Supporting Knowledge Management using a Nomadic Service for Artifact Recommendation

Pablo Gomes Ludermir

Thesis for a Master of Science degree in Telematics from the University of Twente Enschede, The Netherlands

Graduation Committee:
Dr. ir. A. T. van Halteren
R. Silva Souza Guizzardi, MSc.
R. Neisse, MSc.
Dr. E. Munthe-Kaas

Chandler: Hey Joey, where do Dutch people come from?

Joey: Uh... well the Pennsylvania Dutch come from Pennsylvania.

Chandler: and the other Dutch come from somewhere near the Netherlands right?

Joey: Nice try, see the Netherlands is this make believe place where Peter Pan and Tinkerbell come from.
Abstract

Knowledge Management (KM) can be defined as the effective strategies to get the right piece of knowledge to the right person in the right time. Having the main purpose of providing users with information items of their interest, recommender systems seem to be quite valuable for organizational knowledge management environments.

Early KM attempts relied on centralized knowledge bases and have shown a number of drawbacks. For instance, centralization forces users to agree in a common classification for the whole organizational knowledge. This disrespects the personal and distributed nature of knowledge. One approach to solve this problem is the usage of agents to represent actors on KM scenarios, showing effectiveness due to the autonomy, proactive behavior and sociability of agents.

Current mobile devices and wireless network technologies are able to offer information about their context. With that we can think of new kinds of reconfigurable services that take into account the user’s context, i.e. nomadic services. Nomadic mobile service provisioning is a promising new paradigm for service provisioning, where individuals can act as a service provider, offering services on an ad-hoc basis to other users. This paradigm opens up new possibilities for knowledge management systems and services.

This Master Thesis describes the design and implementation of KARe (Knowledgeable Agent for Recommendations), a multi-agent recommender system that supports nomadic users sharing knowledge in a peer-to-peer environment with the support of a nomadic service. Central to this work is the assumption that social interaction is essential for the creation and dissemination of new knowledge. Supporting social interaction, KARe allows users to share knowledge through questions and answers. Furthermore, we assume that nearby users are more suitable for answering his questions in some scenarios and we use this information for choosing the answering partners during the recommendation quest process.
Abstract v

Preface xi

1 Introduction 1
   1.1 Motivation ............................................. 1
   1.2 Objectives ............................................. 3
   1.3 Approach .............................................. 3
   1.4 Structure ............................................. 4

2 Supporting Knowledge Management 5
   2.1 Knowledge Management Definitions ..................... 5
      2.1.1 Data ............................................. 6
      2.1.2 Information ....................................... 6
      2.1.3 Knowledge ......................................... 6
      2.1.4 Knowledge Management ............................. 7
      2.1.5 Knowledge Generation .............................. 8
      2.1.6 Codifying Knowledge ............................... 9
   2.2 Knowledge Management Environments ................... 10
   2.3 Agents Support to Knowledge Management ............... 12
   2.4 Recommender Systems .................................. 12
Preface

This thesis presents the result of a final assignment for a Master of Science degree in Telematics. The assignment has been carried out in the Architecture and Services of Network Applications Group at the University of Twente.

I would like to thank Renata Guizzardi who adopted me not only as a student but as a close friend. Her guidance was very important to achieve the results of this thesis. Besides, without her support I would not be able to write the paper (\cite{30}) that shows the results of our algorithm and made my life much easier during the end of this assignment.

I would like to thank my other supervisors Aart van Halteren, Ricardo Neisse and Ellen Munthe-Kaas whose valuable suggestions helped me carry out this assignment. Their assistance was fundamental in writing this thesis.

I would like to thank also Diego Sona. He helped Renata and I to write our paper and his comments always came at good time and place. During the development of our prototype his suggestions always lead us to the right path.

I want to thank the Architecture and Services of Network Applications Group for giving me the opportunity to carry my master thesis there and to grant me the opportunity of defending the goal of the ASNA soccer team. I hope you can find a better goalkeeper after I am forced to “retire”.

I would like to thank all my friends in the ASNA laboratory for making the process of working in a thesis really pleasant and fun. They provided an excellent working environment during the last months.

I would like to thank all the people that made my life as a foreigner in The Netherlands (and Europe) more pleasant. In special, my “macandrian” friends who made my student housing a place like family. I cannot forget also my friends in Cottbus, because without them I would never thought of staying in Europe any longer. My memories from there
will follow me during my whole life.

Furthermore, I would like to thank all brazilian friends from Enschede. João Paulo, Patrícia, Giancarlo, