UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

IGOR LADEIRA PEREIRA

Decentralized Broker for Context Management and Distribution using Unstructured P2P Networks in a Service-Oriented Architecture

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engineering

Advisor: Prof. Dr. Alberto Egon Schaeffer Filho
Coadvisor: M. Sc. Marcos Rates Crippa

Porto Alegre
January 2018

“It always seems impossible until it is done.”

— NELSON MANDELA

ACKNOWLEDGEMENTS

Many people took part in the tough journey of completing this course, particularly when writing this thesis. Therefore, I would like to express my gratitude and appreciation for them. Firstly, I would like to thank my family, specially my parents, for providing me education and endless support in all possible aspects and for understanding my absence in many moments during these years. Without you, none of my accomplishments would have been possible.

Secondly, I would like to thank UFRGS and the Informatics Institute, including all staff and professors. The experiences which I lived there made me grow and taught me a lot not only about academic knowledge, but life in a general way. Special considerations to my advisor Prof. Dr. Alberto Egon Schaeffer Filho for all the help and guidance provided and for extensively reviewing this work.

Next, I would like to thank TU Kaiserslautern for the unique exchange experience in which I had the pleasure to participate. Thanks to the International School for Graduate Studies (ISGS) for providing support to international students. A special acknowledgement to Prof. Dr. Hans Schotten and the Wireless Communication and Navigation (WICON) group, particularly to my co-advisor M.Sc. Marcos Rates Crippa, who gave me the inspiration and basis to write this work. Thank you for the opportunity to work in such an interesting project and for the help provided during its development.

Last but definitely not least, thanks to my friends and colleagues, the old and new. All of you had a special participation in my accomplishments and it would have been impossible to cope with the difficulties along the way without your help.

ABSTRACT

Context information is present in many application types nowadays. For instance, it may consist of the information or activity related to a user who is registered in a shopping website. With this, the website is able to suggest items based on previous purchases or even on the user country. Another example may correspond to data gathered by humidity sensors in a crop, providing an insight into irrigation problems. Context information is the basis for the interaction between users and computing systems and may comprise a significant amount of data. Thus, this demands the proposal of efficient mechanisms for distribution and management of information. This work proposes a decentralized platform for context management and distribution called Context Broker using unstructured peer-to-peer networks. The platform consists in a set of Brokers that act as context servers, providing storage and retrieval of data to consumers and providers. The design and implementation decisions were taken considering simplicity and performance goals. First, the theoretical foundation that substantiates the development of the platform will be presented. The main concepts related to context-aware systems will be indicated, as well as examples of related work. Next, a description and justification of the design decisions will be given, covering aspects such as the system architecture and the message protocol. Following that, implementation details will be presented and the experimental evaluation will be described. Finally, the results will be presented and discussed.

Keywords: Context. context-aware systems. context awareness. context management. unstructured P2P networks. context broker.

Broker Descentralizado para Gerenciamento e Distribuição de Contexto utilizando Redes P2P Não-Estruturadas em uma Arquitetura Orientada a Serviços

RESUMO

Informações de contexto estão presentes em muitos tipos de aplicação atualmente. Como exemplo, estas podem consistir nas informações ou na atividade relacionada a um usuário que está registado em um website de compras. Com isso, o website é capaz de sugerir items baseado em compras prévias ou até mesmo no país do usuário. Outro exemplo pode corresponder aos dados obtidos por sensores de umidade em uma plantação, fornecendo uma visão sobre possíveis problemas de irrigação. Informações de contexto são a base para a interação entre usuários e sistemas de computação e podem compreender um grande volume de dados. Dessa forma, isso demanda a proposta de mecanismos eficientes para a distribuição e gerenciamento dessas informações. Este trabalho propõe uma plataforma descentralizada para gerenciamento e distribuição de contexto denominada Context Broker utilizando redes peer-to-peer não-estruturadas. A plataforma consiste em um conjunto de Brokers que agem como servidores de contexto, fornecendo armazenamento e obtenção de dados para consumidores e provedores. As decisões de projeto e implementação foram tomadas considerando objetivos de performance e simplicidade. Primeiramente, a base teórica que fundamenta o desenvolvimento da plataforma será apresentada. Os principais conceitos relacionados a sistemas cientes de contexto serão indicados, bem como exemplos de trabalhos relacionados. Posteriormente, uma descrição e justificativas das decisões de projeto serão dadas, abrangendo aspectos tais como a arquitetura do sistema e o protocolo de mensagens. Seguindo, os detalhes da implementação serão apresentados e a avaliação experimental será descrita. Finalmente, os resultados serão apresentados e discutidos.

Palavras-chave: Contexto, sistemas conscientes de contexto, consciência de contexto, gerenciamento de contexto, redes P2P não-estruturadas, agente de contexto.

LIST OF ABBREVIATIONS AND ACRONYMS

GPS	Global Positioning System
IoT	Internet of Things
RDF	Resource Description Framework
OWL	Web Ontology Language
HTTP	Hypertext Transfer Protocol
XML	eXtensible Markup Language
DHT	Distributed Hash Table
CB	Context Broker
CC	Context Consumer
CP	Context Provider
CA	Controller Application
TTL	Time To Live
ACK	Acknowledgement
NACK	Not-Acknowledgement
NTP	Network Time Protocol
P2P	Peer-to-Peer
UDP	User Datagram Protocol
API	Application Programming Interface
CPU	Central Processing Unit
JVM	Java Virtual Machine
JIT	Just In Time
UML	Unified Modeling Language
DB	Distributed Broker

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

This chapter introduces this thesis. A decentralized system for context management and distribution which operates in a service-oriented architecture is proposed. Section 1.1 presents the motivation for this work. Section 1.2 describes the objectives that are expected to be reached. Next, the contributions of this work are indicated in Section 1.3. Lastly, Section 1.4 details how this work is organized.

1.1 Motivation

Technology progress along with the Internet has been changing the way people interact with each other and with computing systems. Low hardware cost, high computational power and increased sensing capability are some of the factors that explain the popularity that computing systems have reached (BROWN; BOVEY; CHEN, 1997). The presence of sensors in devices is an interesting feature, because it provides context-awareness capability and allows the implementation of applications for a great variety of purposes, improving the interaction with these devices (SCHILIT; ADAMS; WANT, 1994).

Let us consider a smartphone. With a GPS, which provides user location, one is able to develop an application that shows weather forecast or displays nearby places which the user might want to visit. The camera allows the user to take pictures, store and share them in social networks, for which an account with an e-mail address and a password is typically required. The user may also listen to music on the smartphone, thus making it possible for an application to send notifications about upcoming concerts based on the most played artists. All this information (location, photos, e-mail address and musical taste) compose a context related to the user.

Internet of Things (IoT) is a recent phenomenon which motivates the study of context awareness. The goal of IoT is to connect objects and enable communication between them by using the Internet and other mechanisms, thus providing the concept of ubiquitous computing and achieving context awareness (PATEL; CHAMPANERIA, 2016). For instance, in a Smart Home scenario, a smartphone may connect to the heating or air conditioning system, so that the working temperature is set according to the outside temperature. Another case would be a smartphone connected to a coffee machine, programming it to start a couple of minutes before the time which the user set his alarm clock

to.

This work was developed considering a project called HiFlecs (BOCKELMANN et al., 2017), (RADIO, 2015) as the background. Previous versions of the Broker platform for this project were developed in (CRIPPA, 2010) and (CRIPPA, 2013), but a decentralized version which had performance as a concern was needed. The platform proposed in this work addresses this issue.

1.2 Objectives

In order to develop context-aware applications, it is necessary to handle context information appropriately, in a way that it is reachable, up-to-date and can be obtained in a reasonable amount of time. The approach should also provide scalability, flexibility and fault tolerance. This thesis proposes the development of a decentralized platform called Context Broker to address this challenge, following the structure and results of (CRIPPA, 2010) and (CRIPPA, 2013). The objectives are to review previous research on context awareness and peer-to-peer networks, indicate the main design decisions and the architecture for the platform based on its requirements, describe the message protocol for interaction between the system components and the implementation of the prototype and present the results of the experimental evaluation along with analyses that validate (or invalidate) the proposed hypotheses.

1.3 Contributions

The work in (CRIPPA, 2010) proposes a centralized Context Broker platform, while (CRIPPA, 2013) presents a decentralized version using structured peer-to-peer networks. Together, these works address key issues related to context management and distribution, considering aspects such as scalability and fault tolerance. However, analyzing the design decisions and used technologies, it is reasonable to affirm that neither had a concern about performance.

The main contributions of this work are summarized below:

- An investigation and study on the state of the art in the corresponding area;
- The proposal of a service-oriented architecture that uses peer-to-peer networks for context management and distribution;

- An experimental analysis comparing different versions of the Context Broker platform.

1.4 Structure of the Text

This work is structured as follows: Chapter 2 presents the research on context-aware systems and peer-to-peer networks, fundamental concepts, key aspects and state of the art in the area. Chapter 3 presents the design and architecture of the Broker platform, including an usage scenario and the message protocol. Chapter 4 details the prototype implementation, describing aspects such as the development and testing environment, as well as system interfaces. This chapter also presents the performed tests and corresponding results, providing the information needed for the analyses. Finally, Chapter 5 concludes this thesis by recapitulating the main concepts from the theoretical foundation, giving an overview of the proposed Context Broker, highlighting the main achievements and indicating future work that can be done to extend and/or improve the platform.

2 THEORETICAL FOUNDATION

This work describes a platform for context management and distribution which uses unstructured peer-to-peer networks in a service-oriented architecture. Therefore, it is important to define and clarify fundamental concepts regarding context-aware systems and peer-to-peer networks, along with details of aspects related to these concepts, in order to provide a correct understanding of the work as a whole. This chapter aims to present these concepts and review related state of the art research on the area. The first three sections (2.1, 2.2 and 2.3) address the basic concepts of context, context-awareness and context management, respectively. Section 2.3 also introduces the Context Life Cycle, an important definition related to context management. Section 2.4 presents previous works in context-aware systems, providing insights into their architecture and operation. Finally, Section 2.5 focuses on peer-to-peer networks and describes three key aspects related to them: structure, search mechanisms and replication.

2.1 Context in Context-Aware Systems

Many definitions for context have already been proposed in the literature. The first work to address context and context awareness was (SCHILIT; THEIMER, 1994), stating that context corresponds to the user location, nearby people and objects, as well as changes to these over time. A similar definition was provided by (BROWN; BOVEY; CHEN, 1997): context corresponds to location, identities of the people around the user, the time of day, season, temperature, etc. Two other more general approaches were given by (DEY; ABOWD; WOOD, 1998) and (PASCOE, 1998). The former defines context as the user's physical, social, emotional or informational state, while the latter considers that context is the subset of physical and conceptual states of interest to a particular entity.

All the presented definitions give us a too specific idea about context. Depending on the scenario, concepts such as location, time of day or emotional state may not be sufficient to properly define context, because other types of information might be used. Therefore, a more general approach is needed in order to substantiate the relevance of this work. The definition which will be used was given by (ABOWD et al., 1999): "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves".

2.2 Context-Awareness in Context-Aware Systems

A definition for context awareness is also provided by (ABOWD et al., 1999) and will be adopted in this work: "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task". This definition is appropriate because it gives a broad and flexible idea, just like the definition for context. Furthermore, the authors of (ABOWD et al., 1999) presented a categorization of features for context-aware applications, combining the ideas of the taxonomies from (PASCOE, 1998) and (SCHILIT; ADAMS; WANT, 1994). There are three categories:

- **Presentation of information and services to a user**: corresponds to the capability of obtaining information and executing commands for the user manually, based on available context;
- **Automatic execution of a service**: relates to the ability of providing or changing the behavior of a service automatically, based on available context;
- **Tagging of context to information for later retrieval**: corresponds to the ability of relating context and data, in the sense that certain data is available for a user when he is in the associated context.

The definition of context awareness provides a way to determine whether an application is context-aware or not and this is useful when specifying the types of applications that need to be supported. The categorization of context-aware features brings us two main benefits. The first is the specification of types of applications that need support from the system. The second is that it shows the types of features that should be thought when building the context-aware applications (ABOWD et al., 1999).

2.3 Context Management in Context-Aware Systems

Although the definition of context-aware systems has already been given, it is important to indicate the purpose of this type of system in a more practical way. Context management is related to the tasks for which a context-aware system is responsible in order to handle context information (CRIPPA, 2010). A more formal definition was provided by (ZIMMERMANN; LORENZ; SPECHT, 2005) based on the authors' work experience in many application domains: context management corresponds to the creation

and administration of context-aware applications, considering related parameters and information sources with the goal of implementing certain behavior. Therefore, context management is the foundation for the development of context-aware applications.

One question that remains is what a context-aware system should be able to do or, in other words, what functionalities should be available. According to (KIANI et al., 2010), a context-aware communication system encompasses several context management functionalities, being two the most important ones: acquisition and provision of contextual information related to an entity. This description already gives us a brief idea about what an architecture of a context-aware system should be like. The authors go further by saying that it is possible to divide the system components involved in context management into either Context Consumers or Context Providers or a combination of these. Other functionalities of context-aware systems are merging correlated context data and locating and accessing context sources (CRIPPA, 2010). The platform considered in this work is based on the provider/consumer architecture outlined above.

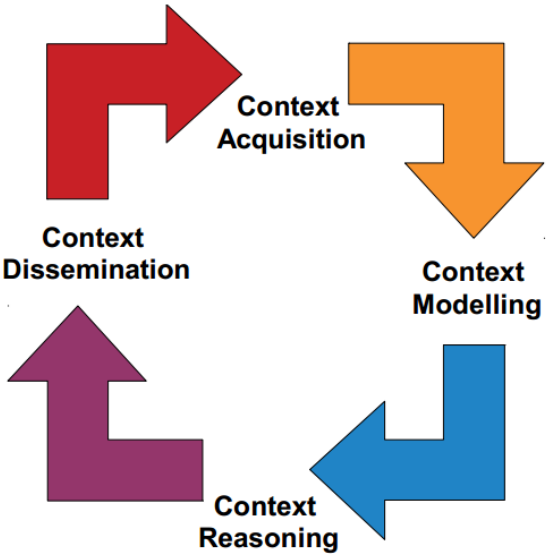
In order to manage context, three models were proposed by (WINOGRAD, 2001). These models are related to the system architecture and aim to coordinate multiple processes and components, a concept that is necessary to implement context-aware systems:

- **Widgets:** Context widgets work as an interface between the application and its operating environment. They hide the complexity of context acquisition and abstract context information to suit application needs (DEY; ABOWD; SALBER, 2001). They provide good efficiency but have the problem of not being robust to component failures;
- **Networked Services:** This model corresponds to a client-server architecture, in which clients look for services and need to establish a connection with them in order to use it (e.g., accessing a database). It is more flexible and robust than the previous due to the independence of the components, but it is also less efficient because a service discovery function is required;
- **Blackboards:** In this approach, applications post data to a shared message board and receive data through subscriptions, according to a specified pattern. In spite of providing loose coupling and robustness, communication efficiency is decreased because every message goes through a centralized server before reaching its final destination.

2.3.1 Context Life Cycle

The tasks related to context management give rise to the Context Life Cycle. According to (PERERA et al., 2014), this cycle shows how data moves from phase to phase in software systems, explaining where the data is generated and where it is consumed. The authors propose a simple Context Life Cycle (depicted in Figure 2.1) after the analysis of ten popular data life cycles.

Figure 2.1: The Context Life Cycle.



Source: (PERERA et al., 2014)

The proposed life cycle is composed by four phases. The first corresponds to the acquisition of context information through sensors. The second relates to the modeling of this information, which needs to be represented appropriately. The next phase is Context Reasoning, in which high-level context information is obtained from low-level sensor data. Lastly, the information is distributed to the consumers in the Context Dissemination phase (PERERA et al., 2014).

The three following subsections present details of important aspects related to context management.

2.3.1.1 Context Acquisition

The chosen method for context acquisition has an impact on the design of context-aware systems, therefore it is relevant to indicate some of the possible options. Three approaches were presented by (CHEN, 2010):

- **Direct access to hardware sensors:** Context data is obtained by directly accessing the physical sensors, providing a good knowledge about the data for the collecting applications. The problem comes when the number of context sources rises, requiring these applications to have the ability to communicate with many different sensors (CHEN, 2010);
- **Facilitated by a middleware infrastructure:** A middleware infrastructure is responsible for managing the low-level sensor data, thus allowing the applications to concern on how context will be used. This approach provides great extensibility and reusability of sensors (CRIPPA, 2010);
- **Acquire context from a context server:** Context data is gathered in a server which runs on a resourceful device and the context-aware applications make requests to this server. As in the middleware approach, great reusability is provided (CRIPPA, 2010).

2.3.1.2 Context Modeling

Context information is acquired by the use of sensors, which provide raw data. In order for a context-aware application to benefit from it, this information must be properly structured according to the application goals. This process is called context modeling (or context representation) and needs to support easy manipulation and extensibility, efficient search and scalability (CRIPPA, 2010).

When modeling context, there are requirements that need to be fulfilled so that the obtained representation can be meaningful for the context-aware system. A set of these requirements was presented in (PERERA et al., 2014) as heterogeneity and mobility, relationships and dependencies, timeliness, imperfection, reasoning, usability of modeling formalisms and efficient context provisioning. The six most popular context modeling techniques were indicated in (STRANG; LINNHOFF-POPIEN, 2004) and are described below:

- **Key-Value Models:** They are the simplest form of context representation. Context information is associated with a unique key and a matching algorithm is used for lookup. In spite of providing easy management, this approach is not scalable and lacks of structure for modeling complex information, thus jeopardizing efficiency of context retrieval;

- **Markup Scheme Models:** Context is modeled using a hierarchical structure of tags. This approach allows efficient data retrieval and provides support for validation, but has limited expressiveness. A popular example of markup language is XML, which is used in the Composite Capabilities/Preference Profile (DEY; ABOWD; SALBER, 2001) and User Agent Profile standards (WAP, 2001);
- **Graphical Models:** Provide more expressiveness in the final result because context is modeled with relationships. One example of this technique is the Unified Modeling Language (UML), which has a generic structure and enables good readability;
- **Object Oriented Models:** Provide encapsulation and reusability by modeling context with hierarchies and relationships (PERERA et al., 2014). Contextual information is accessed through specified interfaces and scalability can be achieved, but validation is difficult to be done;
- **Logic Based Models:** Context is represented as facts, expressions and rules and its management is realized based on these. This technique provides a high degree of formality and more expressiveness than the previous ones;
- **Ontology Based Models:** It is considered the most appropriate way of modeling and managing context. An ontology is an abstract model of a certain phenomenon and offers great flexibility and expressiveness, but it can also decrease the performance in context retrieval (PERERA et al., 2014). Resource Description Framework (RDF) and Web Ontology Language (OWL) are examples of languages used to describe ontologies.

2.3.1.3 Context Reasoning

Context Reasoning corresponds to the process of obtaining high-level context information from low-level raw sensor data, allowing the deduction of knowledge (PERERA et al., 2014). Some of the approaches for this task are described below:

- **Supervised Learning:** in this approach, training examples are collected and then labeled according to the expected results. After that, a function that can generate these results using the training data is derived. The advantages of this approach are accuracy and the existence of mathematical foundation. On the other hand, they require a significant amount of data and are usually more resource intensive (e.g. processing, storage). Examples of techniques in this category include *Decision Trees*, *Bayesian Networks* and *Artificial Neural Networks* (PERERA et al., 2014);

- **Rules:** it is the most popular method of context reasoning (PERERA et al., 2014). It consists in a set of rules in an *if-then-else* format which are used to generate high-level context information. This approach provides easy extensibility and is less resource intensive. However, it must be defined manually, raising the probability of error occurrence due to manual work (PERERA et al., 2014);
- **Probabilistic Logic:** in this approach, decisions are made according to probabilities attached to the facts in the situation which is being considered. It is commonly used to resolve conflicts in context information, allowing the understanding of occurrence of events. The advantages include handling uncertainty and the combination of evidence. On the other hand, it is only possible to reason numerical values and one should know the probabilities of the facts. Examples of techniques are *Dempster-Shafer* and *Hidden Markov Models* (PERERA et al., 2014).

2.3.1.4 Context Dissemination

Context Dissemination relates to how context is distributed in a context-aware system and is one of the most important functionalities in context management. The mechanism used to execute this distribution is influenced by the proposed system architecture and has an impact on the overall performance of the system. According to (PERERA et al., 2014), there are two methods for this task: the first one is the *query* method, in which the context-aware system receives a query from consumer applications and resolves this query to produce a result; the second one is the *subscription* or *publish/subscribe* method, in which consumers subscribe to a certain type of context information and then the system sends updates regarding this information periodically or when an event occurs.

2.4 Peer-to-Peer Networks

Distributed systems have many advantages over centralized systems, including scalability, fault tolerance and performance improvement. The most used architecture for distributed systems nowadays is called *peer-to-peer* (P2P) and provides a resilient solution for many applications. According to (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004), there are two main characteristics of this type of system. The first one is the sharing of computer resources by direct exchange, which reflects the nodes' capacity to independently execute tasks such as message routing and content location. The second

one regards the resilience aspect provided by the fault-tolerance and self-organization mechanisms implemented in these systems.

The definition of peer-to-peer systems adopted in this work is the one proposed by the authors of (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004): a peer-to-peer system is a distributed set of interconnected nodes which have capabilities such as self-organization and failure adaptation while still providing connectivity, with the goal of resource sharing. Content distribution is one of the categories of applications for which peer-to-peer architectures are used the most, thus making a perfect match with the Context Broker platform considered in this work. Peer-to-peer networks are usually called overlay networks because they are implemented on top of the underlying physical computer network (which is typically IP) (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004).

Peer-to-peer overlay network schemes can be categorized into two groups in terms of their structure: unstructured peer-to-peer networks and structured peer-to-peer networks. In this section, three main aspects will be considered: the structure of these networks, search mechanisms and replication.

2.4.1 Structure of Peer-to-Peer Networks

In **structured peer-to-peer networks**, nodes connect to each other according to a specific set of rules. Efficient routing of queries is achieved by using Distributed Hash Tables (DHTs), which are responsible for mapping the data objects to the peer nodes. In spite of providing scalability and efficient location of rare items (LUA et al., 2005), only exact-match queries are supported and a significant overhead is inserted due to the necessity of maintaining the network structure when nodes leave or join (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004), thus making this approach inappropriate if that happens very often. Examples of structured systems are Chord (STOICA et al., 2001) and Tapestry (ZHAO et al., 2006).

In an **unstructured peer-to-peer network**, the connections between nodes are made in a random way, implying that no specific structure is formed. This type of network is usually utilized when the nodes join and leave the network frequently (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004), since no information about the network is maintained and no restructuring is done. The disadvantage concerns the search for data. The location of content is completely independent of the content itself, therefore “brute-force” methods such as flooding the network with queries or more resource-preserving approaches such

as random walks must be used (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004). Examples of unstructured systems are Gnutella (GNU, 2017) and BitTorrent (BITTORRENT, 2017).

2.4.2 Search mechanisms

The mechanism used to find requested data and solve received queries has a huge impact on the performance of a context distribution system. Many techniques have already been proposed in the literature, but there is no perfect solution, since each one has its strengths and weaknesses. Therefore, a brief description of the main proposed approaches is given below, with the goal of selecting the one that best suits the needs of the Context Broker which this work refers to.

2.4.2.1 Search Mechanisms in Structured P2P Networks

Let us first consider the case of structured peer-to-peer networks. The systems in this category are also called *DHT-based* because query routing is executed through the use of Distributed Hash Tables, in which information about data is stored. Keys are generated for the objects and IDs are assigned to the peer nodes. Keys and IDs are both from a same identifier space, implying on a relationship between them. When an application wants to retrieve a data object with a corresponding *{key, value}* pair, the request is routed across the peers until the one that is responsible for storing that key is reached. Each peer has its own routing table containing a set of neighbor peers' IDs and addresses, so that it is able to know the node with the closest ID which the requests should be forwarded to, according to the key (LUA et al., 2005). On average, the most famous systems of this type behave similarly in terms of performance, which is $O(\log N)$ (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004).

This main idea is shared among the different existing DHT-based systems. However, each one has its own particularities regarding routing strategies and network architecture. For instance, Chord is organized in a uni-directional and circular *NodeID* space and uses matching of key and *NodeID* in the lookup protocol, while Pastry is structured in a mesh network (PLAXTON; RAJARAMAN; RICHA, 1997) (similar to a graph and can assume any network topology) and uses matching of key and a prefix in *NodeID* to search for data (LUA et al., 2005). On average, the most famous systems of this type behave sim-

ilarly in terms of performance, which is $O(\log N)$ (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004).

2.4.2.2 Search Mechanisms in Unstructured P2P Networks

Since there is no relationship between the location of files and the network topology in this category, a node that wants to find a file must query its neighbors. The most used technique for searching in unstructured peer-to-peer networks is flooding (GKANTSIDIS; MIHAIL; SABERI, 2005), which provides a good solution for a topology with few nodes. The problem of this technique is scalability. As the number of node increases, the time-to-leave (TTL) needed to reach data also increases, thus generating large loads on the nodes and jeopardizing performance (LV et al., 2002). Selecting the appropriate TTL is also a hard task. Besides, flooding implies on the creation of duplicate messages due to the fact that a node may receive the same query from more than one of its neighbors. Therefore, duplication detection mechanisms are required when using flooding (LV et al., 2002).

Gkantsidis *et al.* (GKANTSIDIS; MIHAIL; SABERI, 2005) propose normalized flooding as an alternative. In this technique, instead of forwarding a query to all neighbors, a node only forwards it to d_{min} neighbors, being d_{min} the minimum degree of the network (the minimum number of connections that a participating node has). It was observed that scalability is improved when using normalized flooding in topologies with high degree nodes. Another approach called *expanding ring* was proposed by (LV et al., 2002). It consists in the execution of successive floods with increasing TTLs. The first flood is done with a certain TTL and, if the search was not successful, the TTL is increased and another flood is executed. This approach showed good results in terms of message overhead and TTL when objects are replicated, if compared to regular flooding.

In an attempt to mitigate the problem of message duplication, a technique called *random walk* has been already proposed and tested in previous works, e.g., (GKANTSIDIS; MIHAIL; SABERI, 2005), (LV et al., 2002), (GKANTSIDIS; MIHAIL; SABERI, 2004). The technique consists in nodes forwarding the queries to a randomly chosen neighbor until the desired data is found. In (LV et al., 2002) the authors showed that the message overhead is decreased using random walks compared to expanding ring, but there is an increase in the delay perceived by the user. In order to reduce this delay, they propose the execution with k walkers, i.e., more than one copy of the query message is sent and each one does its own random walk. This change in the traditional approach

reduces the time needed to find objects, but also generates higher load in the network.

A modification of the random walk technique, called *random walk with lookahead*, was proposed by (GKANTSIDIS; MIHAIL; SABERI, 2005), in which a node executes short random walks with shallow floodings with a small TTL (typically 2). The results presented by the authors show that the number of unique nodes discovered when using random walk with lookahead is similar to the one obtained when using the traditional random walk. However, the response time is significantly smaller in the first method.

2.4.3 Replication

All the search mechanisms presented previously assume that some form of replication is implemented in the network. Content replication is of ultimate importance in peer-to-peer systems, because it increases content availability and provides better performance (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004). Three categories of replication approaches were proposed in (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004) and relate to both structured and unstructured peer-to-peer networks:

- **Passive Replication:** corresponds to the case in which a node requests an object and makes a copy of it when the request is attended;
- **Cache-Based Replication:** In this category, copies of the requested object are made by every node through which the query message passes;
- **Active Replication:** also called *proactive replication*. Nodes replicate or migrate content to others according to a certain policy, even if no request involving that content has been made;

Considering the subject of this work, this subsection will focus on replication in unstructured peer-to-peer networks. (LV et al., 2002) stated that, based on the research in (COHEN; SHENKER, 2002), the optimal method for replication is to replicate objects in a way such that $p \ \alpha \ \sqrt{q_r}$ (p is proportional to $\sqrt{q_r}$), being p the number of replicas of an object and q_r the query rate of this object. This scheme is called *square-root replication* and provides minimization of the overall search traffic (LV et al., 2002). In order to achieve this scheme, the authors have proposed two proactive replication strategies: path replication and random replication. The first one consists in storing a copy of the object in every node through which the query message passes after a successful query. In the second strategy, also considering a successful query, the number of nodes (p) between

the requester and the provider is counted and p nodes that were visited by the k walkers (copies of the query message) are selected to replicate the object (LV et al., 2002). Simulations show that path and random replication achieve results which are very close to the condition of *square-root replication*. Besides, it has been found that random replication performs better than path replication in terms of average number of messages per node (LV et al., 2002). Therefore, the random approach should be chosen, if the implementation is not excessively complex (LV et al., 2002).

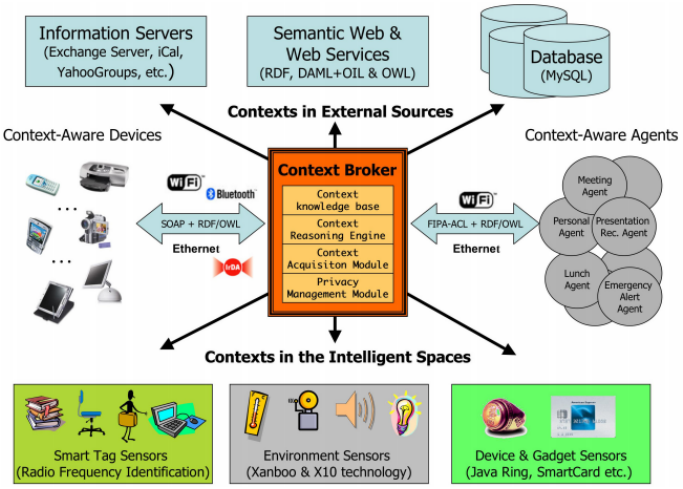
2.5 Related Work in Context-Aware Systems

The Context Broker platform proposed in this work is based on (CRIPPA, 2010) and (CRIPPA, 2013). In (CRIPPA, 2010), a centralized Broker was implemented, in which both methods mentioned in the last section (*query* and *subscription*) were used to distribute context in a straightforward way. In (CRIPPA, 2013), a distributed version of the Broker using structured P2P networks was proposed and this has imposed a bigger challenge regarding the context dissemination aspect. Many other approaches for implementing context-aware systems have already been proposed, each one with its advantages and disadvantages, so it becomes relevant to indicate some of the most significant ones.

CoBrA (CHEN; FININ; JOSHI, 2004) is an architecture for implementing smart spaces using context-aware systems. The architecture design is shown in Figure 2.2. The main component is a broker agent which consists of the following components: context knowledge base, context-reasoning engine, context-acquisition module and policy-management module. Context is modeled using Semantic Web languages such as RDF and OWL, providing a suitable representation for reasoning and knowledge sharing. The proposed architecture is centralized, but a group of brokers can work together through a broker federation (PERERA et al., 2014).

Another platform called Context Toolkit was proposed in (DEY; ABOWD; SALBER, 2001). It consists in a conceptual framework for the design of context-aware applications which is composed by three main entities: context widgets (provide access to context information), interpreters (produce high-level context information from low-level sensor data using reasoning techniques) and aggregators (responsible for gathering related context information in a common repository). To obtain context from the system, applications invoke services and use discoverers to find the components that are able to provide the desired data. The communication between all these components is implemented using

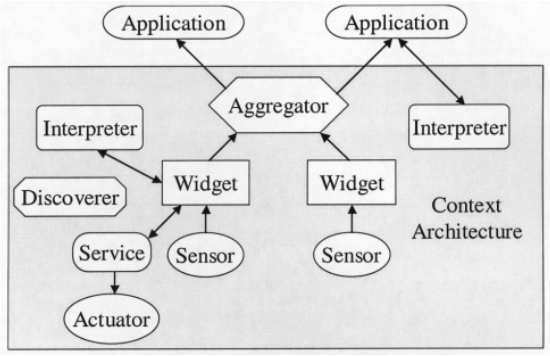
Figure 2.2: CoBrA architecture design.



Source: (CHEN; FININ; JOSHI, 2004)

a protocol based on HTTP and XML. Figure 2.3 shows one possible configuration for this system.

Figure 2.3: Example configuration of the Context Toolkit framework.

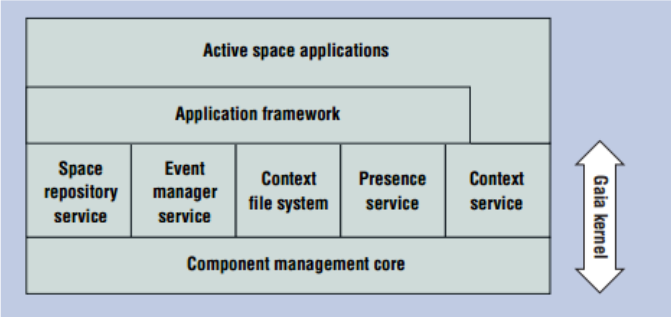


Source: (DEY; ABOWD; SALBER, 2001)

The Context Toolkit was an inspiration to another system called Gaia (ROMAN et al., 2002), whose architecture is shown in Figure 2.4. Gaia is a framework for the development of applications for active spaces, making an abstraction of them and their resources and also storing relevant information. It provides a set of services which support the management of these spaces and access to the resources: event manager, context service, presence service, space repository and context file system. These services enable the execution of functions such as events distribution (state change of applications, for instance), provision of context information, storage of information about the entities in the active space and building a virtual directory structure for the organization and management of data, thus facilitating developers' work.

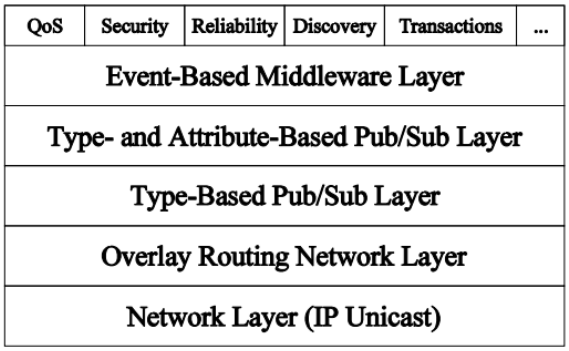
Hermes (PIETZUCH; BACON, 2002) is an event-based middleware architecture for the development of large-scale distributed applications. It is composed by a layered ar-

Figure 2.4: Gaia architecture.



Source: (ROMAN et al., 2002)

Figure 2.5: Hermes' layered architecture.



Source: (ROWSTRON; DRUSCHEL, 2001)

chitecture (shown in Figure 2.5) and utilizes a variation of the publish/subscribe model for the communication between its components, which is based on XML-defined messages. The system consists of two entities, event clients (can publish information or subscribe to receive information) and event brokers (responsible for serving the clients' requests and distributing information). The set of brokers form an overlay routing network (very similar to Pastry (ROWSTRON; DRUSCHEL, 2001)) in which peer-to-peer techniques are used to maintain the network's structure and message routing. Besides scalability, robustness is also reached through the fault tolerance mechanisms that are provided by this network.

3 DISTRIBUTED BROKER DESIGN AND ARCHITECTURE

This work is based on (CRIPPA, 2010) and (CRIPPA, 2013), therefore it inherits the basic characteristics and uses the concepts presented in these previous works. This chapter starts with the presentation of a usage scenario for the distributed Broker platform (Section 3.1), providing a basis for the requirements analysis in Section 3.2, which describes the functional and non-functional requirements. Then, the architecture of a centralized Broker platform is presented, along with key concepts that are also related to this distributed version (Section 3.3). Section 3.4 gives an overview of the distributed Broker design based on Chapter 2 and the requirements analysis. Finally, the Broker message protocol is described in terms of the operations supported by the platform (Section 3.5).

3.1 Usage Scenario

The design presented in this chapter was proposed considering a practical situation, in which a network would be composed by a significant amount of Brokers, probably 100 or more. We may consider the following scenario: the government of the state of Rio Grande do Sul wishes to prevent and monitor the occurrence of fire in vegetation areas. Temperature and humidity sensors would be spread and these would send their raw data to Context Providers, to be installed in measurement stations in these areas. The fire department of each city wishes to have an hourly report of these measures in the nearby vegetation areas, while the government wishes to have daily reports by city. The reports would be stored in a data center in Porto Alegre (the state capital). Let us assume that the appropriate structure for this system to work would be provided by the government.

Rio Grande do Sul has 500 cities and an area of 280.000 km^2, being approximately 40% covered just by native vegetation. One could think of a solution that consists of a single Broker, installed in Porto Alegre, receiving data and attending context requests (considering that every CP and CC is able to reach this Broker). We have a scenario in which a huge amount of sensors (thus, CPs) is sending data and many consumers are interested in receiving this data, even though the request rate is low. Hence, no centralized solution would be able to handle this situation. A more appropriate solution is to divide the load between a set of Brokers, allowing them to communicate and work together. Areas with more CPs/CCs could be served by two or more Brokers (one in each city, for instance), implying on lower response times and avoiding the overload of a single Broker.

3.2 Requirements Analysis

The requirements presented in this section are proposed based on (but not restricted to) the usage scenario of a large and spread sensor network in which context information (temperature, pressure, etc.) is gathered and a set of consumers are interested in receiving this data (as presented in the previous section). The payload in this case is simple and small and consists of measurements made by the sensors, but the number of exchanged messages may be high. Consumers nodes contact a network of Brokers, which may be around a small geographic area, in order to obtain data.

- Functional Requirements
  - *Discoverability*: there must be a way through which providers and consumers are able to find at least one Context Broker;
  - *Communicability*: Context Brokers should be able to communicate with each other in order to attend requests from consumers and providers (CRIPPA, 2013);
- Non-Functional Requirements
  - *Validity*: the content provided by the Brokers should be always up-to-date (CRIPPA, 2010);
  - *Availability*: the platform must implement some form of replication in order to provide appropriate availability of content;
  - *Consistency*: all the components of the architecture must represent context using the same model (CRIPPA, 2010);
  - *Neighborhood Awareness*: Brokers should be aware of nodes leaving and joining the network, avoiding the forwarding of requests to the ones which are not running anymore.

3.3 The Centralized Broker

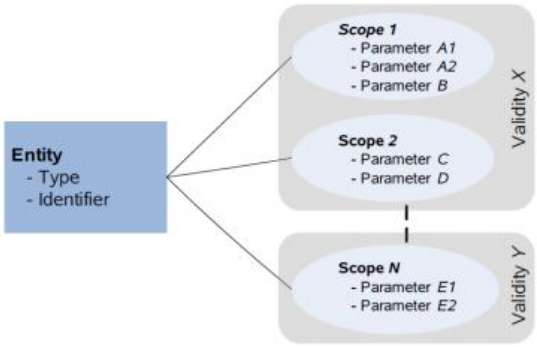
The work in (CRIPPA, 2010) proposes a centralized Broker and introduces some concepts used for this distributed version. This section reviews these concepts in order to provide a better understanding of the Context Broker platform.

3.3.1 Context Entity and Context Scope

A Context Entity (or simply entity) is a subject which context data corresponds to (CRIPPA, 2013). Entities are composed of a **type** and an **identifier**. This identifier is the information used to distinguish a set of entities of the same type. One example of entity is a student enrolled in a university. Every student has a unique registration number among all the other students. In this case, *registration number* is the type and, for instance, *014632* is the identifier.

A Context Scope (or simply scope) is a group of related parameters which belong to a certain context (CRIPPA, 2013). Let us consider the example of a user who wants to log in to his e-mail account. The user needs both his **username** and **password** in order to execute this operation. Therefore, these two parameters belong to the same scope and are always manipulated together. Another observation is that scopes have a validity period with start and end. After the validity expires, data related to that scope is considered invalid (CRIPPA, 2013). Each entity may be related to one or more scopes. The relationship between entity and scope is shown in Figure 3.1.

Figure 3.1: Entity-Scope relationship.



Source: (CRIPPA, 2013)

3.3.2 Centralized Broker Architecture

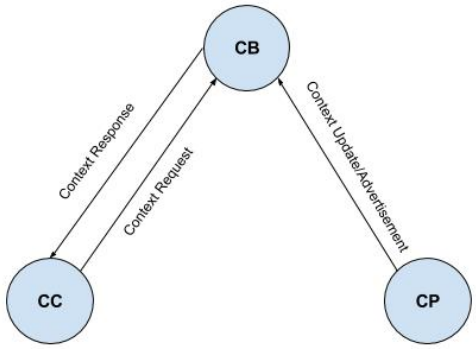
The Context Broker architecture consists of three components: Context Providers (CPs), Context Consumers (CCs) and the Context Broker (CB) itself. This division implements the idea behind a Service-Oriented Architecture (CRIPPA, 2013).

A **Context Provider** (or simply provider) is the component responsible for gathering data from sources (e.g., sensors) and sending it to the Broker, at a certain frequency, through a context update message (CRIPPA, 2010). Every provider must advertise its

presence to the Broker before sending any data. A **Context Consumer** (or simply consumer) is the component which retrieves context data. In this version of the platform, the only way for a consumer to obtain data is by making a context request to the Broker.

The **Context Broker** is the main component of the architecture. It is responsible for storing and retrieving context information based on the requests of providers and consumers, acting as the communication manager in the system (CRIPPA, 2013). Figure 3.2 shows the architecture of a centralized Broker with simplifications when compared to (CRIPPA, 2010). Acknowledgements (ACKs) and not-acknowledgements (NACKs) have been omitted.

Figure 3.2: Centralized Broker architecture.



Source: The authors

3.4 Overview of the Distributed Broker Design

This section indicates the decisions related to the main aspects of the distributed Broker design. These decisions were made based on the research presented in Chapter 2 and on the requirements analysis, considering that an unstructured P2P architecture was chosen.

3.4.1 Context Management in the Distributed Broker Platform

As presented in Chapter 2, context management is the basis of a context-aware system and gives rise to the four phases of the Context Life Cycle: context acquisition, context modeling, context reasoning and context dissemination/distribution. Each phase may be performed through a variety of methods and the choice of these has an impact on

the system design. A summary of the decisions for this work regarding these aspects is given below. The context reasoning phase was not addressed, considering the overhead and complexity that the addition of this mechanism would impose on the system.

- *Context Management* (Section 2.1.3): the chosen model for context management is **Networked Services**, considering that a Service-Oriented Architecture is followed, in which clients (consumers and providers) connect to Brokers in order to make requests;
- *Context Acquisition* (Section 2.1.3.2): context data is acquired from a **context server**. Brokers act as servers which store data and consumers obtain context data from them.
- *Context Modeling* (Section 2.1.3.3): the context modeling technique used in this work is the **key-value** approach, based on the desired performance and simplicity for this implementation of the Context Broker. The platform design was thought in the scenario of applications that deal with small and non-complex payloads, with information such as temperature and humidity being gathered from sensors. Therefore, the other approaches would provide too cumbersome solutions with an unnecessary overhead, compromising performance goals.
- *Context Distribution* (Section 2.1.3.4): context distribution is done through **queries**: consumers must query Brokers in order to obtain data.

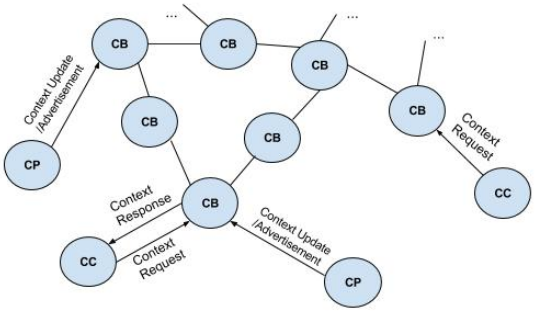
3.4.2 Architecture

The distributed Broker has the same components of the centralized Broker, with a change in how the architecture is structured. Considering a distributed architecture (such as the one proposed in this work), a set of Brokers works cooperatively in order to store and retrieve information. The architecture for the distributed Broker platform is shown in Figure 3.3. Acknowledgements (ACKs) and not-acknowledgements (NACKs) have been omitted.

3.4.3 Node Membership

A joining node receives a list of Broker addresses to connect. These Brokers are assumed to be the most stable and the list is updated when necessary (e.g., a Broker went

Figure 3.3: Distributed Broker architecture.



Source: The authors

offline). If a node is not able to connect to any Broker from this list, this node does not join the network and waits for a new list of addresses. The maximum number of established connections should be relatively small in an attempt to limit the overhead of messages due to nodes leaving the network.

In the case of a node leaving the network, two possibilities are considered:

- *Abrupt exit*: detected when there is no answer from a node when a ping message is sent. The node that detected the exit floods an advertisement message to its neighbors with a small TTL;
- *Normal exit*: a node sends a message stating that it will leave the network. This message is spread using flooding with a small TTL.

The flooding technique was chosen for simplicity reasons and because it shows good performance when the amount of nodes to be covered is small (GKANTSIDIS; MIHAIL; SABERI, 2005). In the situation of a node joining/leaving, only the nodes that are topologically close to the one that left need to know about the exit, in a way that future requests forwarded to nodes which are not online anymore can be appropriately detected. For the considered application scenarios, it is expected that not many nodes will join and leave the network often, so the message overhead is negligible.

3.4.4 Neighborhood Search

A node searches for neighbors based on a **greedy protocol** (WANG; VANNINEN, 2006). Joining nodes receive a list of Brokers and must ping these nodes to discover latencies. Three pings for each node are performed and a connection is established with the nodes that have the lowest average latencies. This greedy approach was chosen taking into consideration the good results it has shown in terms of performance and generated net-

work traffic, while avoiding a too cumbersome implementation in order to obtain network information (WANG; VANNINEN, 2006).

3.4.5 Search For Data and Data Replication

The search mechanism consists in **random walks with lookahead** (performing shallow floodings on each step of the walk). This decision was made considering that the traditional flooding approach would generate too much load in the network due to the amount of propagated messages. By choosing the random walk with lookahead method (Section 2.4.2.2), it is possible to minimize message duplication and the granularity of the coverage, properties that are important according to (LV et al., 2002). When a Broker receives a context request and does not have the desired data, it forwards the request to another Broker, performing a random walk with lookahead through the network.

According to the research presented in Chapter 2, the random replication mechanism performs better than path replication while achieving the *square-root* condition. Therefore, this approach will be used: for each successful search, the number of nodes p on the path between the CC and the Broker that has the requested data is counted and then m ($m \leq p$) nodes that were visited are randomly selected to replicate the object. The decision on whether certain data should be replicated or not is determined by a small time threshold: if the duration (validity period) of the data is lower than the threshold, no replication is performed (even if the Broker was selected to replicate). Replicas are stored in memory as long as a Broker is running and any update made to this replicated data will only be executed by a Broker if it is selected for replication once again, considering a successful query for the corresponding data.

One issue related to data validity and the system functionality in a general way is time synchronization in unstructured P2P networks. This issue is out of the scope of this work. It is assumed that the Brokers are connected to a reliable source of time information (synchronization by third-party), such as through the use of NTP.

3.5 Broker Message Protocol

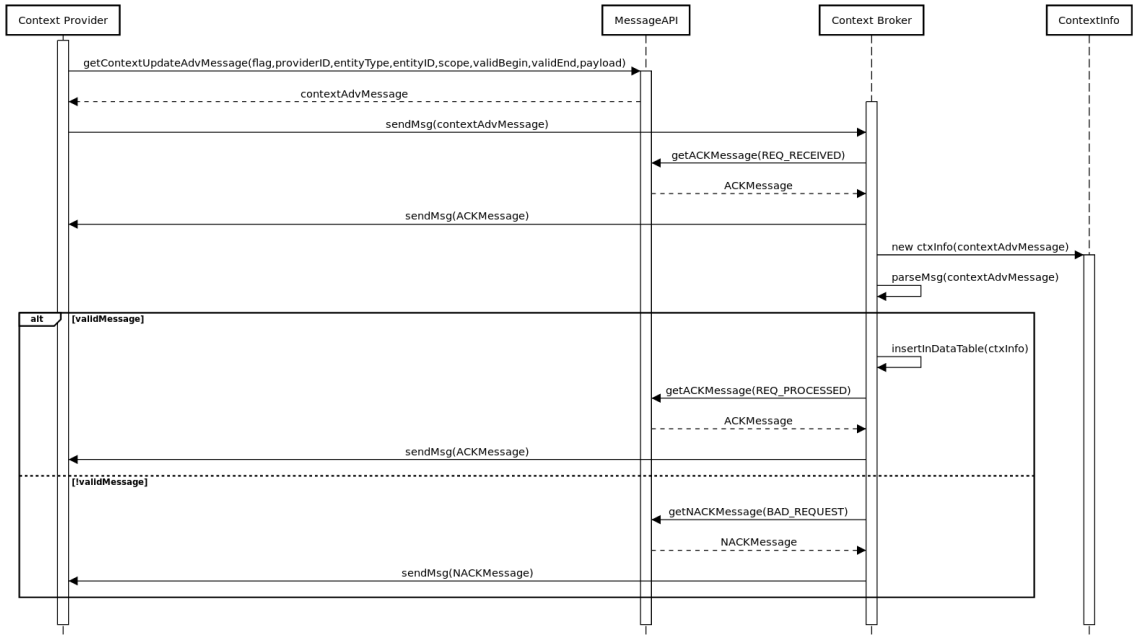
In order for the platform design to satisfy the requirements presented at the beginning of this chapter, a message protocol that provides the expected functionalities to the

corresponding system components must be implemented. For this distributed version of the Context Broker platform, a protocol composed of ten types of messages is proposed. These messages are used by the system components to perform operations. Aside from the three components mentioned previously (Context Broker (CB), Context Provider (CP) and Context Consumer (CC)), this protocol also considers a fourth component, the Controller Application (CA), which is used solely for managing the structure of the Brokers network (node membership). This section lists and describes the operations supported by the platform, allowing for the acquisition and provision of contextual information, the two key functionalities of a context-aware system (Section 2.1.3). A detailed specification of the protocol can be found in the Appendix, at the end of this document.

3.5.1 Context Update/Advertisement

The Context Update/Advertisement operation is performed by Context Providers (CP). A CP uses an Advertisement message to announce its presence and send its data to a Context Broker for the first time. The following messages sent by the CP will be Update Messages and it should keep sending these at a certain rate. The Broker which receives an Update message will either update existing data or add new data, depending on the fields specified in the message. Figure 3.4 shows the steps of the interaction between the system components when this operation is executed.

Figure 3.4: Context Update/Advertisement Operation



Source: The authors

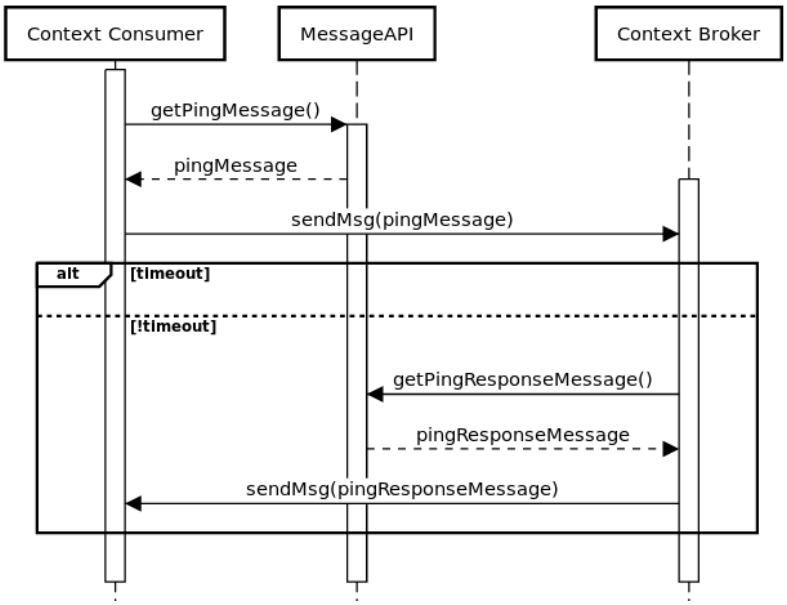
3.5.2 Context Request

The Context Request operation is performed by Context Consumers (CCs). A CC uses a Context Request message to request data to a Broker. If this Broker has the desired data, it will directly send it to the CC. If not, the request will be forwarded to other Brokers in the network until it is found or until the nodes determine that the data does not exist. When a Broker receives a Context Response from another Broker, the former checks whether it should replicate the data (details on this verification can be found in the Appendix). Figure 3.5 shows the steps of the interaction between the system components when this operation is executed.

3.5.3 Ping

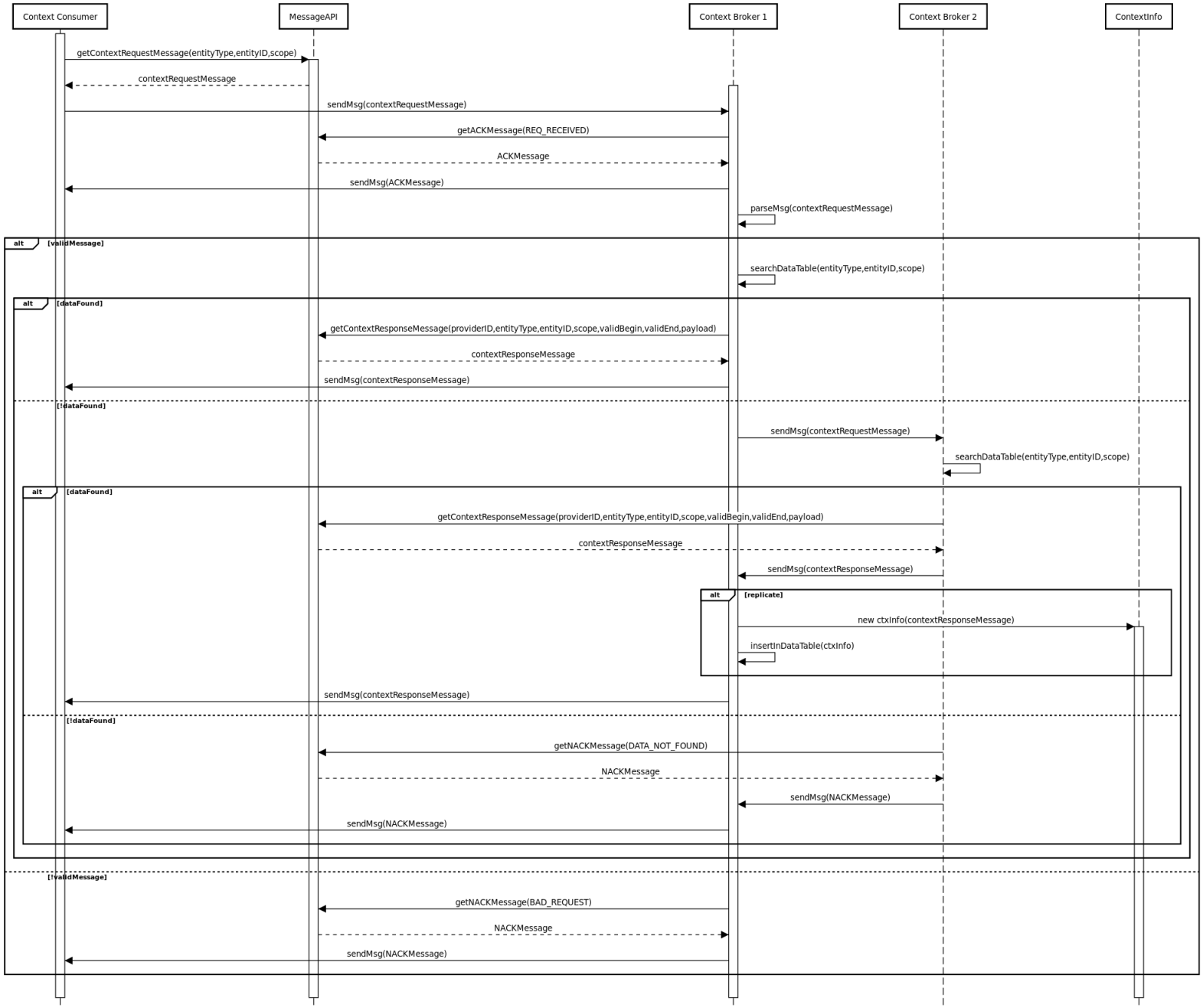
The Ping operation can be performed by any system component: Context Provider, Context Consumer, Context Broker and Controller Application. A Ping is executed to determine if a Broker is alive and able to receive requests. If no response is received within a timeout interval, the Broker is considered to be inactive by the entity who executed the Ping. Figure 3.6 shows the steps of the interaction between the system components when this operation is executed.

Figure 3.6: Ping Operation



Source: The authors

Figure 3.5: Context Request Operation

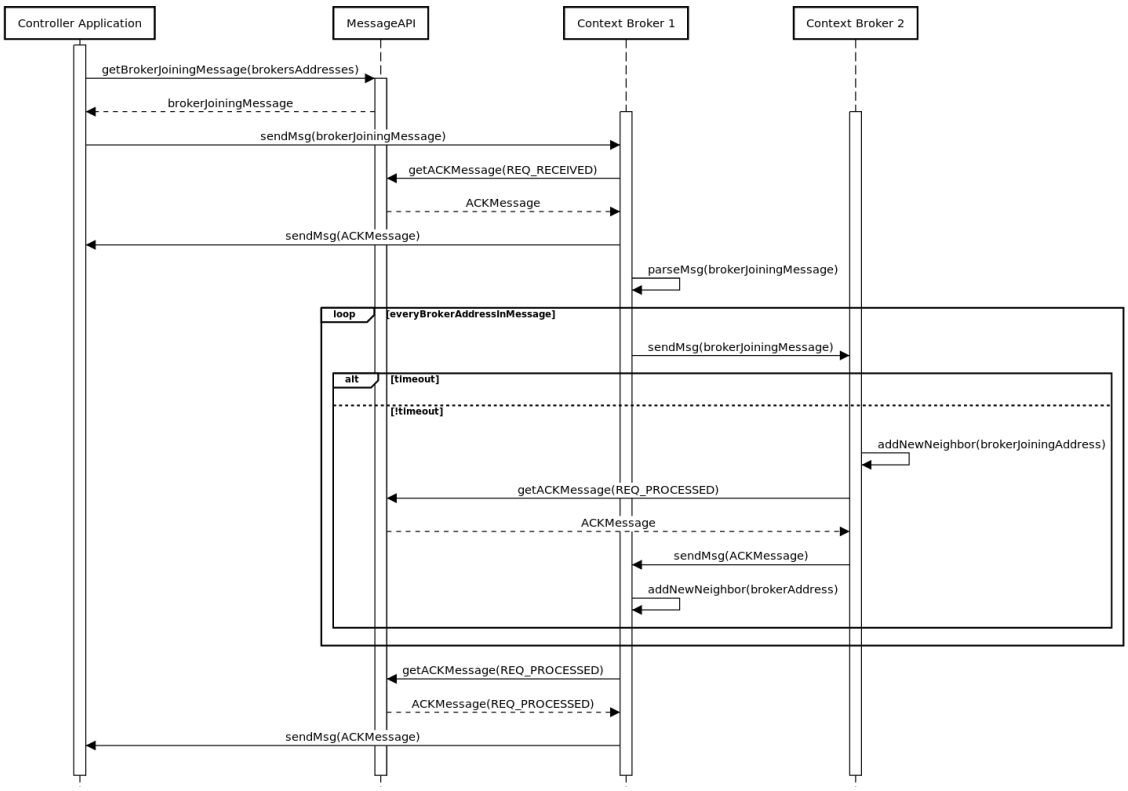


Source: The authors

3.5.4 Broker Joining

The Broker Joining operation is executed by a Controller Application (CA). A CA uses a BrokerJoining message to command a Broker to connect to one or more Brokers. With this, the new Broker joins the network, becoming a new neighbor of the specified nodes that are alive. Figure 3.7 shows the steps of the interaction between the system components when this operation is executed.

Figure 3.7: Broker Joining Operation

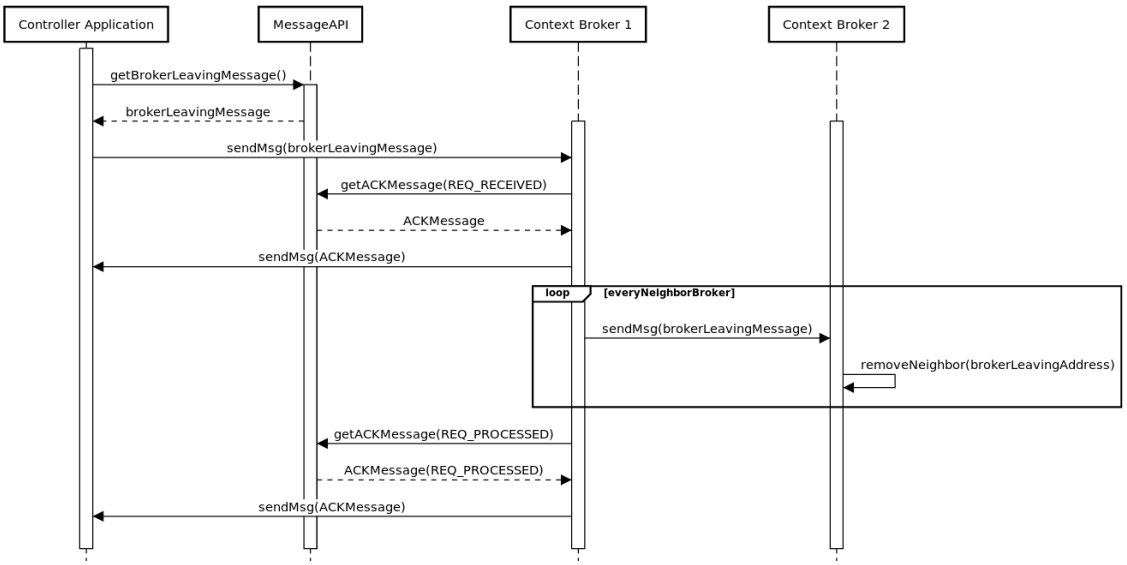


Source: The authors

3.5.5 Broker Leaving

The Broker Leaving operation is executed by a Controller Application (CA). A CA uses a BrokerLeaving Message to command a Broker to leave the network. This Broker sends a leaving advertisement to its neighbors, which remove its address from their neighbors list. Figure 3.8 shows the steps of the interaction between the system components when this operation is executed.

Figure 3.8: Broker Leaving Operation

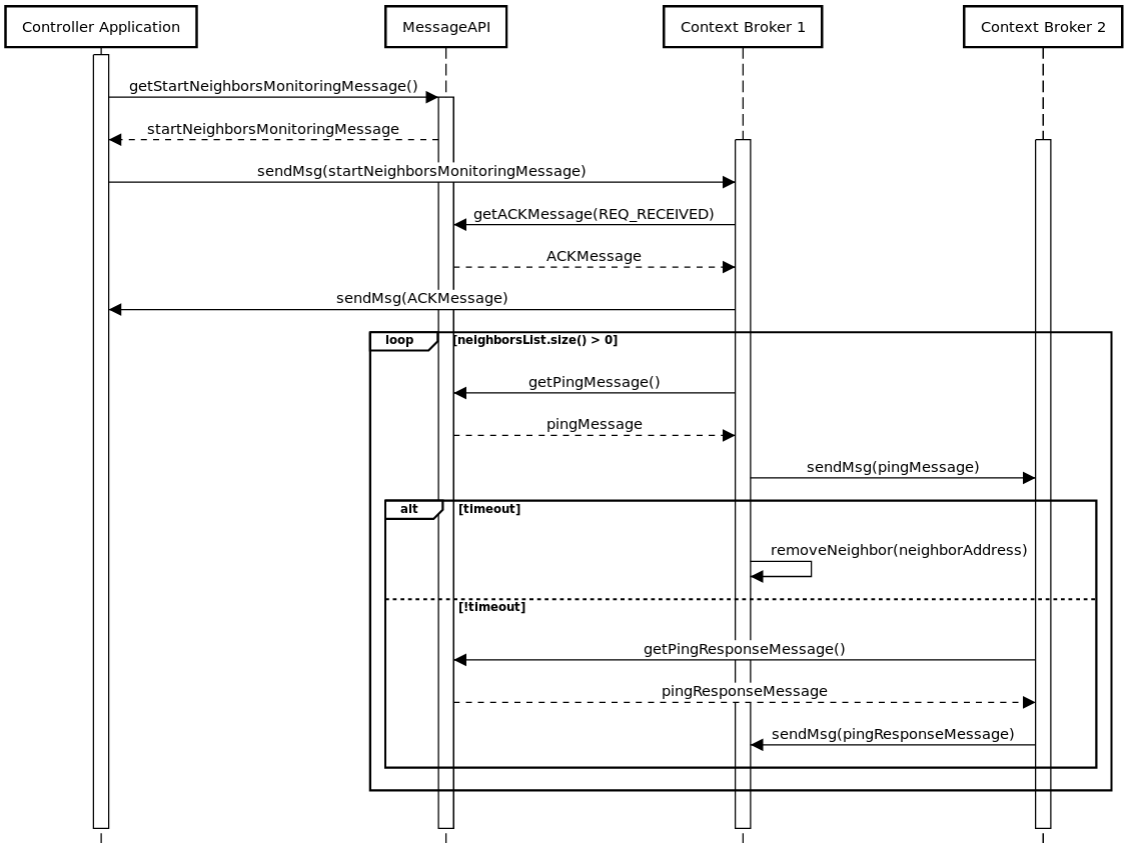


Source: The authors

3.5.6 Start Neighbors Monitoring

The Start Neighbors Monitoring operation is executed by a Controller Application (CA). A CA uses a StartNeighborsMonitoring message to indicate to a Broker that its neighbors are already set up and listening, implying that the monitoring to check their status should be started. Figure 3.9 shows the steps of the interaction between the system components when this operation is executed.

Figure 3.9: Start Neighbors Monitoring Operation



Source: The authors

4 SYSTEM IMPLEMENTATION AND EVALUATION

This chapter discusses the implementation of the Broker platform, the evaluation process and corresponding results. Section 4.1 describes the main aspects of the implemented prototype: the development environment, interfaces and main modules, as well as message parsing and data storage. Section 4.2 addresses the experimental evaluation, presenting the testing environment and configurations, analyzed metrics and chosen parameters. Finally, the results are presented in Section 4.3.

4.1 Broker Platform Prototype

This section describes the main aspects of the Broker platform implementation. Due to resource limitations for testing and the desired analyses, the implementation differs from the design presented previously in two aspects:

- *Neighborhood search*: the neighborhoods are determined manually. Each Broker will connect to the Brokers specified in the BrokerJoining Messages sent by the Controller Application. This approach was chosen because different network topologies were used for testing (more details in Section 4.2.3). Hence, a control over the topology was necessary and the neighborhood search based on latencies would hinder this task;
- *Search for data*: shallow floodings during the random walks were not implemented. These floodings would require that the Brokers kept a history of the received messages and a logic to avoid duplicates, implying on extra and unnecessary complexity in the system. Since the network used for testing was small, executing a flooding in each node would generate a significant amount of messages and most of them would be duplicates, which would be discarded. Thus, this process would imply on a waste of processing time in this simplified scenario.

These small changes have not invalidated the overall functionalities nor changed the main structure of the proposed system.

4.1.1 Development Environment

This platform was developed using the C++ programming language, Standard C++11. This choice was made considering that performance was a concern for this version. The communication between the platform components is done via UDP sockets and, for this, the sockets library *Practical C++ Sockets* (DONAHOO, 2017) was used. UDP was also used for performance reasons, but, in a real scenario, a software layer to handle packet losses would most likely be necessary. The timestamp in all messages is generated through the use of the <*chrono*> library and corresponds to the Unix (epoch) time in milliseconds. The random walk mechanism was implemented by randomly choosing a neighbor from the Broker's neighbors list and the generation of random indexes was done using the <*random*> C++ library[1]. For threads, the <*pthread*> library was used. For code editing, Sublime Text (SUBLIME, 2017) was used and GDB (GDB, 2017) for debugging. The development was done in Ubuntu LTS 14.04. Local tests were performed in an Intel® Core™ i5-2310 CPU @ 2.90GHz machine with 8 GB RAM.

Table 4.1: Broker Message API.

Function	Parameters	Description
getHeader	msgType, payloadSize, payloadExists	Provides the header for every message supported by the platform. This function is called by every other function of the API.
getStartNeighborsMonitoringMessage	-	Provides the message that commands a Broker to start checking if its neighbors are alive.
getPingMessage	-	Provides a ping message to check if a Broker is alive.
getContextUpdateAdvMessage	flag, providerID, entityType, entityID, scope, validBegin, validEnd, payload	Provides a Context Update/Advertisement message based on the fields passed as arguments.
getContextRequestMessage	entityType, entityID, scope	Provides a Context Request message based on the fields passed as arguments.
getContextResponseMessage	providerID, entityType, entityID, scope, validBegin, validEnd, payload	Provides a Context Response message based on the fields passed as arguments.
getACKMessage	operationName, requesterAddress, operationStatus	Provides an ACK message based on the fields passed as arguments.
getNACKMessage	operationName, requesterAddress, operationStatus	Provides a NACK message based on the fields passed as arguments.
getBrokerLeavingMessage	-	Provides the message that commands a Broker to leave a network.
getBrokerJoiningMessage	-	Provides the message that commands a Broker to join a network.
getPingResponseMessage	-	Provides the response to a ping message.

Source: The authors

4.1.2 Broker Platform Interfaces

In this implementation of the Broker platform, four components are considered: Context Broker (CB), Context Provider (CP), Context Consumer (CC) and Controller

[1]The chosen engine for the generation of pseudo-random numbers implements the Mersenne Twister algorithm, proposed in (MATSUMOTO; NISHIMURA, 1998). The chosen distribution for this process was the uniform distribution, so that every generated index has equal probability of occurrence.

Application (CA). The first three are the key system components, while the last one simply has the role of performing the exit and joining of Brokers in the network, since these are always in a passive state waiting for requests. Figure 4.1 shows the structure of the system in a component level. Every operation allowed by the platform is accessed through a message API, which generates the messages in their appropriate formats, according to the parameters (when required) provided by the requester application. The API is shown in Table 4.1. The interface *IBrokerResponse* is provided by the components in order for the Broker to deliver the response of the operations (namely, ACKs, NACKs or context data).

The main communication between Brokers in a network is done by the modules *ServeCtxRequest* and *CtxForwarding* through the *IForwarding* interface, which is provided by every Broker. The *ServeCtxRequest* module is responsible for handling a Context Request. It accesses the Broker data table, forwards the request to another node if the data was not found, determines in which Brokers the data replication will happen and also starts the forwarding of data when it is found. The *CtxForwarding* module is responsible for handling a Context Response message which is being forwarded through the network. It performs the data replication (if the current Broker is supposed to) and sends the message to the next Broker, according to the addresses and ports that have been appended along the request path. The modules *ServeBrokerLeaving* and *ServeBrokerJoining* are also used for communication between Brokers, but they only concern the advertisement of nodes leaving or joining the network.

Table 4.2: Description of the *requestServiceInfo* structure.

Structure field	Description
dTable	Object that represents the structure in which a Broker stores data (details in Section 4.1.6).
receivedMsg	Message containing the request received by a Broker.
semDataTableMutex	Mutex structure to avoid race conditions between modules when accessing the data table.
semLoggerMutex	Mutex structure to avoid race conditions between modules when accessing the log file.
semNeighborsMutex	Mutex structure to avoid race conditions between modules when accessing the neighbors list.
lger	Object that represents the log file used by the Broker.
srcAddr	Address of the entity which sent the message.
srcPort	Port of the entity which sent the message.
neighbors	List of neighbors addresses.

Source: The authors

The Context Broker is composed of a main module *WaitForRequest* (which waits for request messages) and the modules that attend the supported requests. The interaction between this main module and the others is done by using a data structure called *requestServiceInfo* that contains all the information and variables required for processing

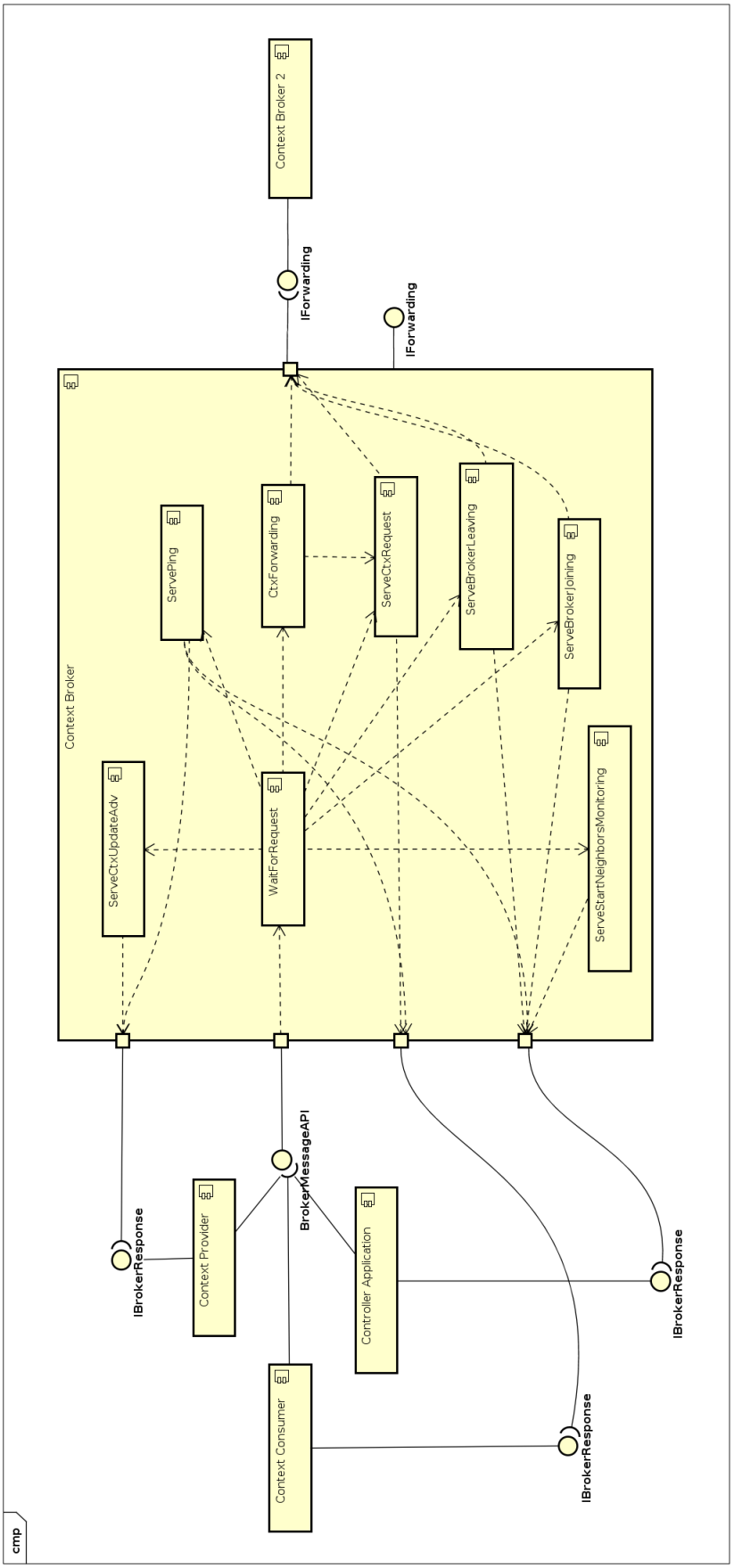


Figure 4.1: Context Broker Component Structure.

Source: The authors

the request, such as the message received, mutexes, requester address, etc. This structure is shown in Table 4.2.

4.1.3 Message Parsing

One of the main activities performed by the Broker is message parsing. It is fundamental for the realization of the platform's core functionalities: storing and retrieving context data. The parsing function (shown below along with the parameters description) transforms the input string into a tokens vector, so that each field of the message can be analyzed and/or stored in appropriate structures for processing. The character used as the field delimiter in all messages is the vertical bar "|".

Listing 4.1: Message Parsing Function.

```
void Tokenize(const string& str,vector<string>& tokens, int& ↩
    nTokens, const string& delimiters) {

    // Skip delimiters at beginning.
    string::size_type lastPos = str.find_first_not_of(delimiters, 0);
    // Find first "non-delimiter".
    string::size_type pos = str.find_first_of(delimiters, lastPos);

    nTokens = 0;

    while (string::npos != pos || string::npos != lastPos) {
        // Found a token, add it to the vector.
        tokens.push_back(str.substr(lastPos, pos - lastPos));
        // Skip delimiters.  Note the "not_of"
        lastPos = str.find_first_not_of(delimiters, pos);
        // Find next "non-delimiter"
        pos = str.find_first_of(delimiters, lastPos);

        nTokens++;
    }
}
```

Parameters:

- *str* (input): string with the message to be parsed

- *tokens* (output): vector in which the message each message field will be stored
- *nTokens* (output): number of tokens (fields) in the message
- *delimiters* (input): characters in the message to be considered as field delimiters

4.1.4 Data Storage and Search

A Context Broker stores data in its data table, implemented by the *dataTable* class. The key element of this class is the *dataMap*, consisting of a C++ *map* structure composed of <*key,value*> pairs. For the considered platform, the keys are strings generated using a hash function and the values correspond to *contextInfo* objects, whose class is shown in Table 4.3 (methods have been omitted for clarity). When a Context Provider sends an Advertisement message to a Broker, the data storage process (implemented by the *insert* method of the *dataTable* class) works as follows: the fields *providerID*, *entityType*, *entityID* and *scope* are concatenated. The resulting string is then passed to a function that generates a hash key using the DJB hashing algorithm. After that, the Broker inserts the pair <*hashKey, contextInfo*> in its *dataMap*.

The context retrieval process is performed by the *search* method. It gets the fields *entityType*, *entityID* and scope from the Context Request message and searches linearly the Broker's *dataMap* for the *contextInfo* entry which has the matching fields.

Table 4.3: Class used to represent context information.

Class attribute	Type
ctxMsgType	char
providerID	string
entityType	string
entityID	string
scope	string
validBegin	string
validEnd	string
payload	string
msgWellFormed	bool
entryID	string

Source: The authors

4.2 Experimental Evaluation

The main goal of this work with respect to evaluation is to show the advantages of a distributed implementation over a centralized version of the platform. Besides avoiding the single point of failure problem in the latter, a distributed version should provide advantage regarding at least two aspects: **response time** and **memory usage**. As the number of Context Consumers rises, a single Broker is expected to take longer to answer requests, because it is the only source of data available and needs to serve one request at a time. As the number of Context Providers is increased, a single Broker is also expected to consume more memory, since more data is being stored and more advertisements and updates must be served. Hence, by distributing the load between more Brokers, on average, lower response times and memory usage should be observed. This section presents the description of the experimental evaluation, including the testing environment, parameters and metrics evaluated.

4.2.1 Testing Environment

Gathering all the resources needed to represent the scenario described in Chapter 3 would be a cumbersome task, mainly due to the amount of machines required to setup a network with a significant amount of Brokers. In an attempt to overcome this problem and use an environment closer to a real scenario, Amazon cloud services were used for testing. Amazon EC2 (AMAZON, 2017) is a web service that provides compute capacity in the cloud, allowing the creation of remote instances (machines) with a wide range of configurations regarding memory, CPU and operating system. For this work, 20 instances were used, all of them with the following configuration:

- Physical Processor: Intel Xeon Family (exact model is not specified), 64-bit or 32-bit architecture (not specified), 2.5 GHz;
- Memory: 1 GB RAM;
- Operating System: Ubuntu Server 16.04 LTS;
- Instances location: US-East;

From the 20[2] machines used, 10[3] were chosen to run Brokers, 5 to run Context

[2] 20 machines is the limit allowed by Amazon in the Free Tier.

[3] Since in (CRIPPA, 2013) 4 machines were used to run Brokers, it was expected that, by using more

Providers and 5 to run Context Consumers. The reason for dividing CPs and CCs into multiple machines is to implement a more realistic scenario (such as the one described in Chapter 3), in which there will be more than a single source of context data and more than a single source of context requests. It is important to mention that the machines which run CPs (or CCs) may run more than one CP (or CC), but the machines running Brokers execute only one instance of the Broker application. In the considered setup, all CPs send the same data (context advertisement followed by context updates) and all CCs ask for the same data. Each machine running CPs and each machine running CCs was randomly assigned to a Broker, so that the load is distributed among the network. This assignment was done manually in order to have a better control of the resulting testing setup: at most two Brokers serve CCs and CPs simultaneously and no Broker serves more than one machine running CCs (or CPs). Each test was performed with a duration of 15 minutes and a delay of 200 milliseconds was introduced between the start of every CP and CC, so that CPs and CCs are not started all at the same time (as in a more realistic scenario).

4.2.2 Evaluation Metrics and Testing Parameters

This work focuses on evaluating the Broker platform with respect to two metrics: **average response time** and **average memory usage**. As mentioned at the beginning of this chapter, it is expected that these metrics reflect the impact when a transition from a centralized Broker to a distributed Broker is done. Aside from defining the evaluation metrics, it is also important to define which parameters have an influence on these metrics, so that the test configurations can be established. The first two parameters that might be considered are the number of CCs and CPs. Since the Broker serves both of these entities, it is reasonable to expect that these parameters will have an impact on the chosen metrics. Related to CCs and CPs, two other parameters which might have an influence are the context update interval and the context request interval. The former corresponds to the frequency at which CPs send Context Update messages to the Broker, while the latter corresponds to the frequency at which CCs send Context Request messages to the Broker.

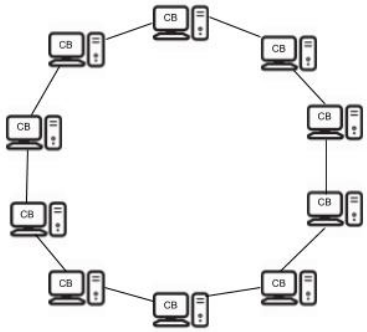
This work considers a distributed version of the Broker platform, in which a set of Brokers work together to store and provide context information. Although no connection is actually established (since UDP is used), the communication between Brokers forms a

machines, the advantages of a distributed version would be better observed.

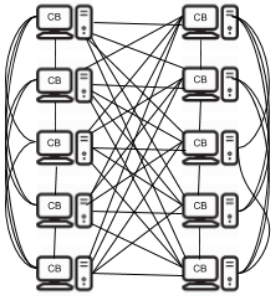
network (practically, a graph), which may have many different topologies depending on how the neighborhood of each Broker is set. In order to analyze the impact of the topology on the previously mentioned metrics, three simple topologies[4] (shown in Figure 4.2) were chosen for testing:

- Ring topology: Each node connects to two other nodes in a ring fashion;
- Full mesh topology: Each node connects to every other node;
- Moderate connectivity topology: A random topology was generated using the Waxman algorithm (WAXMAN, 1988). The topology's average connectivity is 2.

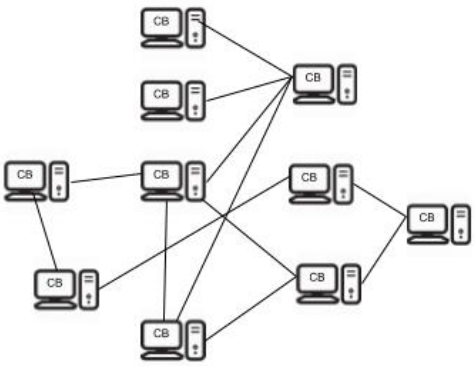
Figure 4.2: Topologies chosen for testing.



(a) Ring Topology



(b) Full Mesh Topology



(c) Moderate Connectivity Topology

Source: The authors

Since many test configurations are already possible, three other relevant parameters were fixed: payload size (fixed at 1 kB), number of Brokers (fixed at 10) and replication index (fixed at 1). The payload size corresponds to the amount of actual context data contained in a Context Request, Update/Advertisement or Context Response message. The replication index determines in how many Brokers context data should be replicated

[4]These topologies were chosen based on the idea of testing an "extreme" case (full mesh), an "average" case (moderate connectivity) and a topology that is often discussed and used in the field of computer networks (ring).

in case of a successful query (when the requested data was found). Considering that half of the used Brokers will serve CPs and that stressing the system capacity of finding rare data was not the goal of this work, this parameter was fixed at 1. Table 4.4 summarizes the parameters chosen to be part of the evaluation process and the values considered for each one of them.

Table 4.4: Evaluation parameters and corresponding values.

Parameter	Value
Number of Brokers (fixed)	10
Payload size (fixed)	1 kB
Replication index (fixed)	1
Network topology	ring, full mesh, moderate connectivity
Number of Context Providers	50, 500, 1000[5]
Number of Context Consumers	50, 500, 1000[5]
Context update interval	1, 30, 60 seconds[6]
Context request interval	1, 30, 60 seconds[6]

Source: The authors

4.3 Results

This section presents the results of the experimental evaluation done on the proposed distributed Broker platform, along with comparison analyses. Before presenting the results, some considerations should be made. Due to the significant amount of testing configurations executed, the results presented in this section are a subset of the total (the complete table with the results of all configurations can be found in the appendix). The idea was to select the parameters to be fixed, vary the others and use the remaining configurations to make comparisons between four different versions of the Broker platform:

- Version V1: centralized Broker, implemented in Java, proposed in (CRIPPA, 2010);

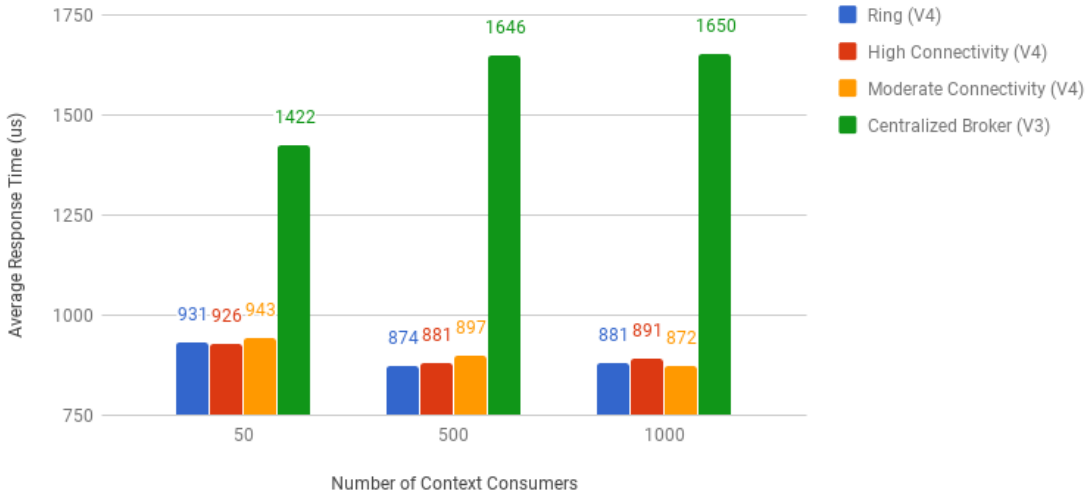
[5]These values were chosen based on previous studies in (CRIPPA, 2013). Since more resources were available for this work, the idea was to explore them and increase the amount of CPs, CCs and Brokers, in order to check the possible effects on the results.

[6]These values were chosen based on previous studies in (CRIPPA, 2013). The idea was to pick a range of values that would encompass small and big intervals considering application scenarios such as the one presented in Chapter 3.

- Version V2: distributed Broker based on structured P2P networks, implemented in Java, proposed in (CRIPPA, 2013);
- Version V3: centralized Broker, implemented in C++. This version corresponds to the same as proposed in this work, but using only one Broker;
- Version V4: distributed Broker based on unstructured P2P networks, implemented in C++, proposed in this work;

According to the results table presented in the appendix, the performed tests showed no major difference in the metrics when the **update interval** was varied, so the average of the response times and memory usage for 1, 30 and 60 seconds was taken. A significant impact on the results was observed when the **request interval** was varied from 1 to 30 seconds or from 1 to 60 seconds, but not from 30 to 60 seconds. Hence, the values of response times and memory usage for a request interval of 30 seconds were chosen. Considering the scenario presented in Section 3.1 and most of the applications in which the Broker platform would be utilized, a valid assumption is that the number of Context Consumers will be always greater than (or at least equal to) the number of Context Providers. Thus, when varying the amount of CCs, the CPs are fixed at 50 and, when varying the amount of CPs, the CCs are fixed at 1000.

Figure 4.3: V3 and V4 - #CCs x Average Response Time.

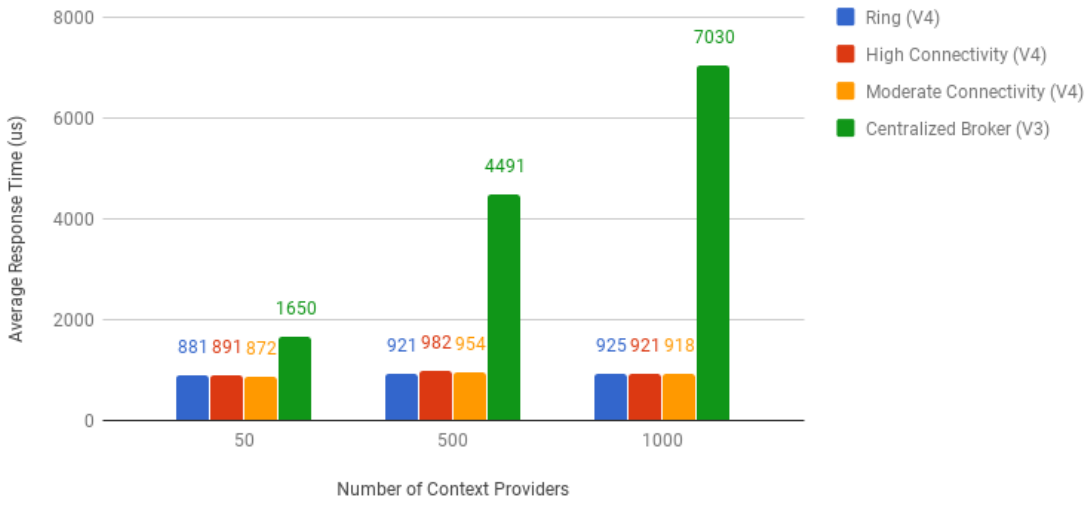


Source: The authors

4.3.1 Average Response Time Analysis - V3 and V4

Figure 4.3 shows the comparison between the centralized Broker (V3) and the distributed (V4), regarding the influence of the number of CCs on the average response time. The behavior of the centralized version confirms the hypothesis stated at the beginning of Section 4.2: the average response time grows as the number of CCs is increased. The distributed Broker showed a decrease in the response time from 50 to 500 Consumers for all 3 topologies. One hypothesis which might explain this is the following: since the delay between the start of each CC is very small (200 ms), the Brokers get overwhelmed with requests for an interval when the CCs are being started, leading to higher response times. Threads along with the structures needed for processing the requests are being created, accesses to the data table are being made and requests are also being forwarded to and received from other Brokers. All this is happening in a very short interval and for the first time, when there is no data in the machines' cache. Moreover, if a Broker is also attending CPs, it has to deal with requests from both entities as soon as the CCs are launched. For small amounts of CCs, it is valid to affirm that the delay in this initial period will dominate the average response time, since less values are being considered in the analysis. However, as the number of CCs grows, this effect should be amortized. Figure 4.3 confirms this hypothesis. The ring and full mesh topologies present lower average response time for 500 CCs and, following the expected behavior, higher response times for 1000 CCs. The moderate connectivity topology still presented lower response time in this last case, but the decrease was smaller than before, indicating that the amortization effect is happening and higher response times should be observed if greater amounts of CCs were considered. Regardless, the difference between the average response time for versions V3 and V4 is evident and reaffirms the benefits of using a decentralized version.

Figure 4.4: V3 and V4: #CPs x Average Response Time.



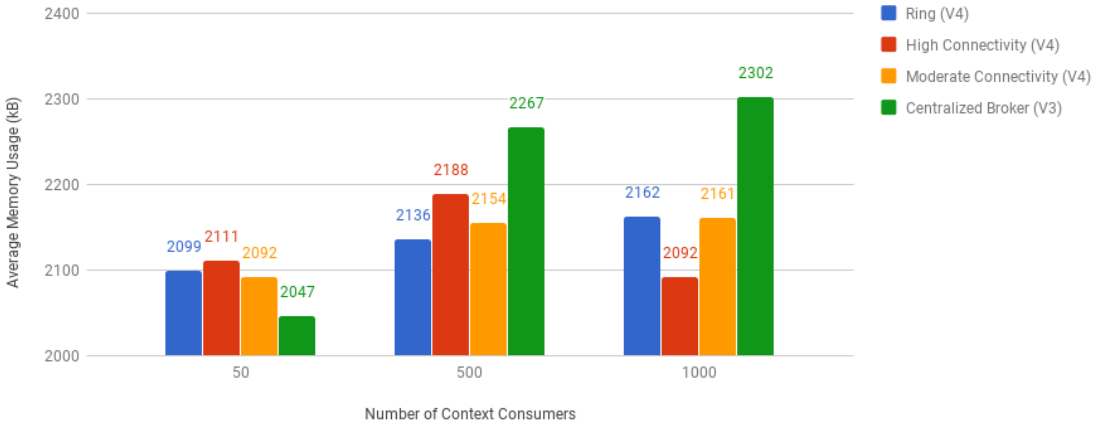
Source: The authors

Figure 4.4 shows the results of V3 and V4 considering the influence of the number of CPs on the average response time. The same behavior present in Figure 4.3 was observed here, but in a more representative way. The average response time reaches higher values as the number of Providers grows in both versions. However, the centralized Broker is much more affected by greater amounts of CPs, while all 3 topologies of the distributed Broker have a more stable behavior with a lower growth rate of response time. This indicates the scalability capacity of the distributed version, already mentioned throughout this work. The full mesh and moderate connectivity topologies present a decrease in the response time when 1000 CPs are considered. Although this goes against the expected behavior, this decrease is negligible and should not represent a meaningful trend in the testing setup. For greater amounts of CPs, the average response time is still expected to grow.

4.3.2 Average Memory Usage Analysis - V3 and V4

Figure 4.5 shows the comparison between V3 and V4 with respect to the impact of the number of CCs on the average memory usage. The first behavior that draws attention is the centralized version presenting better results than the distributed when 50 CCs are considered, for all 3 topologies. This fact makes sense and indicates that, for a small number of CCs, V4 has a higher cost in terms of memory consumption due to the number of Brokers running, giving advantage to V3 and making it the most suitable

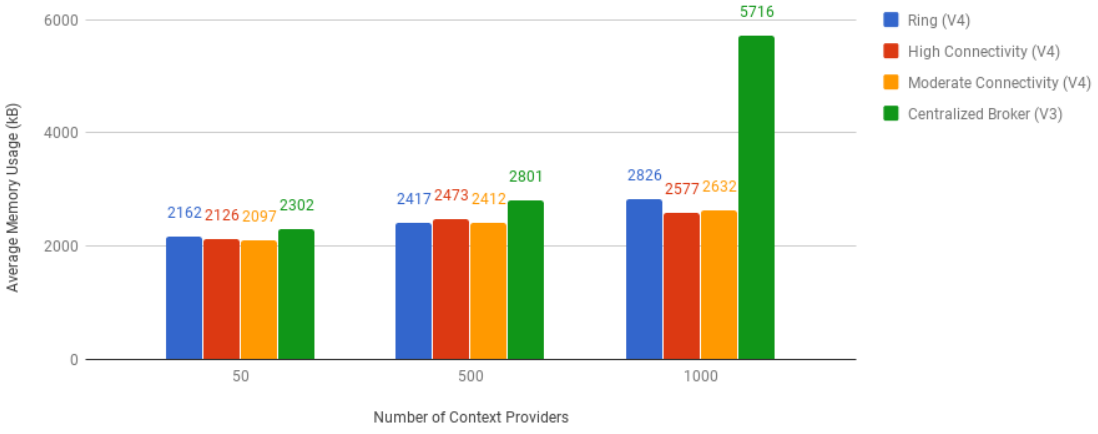
Figure 4.5: V3 and V4 - #CCs x Average Memory Usage.



Source: The authors

option in these cases. For the remaining configurations, however, the distributed version performed better than the centralized, even though the memory usage rises as the number of CCs is increased. The full mesh topology presented slightly higher values than the other topologies, which can be probably explained by the fact that each Broker must keep a larger neighbors list. A small decrease in the average memory usage can be observed in the full mesh topology when 1000 CCs are running, countering previous hypotheses. One possibility is that the tests duration was not long enough to capture the impact of this amount of CCs, since it will take more time for these entities to be initialized and the whole system to be stable, which relates to the startup overloading mentioned in Section 4.3.1, when Figure 4.3 was analyzed.

Figure 4.6: V3 and V4 - #CPs x Average Memory Usage.



Source: The authors

The effect of the number of CPs on the average memory usage for versions V3 and V4 is presented in Figure 4.6. It is interesting to notice that the centralized Broker shows

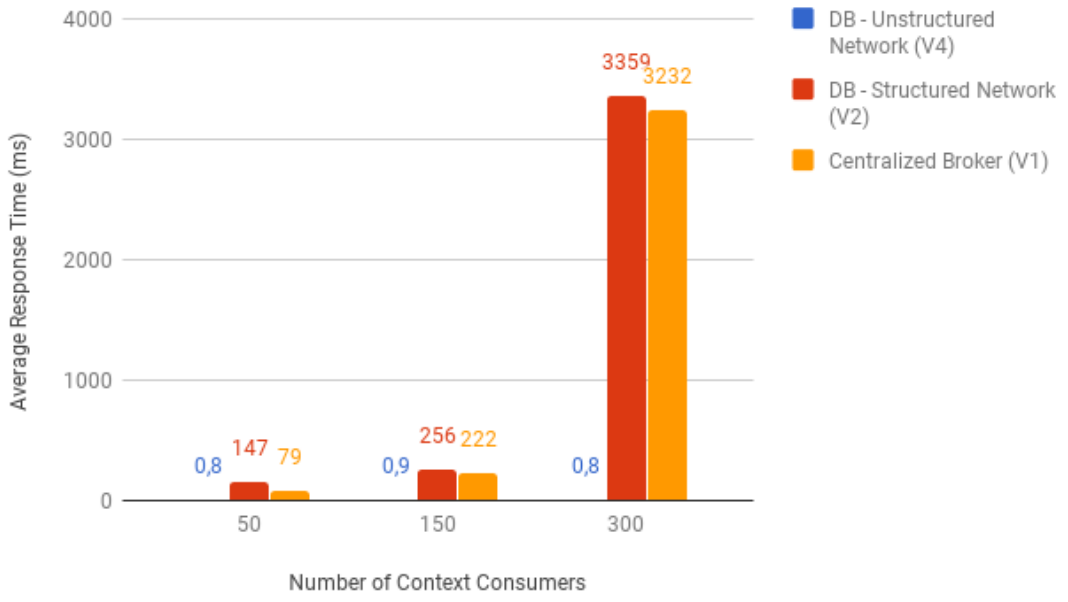
results very close to the ones of the distributed Broker when 50 CPs are considered. However, after that number, the average memory usage starts to rise quickly for the centralized version, while all 3 topologies show a similar behavior as the number of CPs is increased. Tests with smaller amounts of CPs would have to be done in order to check whether the centralized Broker presents less memory usage than every topology proposed. The ring topology showed a slightly higher memory usage with 1000 CPs. One possibility is that, in this case, the assignment of CCs and CPs to Brokers was done in such a way that the Brokers serving CCs are distant (in the topology) from the ones serving CPs, implying that requests must pass through more Brokers and be processed by them, which would lead to more memory consumption on average. Although the difference between V3 and V4 in terms of memory usage is not as high as in the average response times, V3 can consume 122% more memory than V4 when 1000 CPs are used.

4.3.3 Average Response Time Analysis - V1, V2 and V4

Versions V1, V2 and V4 are analyzed in Figure 4.7. The presented data was taken from (CRIPPA, 2013). For this analysis, the configuration used in (CRIPPA, 2013) was replicated for V4: 4 Brokers organized in a ring topology, one machine running 1 CP, one machine running all the CCs, request interval of 10 seconds and update interval of 30 seconds. The sharp difference between the average response times of V2 and V4 can be explained by two facts. The first is that, as mentioned in Chapter 2, structured P2P networks have the disadvantage of the cost incurred to keep the network structure, which might imply on delays when serving context requests. The second regards the programming language used to implement V2 (Java) and V4 (C++). Java was designed to provide portability and speed up development process, while C++ extends the C language, which was designed for efficient execution. Programs written in C++ are directly compiled to native executable machine code and are ready to run on the target machine. Programs written in Java, however, are compiled to byte-code and then are either interpreted by a Java Virtual Machine (JVM) or compiled to machine code by a JIT (just-in-time) compiler to only then run on the actual target machine. Although many improvements have already been made to JIT compilers, the optimizations performed by them are usually not the deepest or most sophisticated ones, since it would imply on an even higher delay in the compiling process. Hence, the resulting machine code might not be fully optimized. Besides, benchmarks (e.g., (BRUCKSCHLEGEL, 2005)) show, for instance, that array

operations have better performance in C++. Other particularities in Java such as the implementation of the libraries and execution of bounds checking for every array access operation may also contribute to the higher response times observed. A complete justification on why the C++ version showed much better performance would require further analysis, which is out of the scope of this work. Nonetheless, the results presented in Figure 4.7 are reasonable.

Figure 4.7: V1, V2 and V4: #CCs x Average Response Time.



Source: The authors

5 CONCLUSION

This chapter concludes this thesis by reviewing the concepts from the theoretical foundation and the decisions regarding the design of the proposed platform, as well as the main results achieved in the experimental evaluation. This is presented in Section 5.1. Moreover, future work is indicated in Section 5.2, showing how the platform can be extended or improved with respect to its operation and how it could be better evaluated.

5.1 Overview and Contributions

This work presented a decentralized version of a context management and distribution platform called Context Broker using unstructured peer-to-peer networks and has the works in (CRIPPA, 2010) and (CRIPPA, 2013) as the baseline. Research was done and related work in context-aware systems and peer-to-peer networks was reviewed in order to indicate and explain fundamental concepts such as context, context-awareness and context management. Three key aspects regarding peer-to-peer networks were considered: network structure, search mechanism and replication, which have a profound impact on the operation and overall performance of a distributed system. The design decisions were proposed based on the theoretical foundation and a usage scenario, which was also used for the requirements extraction. The Broker Message Protocol was described in terms of the operations supported by the platform, allowing the Brokers to work cooperatively to store and retrieve context information. UML diagrams were used in order to provide a better understanding on how the operations work. Next, a description of the prototype implementation was given, presenting a structural diagram of the system and key aspects such as its interfaces and the message parsing mechanism. Finally, the testing methodology was explained and results were presented comparing four different versions of the Context Broker, along with analyses highlighting the main differences in the behavior of these versions with respect to two metrics: average response time and average memory usage.

The Context Broker platform proposed in this work is a distributed system in which nodes (Brokers) are organized in an unstructured peer-to-peer (P2P) network and a Service-Oriented Architecture is used for the storage and retrieval of context information. The platform is composed by three main components: Context Providers, Context Consumers and Context Brokers. CPs and CCs contact CBs to make requests. The search for

data in the Broker network is done through random walks: context requests are forwarded to other Brokers until the desired data is found. Every Broker is always aware of its neighborhood, so no requests are sent to Brokers which are not alive. The implementation was done in C++ for performance reasons and, although it differs slightly from the proposed design, all the implemented functionalities worked as expected and proved to be enough for the validation of the benefits of this version.

The experimental evaluation showed that the distributed Broker platform proposed in this work performs better in terms of both average response time and memory usage than the centralized version which uses the same implementation, with the exception of one testing configuration. Moreover, it presented significantly lower response times when compared to the distributed version in (CRIPPA, 2013) and the centralized version in (CRIPPA, 2010), also exposing the impact of the chosen programming language in the final system. No relevant difference between the behavior of the 3 considered topologies with respect to the analyzed metrics was observed.

5.2 Future Work

Although this work is based on and extends the works in (CRIPPA, 2010) and (CRIPPA, 2013), it does not cover all the possible improvements that could be made to provide a solid context management and distribution platform nor extensively tests the capacity of the implemented system. Hence, it is relevant to indicate how related future work can extend this thesis in terms of operation and experimental evaluation.

The first improvement relates to security and privacy. No concern was considered when it comes to these aspects, which means that every CP and CC can send requests to any Broker in the network and there is no restriction in the communication between Brokers. This issue is important in cases where sensitive data (e.g. personal address, phone number, etc.) is being considered or the system is dealing with specific information belonging to different groups and access to this information should be kept restricted to each group. One way to address the problem is through the implementation of authentication mechanisms between CPs/CCs and Brokers or even between Brokers themselves. As a complement, a logging tool that registers every operation performed in the system with a high detail level could be created, so that any unexpected or forbidden action could be easily identified.

When it comes to the experimental evaluation, the first enhancement which could

be considered is the number of Brokers used to execute the tests. For usage scenarios that have a great amount of CPs and CCs, the benefits of using a decentralized system should be better observed if a network with more CBs is used, since the load can be better distributed. The assignment of CPs and CCs to Brokers is another aspect that potentially has an influence on metrics such as response time and memory consumption. In this work, this assignment was done randomly and manually, but it would be more appropriate if it was automated and performed considering, for instance, the current load in the Brokers. Finally, the platform's capacity of finding data was not stressed in the proposed testing setup. Since 5 out of 10 Brokers were serving CPs and every CC is requesting the same data, it is very unlikely that the desired data will not be found, regardless of the network topology. Therefore, it would be interesting to verify how do random walks perform in this platform when only a small amount of Brokers are storing context information.

REFERENCES

ABOWD, G. D. et al. Towards a better understanding of context and context-awareness. In: **Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing**. Karlsruhe, Germany: Springer-Verlag, 1999. (HUC '99), p. 304–307. ISBN 3-540-66550-1. Available from Internet: <http://dl.acm.org/citation.cfm?id=647985.743843>.

AMAZON. **Amazon EC2**. 2017. <https://aws.amazon.com/ec2/>. Accessed in Dec 6, 2017.

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A survey of peer-to-peer content distribution technologies. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 36, n. 4, p. 335–371, dec. 2004. ISSN 0360-0300. Available from Internet: <http://doi.acm.org/10.1145/1041680.1041681>.

BITTORRENT. **BitTorrent**. 2017. <http://www.bittorrent.com>. Accessed in June 04, 2017.

BOCKELMANN, C. et al. Hiflecs: Innovative technologies for low-latency wireless closed-loop industrial automation systems. In: **22. VDE-ITG-Fachtagung Mobilkommunikation**. Osnabrück, Germany: [s.n.], 2017. Available from Internet: <https://www.vde.com/mobilkommunikation_01>.

