um padrão de elementos contextuais. Essa especificação, basicamente, envolve um padrão que define entidades e aspectos relacionados aos elementos contextuais envolvidos na capacidade em questão. Por exemplo, uma capacidade pode estar relacionada à obtenção do nome de uma localização qualquer. Nesse caso, a especificação poderia conter o aspecto *nome localização* e uma entidade indefinida (qualquer).

Como exemplo, considere um nó em um *smartphone* composto por três agentes: *Agente1*, *Agente2* e *Agente3*. O *Agente1* é capaz de obter dados do acelerômetro do aparelho e disponibilizá-los como contexto de baixo nível. O *Agente2* é capaz também de obter as coordenadas geográficas do usuário *UserA*, a partir do receptor GPS instalado no dispositivo. Já o *Agente3* é capaz de realizar a inferência de dois tipos de atividades de qualquer usuário: caminhando e parado. As capacidades do *Agente1* e do *Agente2* podem ser definidas como abaixo:

$$\begin{aligned} C_{Agente1} = \{&(UserA, AccelerationX), \\ &(UserA, AccelerationY), \\ &(UserA, AccelerationZ)\} \end{aligned} \tag{7.2}$$

$$C_{Agente2} = \{(UserA, Latitude), (UserA, Longitude)\} \tag{7.3}$$

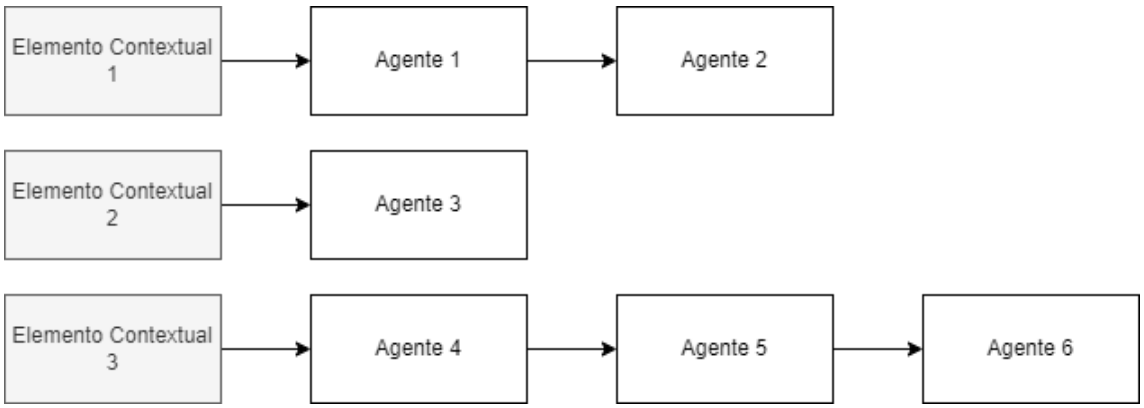
Já as capacidades do Agente3, como mostrado abaixo:

$$C_{Agente3} = \{(Activity)\} \tag{7.4}$$

Deve-se considerar que *AccelerationX*, *AccelerationY*, *AccelerationZ*, *Latitude* e *Longitude* são termos definidos na ontologia base deste trabalho para representar os respectivos aspectos fornecidos por sensores presentes no *smartphone do usuário*. Já *UserA* é um termo definido na ontologia de domínio da aplicação referente à entidade que representa o usuário em questão. Pode-se notar que a especificação de entidade na capacidade do *Agente3* estabelece que este é capaz de obter o aspecto *Activity* para qualquer entidade.

Anúncios de agentes locais provedores de contexto são recebidos pela interface de comunicação de agente agregador do nó em questão (Figura 6.8). O anúncio é então repassado ao componente de gerenciamento de agentes locais, que adiciona o agente e suas capacidades ao registro de agentes locais. Esse registro permite que, ao se receber uma solicitação de determinado elemento contextual, possa-se identificar os agentes que possam fornece-lo. Assim, essa estrutura indexa agentes por elementos contextuais (Figura 7.1, em uma estrutura similar a uma tabela *hash* (em que os elementos contextuais são as chaves e os agentes os respectivos valores).

Figura 7.1: Exemplo hipotético de registro de agentes locais em um agregador.



Fonte: O Autor

Podem existir diversos agentes que forneçam os mesmos elementos contextuais. Portanto, cada entrada do registro pode estar associada a um conjunto de agentes.

7.2 Anúncio de Nós

Um agente agregador, como facilitador de um nó, é responsável por anunciar as capacidades deste, ou seja, os elementos contextuais que podem ser fornecidos por agentes neste nó. Essa anúncio é realizado através de uma mensagem enviada pelo agregador aos intermediadores conhecidos. Dessa forma,

$$C_n = \bigcup_{a \in AG_n} C_a \tag{7.5}$$

onde n representa o nó em questão e AG_n o conjunto de agentes locais neste nó. O agregador possui essa informação, pois mantém um registro das capacidades de todos os agentes do nó em seu componente de gerenciamento de agentes locais (ver seção 7.1).

Um agregador envia um anúncio a um intermediador próximo em duas situações: logo que é criado, e cada vez que há atualizações nas capacidades do nó (novos agentes incluídos ou agentes existentes excluídos). O intermediador, ao receber o anúncio de um agregador, atualiza um registro interno de nós. Esse registro é um conjunto

$$N = \{(Agg_1, C_1), (Agg_2, C_2), ..., (Agg_n, C_n)\} \tag{7.6}$$

onde Agg_i corresponde ao agregador do nó i e C_i é o conjunto de capacidades do nó i, da mesma forma do que o definido na seção 7.1.

7.3 Consultas

Consultas são recebidas pela interface de comunicação e podem ser enviadas por aplicações ou agregadores de outros nós. A estrutura dessas consultas foi apresentada na seção 6.4.

Ao receber uma consulta contendo uma requisição CR, a interface de comunicação a repassa ao gerenciador de consultas. Para processar a consulta recebida, o gerenciador de consultas executa os passos especificados no algoritmo 1.

O algoritmo recebe como entrada o conjunto CR enviado pela aplicação, contendo a especificação das informações de contexto solicitadas, bem como as estratégias de fusão a serem usadas. O processo pode ser dividido em duas etapas básicas: *consulta a agentes locais*, executada pela operação $consultarAgentesLocais$ (linha 2); *consulta a agentes remotos*, executada pela operação $consultarAgentesRemotos$ (linha 8). A con-

Algoritmo 1: CONSULTAR AGREGADOR

```
   Entrada: CR
   Saída: (res_fin, fa_fin)
1  início
2  |  (res_loc, fa_loc) ← consultarAgentesLocais(CR)
3  |  se fa_loc = ∅ então
4  |  |  res_fin ← res_loc
5  |  |  fa_fin ← fa_loc
6  |  fim
7  |  senão
8  |  |  (res_rem, fa_rem) ← consultarAgentesRemotos(fa_loc)
9  |  |  res_fin ← res_loc ∪ res_rem
10 |  |  fa_fin ← fa_rem
11 |  fim
12 fim
13 retorna (res_fin, fa_fin)
```

sulta a agentes locais verifica se há agentes no nó que podem fornecer pelo menos parte dos elementos contextuais solicitados. Essa busca é realizada utilizando-se a estrutura construída a partir dos anúncios dos agentes fornecedores locais, conforme descrito na Seção 7.1. Como resultado, é devolvida uma tupla contendo dois conjuntos. O primeiro, res_{loc}, contém um conjunto de instâncias de elementos contextuais fornecidos pelos agentes locais, uma instância para cada elemento contextual que pôde ser obtido localmente. A representação ontológica de instâncias de elementos contextuais é descrita na Seção 6.2.2. Itens pertencentes a CR que não foram obtidos pelos agentes locais, são informados no conjunto fa_{loc}.

Um resultado com fa_{loc} vazio indica que todos os elementos contextuais solicitados foram obtidos nos agentes locais. Portanto, nenhuma outra consulta é efetuada (a arquitetura prioriza a informação em nós locais). Caso contrário, uma nova consulta a nós remotos é realizada, com o fim de se obter as instâncias faltantes. A lógica dos resultados devolvidos é similar, mas dessa vez o conjunto res_{rem} contém instâncias de elementos contextuais obtidas a partir da consulta a nós remotos, e fa_{rem} contém um subconjunto de itens de fa_{loc} que não puderam ser fornecidos nem através de nós remotos. O conjunto de resultados finais, res_{fin}, é resultado da união dos resultados obtidos por agentes locais e dos resultados obtidos por nós remotos. Já o conjunto de itens de CR que não puderam ser obtidos, fa_{fin}, é igual ao conjunto fa_{rem} devolvido pela operação *consultarAgentesRemotos*.

A operação $consultarAgentesLocais$ é detalhada no algoritmo 2. As linhas 2 e

Algoritmo 2: CONSULTARAGENTESLOCAIS

Entrada: CR
Saída: (res_{loc}, fa_{loc})

início
  $res_{loc} \leftarrow \emptyset$
  $fa_{loc} \leftarrow \emptyset$
  para *cada* $(rd, fs) \in CR$ **faça**
    $Ag \leftarrow buscaAgentesLocais(rd)$
    se $Ag \neq \emptyset$ **então**
      $M \leftarrow criarTabelaHash()$
      para *cada* $a \in Ag$ **faça**
        $I \leftarrow obterValor(M, a)$
        $I \leftarrow I \cup rd$
        $alterarValor(M, a, I)$
      fim
    fim
  fim
  $R \leftarrow criarTabelaHash()$
  para *cada* $(a, REQ) \in M$ **faça**
    $INST \leftarrow consultaAgenteLocal(a, REQ)$
    se $INST \neq \emptyset$ **então**
      para *cada* $i \in INST$ **faça**
        $R' \leftarrow obterValor(R, elementoContextual(i))$
        $R' \leftarrow R' \cup i$
        $alterarValor(R, elementoContextual(i), R')$
      fim
    fim
  fim
  para *cada* $(rd, fs) \in CR$ **faça**
    $CTX \leftarrow obterValor(R, rd)$
    se $CTX = \emptyset$ **então**
      $fa_{loc} \leftarrow fa_{loc} \cup (rd, fs)$
    fim
    senão
      $ctx \leftarrow fundirInstanciasContextuais(fs, CTX)$
      $res_{loc} \leftarrow res_{loc} \cup ctx$
    fim
  fim
fim
retorna (res_{loc}, fa_{loc})

3 do algoritmo iniciam os conjuntos que devem ser retornados. As linhas 4 a 14 efetuam a busca de agentes locais que possam fornecer os elementos contextuais requisitados. A operação *buscaAgentesLocais* (linha 5) verifica o registro de agentes locais (ver Seção 7.1) e retorna aqueles que podem fornecer o elemento contextual especificado em rd. Uma tabela *hash* M é criada para organizar os itens que podem ser fornecidos por cada agente (um mesmo agente pode fornecer mais de um item). Considera-se aqui que a operação $criarTabelaHash$ cria uma nova tabela *hash* vazia, $obterValor$ obtém o valor de uma entrada da tabela *hash* M identificada pela chave informada e, finalmente, a operação $alterarValor$ altera o valor relacionado à chave informada na tabela *hash* M.

As linhas 15 a 25 efetuam a consulta aos agentes locais, enviando uma requisição contendo os itens identificados para cada agente em M. Os resultados obtidos são organizados em uma nova tabela *hash* R, onde as instâncias obtidas são organizadas por elemento contextual. Assim, todas as instâncias obtidas relacionadas a um mesmo elemento contextual estarão em uma mesma entrada em R.

Por fim, as linhas 26 a 35 efetuam a fusão de instâncias encontradas presentes em R. Para cada item presente na requisição recebida CR, tenta-se obter as instâncias relacionadas ao elemento contextual requisitado que estejam presentes em R. Caso não exista nenhuma instância (linha 28), o item em questão é adicionado ao conjunto de falhas. Caso contrário, as instâncias obtidas passam pelo processo de fusão e o resultado é adicionado ao conjunto de resultados res_{loc}. A operação $fundirInstanciasContextuais$ realiza esse processo de fusão. Essa operação executa a estratégia informada no componente fs na requisição. Algumas estratégias de fusão e seleção foram apresentadas na seção 3.6, e podem ser usadas aqui.

O processo de consulta a nós remotos é mostrado no algoritmo 3. Um conjunto de itens de requisição de consulta é recebido através de CR. A primeira ação a ser realizada (linha 2) é uma busca no cache local de agregadores remotos (ver Seção 6.6). Essa busca devolve um conjunto Ag de tuplas (a, C_a), onde a é um agregador e C_a contém o conjunto de capacidades desse agregador (similar ao descrito na Seção 7.1).

As linhas 3 a 8 realizam uma verificação por itens da requisição que não possam ser fornecidos pelos agregadores remotos em Ag. Nesse caso, devem existir itens cujos elementos contextuais requisitados não pertençam às capacidades de nenhum agregador presente em Ag. Se tais itens existirem, serão incluídos no conjunto REM. Nesse caso, realiza-se uma busca aos intermediadores identificados pelo agregador a fim de encontrar agregadores remotos que possam fornecer os itens faltantes (linhas 9 a 16). Esses inter-

Algoritmo 3: CONSULTARAGENTESREMOTOS

Entrada: CR
Saída: (res_{rem}, fa_{rem})

```
1  início
2      Ag ← buscaAgregadoresRemotosEmCache(CR)
3      REM ← ∅
4      para cada (rd, fs) ∈ CR faça
5          se ¬∃(a, C_a)|(a, C_a) ∈ Ag ∧ rd ∈ C_a então
6              REM ← REM ∪ (rd, fs)
7          fim
8      fim
9      se REM ≠ ∅ então
10         INT ← obtemIntermediadores()
11         para cada i ∈ INT faça
12             Ag_rem ← buscaAgregadoresRemotos(i, REM)
13             adicionaACache(Ag_rem)
14         fim
15         Ag ← buscaAgregadoresRemotosEmCache(CR)
16     fim
17     se Ag = ∅ então
18         fa_rem ← CR
19         res_rem ← ∅
20     fim
21     senão
22         M ← criarTabelaHash()
23         para cada (a, C_a) ∈ Ag faça
24             CTX ← consultaAgregador(a, CR ∩ C_a)
25             para cada c ∈ CTX faça
26                 INST ← obterValor(M, elementoContextual(c))
27                 INST ← INST ∪ c
28                 alterarValor(M, elementoContextual(c), INST)
29             fim
30         fim
31         fa_rem ← ∅
32         res_rem ← ∅
33         para cada (rd, fs) ∈ CR faça
34             se rd ∈ chaves(M) então
35                 ctx ←
                     fundirInstanciasContextuais(fs, obterValor(M, rd))
36                 res_rem ← res_rem ∪ ctx
37             fim
38             senão
39                 fa_rem ← fa_rem ∪ (rd, fs)
40             fim
41         fim
42     fim
43 fim
44 retorna (res_rem, fa_rem)
```

mediadores verificam se os agregadores conhecidos por eles podem prover os itens requisitos. Caso exista algum item que não possa ser provido pelos agregadores conhecidos, uma consulta à rede *peer-to-peer* de intermediadores é realizada em busca de agregadores remotos que possam suprir a demanda. Todos os agregadores encontrados pelos intermediares são enviados como resposta ao agregador solicitante e adicionados ao cache local. A busca aos agregadores remotos no cache local é repetida, dessa vez incluindo os novos agregadores identificados (linha 15).

No caso do conjunto Ag ser vazio, ou seja, nenhum agregador remoto foi encontrado para suprir os itens requisitados, todos os itens são colocados no conjunto de falhas fa_{rem} e o conjunto de resultados res_{rem} fica vazio. Caso contrário, uma consulta aos agregadores identificados é realizada (linhas 22 a 30). A cada agregador é enviada uma solicitação de itens requisitos que estejam contidos em suas capacidades. O resultado é um conjunto de instâncias de elementos contextuais CTX. Esta lista é adicionada a uma tabela *hash* M onde cada entrada é identificada por uma chave (um elemento contextual) e contém um conjunto de valores (instâncias retornadas pelos agregadores e relacionadas ao elemento contextual). Agregadores diferentes podem retornar instâncias relacionadas ao mesmo elemento contextual que, depois, devem ser fundidas a fim de se resolver o conflito.

Por fim, o conjunto de resultados res_{rem} e o conjunto de falhas fa_{rem} são preenchidos (linhas 31 a 41). O conjunto de resultados res_{rem} é preenchido a partir da fusão dos valores em cada entrada na tabela *hash* M. Cada entrada gera uma instância em res_{rem}. Caso algum elemento contextual solicitado não esteja presente no conjunto de chaves em M, então o item correspondente é adicionado ao conjunto de falhas em fa_{rem}.

Agentes provedores de contexto locais respondem a requisições de consulta conforme o processo descrito no algoritmo 4. Essas consultas são enviadas por agregadores, quando necessário, conforme descrito no algoritmo 2. Uma requisição de consulta a um agente provedor de contexto local contém um conjunto de elementos contextuais requisitados, bem como dados complementares, assim como o componente de dados de requisição descrito na Seção 6.4. Para cada elemento contextual requisitado, primeiramente o agente tenta executar o plano relacionado ao elemento contextual presente em rd. Caso o plano seja executado com sucesso, uma instância de elemento contextual é adicionada ao conjunto de resultados res.

Em caso do elemento contextual requisitado estar relacionado a um contexto secundário, elementos de entrada serão necessários (ver Seção 2.2). Alguns desses elemen-

Algoritmo 4: CONSULTAAGENTELOCAL

Entrada: CR
Saída: $(res, fa))$

```
1  início
2      res ← ∅
3      fa ← ∅
4      para cada rd ∈ CR faça
5          inst ← executaPlano(rd)
6          se existem dependências então
7              DEP ← obtemDependencias(rd)
8              CR ← ∅
9              para cada rd' ∈ DEP faça
10                 CR ← CR ∪ (rd', obtemEstrategiaFusaoLocal(rd'))
11             fim
12             INST ← consultaAgregador(obtemAgregadorLocal(), CR)
13             para cada i ∈ INST faça
14                 adicionaModuloTemporario(i)
15             fim
16             inst ← executaPlano(rd)
17         fim
18         se houvefalha então
19             fa ← fa ∪ rd
20         fim
21         senão
22             res ← res ∪ inst
23         fim
24     fim
25 fim
26 retorna (res, fa))
```

tos podem não ser fornecidos por módulos locais e uma consulta externa é necessária. Nesse caso, esses elementos de entrada são chamados de *dependências*.

Portanto, é necessário efetuar uma consulta a outros agentes a fim de se obter essas instâncias. Essa consulta é realizada de forma indireta através de uma consulta ao agregador local. A consulta realizada ao agregador é similar à consulta realizada por aplicações. O agente monta uma requisição onde cada item contém os dados de requisição relacionados a cada dependência, bem como uma estratégia de fusão local que o agente utiliza para o elemento contextual em questão. Assim, neste trabalho, optou-se por utilizar uma abordagem em que cada agente pode inferir ou possuir uma estratégia própria de fusão, pré-determinada pelo projetista do agente, para cada situação de consulta externa.

Cada instância obtida do agregador é utilizada para se adicionar módulos temporários ao plano em questão. Esses módulos são do tipo *Getter* e fornecem como resultado uma instância obtida do agregador. Assim, para cada instância obtida é criado um módulo temporário. Dessa forma, o plano é novamente executado e agora contém todas as informações necessárias para sua execução. Os módulos temporários deixam de existir após a execução do plano.

7.4 Inscrições

O processo de inscrição envolve um protocolo diferente do usado em consultas. Um agregador recebe de uma aplicação uma requisição de inscrição para receber notificações relacionadas a instâncias de elementos contextuais. O protocolo utilizado nesta arquitetura para inscrições é baseado no protocolo de inscrições definido pela FIPA (*Foundation for Intelligent Physical Agents*)[1]. No protocolo elaborado neste trabalho, uma aplicação ou agregador externo envia uma solicitação de inscrição a um agregador responsável por um nó. O agregador, por sua vez, ao processar a inscrição deve enviar uma resposta: se ACEITA a solicitação de inscrição, mesmo que parcialmente; ou se REJEITA a solicitação de inscrição. Caso aceite, o agregador começa então a enviar notificações à aplicação ou agente requisitante. O processamento da inscrição no agregador é realizado pelo *gerenciador de inscrições* (seção 6.6).

Os passos desse processamento são mostrados no Algoritmo 5. As linhas 4 a 19 realizam uma busca por agentes locais que possam fornecer os itens requisitados. Cada agente encontrado será adicionado ao conjunto Ag_{sel}, que conterá um conjunto de tuplas

[1] <http://www.fipa.org/specs/fipa00035/SC00035H.html>

Algoritmo 5: PROCESSAINSCRICAOAGREGADOR

Entrada: CR
Saída: $nenhuma$

```
1  início
2      REM ← ∅
3      Ag_sel ← ∅
4      para cada (rd, fs) ∈ CR faça
5          Ag_loc ← buscaAgentesLocais(rd)
6          se Ag_loc = ∅ então
7              REM ← REM ∪ (rd, fs)
8          fim
9          senão
10             para cada a ∈ Ag_loc faça
11                 se ∃(x, y)|(x, y) ∈ Ag_sel ∧ x = a então
12                     y ← y ∪ (rd, fs)
13                 fim
14                 senão
15                     Ag_sel ← Ag_sel ∪ (a, {(rd, fs)})
16                 fim
17             fim
18         fim
19     fim
20     se REM ≠ ∅ então
21         Ag_rem ← buscaAgregadoresRemotosEmCache(REM)
22         (Ag_sel, REM') ←
            organizaRequisicoesPorAgente(Ag_rem, REM, Ag_sel)
23     fim
24     se REM' ≠ ∅ então
25         FAIL ← ∅
26         INT ← obtemIntermediadores()
27         para cada i ∈ INT faça
28             Ag_rem ← buscaAgregadoresRemotos(i, REM)
29             adicionaACache(Ag_rem)
30             (Ag_sel, FAIL') ←
                organizaRequisicoesPorAgente(Ag_rem, REM', Ag_sel)
31             FAIL ← FAIL ∪ FAIL'
32         fim
33         se FAIL ≠ ∅ então
34             enviaFalhas(FAIL)
35         fim
36     fim
37     para cada (ag, REQ) ∈ Ag_sel faça
38         inscricaoAgente(ag, REQ)
39     fim
40     registraInscricoes(Ag_sel)
41 fim
```

(a, I), onde a é um agente e I representa o conjunto de itens requisitados que o agente pode fornecer. O conjunto REM conterá todos os itens requisitados que não podem ser fornecidos por agentes locais.

As linhas 20 a 23 realizam uma busca por nós remotos que possam fornecer os itens restantes presentes em REM. Primeiramente se realiza uma busca por agregadores em cache. A operação $organizaRequisicoesPorAgente$ adiciona um conjunto de agregadores remotos e os itens de requisição que eles podem fornecer no conjunto Ag_{sel}. Os passos executados nessa operação estão descritos no Algoritmo 6. Essa operação devolve o conjunto Ag_{sel} atualizado, bem como um conjunto de itens REM' que não podem ser fornecidos pelos agregadores remotos em cache e que, portanto, continuam em aberto.

Os itens ainda em aberto em REM' são tratados nas linhas 24 a 36. Uma busca por agregadores remotos é realizada nos intermediadores, de forma semelhante ao realizado na operação de consulta. A diferença encontra-se principalmente na linha 30, onde os agregadores encontrados e os itens que podem ser providos por eles são adicionados ao conjunto Ag_{sel} através da operação $organizaRequisicoesPorAgente$.

Itens que não podem ser providos nem por esses agregadores remotos são retornados no conjunto $FAIL'$ e adicionados ao conjunto de falhas $FAIL$. Os itens nesse conjunto, caso existam, são enviados de volta ao requisitante, informando elementos contextuais que não podem ser fornecidos (linhas 33 a 35). Essa operação pode enviar uma mensagem de recusa, caso os itens que falharam corresponda a todos os itens requisitados em CR.

A finalização do processo ocorre nas linhas 37 a 40. Uma requisição de inscrição é enviada a cada agente em Ag_{sel}, contendo os itens REQ que podem ser providos por cada um. As inscrições são registradas no componente *Gerenciador de Inscrições* do agregador (linha 40). Dessa forma, notificações futuras recebidas dos agentes fornecedores podem ser associadas à inscrição correta e enviadas ao requisitante.

O Algoritmo 6 mostra detalhes da operação $organizaRequisicoesPorAgente$. Esta operação, basicamente, associa cada item em REQ com os agentes em $AGENTS$ que possuam em suas capacidades o elemento contextual informado no item. Todas as associações são adicionadas ao conjunto Ag_{sel}. Itens para os quais não são encontrados agentes são retornados como falhas no conjunto $FAIL$.

O processamento de uma inscrição em agentes locais é descrito no Algoritmo 7. O primeiro passo executado no algoritmo é a verificação de dependências para cada item de requisição (linhas 2 a 8). Como apresentado no processo de consulta, dependências são

Algoritmo 6: ORGANIZAREQUISICOESPORAGENTE

Entrada: $AGENTS, REQ, Ag_{sel}$
Saída: $(Ag_{sel}, FAIL)$

```
1  início
2      FAIL ← ∅
3      para cada (rd, fs) ∈ REQ faça
4          Ags ← ∅
5          para cada (ag, C_ag) ∈ AGENTS faça
6              se rd ∈ C_ag então
7                  Ags ← Ags ∪ ag
8                  se ∃(x, y)|(x, y) ∈ Ag_sel ∧ x = ag então
9                      y ← y ∪ (rd, fs)
10                 fim
11                 senão
12                     Ag_sel ← Ag_sel ∪ (ag, {(rd, fs)})
13                 fim
14             fim
15         fim
16         se Ags = ∅ então
17             FAIL ← FAIL ∪ (rd, fs)
18         fim
19     fim
20 fim
21 retorna (Ag_sel, FAIL)
```

elementos contextuais necessários para se obter o elemento contextual requisitado e que não podem ser providos pelo agente em questão (devem ser obtidos de outros agentes). A operação $obtemDependencias$ é usada para se obter o conjunto de dependências de cada item da requisição de inscrição. Todas as dependências encontradas são incluídas no conjunto $DEPS$.

Algoritmo 7: PROCESSAINSCRICAOAGENTELOCAL

```
   Entrada: REQ
   Saída: nenhuma
1  início
2      DEPS ← ∅
3      para cada rd ∈ REQ faça
4          DEP' ← obtemDependencias(rd)
5          se DEP' ≠ ∅ então
6              DEPS ← DEPS ∪ {(rd, DEP')}
7          fim
8      fim
9      se DEPS ≠ ∅ então
10         CR ← ∅
11         para cada rd' ∈ DEPS faça
12             CR ← CR ∪ (rd', obtemEstrategiaFusaoLocal(rd'))
13         fim
14         FAIL' ← inscricaoAgregador(CR)
15         se OK então
16             FAIL ← registraInscricaoAgregador(REQ, CR, FAIL')
17             para cada (rd, fs) ∈ CR faça
18                 se (rd, fs) ∉ FAIL então
19                     adicionaModuloTemporario(rd)
20                 fim
21             fim
22         fim
23         enviaFalhas(FAIL)
24     fim
25     para cada rd ∈ REQ faça
26         se ∄(rd', fs)|(rd, fs) ∈ FAIL ∧ rd' = rd então
27             inscricaoLocal(rd)
28         fim
29     fim
30 fim
```

A linhas 9 a 24 realizam a inscrição do agente junto ao agregador do nó local, a fim de se obter as dependências necessárias. Uma requisição de inscrição é construída nas linhas 10 a 13, onde cada item presente no conjunto $DEPS$ é incluído bem como uma estratégia local para fusão de dados, de forma similar ao realizado na operação de

consulta. A inscrição é enviada ao agregador na linha 14. Três casos podem ocorrer: (1) o agregador realiza a inscrição para todos os itens requisitados, (2) o agregador realiza a inscrição para um subgrupo dos itens requisitados e, (3) o agregador não realiza a inscrição e não pode prover nenhum dos itens requisitados. Nos casos 2 e 3, os itens não providos pelo agregador são colocados no conjunto $FAIL'$.

Caso a inscrição junto ao agregador ocorra com sucesso (linha 15), a inscrição é registrada pelo agente localmente através da operação $registraInscricaoAgregador$. Esse registro envolve também a verificação de quais itens requisitados ao agente não podem ser providos pois suas dependências não podem ser obtidas junto ao agregador. Esses itens são retornados no conjunto $FAIL$. Em outras palavras,

$$\begin{aligned} FAIL = \{(rd, fs) | (rd, fs) \in CR \\ \wedge [\exists rd' | rd' \in deps(rd) \\ \wedge (\exists (x, y) | (x, y) \in FAIL' \wedge x = rd')]\} \end{aligned} \tag{7.7}$$

onde $deps(rd)$ corresponde ao conjunto de dependências necessárias para que o agente obtenha o elemento contextual requisitado rd.

As linhas 17 a 21 realizam a criação de módulos temporários para cada item que não esteja no conjunto $FAIL$ (ou seja, itens que poderão ser fornecidos pelo agente). Estes módulos receberão notificações provenientes do agregador, informando novas instâncias de elementos contextuais de dependência. Essas notificações possibilitarão que o agente execute planos a fim de gerar novas instâncias de elementos contextuais requisitados a ele.

Por fim, as linhas 25 a 29 realizam o registro da inscrição em módulos locais para cada item de requisição que não tenha apresentado falhas. Dessa forma, os módulos gerarão notificações de novos elementos contextuais providos por eles. Essa notificações podem disparar a execução de planos, que geram novos elementos contextuais notificados pelo agente. É importante registrar inscrições para o processamento futuro de notificações.

Toda inscrição recebida por agregadores e/ou agentes locais pode ser cancelada pelo requisitante. No caso de agregadores, uma mensagem de cancelamento recebida leva ao envio de mensagens de cancelamento para cada agente local/agregador remoto que recebeu inscrições relacionadas à inscrição cancelada. No caso de agentes locais, o processamento de uma requisição de cancelamento envolve os seguintes passos:

- Se o item a ser cancelado envolve dependências, o agente deve enviar uma requi-

sição de cancelamento ao agregador, solicitando que as inscrições realizadas antes por essas dependências sejam canceladas. Essa requisição de cancelamento contém apenas dependências que não sejam também dependências de outras inscrições recebidas pelo agente que ainda devem se manter ativas. Os módulos temporários relacionados às dependências canceladas são desativados.

- Inscrições internas em módulos são canceladas.
- O registro interno de inscrição é removido.

7.5 Notificações

Após receber e processar uma requisição de inscrição, agentes locais e agregadores enviam à entidade requisitante (aplicação ou outro agente) notificações contendo instâncias de elementos contextuais. Essas notificações ocorrem quando novas instâncias são obtidas, a fim de notificar o requisitante sobre mudanças no contexto observado.

Um exemplo ocorre quando uma aplicação deseja observar a atividade desempenhada por uma pessoa. Para isso, a aplicação inscreve-se junto ao agregador do nó ao qual está associada solicitando ser notificada toda vez que essa pessoa mudar a atividade que está realizando. O agregador, por sua vez, irá inscrever-se junto aos agentes que possam fornecer tal informação. Podem existir alguns agentes que forneçam informações sobre o elemento contextual requisitado. Quando cada agente observa uma alteração na atividade da pessoa, envia uma notificação ao agregador. O agregador recebe todas as notificações dos agentes contatados e executa um processo de fusão, utilizando uma estratégia repassada pela aplicação em sua solicitação de inscrição.

Portanto, um agregador irá gerar uma nova solicitação quando, após receber notificações dos agentes locais fornecedores de contexto, identificar uma nova instância do elemento contextual observado. Os elementos contextuais recebidos dos agentes fornecedores são fundidos utilizando uma estratégia de fusão recebida pelo agregador durante o processo de inscrição.

Existem algumas questões envolvidas durante o recebimento de notificações, destacadas a seguir:

1. Caso um dos agentes fornecedores apresente uma falha (por exemplo, um sensor pode parar de funcionar), como o agregador deve proceder?
2. Os agentes fornecedores podem não enviar notificações simultaneamente, pois po-

dem identificar mudanças de formas diferentes. Como proceder quando somente alguns agentes enviam notificações e outros não?

No caso da questão 1, o problema é resolvido se o agregador estiver recebendo notificações de outros fornecedores também. Nesse caso, a informação contextual pode ainda ser obtida pela fusão das instâncias recebidas dos demais agentes. Caso todos os agentes fornecedores apresentem falhas, o agregador pode realizar duas ações. Caso sejam agentes locais, nós remotos podem ser contactados para suprir o elemento contextual requisitado. Segundo, caso nós remotos sejam os fornecedores, o agregador também não poderá mais enviar notificações e reportará uma falha, encerrando a inscrição.

A questão 2 envolve um problema mais delicado. As possibilidades para se resolver essa questão poderiam ser as seguintes:

1. *Esperar que todos os fornecedores contactados enviem notificações*. Nesse caso, o agregador só executaria a fusão de instâncias quando todos os agentes enviassem notificações. O problema com esta alternativa, é que alguns agentes podem reconhecer mudanças nos elementos contextuais observados enquanto outros não. Portanto, alguns agentes podem notificar mudanças, enquanto outros não o farão, fazendo com que o agregador espere em vão por notificações desses agentes.
2. *Realizar a checagem de nova instância toda vez que uma notificação for recebida*. Nesse caso, cada vez que o agregador receber uma nova notificação de um fornecedor, executará o processo de fusão, considerando as últimas instâncias enviadas pelos demais agentes. Essa solução resolve o problema da alternativa 1, porém acrescenta outro: se diversos agentes enviarem notificações quase simultaneamente (em um curto intervalo de tempo entre elas), o agregador pode introduzir um *overhead* ao processar diversas notificações em sequência.
3. *Introduzir uma janela temporal de recebimento de notificações*. Nesse caso, o agregador aguarda, dentro de uma janela temporal, o recebimento de notificações. Os elementos contextuais recebidos durante essa janela temporal são fundidos juntamente com os últimos elementos contextuais enviados pelos demais agentes. Ao fim de uma janela, uma nova se inicia. Essa foi a solução adotada neste trabalho.

Quanto a agentes locais, módulos individuais podem gerar notificações sobre novas instâncias, ou o agregador pode notificar novas instâncias de elementos contextuais relacionados a dependências. Nos dois casos, notificações desse tipo geram uma reexecução de um plano. Caso o plano gere uma nova instância de um elemento contextual, esta

é notificada ao agregador local.

8 AVALIAÇÃO E RESULTADOS OBTIDOS

Para materializar e validar a arquitetura desenvolvida neste trabalho, foi implementado um protótipo de *framework*. A construção de um protótipo possibilita descobrir muitos problemas que serão encontrados no sistema real. Também possibilita uma forma de realizar a validação experimental da arquitetura proposta através da variação de suas características, como quantidade de agentes, quantidade de nós, número de elementos contextuais duplicados (fornecidos por mais de um agente) e distribuição de nós.

Neste capítulo o protótipo desenvolvido será descrito bem como o processo de validação desenvolvido a partir do *framework*. Com base no *framework* desenvolvido, diversos experimentos diferentes foram construídos e executados a fim de se avaliar a utilização da arquitetura desenvolvida. A última parte deste capítulo descreve os experimentos realizados e os resultados obtidos.

8.1 Implementação do Framework

A arquitetura multiagente descrita neste trabalho foi implementada utilizando a plataforma *JADE* (*Java Agent DEvelopment Framework*)[1]. Tal plataforma permite o desenvolvimento de aplicações através de um *framework* totalmente desenvolvido na linguagem Java, bem como uma plataforma de execução que permite distribuir os agentes através de diferentes dispositivos conectados em rede. Dentre as funcionalidades da plataforma pode-se citar:

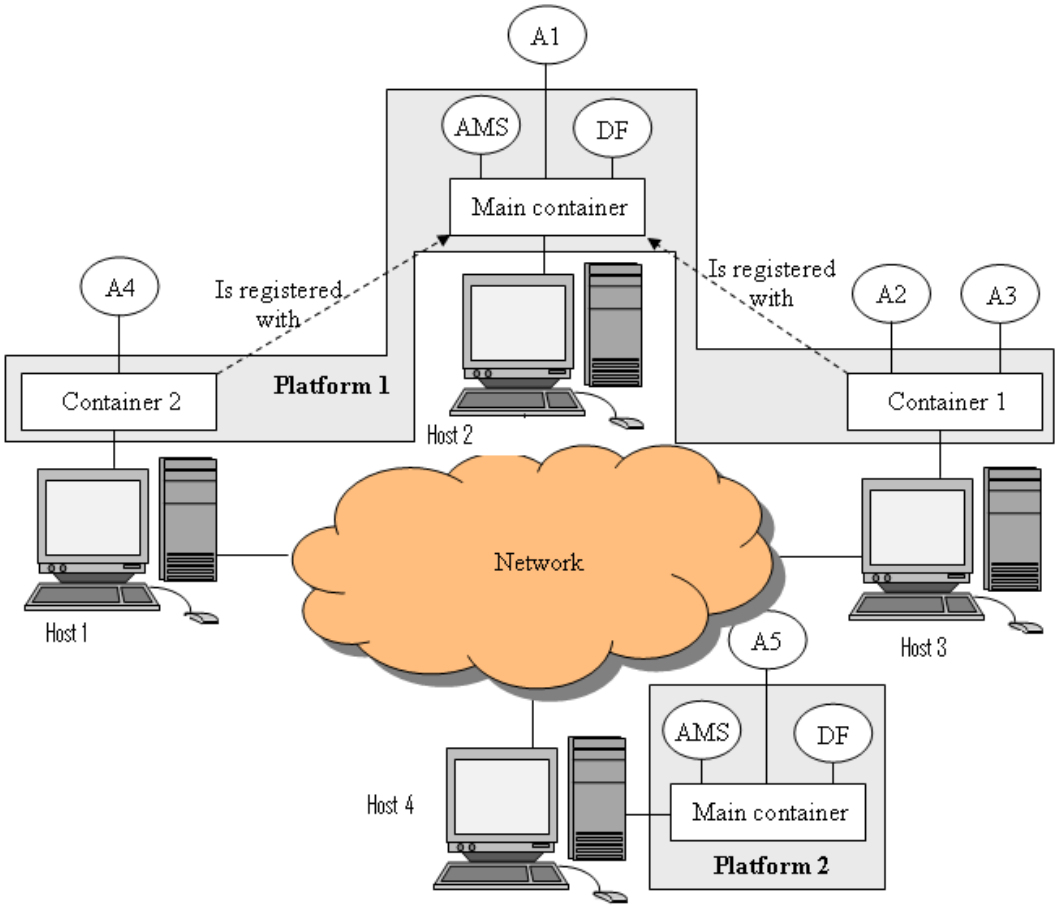
- Um sistema totalmente distribuído, onde os agentes são executados em *threads* separadas e podem estar localizados em diferentes dispositivos. Estes agentes são capazes de se comunicar entre si através de mecanismos de troca de mensagens que abstraem a infraestrutura subjacente de comunicação.
- Total conformidade com as especificações FIPA.
- Gerenciamento do ciclo de vida de cada agente. Cada agente, ao ser criado, recebe uma identificação única global. Uma API e ferramentas gráficas são disponibilizadas para gerenciar o ciclo de vida de agentes, tanto localmente quanto remotamente, envolvendo operações como criar, suspender, reativar, migrar, clonar e finalizar.
- Suporte para migração de agentes entre processos e dispositivos.

[1] <https://jade.tilab.com/>

- Suporte a linguagens de descrição de conteúdo e ontologias, usadas na especificação do conteúdo de mensagens trocadas entre agentes.

A Figura 8.1 mostra a arquitetura geral da plataforma JADE. Os elementos principais da plataforma são os agentes, que podem ser distribuídos ao longo de dispositivos em rede. Cada agente possui um único nome. Na figura mostrada, por exemplo, os nomes *A1, A2, A3, A4* e *A5* são as identificações de cinco agentes diferentes. Os agentes estão inseridos em *containers* que são processos Java que fornecem todo o suporte necessário para hospedar e executar agentes. Na figura mostrada, por exemplo, o *container* identificado como "Container 1"hospeda os agentes *A2* e *A3*.

Figura 8.1: Arquitetura geral da plataforma JADE.



Fonte: JADE Architecture Overview[2]

Um ou mais *containers* fazem parte de uma *plataforma*, que fornece a eles serviços básicos, incluindo comunicação através de troca de mensagens. Toda a plataforma deve possuir um *container* especial chamado de *main container* (*container* principal). Este realiza todas as funções de um *container* convencional, com o acréscimo de:

- deve ser o primeiro *container* a ser criado e todos os demais devem se registrar a ele ao serem iniciados;
- inclui dois agentes especiais obrigatórios: *AMS* (*Agent Management System*, responsável pelo gerenciamento de todos os agentes na plataforma; e *DF* (*Directory Facilitator*) que realiza um serviço de *páginas amarelas*, onde agentes na plataforma podem registrar serviços oferecidos e/ou realizar pesquisas sobre agentes que fornecem determinados serviços.

Containers podem estar localizados em um mesmo *host* do que o *container* principal ou em *hosts* diferentes. Dessa forma, é possível distribuir agentes através de dispositivos distintos em rede.

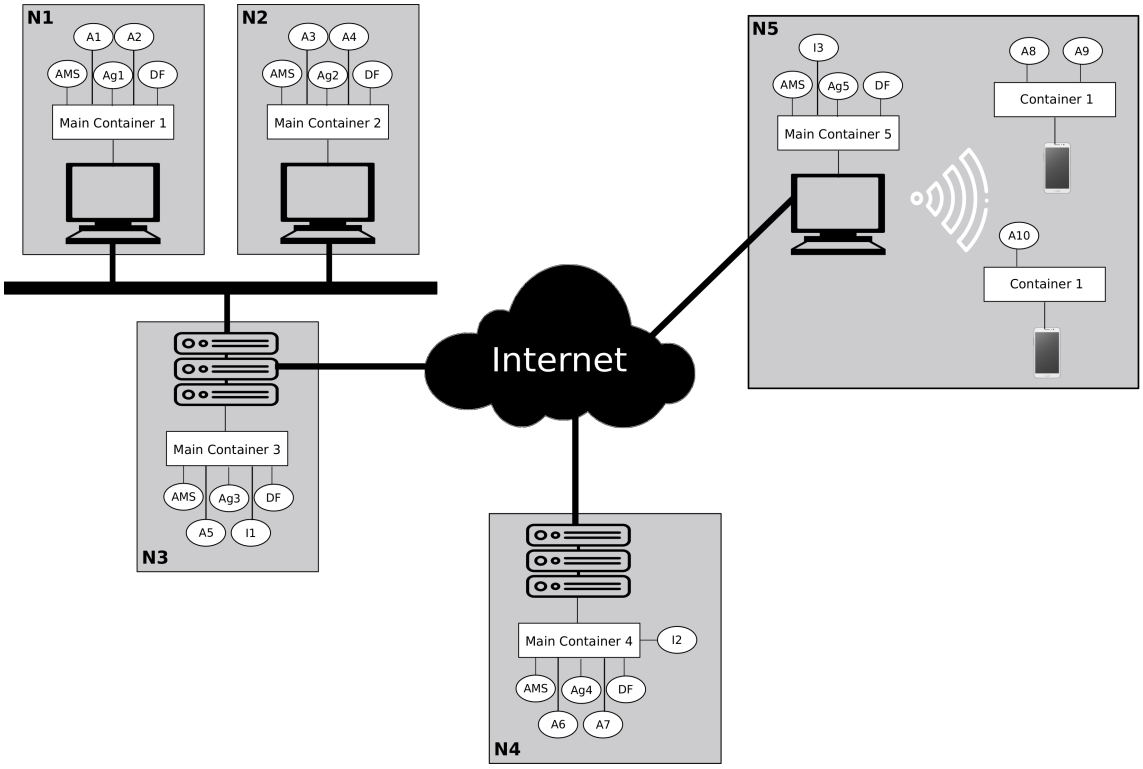
Esse fato foi utilizado para distribuir os agentes da arquitetura deste trabalho em dispositivos em rede. Cada nó da arquitetura foi implementado em uma plataforma JADE distinta. Portanto, os agentes em um mesmo nó podem estar distribuídos em dispositivos diferentes, e mesmo em *containers* diferentes. Ao se criar um nó, um agente agregador é criado e colocado no *container principal*. Agentes intermediadores também podem ser inseridos em um nó, assim como aplicações.

A Figura 8.2 mostra um exemplo de cenário de distribuição em uma cidade inteligente. Neste cenário, os nós N1 e N2 estão localizados em dois computadores distintos e contém apenas dois agentes cada (A1 e A2 em N1; A3 e A4 em N2), além dos agregadores (Ag1 e Ag2). Além deles, há um nó N3 localizado em um pequeno servidor na névoa, contendo um agregador Ag3, um agente A5 e um intermediador I1. Um quarto nó N4 está localizado na nuvem, contendo um agregador Ag4, um intermediador I2 e dois agentes A6 e A7. Um quinto nó N5 na nevoa contém um agregador Ag5, um intermediador I3 e agentes A8, A9 e A10 distribuídos em *containers* ao longo de dispositivos na borda (*smartphones*.

A extensibilidade deste esquema é garantida pela possibilidade provida pelo JADE da adição de novas plataformas à arquitetura à qualquer momento, sem a necessidade de se reiniciar o sistema. Novos *containers* podem ser adicionados à plataformas existentes da mesma forma, assim como novos agentes a cada *container*. Portanto, novos agentes podem ser acrescentados a qualquer momento, sem a necessidade de se reinstalar ou reiniciar o sistema.

Novos agentes podem ser construídos a partir da API orientada a objetos da plataforma JADE, utilizando a linguagem Java. Cada agente deve ser implementado a partir de uma classe que deve estender a classe *Agent* presente no pacote *jade.core* da API.

Figura 8.2: Exemplo de distribuição de agentes da arquitetura utilizando a plataforma JADE.

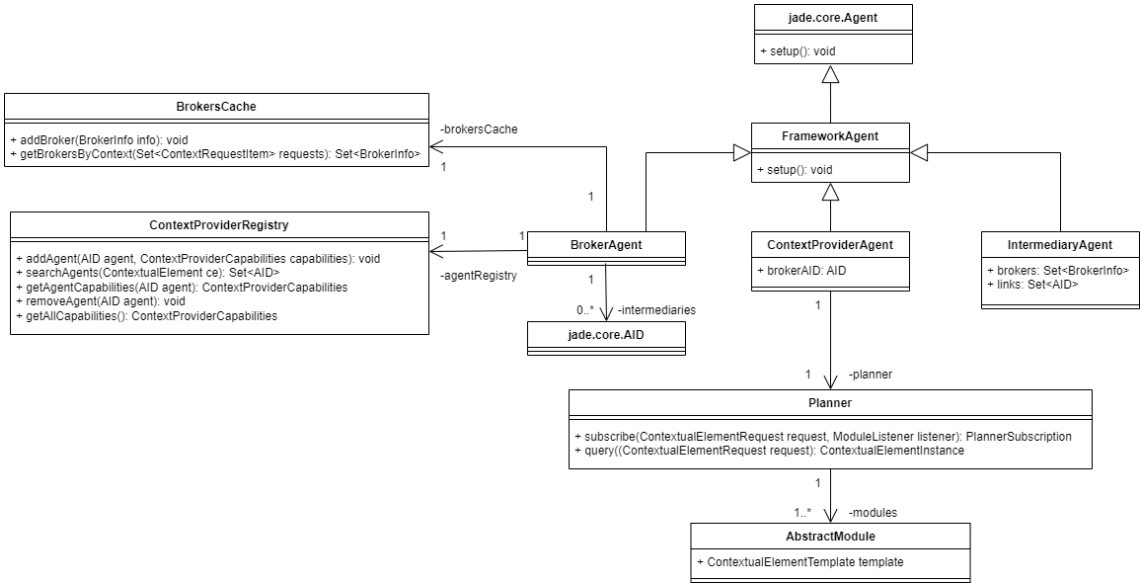


Fonte: O Autor

O *framework* desenvolvido neste trabalho possui uma classe chamada *FrameworkAgent* que estende a classe *Agent*, da qual derivam outras três classes, cada uma associada a um dos três tipos de agentes definidos na arquitetura: *BrokerAgent* (agregadores), *ContextProviderAgent* (agentes provedores de contexto em nós) e *IntermediaryAgent* (intermediadores). Estas classes encontram-se esquematizadas no diagrama de classes simplificado mostrado na Figura 8.3.

Todo agregador (classe *BrokerAgent*) possui um registro de agentes locais (classe *ContextProviderRegistry*, parte do gerenciador de agentes locais (ver Seção 6.6). Este componente permite que o agregador adicione novos agentes locais ao registro, busque por agentes que forneçam determinado elemento contextual (implementação da operação *buscaAgentesLocais*, mostrada no Algoritmo 2 (ver Seção 7.3), e obtenha capacidades registradas no nó (necessário para registro junto a agentes intermediadores). Outro componente presente em agregadores é um cache de agregadores remotos (classe *BrokersCache*), que é parte do gerenciador de comunicação com agentes remotos. Este cache implementa, por exemplo, a operação *buscaAgregadoresRemotosEmCache* presente nos Algoritmos 3 e 5 (ver Seções 7.3 e 7.4). Por fim, cada agregador também possui um con-

Figura 8.3: Diagrama de classes ilustrando a estrutura dos agentes implementados no *framework*.



Fonte: O Autor

junto de intermediadores relacionados a ele, obtido a partir de uma operação de anúncio em *broadcasting* e resposta dos intermediadores próximos. Cada intermediador e representado como uma identificação única de agente (classe *jade.core.AID*).

Agentes fornecedores locais (classe *ContextProviderAgent*) possuem um planejador (classe *Planner*). Cada planejador é capaz de elaborar planos para resposta a consultas e inscrições (ver Seção 6.7.1). Esses planos são elaborados a partir de um conjunto de módulos. Cada módulo deve ser implementado em uma subclasse de *AbstractModule*. No *framework* implementado, foram definidas subclasses dessa classe para cada tipo de módulo descrito na Seção 6.7.1: *Getter*, *Adder* e *Interpreter*.

Por fim, agentes intermediadores foram implementados através da classe *IntermediaryAgent*. Essa classe define um conjunto de informações sobre agregadores registrados (atributo *brokers*) e um conjunto de identificadores para intermediadores conectados (atributo *links*).

Os componentes de comunicação de todos os agentes utilizam os recursos de comunicação por troca de mensagens da plataforma JADE. O sistema de comunicação disponibilizado pela plataforma utiliza a linguagem FIPA-ACL para comunicação entre agentes. O conteúdo das mensagens é codificado utilizando a linguagem FIPA SL (*FIPA Semantic Language*)[3], e os termos utilizados são descritos na ontologia especificada na

[3]<http://www.fipa.org/specs/fipa00008/SC00008I.html>

Seção 6.2.2 e na Seção 6.4. A ontologia foi implementada utilizando o software Protégé versão 5.5 e a API do próprio JADE. Esta API permite o mapeamento de conceitos na ontologia para classes Java. A conexão para acesso à informações presentes na ontologia a partir de objetos Java foi realizada através do *framework* Jena [4].

Ações desempenhadas em agentes JADE são implementadas na forma de comportamentos (*behaviours*). Um agente pode contar diversos comportamentos diferentes. Cada tipo de comportamento novo deve ser uma subclasse da classe *jade.core.Behaviour*. As operações descritas no capítulo 7 foram implementadas na forma de comportamentos em diferentes agentes. Agentes agregadores utilizam três comportamentos: *AnnounceReceiverBehaviour* (que implementa o recebimento de anúncios de agentes fornecedores locais), *BrokerQueryResponderBehaviour* (que implementa a operação de consulta por parte de agregadores) e *BrokerSubscriptionResponder* (que processa solicitações de inscrições e envia notificações). Agentes fornecedores locais utilizam também três comportamentos: *ProviderAnnouncementInitiator* (que envia um anúncio ao agregador), *ProviderQueryResponderBehaviour* (que implementa a operação de consulta no agente local) e *ProviderSubscriptionResponder* (que processa inscrições recebidas pelo agente local e envia notificações). O código fonte completo do *framework* encontra-se disponível na Web[5].

8.2 Cenários de Avaliação

A partir do *framework* desenvolvido, uma aplicação teste foi criada com base no cenário descrito em Roggen et al. (2010). Diferentes experimentos foram aplicados utilizando a aplicação teste.

8.2.1 Descrição do Cenário Utilizado

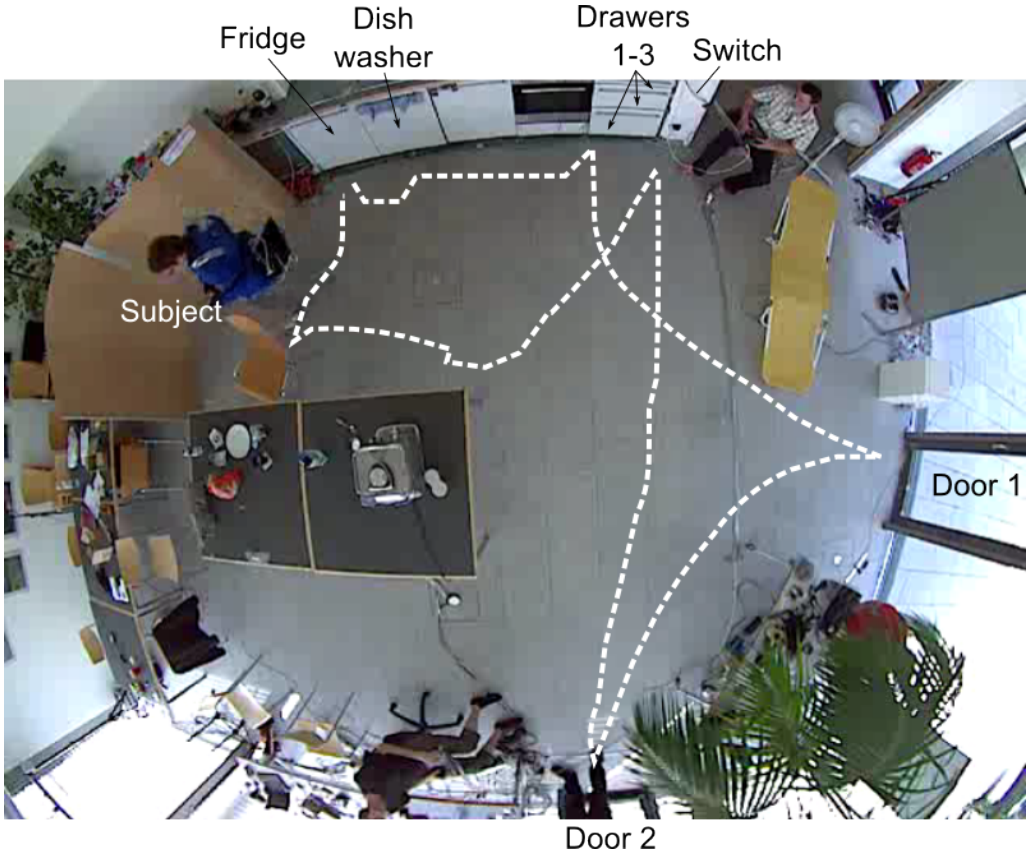
O cenário descrito envolve o reconhecimento de atividade de pessoas através da captação de dados de sensores de movimento enquanto são executadas tarefas rotineiras do dia-a-dia. Um *dataset* chamado *Opportunity* foi gerado a partir dos dados coletados através da execução de sequências de atividades por voluntários em uma sala simulando

[4] <https://jena.apache.org/>

[5] <https://drive.google.com/drive/folders/1q_A43-HAQt7XHWD8BhyAl6oMz9-tMj0J?usp=share_link>

um pequeno apartamento (Figura 8.4). Quatro atividades relacionadas a modos de locomoção foram executadas pelos voluntários: parado em pé (*standing*), caminhando (*walking*), deitado (*lying*) e sentado (*sitting*).

Figura 8.4: Cenário utilizado para captação de dados do *dataset Opportunity*.

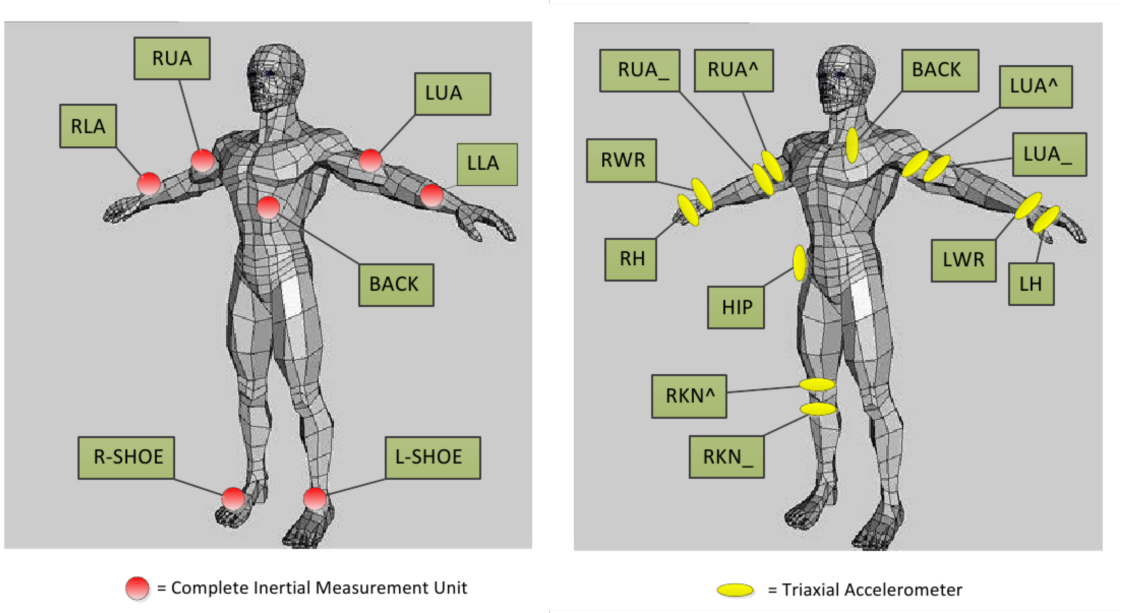


Fonte: (ROGGEN et al., 2010)

Os dados usados para obtenção das atividades foram gerados a partir de sensores vestíveis (*wearables*) usados por cada voluntário. A localização dos sensores utilizados é mostrada na Figura 8.5. Os sensores incluem 12 acelerômetros 3D (mostrados em amarelo na figura) e sete sistemas inerciais IMU (*Inertial Measurement Unit*) identificados em vermelho na figura. Os cinco IMUs localizados na parte superior do corpo dos voluntários (RLA, RUA, LUA, LLA e BACK) são capazes de gerar 9 parâmetros relacionados a movimento (aceleração 3D, rotação 3D e campo magnético 3D). Os IMUs localizados nos pés de cada voluntário (R-SHOE e L-SHOE) fornecem outros 32 parâmetros. No total, 113 parâmetros foram gerados por estes sensores e registrados no *dataset*.

O cenário descrito ainda continha sensores em objetos espalhados pelo ambiente. Porém, os dados desses sensores não foram utilizados para obtenção das atividades consideradas e foram ignorados neste processo de avaliação.

Figura 8.5: Localização de sensores utilizados para a captação de dados para o dataset *Opportunity*



Fonte: (CHAVARRIAGA et al., 2013)

8.2.2 Aplicação OPPORTUNITY

Uma aplicação foi implementada utilizando o *dataset Opportunity* e o *framework* desenvolvido. Cada um dos acelerômetros e IMUs foi implementado como um sensor simulado, obtendo dados do *dataset*. Um módulo *Getter* foi criado para cada um dos 113 parâmetros. Um módulo *Adder* foi criado para obtenção da média de um conjunto de valores e um módulo *Interpreter* foi criado para inferência da atividade a partir da média dos valores em uma janela de leitura dos dados de cada parâmetro. A inferência foi realizada através de um classificador KNN treinado a partir dos dados de treinamento disponíveis no *dataset*. O classificador foi implementado utilizando o *framework* Weka[6]. O classificador espera receber uma média de cada parâmetro a partir de uma janela de 500 ms de leitura para cada um, a uma taxa de amostragem de 30 Hz com sobreposição de 50%.

A aplicação utiliza uma extensão da ontologia da arquitetura. Na ontologia da aplicação foram especificados parâmetros dos sensores modelados, entidades presentes (um voluntário e mais uma entidade para cada sensor, que foi considerado uma entidade por medir dados de movimento de partes diferentes do corpo) e elementos contextuais

[6]<https://www.cs.waikato.ac.nz/ml/weka/>

utilizados (um para cada um dos 113 parâmetros).

Um pequeno programa Java foi escrito que simplesmente realiza uma consulta pela atividade do voluntário no momento. Essa consulta é direcionada a um nó da aplicação.

8.3 Experimentos

Diferentes experimentos foram realizados com base na aplicação *Opportunity*, com o objetivo de verificar se seria possível colocar em funcionamento um sistema de obtenção de contexto a partir da arquitetura desenvolvida e também avaliar o comportamento de tal sistema quanto ao tempo de resposta, e o quanto a redundância utilizada para fusão e tolerância a falhas impactaria a performance do sistema. Além disso, o desempenho de mecanismos de tolerância a falhas também foi avaliado. Estes experimentos foram escolhidos a fim de demonstrar que a arquitetura desenvolvida neste trabalho é passível de implementação prática e o quanto suas características afetariam o desempenho de aplicações que consumiriam contexto fornecido por ela. Cada um dos experimentos realizados e resultados obtidos são descritos à seguir.

8.3.1 Avaliação do Impacto da Distribuição de Sensores e Agentes

A fim de avaliar como a distribuição de módulos *Getter* associados aos aspectos fornecidos pelos sensores e os agentes podem afetar o desempenho de nós foram realizados 8 experimentos descritos abaixo. Estes experimentos foram escolhidos pois permitem avaliar o quanto a distribuição de fontes de contexto e seus agentes associados afeta o desempenho geral da arquitetura ao responder a uma requisição de consulta. Os dados obtidos são importantes para se embasar o planejamento da distribuição de agentes em uma situação real, a partir da performance exigida por um sistema. Em cada experimento a aplicação foi executada 50 vezes e o tempo médio de execução da consulta foi calculado.

- **EXPERIMENTO 1 (E1)**
  - **Descrição:** Um único nó em um único dispositivo contendo um único agente provedor de contexto com todos os módulos (um *interpreter*, um *adder* e 113 *getters*).

- **Configuração do Dispositivo:** Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits

- **EXPERIMENTO 2 (E2)**
  - **Descrição:** Um único nó em um único dispositivo contendo dois agentes: o primeiro agente contém dois módulos (um *interpreter*, um *adder*) e outro agente com os demais módulos (113 *getters*).
  - **Configuração do Dispositivo:** Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits

- **EXPERIMENTO 3 (E3)**
  - **Descrição:** Um único nó em um único dispositivo contendo 114 agentes: o primeiro agente contém dois módulos (um *interpreter*, um *adder*) e cada um dos demais 113 agentes contém um único diferente módulo *getter*.
  - **Configuração do Dispositivo:** Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits

- **EXPERIMENTO 4 (E4)**
  - **Descrição:** Um único nó distribuído em dois *containers*, C1 e C2, localizados em dispositivos diferentes. O *container* C1 contém um único agente contendo dois módulos (um *interpreter*, um *adder*). O *container* C2 é composto por um único agente com os demais módulos (113 *getters*). Os dispositivos estão conectados em uma LAN Ethernet 100 Mb/s.
  - **Configuração do Dispositivo (C1):** Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits
  - **Configuração do Dispositivo (C2):** Desktop Intel Core Duo 3 GHz, 4 GB RAM, Windows 10 64 bits

- **EXPERIMENTO 5 (E5)**
  - **Descrição:** Um único nó distribuído em dois *containers*, C1 e C2, localizados em dispositivos diferentes. O *container* C1 contém um único agente contendo dois módulos (um *interpreter*, um *adder*). O *container* C2 é composto por 113 agentes, cada um contendo um único diferente módulo *getter*. Os dispositivos estão conectados em uma LAN Ethernet 100 Mb/s.
  - **Configuração do Dispositivo (C1):** Laptop Intel Core i5 2.5 GHz, 8GB RAM

DDR 4, Windows 10 64 bits

- **Configuração do Dispositivo (C2):** Desktop Intel Core Duo 3 GHz, 4 GB RAM, Windows 10 64 bits

- **EXPERIMENTO 6 (E6)**
  - **Descrição:** Dois nós, N1 e N2, localizados em dispositivos diferentes. O nó N1 contém um único agente contendo dois módulos (um *interpreter*, um *adder*). O nó N2 é composto por um único agente com os demais módulos (113 *getters*). Os dispositivos estão conectados em uma LAN Ethernet 100 Mb/s. Aplicação executada no Nó N1.
  - **Configuração do Dispositivo (N1):** Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits
  - **Configuração do Dispositivo (N2):** Desktop Intel Core Duo 3 GHz, 4 GB RAM, Windows 10 64 bits
- **EXPERIMENTO 7 (E7)**
  - **Descrição:** Dois nós, N1 e N2, localizados em dispositivos diferentes. O nó N1 contém um único agente contendo dois módulos (um *interpreter*, um *adder*). O nó N2 é composto por 113 agentes, cada um contendo um único diferente módulo *getter*. Os dispositivos estão conectados em uma LAN Ethernet 100 Mb/s. Aplicação executada no Nó N1.
  - **Configuração do Dispositivo (N1):** Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits
  - **Configuração do Dispositivo (N2):** Desktop Intel Core Duo 3 GHz, 4 GB RAM, Windows 10 64 bits
- **EXPERIMENTO 8 (E8)**
  - **Descrição:** Um total de 20 nós foi distribuído em 20 dispositivos conectados em uma LAN Ethernet 100 Mb/s. O primeiro nó N1 contém um único agente contendo dois módulos (um *interpreter*, um *adder*). Os demais nós, N2 a N20, são compostos, cada um, por um agente correspondente a um diferente sensor. Portanto, cada agente pode conter de 3 a 16 *getters*. Aplicação executada no Nó N1.
  - **Configuração do Dispositivo (N1):** Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits

- **Configuração dos Dispositivos (N2 a N20):** 1 Desktop Intel Core i7 2 GHz, 16GB RAM, Windows 10 64 bits; 2 Desktops Intel Core i5 3.10 GHz, 8GB RAM, Windows 10 64 bits; 3 Desktops Intel Core Duo 3 GHz, 4 GB RAM, Windows 10 64 bits; 13 Desktops AMD Phenom II 550 3.10 GHz, 4GB RAM, Windows 10 64 bits.

Os experimentos apresentados mostram que a arquitetura desenvolvida neste trabalho permite a distribuição de fontes de contexto de diferentes formas, dentro de uma mesma LAN. Portanto é possível utilizar a arquitetura em diversos tipos de ambientes inteligentes dentro de uma cidade, como pequenos aeroportos, rodoviárias, estações de trem e metrô, além de ambientes como casas inteligentes, salas de reuniões inteligentes e salas de aula inteligentes.

O tempo médio de execução da aplicação calculado para cada experimento é mostrado na Tabela 8.1 e na Figura 8.6. Este tempo leva em conta o envio da solicitação de consulta por parte da aplicação, o processamento da consulta pelo *framework* e o envio da resposta de volta à aplicação. A aplicação, neste caso, entra na arquitetura como um *container* que se conecta a um dos nós envolvidos. Todos os experimentos apresentam tempo maior do que 500 ms, o tempo necessário para obtenção da janela de leituras de todos os sensores.

Tabela 8.1: Tempo médio de execução de consultas nos experimentos 1 a 8 (distribuição de agentes e sensores)

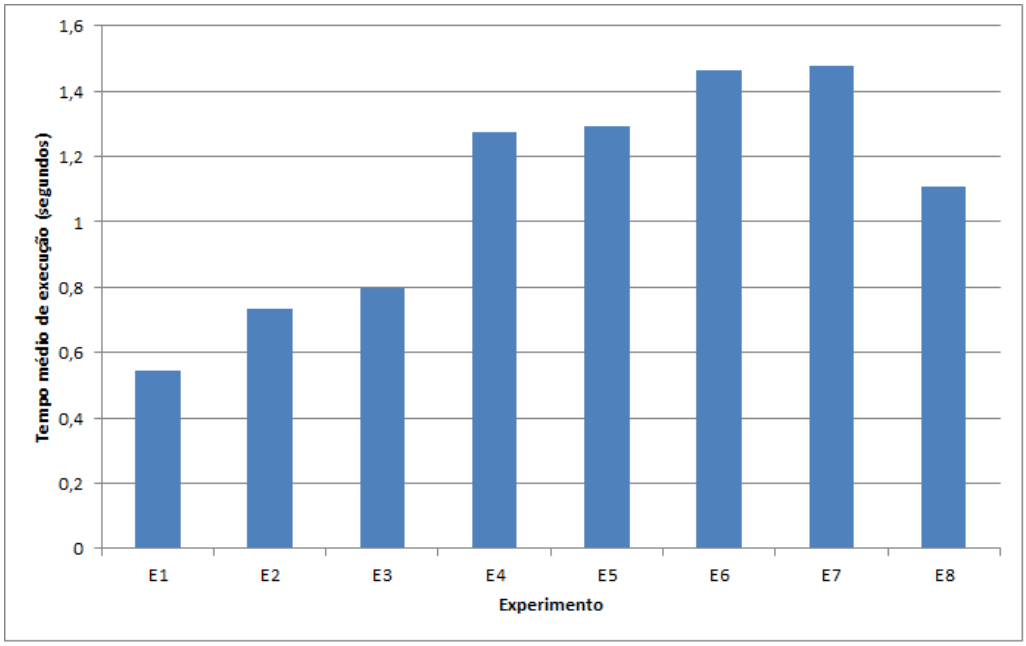
Experimento	Tempo Médio de Execução (segundos)
E1	0,544
E2	0,732
E3	0,798
E4	1,276
E5	1,293
E6	1,462
E7	1,478
E8	1,108

Fonte: O Autor

No primeiro experimento E1, há um pequeno tempo extra para extração de dados da solicitação de consulta recebida da aplicação pelo agente agregador, cálculo das médias dos valores obtidos para cada sensor (113), o tempo para inferência da atividade com base nesses dados e a montagem da mensagem de resposta. Como tudo ocorreu a partir de um plano em um único agente, o tempo médio de execução foi relativamente rápido.

Os experimentos 2 e 3 envolvem a separação dos sensores em agentes diferentes.

Figura 8.6: Tempo médio de execução de consultas nos experimentos 1 a 8 (distribuição de agentes e sensores)



Fonte: O Autor

Há um aumento no tempo médio de execução, já que desta vez há a necessidade dos agentes que contém os sensores montarem uma mensagem contendo seus resultados e enviá-la ao agente agregador. Este último, por sua vez, precisa analisar a mensagem e extrair os resultados. Isto está contribuindo mais para o aumento do tempo de execução, já que nenhuma otimização no processamento de mensagens foi realizado, o que poderia melhorar a performance destas consultas.

Os experimentos 4 e 5 distribuem os agentes de inferência e os de obtenção de dados de sensores em dispositivos diferentes, porém dentro do mesmo nó. É possível observar que há um aumento no tempo de execução devido principalmente à comunicação entre os agentes fornecedores e o agregador passar por uma rede. Não há diferença significativa entre utilizar um único agente fornecedor ou distribuir os sensores entre diversos agentes.

Os experimentos 6 e 7 são similares aos efetuados nos experimentos 4 e 5, porém agora os agentes com sensores estão em nós separados do agente de inferência. Neste caso, há uma etapa a mais, onde os agentes com sensores devem encaminhar os resultados para o agregador local de seu nó, que repassa o resultado para o agregador do nó contendo o agente de inferência. Há um pequeno acréscimo de tempo em ambos os experimentos, devido a existência de mais um agente intermediário no processo.

É possível observar que a maior distribuição dos agentes e sensores realizada no experimento 8 não piorou significativamente a performance das consultas quando comparado com os experimentos anteriores. Esse experimento realizou uma distribuição que seria plausível, considerando cada sensor físico como um agente distribuído em uma rede de comunicação. Esse paralelismo produziu, inclusive, resultados ligeiramente melhores que os experimentos 4, 5, 6 e 7, onde os sensores encontravam-se centralizados em um único nó.

8.3.2 Avaliação do Impacto da Replicação de Elementos Contextuais e Fusão de Dados

A arquitetura desenvolvida neste trabalho permite que diversos agentes possam fornecer os mesmos elementos contextuais, permitindo replicação de dados. As instâncias desses elementos contextuais obtidas são fundidas pelo agregador.

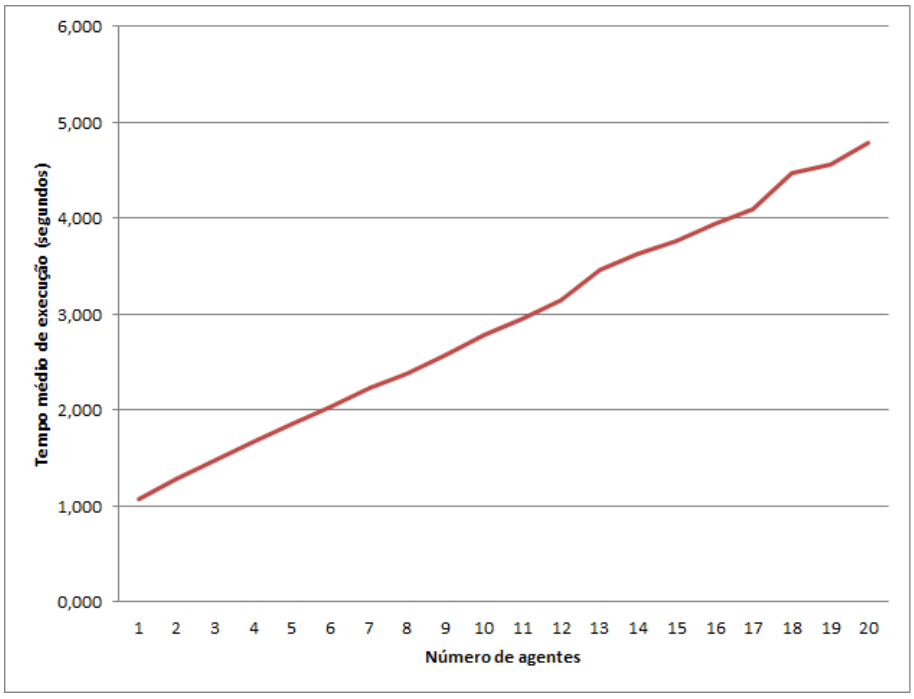
Com o objetivo de avaliar o impacto da duplicação de fontes de contexto no desempenho de um sistema, foram realizados 2 experimentos descritos a seguir. Em cada experimento a aplicação de consulta foi executada 50 vezes e o tempo médio de execução da consulta foi calculado. É importante compreender o quanto a escala de replicação impacta no tempo de execução de consultas a fim de se entender os limites da arquitetura quando aplicada a sistemas que são críticos quanto ao tempo.

- **EXPERIMENTO 9 (E9)**
  - **Descrição:** Um único nó em um único dispositivo contendo um agente provedor de contexto com dois módulos (um *interpreter* e um *adder*) e um outro agente contendo os 113 *getters* em um *container* separado, dentro do mesmo dispositivo. O experimento foi repetido outras 19 vezes, acrescentando um número crescente de agentes com os 113 *getters* no segundo container, de forma a criar um número crescente de réplicas para todos os 113 dados de sensores. Assim, o experimento foi executado com 1, 2, 3, até 20 agentes fornecedores com os módulos *getter*.
  - **Configuração do Dispositivo:** Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits
- **EXPERIMENTO 10 (E10)**

- **Descrição:** Uma quase repetição do experimento anterior, porém cada agente fornecedor contendo os módulos *getter* foi colocado em um nó diferente. A aplicação é executada sempre no nó contendo o agente com o módulo de inferência. Dessa forma, os agentes fornecedores com os módulos *getter* serão tratados como agentes remotos.
- **Configuração dos Dispositivos Utilizados:** 1 Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits; 1 Desktop Intel Core i7 2 GHz, 16GB RAM, Windows 10 64 bits; 2 Desktops Intel Core i5 3.10 GHz, 8GB RAM, Windows 10 64 bits; 3 Desktops Intel Core Duo 3 GHz, 4 GB RAM, Windows 10 64 bits; 13 Desktops AMD Phenom II 550 3.10 GHz, 4GB RAM, Windows 10 64 bits.

Os experimentos descritos aqui mostram que é possível implementar a arquitetura com o recurso de duplicação de fontes de contexto, além da possibilidade de se estender a arquitetura com novos agentes em um nó com com novos nós. O tempo médio de execução da aplicação calculado para cada experimento é mostrado nas Figuras 8.7 e 8.8. O tempo aqui foi medido de forma similar ao executado nos experimentos anteriores.

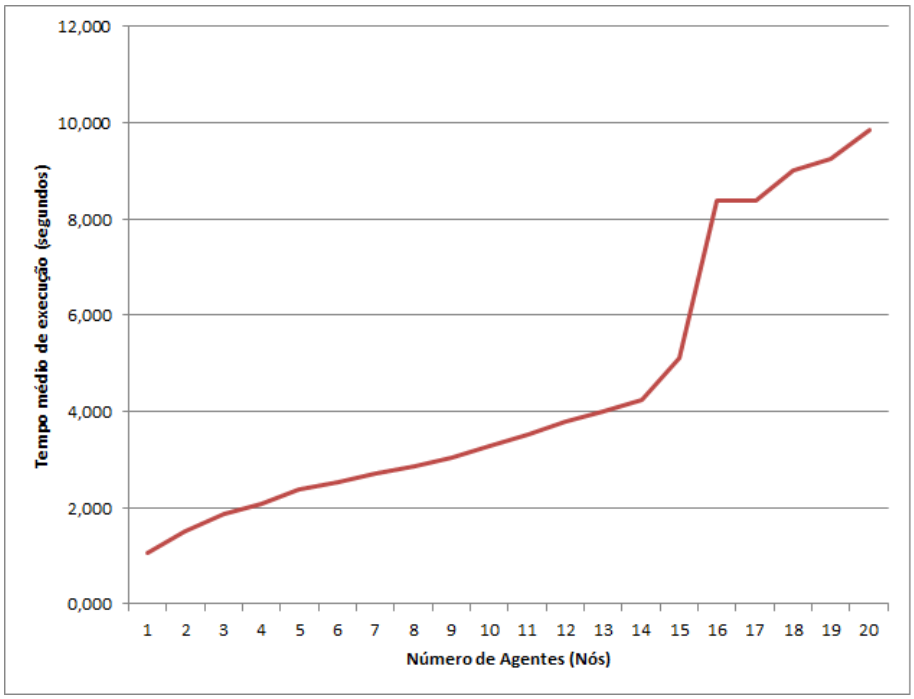
Figura 8.7: Tempo médio de execução de consultas no experimento E9



Fonte: O Autor

Cabe ressaltar que os tempos medidos em cada experimento incluem o envio da

Figura 8.8: Tempo médio de execução de consultas no experimento E10



Fonte: O Autor

solicitação de consulta pela aplicação (que é executada como um *container* no mesmo dispositivo do nó onde encontra-se o agente responsável pela inferência), o envio da mensagem do agregador para os agentes fornecedores que contém os módulos *getter* (nó experimento 10 isso envolve o contato do agregador de cada nó fornecedor com intermediário), o envio da mensagem de cada fornecedor para o agregador, o processamento dessas mensagens pelo agregador, a fusão dos dados redundantes, o envio dos dados de sensores ao agente de inferência, o processamento da inferência e o envio do resultado da consulta à aplicação.

O envio dos dados de contexto produzidos pelos módulos *getter* inclui valores de janelas para 113 aspectos, o que gera uma mensagem consideravelmente grande. A cada aumento do número desses agentes, há um aumento também no número de mensagens enviadas. Isso gera um maior tráfego na rede e também um maior tempo para que o agente agregador do nó principal processe essas mensagens e faça a fusão dos dados redundantes (todos os 113 aspectos são redundantes, quando o número de agentes é maior do que 1).

É possível observar no experimento 9 que o tempo de resposta cresce linearmente de acordo com o aumento do número de agentes fornecendo dados redundantes. Pode-se dizer que o sistema escala bem, sem um aumento exponencial no tempo de execução à medida que se aumenta a escala de agentes redundantes. Além disso, os experimentos

foram realizados com um número relativamente alto de elementos contextuais com redundância (113). Números menores de fontes de redundância geram tempos de resposta menores.

No experimento 10 cada agente foi executado em um nó diferente em dispositivos diferentes, em uma rede Ethernet 100 Mb/s. O tempo de resposta se torna maior, já que existe o agregador de cada nó como intermediário no caminho. O tempo de execução cresceu linearmente até a execução com 14 nós, quando o tempo tem um aumento brusco. Percebe-se assim que, a partir de um certo número de agentes redundantes em nós diferentes, podem ocorrer aumentos grandes no tempo de execução. Mecanismos de limitação no número de agentes utilizados podem ser necessários a fim de garantir tempos de resposta aceitáveis, principalmente para sistemas onde o tempo de resposta seja crítico. Tais mecanismos podem incluir, por exemplo, abordagens de seleção baseados em melhores taxas de transmissão, ou agentes com fornecimentos de melhores valores de QoC, etc.

8.3.3 Avaliação dos Mecanismos de Tolerância a Falhas

A fim de avaliar os mecanismos de tolerância a falhas propostos na arquitetura deste trabalho, um experimento foi realizado, conforme descrito abaixo:

- **EXPERIMENTO 11 (E11)**
  - **Descrição:** Um único nó em um único dispositivo contendo um agente provedor de contexto com dois módulos (um *interpreter* e um *adder*) e outros dois agentes contendo os 113 módulos *getter*, de forma que ambos oferecem informações redundantes.
  - **Configuração do Dispositivo:** Laptop Intel Core i5 2.5 GHz, 8GB RAM DDR 4, Windows 10 64 bits

Neste experimento foram a aplicação teste foi executada 50 vezes em 100 séries, em cada série com uma probabilidade de falha diferente para os sensores, considerando probabilidades de falha de 0 a 1%. Uma probabilidade de falha de 0,01% significa que o sensor em questão pode apresentar uma falha a cada 10000 consultas. No experimento, a probabilidade em questão foi aplicada a todos os sensores, usados pelos dois agentes que fornecem seus dados. Os resultados do experimento são mostrados na Tabela 8.2.

Uma falha em um sensor causa uma falha na consulta a um módulo *getter* que,

Tabela 8.2: Quantidade de falhas encontradas no experimento 11

P. F. S.	F. S.	F. C.	P. F. S.	F. S.	F. C.	P. F. S.	F. S.	F. C.
0,00	0	0	0,34	547	10	0,68	1111	30
0,01	22	0	0,35	625	11	0,69	1082	31
0,02	38	0	0,36	580	16	0,70	1047	32
0,03	50	1	0,37	596	16	0,71	1148	32
0,04	66	0	0,38	637	16	0,72	1166	32
0,05	78	0	0,39	611	11	0,73	1159	33
0,06	98	0	0,40	643	15	0,74	1175	39
0,07	88	1	0,41	664	15	0,75	1252	39
0,08	130	1	0,42	683	14	0,76	1186	40
0,09	151	1	0,43	705	22	0,77	1264	37
0,10	172	1	0,44	713	14	0,78	1232	36
0,11	157	0	0,45	781	19	0,79	1266	39
0,12	179	2	0,46	743	18	0,80	1300	40
0,13	223	2	0,47	763	21	0,81	1251	32
0,14	254	0	0,48	815	26	0,82	1314	37
0,15	277	3	0,49	813	22	0,83	1307	44
0,16	282	7	0,50	836	22	0,84	1337	42
0,17	283	1	0,51	874	25	0,85	1438	42
0,18	293	10	0,52	852	20	0,86	1391	40
0,19	311	0	0,53	911	24	0,87	1458	41
0,20	351	6	0,54	880	26	0,88	1379	41
0,21	333	4	0,55	865	26	0,89	1402	39
0,22	386	3	0,56	874	25	0,90	1444	38
0,23	403	13	0,57	986	30	0,91	1441	44
0,24	413	12	0,58	974	27	0,92	1489	41
0,25	425	4	0,59	955	24	0,93	1489	43
0,26	423	7	0,60	1005	32	0,94	1480	42
0,27	439	9	0,61	1047	25	0,95	1543	45
0,28	508	15	0,62	1051	33	0,96	1524	45
0,29	457	7	0,63	1037	32	0,97	1591	45
0,30	455	8	0,64	1102	27	0,98	1496	41
0,31	523	9	0,65	990	30	0,99	1557	46
0,32	488	10	0,66	1068	30	1,00	1554	39
0,33	557	12	0,67	1046	32			

Fonte: O Autor. Significado das siglas: P.F.S. (Probabilidade de Falha no Sensor), F.S. (Falhas em Sensores), F.C. (Falhas em Consultas)

no caso do cenário utilizado, causa uma falha na consulta, caso não exista outra fonte redundante para ser utilizada. Falhas em sensores dizem respeito a qualquer falha que impeça o agente de ter acesso a seus dados (problemas de hardware, problemas de rede, etc.) No caso do experimento 11, uma falha de consulta só ocorre quando ambos os agentes apresentarem falhas simultâneas. Com dois agentes redundantes, falhas em consultas começam a se tornar comuns em torno de 300 falhas de sensores, o que é uma quanti-

dade alta de falhas. Esses valores podem melhorar com o uso de maior redundância, que deve ser usada com a cautela quando existirem restrições de tempo (conforme discutido na seção anterior).

8.4 Discussão dos Resultados

Neste capítulo foram apresentadas a implementação da arquitetura na forma de um *framework* multi-agente, que permite a distribuição dos agentes em diferentes dispositivos, distribuídos em rede. A partir do *framework* implementado, foram realizados diversos experimentos, utilizando diferentes configurações de nós e agentes a partir de um mesmo cenário. Também foram explorados o impacto da replicação de dados e também dos mecanismos de tolerância a falhas implementados.

O desenvolvimento do *framework* e os experimentos realizados mostram que a questão de pesquisa levantada nesta tese, se é possível integrar fontes de contexto em uma cidade inteligente de forma que aplicações possam acessar tais informações de forma transparente, foi respondida satisfatoriamente. Além disso, demonstrou-se também que a implementação prática da arquitetura é viável. A implementação realizada permite a expansão da arquitetura com a adição de novos agentes e nós através da inclusão de novos *containers* à plataforma JADE utilizada, como executado nos experimentos 9 e 10. Além disso, aplicações podem ser executadas a partir de qualquer nó, o que possibilita a execução delas a partir de qualquer ponto da rede, sem a necessidade de um nó central de execução.

O experimento 11 mostrou que os mecanismos de tolerância a falhas implementados diminuem a incidência de falhas em consultas, mesmo na presença de falhas nas fontes de dados utilizadas. O uso de mais agentes com elementos contextuais duplicados pode proporcionar um desempenho quanto a falhas melhor ainda, porém com ônus ao desempenho geral de consultas, como mostrado nos experimentos 9 e 10. Portanto, em aplicações onde o tempo de execução é um fator importante, deve haver um limite ao número de agentes consultados pelo agregador. Isso pode ser realizado pelos desenvolvedores (estabelecendo um número de agentes cujo processamento de suas respostas fique dentro do tempo limite) ou através de algum mecanismo a ser acrescentado à arquitetura para restrição ao número de agentes consultados de acordo com restrições de tempo informadas pelas aplicações.

Por fim, os experimentos demonstraram que a composição dinâmica ocorre auto-

maticamente. Agregadores recebem solicitações de elementos contextuais e selecionam automaticamente agentes locais para fornecê-los. É possível ver, por exemplo, que os experimentos realizados utilizam diferentes configurações de agentes e, em todas as situações, a consulta foi realizada com sucesso, de forma que os agentes disponíveis foram localizados e uma composição deles foi organizada a fim de obter o elemento contextual requisitado.

9 CONCLUSÃO

Cidades inteligentes são um ramo de pesquisa bastante aquecido na atualidade. Sistemas sensíveis ao contexto também representam uma área de extrema importância, dada a demanda crescente por serviços personalizados e que sejam mais adequados à situação em que as pessoas se encontram.

Arquiteturas de reconhecimento de contexto já foram propostas para a obtenção de contexto em cidades inteligentes. Porém, este trabalho traz contribuições diferenciadas por tratar as seguintes questões:

- *Computação distribuída*. Como já mostrado na seção 6.3, a arquitetura proposta é inerentemente distribuída. Nós podem estar localizados em computadores na borda (como *smartphones*, PCs, *tablets*, etc.), ou até em servidores na névoa ou na nuvem. Cada nó é interligado por intermediadores responsáveis por interligar estes nós distribuídos. Intermediadores também podem estar distribuídos pela névoa ou nuvem.
- *Descentralização*. A arquitetura foi desenvolvida de forma que não há a necessidade de um nó ou intermediador central que controla todo o sistema. Portanto, é naturalmente descentralizada.
- *Escalabilidade da Arquitetura*. A escolha de uma arquitetura multiagente possibilita que novos agentes possam facilmente ser agregados ao sistema. Isso contribui grandemente para a escalabilidade da arquitetura quanto à inclusão de novos sensores e novas funcionalidades. Para que novas informações contextuais possam ser geradas a partir de novos sensores, basta incluir novos agentes que acessem esses sensores e processem os dados gerados por eles para obtenção dessas informações.
- *Resiliência da Arquitetura*. A arquitetura desenvolvida neste trabalho incorpora mecanismos de resiliência ao suportar a substituição de agentes por outros agentes equivalentes, capazes de fornecer o mesmo tipo de informação. Esses agentes podem ser tanto locais (presentes no mesmo nó), quanto remotos (presentes em outros nós).
- *Fusão de Dados* Com a grande quantidade de fontes de dados redundantes que podem existir em uma cidade inteligente, mecanismos de fusão de dados são importantes. A arquitetura desenvolvida neste trabalho prevê a fusão automática de instâncias de um mesmo elemento contextual providas por agentes diferentes. Esse

processo é executado pelo agente agregador.

- *Composição Dinâmica*. A arquitetura permite que diversos mecanismos de inferência e fontes de dados contextuais sejam adicionados. Além do mais, esses mecanismos são selecionados a cada requisição, levando-se em conta os requisitos de cada aplicação sobre informações de contexto e qualidade. Assim, agentes diferentes podem ser selecionados para a mesma tarefa em momentos diferentes e para aplicações com necessidades diferentes.

Uma ontologia foi construída para que os agentes possuam um vocabulário comum relacionado aos termos de contexto utilizados. A ontologia é genérica o suficiente para ser utilizada, inclusive, em outras plataformas onde seja necessário representar contexto.

Um *framework* foi desenvolvido, onde a arquitetura foi implementada. Os elementos da arquitetura foram codificados na linguagem Java utilizando a plataforma JADE, permitindo que experimentos fossem realizados em diferentes configurações e cenários. Os experimentos mostraram que a arquitetura pode ser implementada na prática e que fornece bons tempos de resposta para requisições de aplicações. Esses tempos podem ser melhorados através da otimização de alguns processos, como o processamento de mensagens.

9.1 Contribuições

A seguir são destacadas as contribuições desta tese em duas grandes frentes. Na primeira, são pontuadas as contribuições diretamente relacionadas à arquitetura desenvolvida. Na segunda, são apresentadas as contribuições no âmbito acadêmico, isto é, as publicações e demais atividades realizadas que demonstram a extensão desta tese, seus relacionamentos, sua aplicabilidade, bem como o trabalho já desenvolvido e a experiência adquirida.

- Uma arquitetura multiagente que permite o acesso de aplicações a elementos contextuais de forma transparente. Isso ocorre através da integração de diferentes fontes de contexto em um ambiente inteligente de forma descentralizada e tolerante a falhas. Instâncias de elementos contextuais podem ser fundidas e as fontes de contexto são selecionadas de forma dinâmica, podendo ser incluídas a qualquer momento.

- Um *framework* na linguagem Java ainda em estava de testes, mas que pode ser aprimorado para o desenvolvimento de futuras aplicações sensíveis ao contexto em ambientes inteligentes.
- Uma ontologia de alto nível para modelagem de elementos contextuais e suas instâncias em ambientes inteligentes. A ontologia pode ser estendida para modelar diferentes subdomínios e ambientes.
- A realização de experimentos que avaliam o desempenho do *framework* desenvolvido a capacidade da arquitetura de prover contexto em diferentes situações.

A seguir são apresentados os artefatos de pesquisa já publicados, submetidos à publicação e/ou em processo de redação durante o período de doutorado, em ordem cronológica, do mais atual até o mais antigo. Ao todo, são sete publicações sendo, duas submissões em conferências e cinco em *journals*. Abaixo, são apresentadas as publicações, como primeiro autor, diretamente relacionadas à esta tese:

1. **ActCity - An Ontology for User Activities on Urban Environment** - este artigo apresenta uma extensão da ontologia desenvolvida neste trabalho. A ideia é modelar atividades de pessoas em uma cidade inteligente. Artigo em fase de finalização da redação a ser submetido ao *Seminário de Ontologias do Brasil* (ONTOBRAS 2023).
2. **An Ontology for Context Modeling in Smart Spaces** - este artigo apresenta a ontologia desenvolvida nesta tese, ampliando a aplicabilidade da mesma para diferentes ambientes inteligentes. Artigo submetido à conferência *ER - Conceptual Modeling* (ER 2023).
3. **Decentralized, distributed and fault-tolerant context recognition architectures for smart cities: A systematic mapping** - artigo que mostra em detalhes o mapeamento sistemático da literatura desenvolvido para se levantar os trabalhos relacionados a esta tese e descritos no Capítulo 4 (NASCIMENTO; OLIVEIRA, 2021). Foi publicado no *journal Reviews on Computer Science* da Sociedade Brasileira de Computação.
4. **Context recognition and ubiquitous computing in smart cities: a systematic mapping** - artigo que descreve o mapeamento sistemático da literatura desenvolvido na fase inicial de desenvolvimento desta tese, a fim de se conhecer e identificar como contexto é tratado em aplicações sensíveis a ele em cidades inteligentes. Parte dos resultados obtidos encontra-se descrita na Seção 2.8 (NASCIMENTO et

al., 2021). Artigo publicado no *journal Springer Computing*. QUALIS A1.

Pretende-se também redigir um artigo com a descrição da arquitetura e sua avaliação, para publicação futura. A intenção é submetê-lo ao *journal Elsevier Applied Soft Computing*, QUALIS A1.

Por fim, são apresentadas as publicações concebidas em coautoria e colaboração com os trabalhos de outros colegas:

1. **Smart Campuses e Ontologias: Cenário Atual e Indicacão de Trabalhos Futuros a partir de um Mapeamento Sistemático da Literatura** - este artigo teve o objetivo de realizar um mapeamento sistemático da literatura a fim de identificar ontologias aplicadas a *smart campuses*. O trabalho identificou o estado da arte na área e como este pode evoluir através da proposição de trabalhos futuros. Artigo submetido à *Revista Novas Tecnologias na Educação* (RENOTE). QUALIS A4.
2. **Recommender Systems in IoT-driven Smart Cities: A systematic Mapping Towards a Service-based Architecture** - este artigo apresenta um mapeamento sistemático da literatura, buscando evidenciar o potencial de concepção de uma arquitetura baseada em serviços para prover dados a sistemas de recomendação em cidades inteligentes. Trabalho submetido ao *journal Elsevier Future Generation Computer Systems*. QUALIS A1.
3. **Service Recommendation System for a Smart City** - este artigo apresenta um sistema de recomendação para recomendação de pontos de interesse em uma cidade inteligente baseado em ontologias. Uma ontologia para representação de pontos de interesse é apresentada, bem como regras SWRL para a realização da recomendação (PICCOLO et al., 2022). Artigo publicado no volume 11, número 1 dos Cadernos de Informática do Instituto de Informática da UFRGS: RENASCE - Recomendação sensível ao contexto de recursos e eventos em campus universitários utilizando aprendizagem de máquina (2022).

As produções e os desdobramentos desta tese estão intimamente ligados aos projetos de pesquisa aos quais o autor está atrelado. O primeiro deles é o projeto CNPq, chamada universal, submetido pelo prof. orientador e, aprovado em 2017, denominado: Recomendação adaptativa para cidades inteligentes. Tem como objetivo recomendar recursos informacionais adaptados de acordo com o nível de conhecimentos e contexto do usuário. Acredita-se que a recomendação de recursos informacionais de tipos diversos pode estimular usuários de dispositivos móveis a tornarem-se mais inseridos socialmente

em suas cidades, contribuindo assim para o aperfeiçoamento do nível de participação cidadã da população.

Em 2021, outro projeto de pesquisa, sob chamada CNPq/MCTI/FNDCT Nº 18/ 2021, voltado para grupos de pesquisa emergentes, foi aprovado: Renasce - Recomendação sensível ao contexto de recursos e eventos em campus universitários utilizando aprendizagem de máquina. Este projeto de pesquisa apresenta a proposta de conceituação e definição de um sistema de recomendação de recursos e eventos em campus universitários inteligentes, que implementa um modelo de integração que atua em conjunto com a definição formal de contexto através de ontologias para suportar recomendações sensíveis ao contexto em campus universitários. O autor dessa tese é colaborador do projeto, juntamente com o professor orientador.

9.2 Trabalhos Futuros

Como trabalhos futuros, aponta-se, como primeira possibilidade, a otimização e extensão do *framework* implementado para o suporte a agentes executando em dispositivos móveis e vestíveis, assim como dispositivos em redes IoT. Existe suporte atualmente da plataforma JADE para implementação de *containers* em *smartphones*. Portanto, seria necessário analisar a viabilidade e performance de se implementar agentes nesses dispositivos. O desenvolvimento do suporte a esses dispositivos possibilitará a construção de aplicações reais e experimentos com usuários utilizando, por exemplo, seus próprios *smartphones*.

Uma segunda possibilidade vislumbra-se o uso da arquitetura e do *framework* no desenvolvimento de aplicações reais em cidades inteligentes. Possíveis aplicações incluem: aplicações visando acessibilidade, educação e realidade aumentada.

Outra possibilidade de trabalho é aplicar a arquitetura construída em aplicações voltadas a *smart campuses*, em consonância com o projeto de pesquisa ao qual o autor desta tese é colaborador. A aplicação envolveria a coleta de elementos contextuais no ambiente de um campus para apoio a um sistema de recomendação. A possibilidade de aplicação é real, pois é possível pensar em um *campus* universitário como uma versão reduzida de uma cidade.

Uma quarta possibilidade diz respeito à análise e desenvolvimento de suporte a segurança e privacidade na arquitetura. Este é um requisito importante para ao desenvolvimento de aplicações reais onde dados de usuários sejam transmitidos em rede, inclusive

por questões legais. A arquitetura, atualmente, privilegia o uso de fontes de contexto locais para evitar, o quanto possível, a transmissão desses dados via rede. Porém, mecanismos específicos precisam ser utilizados para garantir segurança e privacidade. Alguns desses mecanismos foram revisados em trabalho do autor (NASCIMENTO; OLIVEIRA, 2021) e podem ser adaptados à arquitetura desenvolvida.

Por fim, a arquitetura atual não especifica como o histórico de contextos será gerenciado. As instâncias de elementos contextuais possuem um componente temporal associado. Como trabalhos futuros pretende-se especificar como esses elementos serão armazenados nos nós e como será feito o acesso ao histórico (as consultas permitem apenas acesso ao contexto atual de usuários).

REFERÊNCIAS

ABOWD, G. D. et al. Towards a better understanding of context and context-awareness. In: SPRINGER. **International symposium on handheld and ubiquitous computing**. [S.l.], 1999. p. 304–307.

ADOMAVICIUS, G. et al. Incorporating contextual information in recommender systems using a multidimensional approach. **ACM Transactions on Information Systems (TOIS)**, ACM New York, NY, USA, v. 23, n. 1, p. 103–145, 2005.

AGUILAR, J.; JEREZ, M.; RODRÍGUEZ, T. Cameonto: Context awareness meta ontology modeling. **Applied computing and informatics**, Elsevier, v. 14, n. 2, p. 202–213, 2018.

AL-SHARGABI, A.; SIEWE, F.; ZAHARY, A. Quality of context in context-aware systems. Journal of Context-aware Systems and Applications, 2017.

ALAM, M. A. U. et al. Infrequent non-speech gestural activity recognition using smart jewelry: Challenges and opportunities for large-scale adaptation. In: **Handbook of Large-Scale Distributed Computing in Smart Healthcare**. [S.l.]: Springer, 2017. p. 443–471.

ALAVI, A. H. et al. Internet of things-enabled smart cities: State-of-the-art and future trends. **Measurement**, Elsevier, v. 129, p. 589–606, 2018.

ALBINO, V.; BERARDI, U.; DANGELICO, R. M. Smart cities: Definitions, dimensions, performance, and initiatives. **Journal of urban technology**, Taylor & Francis, v. 22, n. 1, p. 3–21, 2015.

ALVAREZ-NAPAGAO, S. et al. Urban context detection and context-aware recommendation via networks of humans as sensors. In: **Agent Technology for Intelligent Mobile Services and Smart Societies**. [S.l.]: Springer, 2014. p. 68–85.

ARANDA, J. A. S. et al. A computational model for adaptive recording of vital signs through context histories. **Journal of Ambient Intelligence and Humanized Computing**, Springer, p. 1–15, 2021.

ARASTEH, H. et al. Iot-based smart cities: a survey. In: IEEE. **2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)**. [S.l.], 2016. p. 1–6.

ASIMAKOPOULOU, E.; BESSIS, N. Buildings and crowds: Forming smart cities for more effective disaster management. In: IEEE. **2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing**. [S.l.], 2011. p. 229–234.

BARBA, C. T. et al. Smart city for vanets using warning messages, traffic statistics and intelligent traffic lights. In: IEEE. **2012 IEEE intelligent vehicles symposium**. [S.l.], 2012. p. 902–907.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software architecture in practice: Third Edition**. [S.l.]: Addison-Wesley Professional, 2012.

BAZIRE, M.; BRÉZILLON, P. Understanding context before using it. In: SPRINGER. **International and Interdisciplinary Conference on Modeling and Using Context**. [S.l.], 2005. p. 29–40.

BERNARDOS, A. M.; MADRAZO, E.; CASAR, J. R. An embeddable fusion framework to manage context information in mobile devices. In: SPRINGER. **International Conference on Hybrid Artificial Intelligence Systems**. [S.l.], 2010. p. 468–477. <https://link.springer.com/chapter/10.1007/978-3-642-13803-4_58>.

BETTINI, C. et al. A survey of context modelling and reasoning techniques. **Pervasive and mobile computing**, Elsevier, v. 6, n. 2, p. 161–180, 2010.

BUNNINGEN, A. H. V.; FENG, L.; APERS, P. M. Context for ubiquitous data management. In: IEEE. **International Workshop on Ubiquitous Data Management**. [S.l.], 2005. p. 17–24.

CABRERA, O.; FRANCH, X.; MARCO, J. 3lconont: a three-level ontology for context modelling in context-aware computing. **Software & Systems Modeling**, Springer, v. 18, n. 2, p. 1345–1378, 2019.

CASSENS, J.; SCHMITT, F.; HERCZEG, M. Cake–distributed environments for context-aware systems. In: SPRINGER. **International Joint Conference on Ambient Intelligence**. [S.l.], 2013. p. 275–280.

CHAVARRIAGA, R. et al. The opportunity challenge: A benchmark database for on-body sensor-based activity recognition. **Pattern Recognition Letters**, Elsevier, v. 34, n. 15, p. 2033–2042, 2013.

CHEN, H.; FININ, T.; JOSHI, A. An ontology for context-aware pervasive computing environments. **The knowledge engineering review**, Cambridge University Press, v. 18, n. 3, p. 197–207, 2003.

CHEN, H. et al. Soupa: Standard ontology for ubiquitous and pervasive applications. In: IEEE. **The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.** [S.l.], 2004. p. 258–267.

DASARATHY, B. V. Sensor fusion potential exploitation-innovative architectures and illustrative applications. **Proceedings of the IEEE**, IEEE, v. 85, n. 1, p. 24–38, 1997.

DEMAZEAU, Y.; COSTA, A. R. Populations and organizations in open multi-agent systems. In: **Proceedings of the 1st National Symposium on Parallel and Distributed AI**. [S.l.: s.n.], 1996. p. 1–13.

DERRICK, N.; RENFA, L.; YONGHENG, W. Particle swarm optimization and dempster shafer approach to achieve internet of things context fusion using quality of context. **International Journal of Multimedia and Ubiquitous Engineering**, Global Vision Press, v. 11, n. 2, p. 247–264, 2016.

DEY, A. K.; ABOWD, G. D.; SALBER, D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. **Human–Computer Interaction**, Taylor & Francis, v. 16, n. 2-4, p. 97–166, 2001.

DEY, A. K.; MANKOFF, J. Designing mediation for context-aware applications. **ACM Transactions on Computer-Human Interaction (TOCHI)**, ACM New York, NY, USA, v. 12, n. 1, p. 53–80, 2005. <https://dl.acm.org/doi/abs/10.1145/1057237.1057241>.

DJAHEL, S. et al. A communications-oriented perspective on traffic management systems for smart cities: Challenges and innovative approaches. **IEEE Communications Surveys & Tutorials**, IEEE, v. 17, n. 1, p. 125–151, 2014.