BROWN, P. J.; BOVEY, J. D.; CHEN, X. Context-aware applications: from the laboratory to the marketplace. **IEEE Personal Communications**, v. 4, n. 5, p. 58–64, Oct 1997. ISSN 1070-9916.

BRUCKSCHLEGEL, T. **Microbenchmarking C++, C# and Java | Dr Dobb's**. 2005. <http://www.drdobbs.com/cpp/microbenchmarking-c-c-and-java/184401976?pgno=1>. Accessed in Dec 21, 2017.

CHEN, H.; FININ, T.; JOSHI, A. Semantic web in the context broker architecture. In: **Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications**. Piscataway, NJ, USA: IEEE, 2004. p. 277–286.

CHEN, H. L. **An Intelligent Broker Architecture for Pervasive Context-Aware Systems**. Thesis (PhD) — University of Maryland, 2010.

COHEN, E.; SHENKER, S. Replication strategies in unstructured peer-to-peer networks. In: **Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications**. New York, NY, USA: ACM, 2002. (SIGCOMM '02), p. 177–190. ISBN 1-58113-570-X. Available from Internet: <http://doi.acm.org/10.1145/633025.633043>.

CRIPPA, M. R. **Design and implementation of a broker for a service-oriented context management and distribution architecture**. Dissertation (Bachelor's Thesis) — Federal University of Rio Grande do Sul (UFRGS), 2010. Available from Internet: <http://hdl.handle.net/10183/26352>.

CRIPPA, M. R. **Federation of Brokers for a Context Distribution and Management Architecture**. Dissertation (Master) — Technische Universität Kaiserslautern, 2013.

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**, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, v. 16, n. 2, p. 97–166, dec. 2001. ISSN 0737-0024. Available from Internet: <http://dx.doi.org/10.1207/S15327051HCI16234_02>.

DEY, A. K.; ABOWD, G. D.; WOOD, A. Cyberdesk: A framework for providing self-integrating context-aware services. In: **Proceedings of the 3rd International Conference on Intelligent User Interfaces**. San Francisco, California, USA: ACM, 1998. (IUI '98), p. 47–54. ISBN 0-89791-955-6. Available from Internet: <http://doi.acm.org/10.1145/268389.268398>.

DONAHOO, M. J. **Practical C++ Sockets**. 2017. <http://cs.ecs.baylor.edu/~donahoo/practical/CSockets/practical/>. Accessed in Nov 23, 2017.

GDB. **GDB: The GNU Project Debugger**. 2017. <https://www.gnu.org/software/gdb/>. Accessed in Dec 3, 2017.

GKANTSIDIS, C.; MIHAIL, M.; SABERI, A. Random walks in peer-to-peer networks. In: **IEEE INFOCOM**. Hong Kong, China: IEEE, 2004. v. 1, p. 130. ISSN 0743-166X.

GKANTSIDIS, C.; MIHAIL, M.; SABERI, A. Hybrid search schemes for unstructured peer-to-peer networks. In: **IEEE INFOCOM**. Piscataway, NJ, USA: IEEE, 2005.

GNU. **Gnutella**. 2017. <https://www.gnu.org/philosophy/gnutella.html>. Accessed in June 02, 2017.

KIANI, S. L. et al. A federated broker architecture for large scale context dissemination. In: **10th IEEE International Conference on Computer and Information Technology**. [S.l.: s.n.], 2010. p. 2964–2969.

LUA, E. K. et al. A survey and comparison of peer-to-peer overlay network schemes. **Communnication Surveys and Tutorials**, IEEE Press, Piscataway, NJ, USA, v. 7, n. 2, p. 72–93, abr. 2005. ISSN 1553-877X. Available from Internet: <http://dx.doi.org/10.1109/COMST.2005.1610546>.

LV, Q. et al. Search and replication in unstructured peer-to-peer networks. In: **Proceedings of the 16th International Conference on Supercomputing**. New York, NY, USA: ACM, 2002. (ICS '02), p. 84–95. ISBN 1-58113-483-5. Available from Internet: <http://doi.acm.org/10.1145/514191.514206>.

MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. **ACM Trans. Model. Comput. Simul.**, ACM, New York, NY, USA, v. 8, n. 1, p. 3–30, jan. 1998. ISSN 1049-3301. Available from Internet: <http://doi.acm.org/10.1145/272991.272995>.

PASCOE, J. Adding generic contextual capabilities to wearable computers. In: **Proceedings of the 2nd IEEE International Symposium on Wearable Computers**. Washington, DC, USA: IEEE Computer Society, 1998. p. 92–99.

PATEL, A.; CHAMPANERIA, T. A. Fuzzy logic based algorithm for context awareness in iot for smart home environment. In: **IEEE Region 10 Conference (TENCON)**. Singapore, Singapore: IEEE, 2016. p. 1057–1060.

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, First 2014. ISSN 1553-877X.

PIETZUCH, P. R.; BACON, J. M. Hermes: a distributed event-based middleware architecture. In: **Proceedings of the 22nd International Conference on Distributed Computing Systems**. Vienna, Austria: IEEE, 2002. p. 611–618.

PLAXTON, C. G.; RAJARAMAN, R.; RICHA, A. W. Accessing nearby copies of replicated objects in a distributed environment. In: **Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures**. Newport, Rhode Island, USA: ACM, 1997. (SPAA '97), p. 311–320. ISBN 0-89791-890-8. Available from Internet: <http://doi.acm.org/10.1145/258492.258523>.

RADIO, I. **HiFlecs**. 2015. <http://www.industrialradio.de/Projects/Home/HiFlecs>. Accessed in Jan 16, 2018.

ROMAN, M. et al. A middleware infrastructure for active spaces. **IEEE Pervasive Computing**, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 1, n. 4, p. 74–83, Oct 2002. ISSN 1536-1268.

ROWSTRON, A.; DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: GUERRAOUI, R. (Ed.). **Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms**. Heidelberg, Germany: Springer, 2001. p. 329–350. ISBN 978-3-540-45518-9. Available from Internet: <http://dx.doi.org/10.1007/3-540-45518-3_18>.

SCHILIT, B.; ADAMS, N.; WANT, R. Context-aware computing applications. In: **Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications**. Washington, DC, USA: IEEE Computer Society, 1994. (WMCSA '94), p. 85–90. ISBN 978-0-7695-3451-0. Available from Internet: <http://dx.doi.org/10.1109/WMCSA.1994.16>.

SCHILIT, B. N.; THEIMER, M. M. Disseminating active map information to mobile hosts. **IEEE Network**, IEEE Press, Piscataway, NJ, USA, v. 8, n. 5, p. 22–32, September 1994. ISSN 0890-8044.

STOICA, I. et al. Chord: A scalable peer-to-peer lookup service for internet applications. In: **Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications**. New York, NY, USA: ACM, 2001. (SIGCOMM '01), p. 149–160. ISBN 1-58113-411-8. Available from Internet: <http://doi.acm.org/10.1145/383059.383071>.

STRANG, T.; LINNHOFF-POPIEN, C. A context modeling survey. In: **First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp**. Nottingham, England: University of Southampton, 2004. Available from Internet: <http://elib.dlr.de/7444/>.

SUBLIME. **Sublime Text - A sophisticated text editor for code, markup and prose**. 2017. <https://www.sublimetext.com/>. Accessed in Dec 3, 2017.

WANG, J.; VANNINEN, M. Self-configuration protocols for P2P networks. **Web Intelligence and Agent Systems: An international journal**, v. 4, p. 61–76, 2006.

WAP. **WAP. User Agent Profile (UAProf)**. 2001. <http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf>. Accessed in June 20, 2017.

WAXMAN, B. M. Routing of multipoint connections. **IEEE Journal on Selected Areas in Communications**, v. 6, n. 9, p. 1617–1622, Dec 1988. ISSN 0733-8716.

WINOGRAD, T. Architectures for context. **Human-Computer Interaction**, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, v. 16, n. 2, p. 401–419, dec. 2001. ISSN 0737-0024. Available from Internet: <http://dx.doi.org/10.1207/S15327051HCI16234_18>.

ZHAO, B. Y. et al. Tapestry: A resilient global-scale overlay for service deployment. **IEEE Journal on Selected Areas in Communication**, IEEE Press, Piscataway, NJ, USA, v. 22, n. 1, p. 41–53, sep. 2006. ISSN 0733-8716. Available from Internet: <http://dx.doi.org/10.1109/JSAC.2003.818784>.

ZIMMERMANN, A.; LORENZ, A.; SPECHT, M. Applications of a context-management system. In: DEY, A. et al. (Ed.). **Proceedings of the 5th International and Interdisciplinary Conference CONTEXT on Modeling and Using Context**. Paris, France: Springer, 2005. p. 556–569. ISBN 978-3-540-31890-3. Available from Internet: <http://dx.doi.org/10.1007/11508373_42>.

APPENDIX A — BROKER MESSAGE PROTOCOL SPECIFICATION

Every message has a header in the following format:

MessageType | Timestamp

MessageType is an integer and *Timestamp* is the time in which the message was sent (in milliseconds). For the Context Update/Advertisement and Response messages, the header also includes the size of the payload (in bytes):

MessageType | Timestamp | PayloadSize

The messages supported by the platform, along with their description, are listed below:

Message types:

- **StartNeighborsMonitoring** - Code 0
- **Ping** - Code 1
- **Context Update/Advertisement** - Code 2
- **Context Request** - Code 3
- **Context Response** - Code 4
- **ACK** - Code 5
- **NACK** - Code 6
- **BrokerLeaving** - Code 7
- **BrokerJoining** - Code 8
- **Ping Response** - Code 9

Messages fields:

- **Flag**: flag to indicate if it is an Update or Advertisement message;
- **PayloadSize**: size of the actual message payload;
- **ProviderID**: ID of the provider sending the message;
- **EntityType**: type of the entity which the data refers to;
- **EntityID**: ID of the entity which the data refers to;
- **Scope**: scope which the data refers to;
- **ValidBegin**: time from which context data will be considered valid;
- **ValidEnd**: time from which context data will be considered invalid;

- **Payload**: data to be stored. The format used for the payload is (key, value) -> *K=V;K=V;...*;
- **StatusCode**: code used in the ACK and NACK messages to inform a requester application about the status of its request. The platform provides five codes:
  - REQ_RECEIVED (101): used in ACK messages. Indicates that the request has been received, but not yet processed;
  - REQ_PROCESSED (102): used in ACK messages. Indicates that the request has been processed with no errors;
  - I_ERROR (201): used in NACK messages. Indicates that an error internal to the platform has occurred, such as a failure in the creation of a thread or the throw of an exception;
  - DATA_NOT_FOUND (202): used in NACK messages. Indicates that no valid data has been found for a Context Request;
  - BAD_REQUEST (301): used in NACK messages. Indicates that the requester application has sent a message in a format that is not recognized by the platform.

A.0.1 StartNeighborsMonitoring Message

The StartNeighborsMonitoring message is sent from a Controller Application (CA) to a Broker. It indicates to the Broker that its neighbors are already set up and listening and that the monitoring (to check if they are alive) can start. After sending the message, the Controller Application receives an ACK, indicating that the monitoring has started, or a NACK, indicating that there was an error and the operation was not started. The message format is “0 | Timestamp”.

A.0.2 Ping Message

The Ping message is sent from a Context Provider or Consumer to a Broker, or from a Broker to another Broker. It is used to check if the destination node is alive and able to receive requests. A Broker uses its *neighborsAlive* module to send ping messages periodically to all neighbors, keeping its neighbors list always updated. The message

format is “1 | Timestamp”.

A.0.3 Context Update/Advertisement Message

The Context Update/Advertisement message is sent from a Context Provider to a Broker. In order to store new context data in a Broker, a CP must send an Advertisement message and later, if desired, an Update message to change the stored information. The message format is “2 | Timestamp | PayloadSize | Flag | ProviderID | EntityType | EntityID | Scope | ValidBegin | ValidEnd | Payload”.

A.0.4 Context Request Message

The Context Request message is sent from a Context Consumer to a Broker, or from a Broker to another one. The first case corresponds to when the CC requests data from a Broker, while the latter corresponds to when the Broker does not have the desired data and forwards the request to a neighbor, using the random walk mechanism (explained previously in this chapter). In this last scenario, each Broker appends to the Request message the address and port of the last Broker through which the query has passed, until the data is found. The message format is “3 | Timestamp | EntityType | EntityID | Scope | <BrokerAddress1> | <BrokerPort1> | <BrokerAddress2>...”, where the fields in angle brackets may appear or not.

A.0.5 Context Response Message

The Context Response message is sent from a Broker to a Context Consumer, or to another Broker. The first case corresponds to when the Broker which finds the requested data is the one that received the request, so that it is able to send this data directly to the CC. The second case is when a Broker finds the data, but needs to forward it to one or more Brokers until the Broker which received the request from the CC is reached. Considering this last scenario, when the desired data is found, the list of addresses and ports from the Context Request is appended to the Context Response message, so that each Broker will know the path back to the CC. The determination of the Brokers in which the data will be replicated also happens in this moment. Asterisks are appended randomly between

the *(BrokerAddress, BrokerPort)* pairs during the transfer from the Context Request to the Response. Hence, when a Broker receives a Response, it checks if there is an asterisk at the end of the message. If so, it replicates the data in its data table (either adding or updating it), removes the asterisk and then checks the last address and port, also removing them from the message and forwarding it to the corresponding Broker.

The message format is "4 | Timestamp | PayloadSize | ProviderID | EntityType | EntityID | Scope | ValidBegin | ValidEnd | Payload | <BrokerAddress1> | <BrokerPort1> | <*> | <BrokerAddress2>...", where the fields in angle brackets may appear or not.

A.0.6 Acknowledgement Message (ACK)

The ACK message is sent from a Broker to a Context Consumer, Context Provider or Controller Application. It is used to inform a positive status of a request made by one of these entities. The following operations supported by the broker generate two ACK messages, one indicating that the request was received and another indicating that the request was processed:

- Context Update/Advertisement
- Context Request
- BrokerJoining

Except for the Ping message (which has its own response), the remaining operations generate only one ACK, indicating that the request was received:

- BrokerLeaving
- StartNeighborsAlive

The ACK message format is "5 | Timestamp | 'ACK' | 'Status:'<StatusCode> | 'Operation' <OperationName> | 'Requested by' <RequesterAddres>".

A.0.7 Not-Acknowledgement Message (NACK)

The NACK message can be sent from a Broker to a Context Consumer, Context Provider, Controller Application or even to another Broker. It is used to inform a negative status of a request made by one of these entities. The case in which a Broker sends a NACK to another Broker corresponds to when the requested data is not found in the

network, arising the necessity to forward this NACK through the nodes until it reaches the corresponding CC. Every operation supported by the Broker generates at most one NACK message. The format is “6 | Timestamp | ‘NACK’ | ‘Status:’<StatusCode> | ‘Operation’ <OperationName> | ‘Requested by’ <RequesterAddress>”.

A.0.8 BrokerLeaving Message

The BrokerLeaving message is sent from a Controller Application to a Broker, or from a Broker to a neighbor node. It is used to command the Broker to exit the network, making it send a leaving advertisement to its neighbors. When a Broker receives a BrokerLeaving message from a Controller Application, it appends an asterisk to the message (indicating to the other Brokers that they should not further forward it) and sends it to the neighbor nodes. The message format is “7 | Timestamp | <*>”, where the field in angle brackets may appear or not.

A.0.9 BrokerJoining Message

The *BrokerJoining* message is sent from a Controller Application to a Broker, or from a Broker to a neighbor node. It is used to command the Broker to connect to a set of neighbors, which are parameters of the Controller Application. When a Broker receives a *BrokerJoining* message from this application, it appends an asterisk to the message (indicating to the other Brokers that they should not execute the same actions) and sends it to the neighbors indicated in the message. The Broker, then, waits for an ACK from each of the nodes with which the communication was attempted. If the response was received within the established timeout interval, the addresses of those that answered are added to its neighbors list, while they add the address of the new Broker. The message format is “8 | Timestamp | NeighborAddress1 | <NeighborAddress2> | <NeighborAddress3>...”, where the fields in angle brackets may appear or not.

A.0.10 Ping Response Message

The Ping Response message is sent from a Broker to a Context Provider or Consumer. It indicates that the Broker is alive and able to receive requests. The message

format is “9 | Timestamp | Ctx_Broker | BrokerAddress”.

APPENDIX B — RESULTS TABLE

This chapter presents the complete table with the results regarding average response time and average memory usage for all performed tests.

Table B.1: Results Table.

Test Config.	Network Topology	Num. of CPs	Num. of CCs	Upd. Interval (s)	Req. Interval (s)	Avg. Resp. Time (us)	Avg. Mem. Usage (kB)
1	Ring	50	50	1	1	465.09	2129.35
2	Ring	50	50	1	30	917.66	2386.54
3	Ring	50	50	1	60	982.59	2567.52
4	Ring	50	50	30	1	453.29	1984.64
5	Ring	50	50	30	30	922.05	1903.77
6	Ring	50	50	30	60	950.05	1885.23
7	Ring	50	50	60	1	501.03	2263.46
8	Ring	50	50	60	30	953.79	2006.28
9	Ring	50	50	60	60	997.23	2235.41
10	Ring	50	500	1	1	264.73	2634.34
11	Ring	50	500	1	30	892.08	2003.37
12	Ring	50	500	1	60	953.72	2499.84
13	Ring	50	500	30	1	237.16	2511.45
14	Ring	50	500	30	30	820.26	2122.70
15	Ring	50	500	30	60	987.75	2015.65
16	Ring	50	500	60	1	281.66	2441.47
17	Ring	50	500	60	30	908.69	2281.99
18	Ring	50	500	60	60	996.47	2061.32
19	Ring	50	1000	1	1	269.42	3057.06
20	Ring	50	1000	1	30	885.09	2246.53
21	Ring	50	1000	1	60	976.07	2050.46
22	Ring	50	1000	30	1	233.83	2657.40
23	Ring	50	1000	30	30	855.12	2036.59
24	Ring	50	1000	30	60	903.05	2047.07
25	Ring	50	1000	60	1	229.28	2664.20
26	Ring	50	1000	60	30	903.61	2202.01
27	Ring	50	1000	60	60	914.91	1906.94
28	Ring	500	50	1	1	359.99	2796.91
29	Ring	500	50	1	30	1004.99	2645.34
30	Ring	500	50	1	60	1177.72	2258.36
31	Ring	500	50	30	1	424.73	2360.34
32	Ring	500	50	30	30	1148.45	2073.60
33	Ring	500	50	30	60	1093.46	2230.21
34	Ring	500	50	60	1	481.24	2430.94
35	Ring	500	50	60	30	1147.32	2044.85
36	Ring	500	50	60	60	1198.47	1999.78
37	Ring	500	500	1	1	241.46	2609.66
38	Ring	500	500	1	30	1015.91	2519.95
39	Ring	500	500	1	60	1002.61	2270.96
40	Ring	500	500	30	1	269.80	2525.25
41	Ring	500	500	30	30	1090.30	2052.55
42	Ring	500	500	30	60	1126.23	2028.21
43	Ring	500	500	60	1	237.98	2799.76
44	Ring	500	500	60	30	1053.33	2278.52
45	Ring	500	500	60	60	1047.64	2054.70
46	Ring	500	1000	1	1	199.74	3155.66
47	Ring	500	1000	1	30	929.18	2590.58
48	Ring	500	1000	1	60	1033.76	2760.31
49	Ring	500	1000	30	1	208.02	2972.77
50	Ring	500	1000	30	30	925.79	2197.78
51	Ring	500	1000	30	60	1095.70	2075.90
52	Ring	500	1000	60	1	213.42	3429.51
53	Ring	500	1000	60	30	909.1	2461.46
54	Ring	500	1000	60	60	1084.42	2056.00
55	Ring	1000	50	1	1	394.14	2725.13
56	Ring	1000	50	1	30	1049.53	2942.78
57	Ring	1000	50	1	60	1038.36	2980.25
58	Ring	1000	50	30	1	427.13	2574.78
59	Ring	1000	50	30	30	1169.16	2319.61
60	Ring	1000	50	30	60	925.59	2265.57
61	Ring	1000	50	60	1	411.82	2305.50
62	Ring	1000	50	60	30	1067.51	2101.66

Test Config.	Network Topology	Num. of CPs	Num. of CCs	Upd. Interval (s)	Req. Interval (s)	Avg. Resp. Time (us)	Avg. Mem. Usage (kB)
63	Ring	1000	50	60	60	1149.36	2165.39
64	Ring	1000	500	1	1	228.31	2915.51
65	Ring	1000	500	1	30	897.35	2848.27
66	Ring	1000	500	1	60	995.05	2520.51
67	Ring	1000	500	30	1	253.29	2914.22
68	Ring	1000	500	30	30	1128.41	2125.97
69	Ring	1000	500	30	60	1152.61	2096.64
70	Ring	1000	500	60	1	256.86	2749.07
71	Ring	1000	500	60	30	1127.74	2471.10
72	Ring	1000	500	60	60	896.23	2297.03
73	Ring	1000	1000	1	1	215.27	3070.49
74	Ring	1000	1000	1	30	944.7	3555.47
75	Ring	1000	1000	1	60	901.75	2617.22
76	Ring	1000	1000	30	1	231.26	2706.91
77	Ring	1000	1000	30	30	919.6	2356.81
78	Ring	1000	1000	30	60	899.24	2112.82
79	Ring	1000	1000	60	1	221.39	2549.70
80	Ring	1000	1000	60	30	909.47	2565.06
81	Ring	1000	1000	60	60	888.19	2116.00
82	High Connectivity	50	50	1	1	428.62	2913.33
83	High Connectivity	50	50	1	30	889.94	2320.19
84	High Connectivity	50	50	1	60	938.22	2611.96
85	High Connectivity	50	50	30	1	424.55	2702.08
86	High Connectivity	50	50	30	30	931.12	2073.28
87	High Connectivity	50	50	30	60	934.05	2067.35
88	High Connectivity	50	50	60	1	392.63	2264.75
89	High Connectivity	50	50	60	30	957.11	1939.14
90	High Connectivity	50	50	60	60	925.64	2023.97
91	High Connectivity	50	500	1	1	266.24	2611.17
92	High Connectivity	50	500	1	30	890.28	2139.71
93	High Connectivity	50	500	1	60	920.21	2251.20
94	High Connectivity	50	500	30	1	255.79	2233.99
95	High Connectivity	50	500	30	30	866.31	2062.87
96	High Connectivity	50	500	30	60	909.46	2218.66
97	High Connectivity	50	500	60	1	266.92	2926.04
98	High Connectivity	50	500	60	30	887.47	2362.42
99	High Connectivity	50	500	60	60	898.87	2241.88
100	High Connectivity	50	1000	1	1	220.68	2903.41
101	High Connectivity	50	1000	1	30	926.39	2062.87
102	High Connectivity	50	1000	1	60	893.70	2140.05
103	High Connectivity	50	1000	30	1	314.31	3283.10
104	High Connectivity	50	1000	30	30	855.39	2104.02
105	High Connectivity	50	1000	30	60	879.65	2043.61
106	High Connectivity	50	1000	60	1	290.08	2888.44
107	High Connectivity	50	1000	60	30	889.89	2210,45
108	High Connectivity	50	1000	60	60	922.70	2089.23
109	High Connectivity	500	50	1	1	631.42	2910.70
110	High Connectivity	500	50	1	30	1027.39	2621.23
111	High Connectivity	500	50	1	60	1019.70	2469.04
112	High Connectivity	500	50	30	1	812.33	2086.48
113	High Connectivity	500	50	30	30	1135.74	2232.87
114	High Connectivity	500	50	30	60	1207.14	2015.54
115	High Connectivity	500	50	60	1	863.59	2536.12
116	High Connectivity	500	50	60	30	1194.86	1974.99
117	High Connectivity	500	50	60	60	1166.81	1991.07
118	High Connectivity	500	500	1	1	275.87	2723.82
119	High Connectivity	500	500	1	30	906.50	2165.71
120	High Connectivity	500	500	1	60	920.15	2484.72
121	High Connectivity	500	500	30	1	279.31	2699.26
122	High Connectivity	500	500	30	30	972.71	2056.42
123	High Connectivity	500	500	30	60	987.57	2298.26
124	High Connectivity	500	500	60	1	302.81	2512.22
125	High Connectivity	500	500	60	30	987.71	1986.52
126	High Connectivity	500	500	60	60	1004.84	2290.40
127	High Connectivity	500	1000	1	1	281.13	3268.75
128	High Connectivity	500	1000	1	30	1126.34	2869.44
129	High Connectivity	500	1000	1	60	925.22	2946.19
130	High Connectivity	500	1000	30	1	346.66	3342.91
131	High Connectivity	500	1000	30	30	915.28	2505.64
132	High Connectivity	500	1000	30	60	981.59	2288.80
133	High Connectivity	500	1000	60	1	305.45	3075.39
134	High Connectivity	500	1000	60	30	904.22	2042.92
135	High Connectivity	500	1000	60	60	976.91	1995.84
136	High Connectivity	1000	50	1	1	661.84	2759.52
137	High Connectivity	1000	50	1	30	967.24	2852.67
138	High Connectivity	1000	50	1	60	979.42	2525.30
139	High Connectivity	1000	50	30	1	720.19	2621.79
140	High Connectivity	1000	50	30	30	1152.13	2099.18

Test Config.	Network Topology	Num. of CPs	Num. of CCs	Upd. Interval (s)	Req. Interval (s)	Avg. Resp. Time (us)	Avg. Mem. Usage (kB)
141	High Connectivity	1000	50	30	60	1082.89	2203.97
142	High Connectivity	1000	50	60	1	720.90	2104.20
143	High Connectivity	1000	50	60	30	1241.90	2159.41
144	High Connectivity	1000	50	60	60	1289.41	2329.03
145	High Connectivity	1000	500	1	1	852.55	3561.00
146	High Connectivity	1000	500	1	30	931.83	2997.86
147	High Connectivity	1000	500	1	60	1009.59	3377.55
148	High Connectivity	1000	500	30	1	281.66	3026.01
149	High Connectivity	1000	500	30	30	1271.23	2591.24
150	High Connectivity	1000	500	30	60	1270.38	2560.50
151	High Connectivity	1000	500	60	1	297.46	2808.73
152	High Connectivity	1000	500	60	30	1164.46	2531.53
153	High Connectivity	1000	500	60	60	1280.61	2521.03
154	High Connectivity	1000	1000	1	1	268.01	3734.13
155	High Connectivity	1000	1000	1	30	928.77	3208.97
156	High Connectivity	1000	1000	1	60	1055.51	2827.62
157	High Connectivity	1000	1000	30	1	293.39	2969.01
158	High Connectivity	1000	1000	30	30	916.24	2371.43
159	High Connectivity	1000	1000	30	60	1206.15	2350.38
160	High Connectivity	1000	1000	60	1	294.79	3093.19
161	High Connectivity	1000	1000	60	30	917.49	2151.18
162	High Connectivity	1000	1000	60	60	1271.65	2563.68
163	Moderate Connectivity	50	50	1	1	776.70	2474.47
164	Moderate Connectivity	50	50	1	30	933.99	2214.86
165	Moderate Connectivity	50	50	1	60	924.40	2411.47
166	Moderate Connectivity	50	50	30	1	784.32	2291.95
167	Moderate Connectivity	50	50	30	30	945.93	2025.37
168	Moderate Connectivity	50	50	30	60	895.12	1983.77
169	Moderate Connectivity	50	50	60	1	753.73	2169.06
170	Moderate Connectivity	50	50	60	30	949.58	2036.57
171	Moderate Connectivity	50	50	60	60	844.97	2264.92
172	Moderate Connectivity	50	500	1	1	274.32	3496.09
173	Moderate Connectivity	50	500	1	30	892.86	2147.36
174	Moderate Connectivity	50	500	1	60	877.50	2249.22
175	Moderate Connectivity	50	500	30	1	451.08	2385.24
176	Moderate Connectivity	50	500	30	30	885.27	1984.32
177	Moderate Connectivity	50	500	30	60	854.56	2283.39
178	Moderate Connectivity	50	500	60	1	302.61	2735.29
179	Moderate Connectivity	50	500	60	30	914.09	2331.22
180	Moderate Connectivity	50	500	60	60	852.53	2257.93
181	Moderate Connectivity	50	1000	1	1	305.93	2967.91
182	Moderate Connectivity	50	1000	1	30	860,1	2152.77
183	Moderate Connectivity	50	1000	1	60	849.66	2625.55
184	Moderate Connectivity	50	1000	30	1	300.39	3321.64
185	Moderate Connectivity	50	1000	30	30	848.99	2032.19
186	Moderate Connectivity	50	1000	30	60	858.10	2530.70
187	Moderate Connectivity	50	1000	60	1	277.45	2940.82
188	Moderate Connectivity	50	1000	60	30	907.81	2105.74
189	Moderate Connectivity	50	1000	60	60	851.15	2347.91
190	Moderate Connectivity	500	50	1	1	737.95	3028.89
191	Moderate Connectivity	500	50	1	30	927.82	3068.86
192	Moderate Connectivity	500	50	1	60	894.82	3040.51
193	Moderate Connectivity	500	50	30	1	746.80	2918.59
194	Moderate Connectivity	500	50	30	30	918.79	2956.59
195	Moderate Connectivity	500	50	30	60	884.98	2419.56
196	Moderate Connectivity	500	50	60	1	760.36	2670.63
197	Moderate Connectivity	500	50	60	30	949.04	2406.40
198	Moderate Connectivity	500	50	60	60	890.93	2397.63
199	Moderate Connectivity	500	500	1	1	247.05	3073.50
200	Moderate Connectivity	500	500	1	30	856.09	2935.93
201	Moderate Connectivity	500	500	1	60	852.86	3319.14
202	Moderate Connectivity	500	500	30	1	284.82	2724.33
203	Moderate Connectivity	500	500	30	30	859.22	2513.91
204	Moderate Connectivity	500	500	30	60	827.94	2602.87
205	Moderate Connectivity	500	500	60	1	292.04	3173.76
206	Moderate Connectivity	500	500	60	30	897.21	2520.39
207	Moderate Connectivity	500	500	60	60	910.02	2304.25
208	Moderate Connectivity	500	1000	1	1	295.64	3626.72
209	Moderate Connectivity	500	1000	1	30	975.7	2761.29
210	Moderate Connectivity	500	1000	1	60	893.89	3041.64
211	Moderate Connectivity	500	1000	30	1	329.49	3219.83
212	Moderate Connectivity	500	1000	30	30	984.04	2386.06
213	Moderate Connectivity	500	1000	30	60	901.56	2452.82
214	Moderate Connectivity	500	1000	60	1	387.53	3028.09
215	Moderate Connectivity	500	1000	60	30	902.84	2088.01
216	Moderate Connectivity	500	1000	60	60	898.28	2419.48
217	Moderate Connectivity	1000	50	1	1	783.88	3105.93
218	Moderate Connectivity	1000	50	1	30	953.26	3251.37
219	Moderate Connectivity	1000	50	1	60	951.25	3122.86
220	Moderate Connectivity	1000	50	30	1	760.30	2607.27
221	Moderate Connectivity	1000	50	30	30	938.16	2522.17
222	Moderate Connectivity	1000	50	30	60	942.37	2826.38

Test Config.	Network Topology	Num. of CPs	Num. of CCs	Upd. Interval (s)	Req. Interval (s)	Avg. Resp. Time (us)	Avg. Mem. Usage (kB)
223	Moderate Connectivity	1000	50	60	1	769.20	2751.31
224	Moderate Connectivity	1000	50	60	30	959.44	2539.52
225	Moderate Connectivity	1000	50	60	60	925.31	2654.96
226	Moderate Connectivity	1000	500	1	1	277.40	3813.56
227	Moderate Connectivity	1000	500	1	30	908.90	2869.80
228	Moderate Connectivity	1000	500	1	60	839.68	3443.54
229	Moderate Connectivity	1000	500	30	1	299.94	2746.38
230	Moderate Connectivity	1000	500	30	30	836.02	2334.70
231	Moderate Connectivity	1000	500	30	60	842.03	2288.66
232	Moderate Connectivity	1000	500	60	1	285.39	2785.67
233	Moderate Connectivity	1000	500	60	30	856.20	2430.09
234	Moderate Connectivity	1000	500	60	60	848.33	2587.51
235	Moderate Connectivity	1000	1000	1	1	313.73	3552.94
236	Moderate Connectivity	1000	1000	1	30	911.02	3302.01
237	Moderate Connectivity	1000	1000	1	60	859.98	3093.53
238	Moderate Connectivity	1000	1000	30	1	293.11	3355.19
239	Moderate Connectivity	1000	1000	30	30	931.49	2417.76
240	Moderate Connectivity	1000	1000	30	60	854.06	2591.60
241	Moderate Connectivity	1000	1000	60	1	294.82	3360.80
242	Moderate Connectivity	1000	1000	60	30	910.95	2177.5
243	Moderate Connectivity	1000	1000	60	60	851.06	2791.13
244	Ring	1	50	30	10	840	Not analyzed
245	Ring	1	150	30	10	880	Not analyzed
246	Ring	1	300	30	10	830	Not analyzed
247	Centralized Broker	50	50	30	30	1421.74	2046.89
248	Centralized Broker	50	500	30	30	1646.08	2267.06
249	Centralized Broker	50	1000	30	30	1650.22	2302,39
250	Centralized Broker	500	1000	30	30	4491.3	2800.9
250	Centralized Broker	1000	1000	30	30	7029.55	5715.83

Source: The authors

APPENDIX C — GRADUATION WORK 1

Decentralized Broker for Context Management and Distribution using Unstructured P2P Networks in a Service-Oriented Architecture

Igor L. Pereira[1], **Alberto Egon S. Filho**[1], **Marcos R. Crippa**[2]

[1]Informatics Institute – Federal University of Rio Grande do Sul (UFRGS)
Mailbox 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

[2]Department of Electrical and Computer Engineering
Technische Universität Kaiserslautern (TUKL) – Kaiserslautern, Germany

{ilpereira,alberto}@inf.ufrgs.br, crippa@eit.uni-kl.de

Abstract. *Context information is present in many application types nowadays. This information is the basis for the interaction between users and computing systems and may consist in a great data volume. Thus, this demands the proposal of efficient mechanisms for distribution and management of the information. This work proposes a decentralized platform for context management and distribution called Context Broker using unstructured peer-to-peer networks. The theoretical foundation that substantiates the development of the platform and a description and justification of the design decisions and used technologies will be presented, along with a task schedule regarding the implementation and evaluation processes.*

1. Introduction

Technology progress along with the Internet has been changing the way people interact with each other and with computing systems. Low hardware cost, high computational power and increased sensing capability are some of the factors that explain the popularity that computing systems have reached [Brown et al. 1997]. The presence of sensors in devices is an interesting feature, because it provides context-awareness capability and allows the implementation of applications for a great variety of purposes, improving the interaction with these devices [Schilit et al. 1994].

Let us consider a smartphone. With a GPS, which provides user location, one is able to develop an application that shows weather forecast or displays nearby places which the user might want to visit. The camera allows the user to take pictures, store and share them in social networks, for which an account with an e-mail address and a password is typically required. The user may also listen to music on the smartphone, thus making it possible for an application to send notifications about upcoming concerts based on the most played artists. All this information (location, photos, e-mail address and musical taste) compose a context related to the user.

In order to develop context-aware applications, it is necessary to handle context information appropriately, in a way that it is reachable, up-to-date and can be obtained in a reasonable amount of time. The approach should also provide scalability, flexibility and fault tolerance. This paper (Graduation Work I) proposes the development of a decentralized platform called Context Broker to address this challenge, following the structure

and results of [Crippa 2010] and [Crippa 2013]. The objectives of this work are to review previous research on context awareness and peer-to-peer networks and indicate the main design decisions for the platform based on its requirements. Implementation and testing will be presented in Graduation Work II.

This work is structured as follows: section 2 presents the research on context-aware systems, the fundamental concepts and state of the art in the area. Section 3 addresses peer-to-peer networks, presenting basic definitions and two key aspects related to them. Section 4 describes the proposed platform's components along with a requirements analysis and the main design decisions. Section 5 shows the task schedule for the next phase of the work (Graduation Work II). Section 6 concludes the paper and indicates the next steps.

2. Context-Aware Systems

This work describes a platform for context management and distribution. Therefore, it is important to define and clarify some fundamental concepts such as *context*, *context awareness* and *context management* (along with a detailing of important related aspects), in order to provide a correct understanding of the work as a whole. This section aims to present these concepts and review related research on the area.

2.1. Context in Context-Aware Systems

Many definitions for context have already been proposed in the literature. The first work to address context and context awareness was [Schilit and Theimer 1994], stating that context corresponds to the user location, nearby people and objects, as well as changes to these over time. A similar definition was provided by [Brown et al. 1997]: context corresponds to location, identities of the people around the user, the time of day, season, temperature, etc. Two other more general approaches were given by [Dey et al. 1998] and [Pascoe 1998]. The former defines context as the user's physical, social, emotional or informational state, while the latter considers that context is the subset of physical and conceptual states of interest to a particular entity.

All the presented definitions give us a too specific idea about context. Depending on the scenario, concepts such as location, time of day or emotional state may not be sufficient to properly define context, because other types of information might be used. Therefore, a more general approach is needed in order to substantiate the relevance of this work. The definition which will be used was given by [Abowd et al. 1999]: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves".

2.2. Context-Awareness in Context-Aware Systems

A definition for context awareness is also provided by [Abowd et al. 1999] and will be adopted in this work: "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task". This definition is appropriate because it gives a broad and flexible idea, just like the definition for context. Furthermore, the authors of [Abowd et al. 1999] presented a categorization of features for context-aware applications, combining the ideas of the taxonomies from [Pascoe 1998] and [Schilit et al. 1994]. There are three categories:

1. Presentation of information and services to a user: corresponds to the capability of obtaining information and executing commands for the user manually, based on available context;
2. Automatic execution of a service: relates to the ability of providing or changing the behavior of a service automatically, based on available context;
3. Tagging of context to information for later retrieval: corresponds to the ability of relating context and data, in the sense that certain data is available for a user when he is in the associated context.

The definition of context awareness provides a way to determine whether an application is context-aware or not and this is useful when specifying the types of applications that need to be supported. The categorization of context-aware features brings us two main benefits. The first is the specification of types of applications that need support from the system. The second is that it shows the types of features that should be thought when building the context-aware applications [Abowd et al. 1999].

2.3. Context Management in Context-Aware Systems

Although the definition of context-aware systems has already been given, it is important to indicate the purpose of this type of system in a more practical way. Context management is related to the tasks for which a context-aware system is responsible in order to handle context information [Crippa 2010]. A more formal definition was provided by [Zimmermann et al. 2005] based on the authors' work experience in many application domains: context management corresponds to the creation and administration of context-aware applications, considering related parameters and information sources with the goal of implementing certain behavior. Therefore, context management is the foundation for the development of context-aware applications.

One question that remains is what should a context-aware system be able to do or, in other words, what functionalities should be available. According to [Kiani et al. 2010], a context-aware communication system encompasses several context management functionalities, being two the most important ones: acquisition and provision of contextual information related to an entity. This description already gives us a brief idea about what an architecture of a context-aware system should be like. The authors go further by saying that it is possible to divide the system components involved in context management into either context consumers or context providers or a combination of these. Other functionalities of context-aware systems are merging correlated context data and locating and accessing context sources [Crippa 2010]. The platform considered in this work is based on the provider/consumer architecture outlined above.

In order to manage context, three models were proposed by [Winograd 2001]. These models are related to the system architecture and aim to coordinate multiple processes and components, a concept that is necessary to implement context-aware systems:

- **Widgets:** Context widgets work as an interface between the application and its operating environment. They hide the complexity of context acquisition and abstract context information to suit application needs [Dey et al. 2001]. They provide good efficiency but have the problem of not being robust to component failures;

- **Networked Services:** This model corresponds to a client-server architecture, in which clients look for services and need to establish a connection with them in order to use it (e.g., accessing a database). It is more flexible and robust than the previous due to the independence of the components, but it is also less efficient because a service discovery function is required;
- **Blackboards:** In this approach, applications post data to a shared message board and receive data through subscriptions, according to a specified pattern. In spite of providing loose coupling and robustness, communication efficiency is decreased because every message goes through a centralized server before reaching its final destination.

2.3.1. Context Acquisition

The chosen method for context acquisition has an impact on the design of context-aware systems, therefore it is relevant to indicate some of the possible options. Three approaches were presented by [Chen 2010]:

- **Direct access to hardware sensors:** Context data is obtained by directly accessing the physical sensors, providing a good knowledge about the data for the collecting applications. The problem comes when the number of context sources rises, requiring these applications to have the ability to communicate with many different sensors [Chen 2010];
- **Facilitated by a middleware infrastructure:** A middleware infrastructure is responsible for managing the low-level sensor data, thus allowing the applications to concern on how context will be used. This approach provides great extensibility and reusability of sensors [Crippa 2010];
- **Acquire context from a context server:** Context data is gathered in a server which runs on a resourceful device and the context-aware applications make requests to this server. As in the middleware approach, great reusability is provided [Crippa 2010].

2.3.2. Context Modeling

Context information is acquired by the use of sensors, which provide raw data. In order for a context-aware application to benefit from it, this information must be properly structured according to the application goals. This process is called context modeling (or context representation) and needs to support easy manipulation and extensibility, efficient search and scalability [Crippa 2010].

When modeling context, there are requirements that need to be fulfilled so that the obtained representation can be meaningful for the context-aware system. A set of these requirements was presented in [Perera et al. 2014] as heterogeneity and mobility, relationships and dependencies, timeliness, imperfection, reasoning, usability of modeling formalisms and efficient context provisioning. The six most popular context modeling techniques were indicated in [Strang and Linnhoff-Popien 2004]. Due to space limitations, three of them are described below:

- **Key-Value Models:** They are the simplest form of context representation. Context information is associated with a unique key and a matching algorithm is used for lookup. In spite of providing easy management, this approach is not scalable and lacks of structure for modeling complex information, thus jeopardizing efficiency of context retrieval;
- **Graphical Models:** Provide more expressiveness in the final result because context is modeled with relationships. One example of this technique is the Unified Modeling Language (UML), which has a generic structure and enables good readability;
- **Ontology Based Models:** It is considered the most appropriate way of modeling and managing context. An ontology is an abstract model of a certain phenomenon and offers great flexibility and expressiveness, but it can also decrease the performance in context retrieval [Perera et al. 2014]. Resource Description Framework (RFD) and Web Ontology Language (OWL) are examples of languages used to describe ontologies.