EFSTRATIOU, C. et al. An architecture for the effective support of adaptive context-aware applications. In: SPRINGER. **International Conference on Mobile Data Management**. [S.l.], 2001. p. 15–26. <https://link.springer.com/chapter/10.1007/3-540-44498-X_2>.

FERREIRA, D.; KOSTAKOS, V.; DEY, A. K. Aware: mobile context instrumentation framework. **Frontiers in ICT**, Frontiers, v. 2, p. 6, 2015.

FLEISCHMANN, A. M. P. **Sensibilidade à situação em sistemas educacionais na web**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 7 2012.

FORTES, S. et al. The campus as a smart city: University of málaga environmental, learning, and research approaches. **Sensors**, MDPI, v. 19, n. 6, p. 1349, 2019.

GAIRE, R. et al. Internet of things (iot) and cloud computing enabled disaster management. In: **Handbook of Integration of Cloud Computing, Cyber Physical Systems and Internet of Things**. [S.l.]: Springer, 2020. p. 273–298.

GALACHE, J. A. et al. Clout: Leveraging cloud computing techniques for improving management of massive iot data. In: IEEE. **2014 IEEE 7th International Conference on Service-Oriented Computing and Applications**. [S.l.], 2014. p. 324–327.

GEIHS, K.; WAGNER, M. Context-awareness for self-adaptive applications in ubiquitous computing environments. In: SPRINGER. **International Conference on Context-Aware Systems and Applications**. [S.l.], 2012. p. 108–120.

GONZÁLEZ, D. P.; DÍAZ, R. D. et al. Public services provided with ict in the smart city environment: the case of spanish cities. Graz University of Technology-, 2015.

GUILLEMIN, P.; FRIESS, P. et al. Internet of things strategic research roadmap. **The Cluster of European Research Projects, Tech. Rep**, 2009.

GUNDERSEN, O. E. Situational awareness in context. In: SPRINGER. **International and Interdisciplinary Conference on Modeling and Using Context**. [S.l.], 2013. p. 274–287.

HAMMI, B. et al. Iot technologies for smart cities. **IET Networks**, IET, v. 7, n. 1, p. 1–13, 2017.

HEGDE, R. M.; KURNIAWAN, J.; RAO, B. D. On the design and prototype implementation of a multimodal situation aware system. **IEEE transactions on multimedia**, IEEE, v. 11, n. 4, p. 645–657, 2009.

HEIDARI, A.; NAVIMIPOUR, N. J. Service discovery mechanisms in cloud computing: a comprehensive and systematic literature review. **Kybernetes**, Emerald Publishing Limited, 2021.

HORLING, B.; LESSER, V. A survey of multi-agent organizational paradigms. **Knowledge Engineering Review**, Cambridge [England]; New York, NY: Cambridge University Press,[1984-, v. 19, n. 4, p. 281–316, 2004.

HUEBSCHER, M. C.; MCCANN, J. A. An adaptive middleware framework for context-aware applications. **Personal and Ubiquitous Computing**, Springer, v. 10, n. 1, p. 12–20, 2006.

HUSSAIN, A. et al. Health and emergency-care platform for the elderly and disabled people in the smart city. **Journal of Systems and Software**, Elsevier, v. 110, p. 253–263, 2015.

INDULSKA, J. et al. Scalable location management for context-aware systems. In: SPRINGER. **IFIP International Conference on Distributed Applications and Interoperable Systems**. [S.l.], 2003. p. 224–235.

INDULSKA, J. et al. Experiences in using cc/pp in context-aware systems. In: SPRINGER. **International Conference on Mobile Data Management**. [S.l.], 2003. p. 247–261.

JENNINGS, N. R. An agent-based approach for building complex software systems. **Communications of the ACM**, ACM New York, NY, USA, v. 44, n. 4, p. 35–41, 2001.

JOÃO, L. et al. Uma contribuição à ciência de contexto no middleware exehda explorando fog computing. In: SBC. **Anais do XLV Seminário Integrado de Software e Hardware**. [S.l.], 2018.

KHATOUN, R.; ZEADALLY, S. Smart cities: concepts, architectures, research opportunities. **Communications of the ACM**, ACM, v. 59, n. 8, p. 46–57, 2016.

KON, F.; SANTANA, E. F. Z. Cidades inteligentes: Conceitos, plataformas e desafios. In: **Congresso da Sociedade Brasileira de Computação**. [S.l.: s.n.], 2016. p. 2–49.

KUMARI, P.; KAUR, P. A survey of fault tolerance in cloud computing. **Journal of King Saud University-Computer and Information Sciences**, Elsevier, v. 33, n. 10, p. 1159–1176, 2021.

LAU, B. P. L. et al. A survey of data fusion in smart city applications. **Information Fusion**, Elsevier, v. 52, p. 357–374, 2019.

MACHADO, G. M. et al. A systematic mapping on adaptive recommender approaches for ubiquitous environments. **Computing**, Springer, v. 100, n. 2, p. 183–209, 2018.

MACIEL, C. et al. A multi-agent architecture to support ubiquitous applications in smart environments. In: **Agent Technology for Intelligent Mobile Services and Smart Societies**. [S.l.]: Springer, 2014. p. 106–116.

MALANDRINO, D. et al. Mimosa: context-aware adaptation for ubiquitous web access. **Personal and ubiquitous computing**, Springer, v. 14, n. 4, p. 301–320, 2010. <https://dl.acm.org/doi/abs/10.1007/s00779-009-0232-9>.

MANZOOR, A.; TRUONG, H.-L.; DUSTDAR, S. Using quality of context to resolve conflicts in context-aware systems. In: SPRINGER. **International Workshop on Quality of Context**. [S.l.], 2009. p. 144–155.

MANZOOR, A.; TRUONG, H.-L.; DUSTDAR, S. Quality of context: models and applications for context-aware systems in pervasive environments. **The Knowledge Engineering Review**, Cambridge University Press, v. 29, n. 2, p. 154–170, 2014.

MARTINI, B. G. et al. Indoorplant: A model for intelligent services in indoor agriculture based on context histories. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 21, n. 5, p. 1631, 2021.

MEHMOOD, Y. et al. Internet-of-things-based smart cities: Recent advances and challenges. **IEEE Communications Magazine**, IEEE, v. 55, n. 9, p. 16–24, 2017.

MOUSTAKA, V.; VAKALI, A.; ANTHOPOULOS, L. G. A systematic review for smart city data analytics. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 51, n. 5, p. 1–41, 2018.

NASCIMENTO, L. V. do et al. Context recognition and ubiquitous computing in smart cities: a systematic mapping. **Computing**, Springer, p. 1–25, 2021.

NASCIMENTO, L. V. do; OLIVEIRA, J. P. M. de. Decentralized, distributed and fault-tolerant context recognition architectures for smart cities: A systematic mapping. **SBC Reviews on Computer Science (ROCS)**, v. 1, n. 1, 2021.

NORVIG, P.; RUSSELL, S. Inteligência artificial. **Rio de Janeiro: Grupo GEN**, 2013.

NOY, N. F.; MCGUINNESS, D. L. et al. **Ontology development 101: A guide to creating your first ontology**. [S.l.]: Stanford knowledge systems laboratory technical report KSL-01-05 and . . . , 2001.

NUAIMI, E. A. et al. Applications of big data to smart cities. **Journal of Internet Services and Applications**, SpringerOpen, v. 6, n. 1, p. 1–15, 2015.

NWEKE, H. F. et al. Data fusion and multiple classifier systems for human activity detection and health monitoring: Review and open research directions. **Information Fusion**, Elsevier, v. 46, p. 147–170, 2019.

ORREGO, R. B. S.; BARBOSA, J. L. V. A model for resource management in smart cities based on crowdsourcing and gamification. **Journal of Universal Computer Science**, GRAZ UNIV TECHNOLGOY, INST INFORMATION SYSTEMS COMPUTER MEDIA-IICM . . . , v. 25, n. 8, p. 1018–1038, 2019.

PERERA, C. et al. Context aware computing for the internet of things: A survey. **IEEE Communications Surveys Tutorials**, v. 16, n. 1, p. 414–454, 2014. ISSN 1553-877X.

PERERA, C. et al. Sensing as a service model for smart cities supported by internet of things. **Transactions on emerging telecommunications technologies**, Wiley Online Library, v. 25, n. 1, p. 81–93, 2014.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. **Information and Software Technology**, Elsevier, v. 64, p. 1–18, 2015.

PICCOLO, P. et al. Service recommendation system for a smart city. **Cadernos de Informática**, v. 11, n. 1, p. 27–38, 2022.

PREUVENEERS, D. et al. Towards an extensible context ontology for ambient intelligence. In: SPRINGER. **European Symposium on Ambient Intelligence**. [S.l.], 2004. p. 148–159.

PU, F. et al. P2p-based semantic policy infrastucture for pervasive computing. In: IEEE. **2010 International Conference on Computational Intelligence and Security**. [S.l.], 2010. p. 541–545.

RAGHAVAN, S. et al. Data integration for smart cities: opportunities and challenges. **Computational Science and Technology: 6th ICCST 2019, Kota Kinabalu, Malaysia, 29-30 August 2019**, Springer, p. 393–403, 2020.

RANGANATHAN, A.; CAMPBELL, R. H. An infrastructure for context-awareness based on first order logic. **Personal and Ubiquitous Computing**, Springer, v. 7, n. 6, p. 353–364, 2003. <https://dl.acm.org/doi/abs/10.1007/s00779-003-0251-x>.

RAZZAQUE, M. A. et al. Middleware for internet of things: a survey. **IEEE Internet of things journal**, IEEE, v. 3, n. 1, p. 70–95, 2015.

RESNICK, P. et al. Grouplens: an open architecture for collaborative filtering of netnews. In: **Proceedings of the 1994 ACM conference on Computer supported cooperative work**. [S.l.: s.n.], 1994. p. 175–186.

RODA, C. et al. Past and future of software architectures for context-aware systems: A systematic mapping study. **Journal of Systems and Software**, Elsevier, v. 146, p. 310–355, 2018.

ROGGEN, D. et al. Collecting complex activity datasets in highly rich networked sensor environments. In: IEEE. **2010 Seventh international conference on networked sensing systems (INSS)**. [S.l.], 2010. p. 233–240.

ROSA, J. H. da; BARBOSA, J. L.; RIBEIRO, G. D. Oracon: An adaptive model for context prediction. **Expert Systems with Applications**, Elsevier, v. 45, p. 56–70, 2016.

RUSSELL, S.; NORVIG, P. **Artificial intelligence: A Modern Approach**. [S.l.]: Prentice-Hall, 1995.

RYU, H. et al. A task decomposition scheme for context aggregation in personal smart space. In: SPRINGER. **IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems**. [S.l.], 2007. p. 20–29. <https://link.springer.com/chapter/10.1007/978-3-540-75664-4_3>.

SABAGH, A. A. A.; AL-YASIRI, A. Gecaf: a framework for developing context-aware pervasive systems. **Computer Science-Research and Development**, Springer, v. 30, n. 1, p. 87–103, 2015.

SÁNCHEZ-CORCUERA, R. et al. Smart cities survey: Technologies, application domains and challenges for the cities of the future. **International Journal of Distributed Sensor Networks**, SAGE Publications Sage UK: London, England, v. 15, n. 6, p. 1550147719853984, 2019.

SANTOS, V. V. d. Cemantika: a domain-independent framework for designing context sensitive systems. Universidade Federal de Pernambuco, 2008.

SCHILIT, B.; ADAMS, N.; WANT, R. Context-aware computing applications. In: IEEE. **1994 first workshop on mobile computing systems and applications**. [S.l.], 1994. p. 85–90.

SEBBAK, F. et al. Context-aware ubiquitous framework services using jade-osgi integration framework. In: IEEE. **2010 International Conference on Machine and Web Intelligence**. [S.l.], 2010. p. 48–53.

STRANG, T.; LINNHOFF-POPIEN, C.; FRANK, K. Cool: A context ontology language to enable contextual interoperability. In: SPRINGER. **IFIP International Conference on Distributed Applications and Interoperable Systems**. [S.l.], 2003. p. 236–247.

STUDER, R.; BENJAMINS, V. R.; FENSEL, D. Knowledge engineering: principles and methods. **Data & knowledge engineering**, Elsevier, v. 25, n. 1-2, p. 161–197, 1998.

SYLLA, T. et al. Towards a context-aware security and privacy as a service in the internet of things. In: SPRINGER. **IFIP International Conference on Information Security Theory and Practice**. [S.l.], 2019. p. 240–252.

TALARI, S. et al. A review of smart cities based on the internet of things concept. **Energies**, Multidisciplinary Digital Publishing Institute, v. 10, n. 4, p. 421, 2017.

United Nations. **The World's Cities in 2018**. 2018. Available from Internet: <https://www.un.org/en/events/citiesday/assets/pdf/the_worlds_cities_in_2018_data_booklet.pdf>.

VAHDAT-NEJAD, H. Camid: architectural support of middleware for multiple-domain ubiquitous computing and iot. **The Journal of Supercomputing**, Springer, p. 1–18, 2022.

VAKALI, A.; ANTHOPOULOS, L.; KRCO, S. Smart cities data streams integration: experimenting with internet of things and social data flows. In: **Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)**. [S.l.: s.n.], 2014. p. 1–5.

VILLEGAS-CH, W.; PALACIOS-PACHECO, X.; LUJÁN-MORA, S. Application of a smart city model to a traditional university campus with a big data architecture: A sustainable smart campus. **Sustainability**, MDPI, v. 11, n. 10, p. 2857, 2019.

VILLEGAS, N. M.; MÜLLER, H. A. Managing dynamic context to optimize smart interactions and services. In: **The Smart Internet: Current Research and Future Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 289–318. ISBN 978-3-642-16599-3.

WANG, X. H. et al. Ontology based context modeling and reasoning using owl. In: IEEE. **IEEE annual conference on pervasive computing and communications workshops, 2004. Proceedings of the second**. [S.l.], 2004. p. 18–22.

WEI, X. et al. Uavfog-assisted data-driven disaster response: Architecture, use case, and challenges. In: SPRINGER. **International Conference on Web Information Systems Engineering**. [S.l.], 2020. p. 591–606.

WEISER, M. The computer for the 21 st century. **Scientific american**, JSTOR, v. 265, n. 3, p. 94–105, 1991.

WOOLDRIDGE, M. **An introduction to multiagent systems**. [S.l.]: John Wiley & Sons, 2009.

XU, N. et al. Cacont: A ontology-based model for context modeling and reasoning. In: TRANS TECH PUBL. **Applied Mechanics and Materials**. [S.l.], 2013. v. 347, p. 2304–2310.

YAMAMOTO, S. et al. Using materialized view as a service of scallop4sc for smart city application services. In: **Soft Computing in Big Data Processing**. [S.l.]: Springer, 2014. p. 51–60.

YÜRÜR, Ö. et al. Context-awareness for mobile sensing: A survey and future directions. **IEEE Communications Surveys & Tutorials**, IEEE, v. 18, n. 1, p. 68–93, 2016.

ZANELLA, A. et al. Internet of things for smart cities. **IEEE Internet of Things journal**, IEEE, v. 1, n. 1, p. 22–32, 2014.