2.3.3. Context Dissemination

Context Dissemination relates to how context is distributed in a context-aware system and is one of the most important functionalities in context management. The mechanism used to execute this distribution is influenced by the proposed system architecture and has an impact on the overall performance of the system. According to [Perera et al. 2014], there are two methods for this task: the first one is the *query* method, in which the context-aware system receives a query from consumer applications and resolves this query to produce a result; the second one is the *subscription* or *publish/subscribe* method, in which consumers subscribe to a certain type of context information and then the system sends updates regarding this information periodically or when an event occurs.

2.4. Related Work in Context-Aware Systems

The Context Broker platform proposed in this work is based on [Crippa 2010] and [Crippa 2013]. In [Crippa 2010], a centralized Broker was implemented, in which both methods were used to distribute context in a straightforward way. In [Crippa 2013], a distributed version of the Broker using structured P2P networks was proposed and this has imposed a bigger challenge regarding the context dissemination aspect. Many other approaches for implementing context-aware systems have already been proposed, each one with its advantages and disadvantages, so it becomes relevant to indicate some of the most significant ones.

CoBrA [Chen et al. 2004] is an architecture for implementing smart spaces using context-aware systems. The architecture design is shown in Figure 1. The main component is a broker agent which consists of the following components: context knowledge base, context-reasoning engine, context-acquisition module and policy-management module. Context is modeled using Semantic Web languages such as RDF and OWL, providing a suitable representation for reasoning and knowledge sharing. The proposed architecture is centralized, but a group of brokers can work together through a broker federation [Perera et al. 2014].

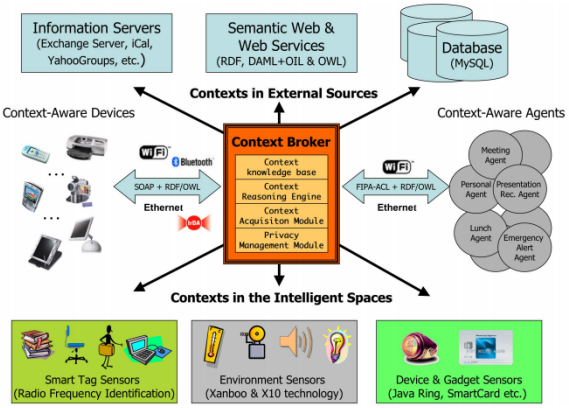


Figure 1. CoBrA architecture design [Chen et al. 2004]

Another platform, called Context Toolkit, was proposed in [Dey et al. 2001]. It consists in a conceptual framework for the design of context-aware applications which is composed by three main entities: context widgets (provide access to context information), interpreters (produce high-level context information from low-level sensor data using reasoning techniques) and aggregators (responsible for gathering related context information in a common repository). To obtain context from the system, applications invoke services and use discoverers to find the components that are able to provide the desired data. The communication between all these components is implemented using a protocol based on HTTP and XML. Figure 2 shows one possible configuration for this system.

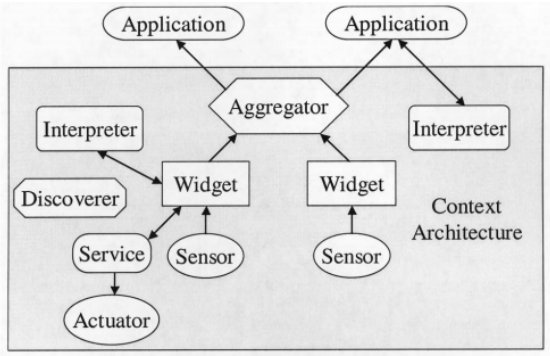


Figure 2. Example configuration of the Context Toolkit framework [Dey et al. 2001]

3. Peer-to-Peer Networks

As mentioned in Section 1, distributed systems have many advantages over centralized systems, including scalability, fault tolerance and performance. The most used architecture for distributed systems nowadays is called *peer-to-peer* (P2P) and provides a resilient solution for many applications. According to [Androutsellis-Theotokis and Spinellis 2004], there are two main characteristics of this type of system. The first one is the sharing of computer resources by direct exchange, which reflects the nodes capacity to independently execute tasks such as message routing and content location. The second one regards the resilience aspect provided by the fault-tolerance and self-organization mechanisms implemented in these systems.

The definition of peer-to-peer systems adopted in this work is the one proposed by the authors of [Androutsellis-Theotokis and Spinellis 2004]: a peer-to-peer system is a distributed set of interconnected nodes which have capabilities such as self-organization and failure adaptation while still providing connectivity, with the goal of resource sharing. Content distribution is one of the categories of applications for which peer-to-peer architectures are used the most, thus making a perfect match with the Context Broker platform considered in this work. Peer-to-peer networks are usually called overlay networks because they are implemented on top of the underlying physical computer network (which is typically IP) [Androutsellis-Theotokis and Spinellis 2004].

Peer-to-peer overlay network schemes can be categorized into two groups in terms of their structure: unstructured peer-to-peer networks and structured peer-to-peer networks. In this section, three main aspects will be considered: the structure of these networks, search mechanisms and replication.

3.1. Structure of Peer-to-Peer Networks

In structured peer-to-peer networks, nodes connect to each other according to a specific set of rules. Efficient routing of queries is achieved by using Distributed Hash Tables (DHTs), which are responsible for mapping the data objects to the peer nodes. In spite of providing scalability and efficient location of rare items [Lua et al. 2005], only exact-match queries are supported and a significant overhead is inserted due to the necessity of maintaining the network's structure when nodes leave or join [Androutsellis-Theotokis and Spinellis 2004], thus making this approach inappropriate if that happens very often. Examples of structured systems are Chord [Stoica et al. 2001] and Tapestry [Zhao et al. 2006].

In an unstructured peer-to-peer network, the connections between nodes are made in a random way, implying that no specific structure is formed. This type of network is usually utilized when the nodes join and leave the network frequently [Androutsellis-Theotokis and Spinellis 2004], since no information about the network is maintained and no restructuring is done. The disadvantage concerns the search for data. The location of content is completely independent of the content itself, therefore "brute-force" methods such as flooding the network with queries or more resource-preserving approaches such as random walks must be used [Androutsellis-Theotokis and Spinellis 2004]. Examples of unstructured systems are Gnutella [Gnutella 2017] and BitTorrent [BitTorrent 2017].

3.2. Search mechanisms

The mechanism used to find requested data and solve received queries has a huge impact on the performance of a context distribution system. Many techniques have already been proposed in the literature, but there is no perfect solution, since each one has its strengths and weaknesses. Therefore, a brief description of the main proposed approaches is given below, with the goal of selecting the one that best suits the needs of the Context Broker which this work refers to.

Let us first consider the case of structured peer-to-peer networks. The systems in this category are also called *DHT-based* because query routing is executed through the use of Distributed Hash Tables, in which information about data is stored. Keys

are generated for the objects and IDs are assigned to the peer nodes. Keys and IDs are both from a same identifier space, implying on a relationship between them. When an application wants to retrieve a data object with a corresponding {key, value} pair, the request is routed across the peers until the one that is responsible for storing that key is reached. Each peer has its own routing table containing a set of neighbor peers' IDs and addresses, so that it is able to know the node with the closest ID which the requests should be forwarded to, according to the key [Lua et al. 2005]. On average, the most famous systems of this type behave similarly in terms of performance, which is $O(\log N)$ [Androutsellis-Theotokis and Spinellis 2004].

Now let us focus on unstructured peer-to-peer networks. Since there is no relationship between the location of files and the network topology, a node that wants to find a file must query its neighbors. The most used technique for searching in unstructured peer-to-peer networks is flooding [Gkantsidis et al. 2005], which provides a good solution for a topology with few nodes. The problem of this technique is scalability. As the number of node increases, the time-to-live (TTL) needed to reach data also increases, thus generating large loads on the nodes and jeopardizing performance [Lv et al. 2002]. Selecting the appropriate TTL is also a hard task. Besides, flooding implies on the creation of duplicate messages due to the fact that a node may receive the same query from more than one of its neighbors. Therefore, duplication detection mechanisms are required when using flooding [Lv et al. 2002].

In an attempt to mitigate the problem of message duplication, a technique called *random walk* has been already proposed and tested in previous works, e.g., [Gkantsidis et al. 2005], [Lv et al. 2002], [Gkantsidis et al. 2004]. The technique consists in nodes forwarding the queries to a randomly chosen neighbor until the desired data is found. A modification of the random walk technique, called *random walk with lookahead*, was proposed by [Gkantsidis et al. 2005], in which a node executes short random walks with shallow floodings with a small TTL (typically 2). The results presented by the authors show that the number of unique nodes discovered when using random walk with lookahead is similar to the one obtained when using the traditional random walk. However, the response time is significantly smaller in the first method.

3.3. Replication

All the search mechanisms presented previously assume that some form of replication is implemented in the network. Content replication is of ultimate importance in peer-to-peer systems, because it increases content availability and provides better performance [Androutsellis-Theotokis and Spinellis 2004]. Three categories of replication approaches were proposed in [Androutsellis-Theotokis and Spinellis 2004] and relate to both structured and unstructured peer-to-peer networks:

- **Passive Replication:** corresponds to the case in which a node requests an object and makes a copy of it when the request is attended;
- **Cache-Based Replication:** In this category, copies of the requested object are made by every node through which the query message passes;
- **Active Replication:** also called *proactive replication*. Nodes replicate or migrate content to others according to a certain policy, even if no request involving that content has been made;

Due to space limitations, this subsection will focus on replication in unstructured peer-to-peer networks. [Lv et al. 2002] stated that, based on the research in [Cohen and Shenker 2002], the optimal method for replication is to replicate objects in a way such that $p \propto \sqrt{q_r}$ (p is proportional to $\sqrt{q_r}$), being p the number of replicas of an object and q_r the query rate of this object. This scheme is called *square-root replication* and provides minimization of the overall search traffic [Lv et al. 2002]. In order to achieve this scheme, the authors have proposed random replication, a proactive replication strategy. Considering a successful query, the number of nodes (p) between the requester and the provider is counted and (p) nodes that were visited in the random walk are selected to replicate the object [Lv et al. 2002]. Simulations show that random replication achieves results which are very close to the condition of *square-root replication*. Besides, it has been found that random replication performs better than path replication (another approach proposed by the authors) in terms of average number of messages per node [Lv et al. 2002].

4. Distributed Broker Design and Architecture

This work is based on [Crippa 2010] and [Crippa 2013], therefore it inherits the basic characteristics and uses the basic concepts presented in these previous works. This section briefly describes these definitions, analyzes the requirements that the platform must attend and also gives a first overview of the distributed Broker design, based on the theoretical foundation presented in the last two sections.

4.1. Context Entity and Context Scope

A Context Entity (or simply entity) is a subject which context data corresponds to [Crippa 2013]. Entities are composed by a **type** and an **identifier**. This identifier is the information used to distinguish a set of entities of the same type. One example of entity is a student enrolled in a university. Every student has a unique registration number among all the other students. In this case, *registration number* is the type and, for instance, *014632* is the identifier.

A Context Scope (or simply scope) is a group of related parameters which belong to a certain context [Crippa 2013]. Let us consider the example of a user who wants to log in to his e-mail account. The user needs both his **username** and **password** in order to execute this operation. Therefore, these two parameters belong to the same scope and are always manipulated together. Another observation is that scopes have a validity period with start and end. After the validity expires, data related to that scope is considered invalid [Crippa 2013]. Each entity may be related to one or more scopes.

4.2. Requirements Analysis

The requirements presented in this subsection are proposed based on (but not restricted to) the usage scenario of a large and spread sensor network in which context information (temperature, pressure, etc.) is gathered and a set of consumers are interested in receiving this data. The payload in this case is simple and small, but the number of exchanged messages may be high. The consumers contact a network of Brokers, which may be spread around different rooms, in order to obtain data. The goal is to establish a good compromise between performance, flexibility, ease of maintenance and resource usage.

- *Discoverability*: there must be a way through which providers and consumers are able to find at least one Context Broker;
- *Validity*: the content provided by the Brokers should be always up-to-date [Crippa 2010];
- *Availability*: the platform must implement some form of replication in order to provide appropriate availability of content;
- *Consistency*: all the components of the architecture must represent context using the same model [Crippa 2010];
- *Communicability*: Context Brokers should be able to communicate with each other in order to attend requests from consumers and providers [Crippa 2013];
- *Resilience*: the set of Brokers should be resilient with respect to nodes leaving and joining the network, in the sense of being aware of these situations and able to deal with them.

4.3. Broker Components

The Context Broker platform consists of three components: Context Providers (CPs), Context Consumers (CCs) and the Context Broker (CB) itself. This division implements the idea behind a Service-Oriented Architecture [Crippa 2013].

A **Context Provider** (or simply provider) is the component responsible for gathering data from sources (e.g., sensors) and sending it to the Broker, with a certain frequency, through a context update message [Crippa 2010]. Every provider must advertise its presence to the Broker before sending any data. A **Context Consumer** (or simply consumer) is the component which retrieves context data. In this version of the platform, the only way for a consumer to obtain data is by making a context request to the Broker.

The **Context Broker** is the main component of the architecture. It is responsible for storing and retrieving context information based on the requests of providers and consumers, acting as the communication manager in the system [Crippa 2013]. The Broker keeps a list of providers that are registered to it and is capable of sending data to a database for further uses. Considering a distributed architecture (such as the one proposed in this paper), a set of Brokers works cooperatively in order to store and retrieve information. Figure 3 shows the architecture of a centralized Broker with simplifications when compared to [Crippa 2010]. Acknowledgements (ACKs) and non-acknowledgements (NACKs) have been omitted.

The chosen model for context management is **Networked Services**, considering that a Service-Oriented Architecture is followed, in which clients (consumers and providers) connect to Brokers in order to make requests. Context data is acquired from a **context server**. Brokers act as servers which store data and consumers obtain context data from them. The context modeling technique used in this work is the **key-value** approach, based on the desired performance and simplicity for this implementation of the Context Broker. The platform design was thought in the scenario of applications that deal with small and non-complex payloads, with information such as temperature and humidity being gathered from sensors. Therefore, the other approaches would provide too cumbersome solutions with an unnecessary overhead, compromising performance goals. Context distribution is done through **queries**: consumers must query Brokers in order to obtain data.

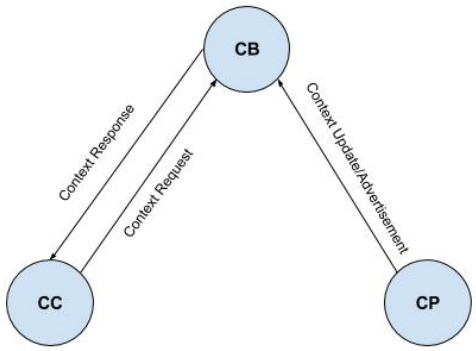


Figure 3. Centralized Broker Architecture

4.4. Overview of the Distributed Broker Design

This subsection indicates the decisions related to the main aspects of the distributed Broker design. These decisions were made based on the research presented in sections 2 and 3 and on the requirements analysis.

4.4.1. Node Joining/Leaving The Network and Search For Neighbors

A joining node receives a list of Broker addresses to connect. These Brokers are assumed to be more stable and the list is updated when necessary (e.g., a Broker went offline). After establishing this connection, the new node must connect to more brokers (e.g., 2). If it is not possible, the node does not join the network and keeps searching for other neighbors. The maximum number of connections should be relatively small (e.g., 20) in an attempt to limit the overhead of messages due to nodes leaving the network.

In the case of a node leaving the network, two possibilities are considered:

- Abrupt exit: detected when there is no answer from a node when data is requested. The node that detected the exit floods an advertisement message to the others with a small TTL (e.g., 5 hops).
- Normal exit: a node sends a message stating that it will leave the network. This message is spread using flooding with a small TTL;

The flooding technique was chosen for simplicity reasons and because it shows good performance when the amount of nodes to be covered is small [Gkantsidis et al. 2005]. In the situation of a node joining/leaving, only the nodes that are topologically close to the one that left need to know about the exit, in a way that future requests forwarded to nodes which are not online anymore can be appropriately detected. For the considered application scenarios, it is expected that not many nodes will keep joining and leaving the network, so the message overhead is not meaningful.

A node searches for neighbors based on a greedy protocol [Wang and Vanninen 2006]. Joining nodes must ping other nodes to discover latencies and a connection is established with nodes that have the lowest latencies. This approach was chosen taking into consideration the good results it has shown in terms of performance and generated network traffic, while avoiding a too cumbersome implementation in order to obtain network information [Wang and Vanninen 2006].

4.4.2. Search For Data and Data Replication

The search mechanism consists of random walks with lookahead (performing shallow floodings on each step of the walk). This decision was made considering that the traditional flooding approach would generate too much load in the network due to the amount of propagated messages. By choosing the random walks with lookahead method, it is possible to minimize message duplication and the granularity of the coverage, properties that are important according to [Lv et al. 2002].

According to the research presented in Section 3, the random replication mechanism performs better than path replication while achieving the *square-root* condition. Therefore, this approach will be used: for each successful search, the number of nodes p on the path between the requester and the provider is counted and then p nodes that were visited are randomly selected to replicate the object. The decision on whether certain data should be replicated or not is determined by a a small time threshold (e.g., 1 minute): if the duration of the data is lower than the threshold, no replication is performed.

One issue related to data validity and the system functionality in a general way is time synchronization in unstructured P2P networks. This issue is out of the scope of this work. It is assumed that the Brokers are connected to a reliable source of time information (synchronization by third-party), such as through the use of NTP.

5. Task Schedule

The tasks to be done in the second phase of the work (Graduation Work II) are listed below. Table 1 presents the task schedule for this next phase.

1. Research on related work for improvement of the theoretical foundation and detailing the design decisions of the platform;
2. Implementation of the platform prototype;
3. Experimental evaluation of the platform through simulations. Metrics such as memory usage, number of nodes visited and average response time will be considered.
4. Writing of Graduation Work II;
5. Presentation of Graduation Work II.

Table 1. Task schedule for Graduation Work II

Task	2017					
	Jul	Aug	Sept	Oct	Nov	Dez
1	X					
2	X	X	X	X		
3					X	
4		X	X	X	X	
5						X

6. Conclusion

This work presented a decentralized version of a context management and distribution platform called Context Broker using unstructured peer-to-peer networks. Related work

in the area has been reviewed and the design decisions have been proposed based on this theoretical foundation. Two key aspects regarding peer-to-peer networks have been considered: search mechanism and replication, which have a profound impact on the operation and overall performance of a distributed system. The next steps of this work are to provide a more detailed insight about the design decisions, implement a prototype and evaluate the platform through simulations, after defining the appropriate metrics. Small changes in the platform design might be considered. The final results will be presented in Graduation Work II.

References

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). *Towards a Better Understanding of Context and Context-Awareness*, pages 304–307. Springer, Berlin, Heidelberg.

Androutsellis-Theotokis, S. and Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371.

BitTorrent 2017. Bittorrent. `http://www.bittorrent.com`. Accessed in June 04, 2017.

Brown, P. J., Bovey, J. D., and Chen, X. (1997). Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64.

Chen, H., Finin, T., and Joshi, A. (2004). Semantic web in the context broker architecture. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications*, pages 277–286, Piscataway, NJ, USA. IEEE.

Chen, H. L. (2010). *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland.

Cohen, E. and Shenker, S. (2002). Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 177–190, New York, NY, USA. ACM.

Crippa, M. R. (2010). Design and implementation of a broker for a service-oriented context management and distribution architecture. Bachelor's thesis, Federal University of Rio Grande do Sul (UFRGS).

Crippa, M. R. (2013). Federation of brokers for a context distribution and management architecture. Master's thesis, Technische Universität Kaiserslautern.

Dey, A. K., Abowd, G. D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166.

Dey, A. K., Abowd, G. D., and Wood, A. (1998). Cyberdesk: A framework for providing self-integrating context-aware services. In *Proceedings of the 3rd International Conference on Intelligent User Interfaces*, IUI '98, pages 47–54, New York, NY, USA. ACM.

Gkantsidis, C., Mihail, M., and Saberi, A. (2004). Random walks in peer-to-peer networks. In *IEEE INFOCOM*, volume 1, page 130.

Gkantsidis, C., Mihail, M., and Saberi, A. (2005). Hybrid search schemes for unstructured peer-to-peer networks. In *IEEE INFOCOM*, Piscataway, NJ, USA. IEEE.

Gnutella 2017. Gnutella. `https://www.gnu.org/philosophy/gnutella.html`. Accessed in June 02, 2017.

Kiani, S. L., Knappmeyery, M., Baker, N., and Moltchanov, B. (2010). A federated broker architecture for large scale context dissemination. In *10th IEEE International Conference on Computer and Information Technology*, pages 2964–2969.

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *Communnication Surveys and Tutorials*, 7(2):72–93.

Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. (2002). Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing*, ICS '02, pages 84–95, New York, NY, USA. ACM.

Pascoe, J. (1998). Adding generic contextual capabilities to wearable computers. In *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, pages 92–99, Washington, DC, USA. IEEE Computer Society.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454.

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

Schilit, B. N. and Theimer, M. M. (1994). Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, New York, NY, USA. ACM.

Strang, T. and Linnhoff-Popien, C. (2004). A context modeling survey. In *First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004, Nottingham, England, September 7, 2004*.

Wang, J. and Vanninen, M. (2006). Self-configuration protocols for p2p networks. 4:61–76.

Winograd, T. (2001). Architectures for context. *Human-Computer Interaction*, 16(2):401–419.

Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., and Kubiatowicz, J. D. (2006). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communication*, 22(1):41–53.

Zimmermann, A., Lorenz, A., and Specht, M. (2005). *Applications of a Context-Management System*, pages 556–569. Springer, Berlin, Heidelberg.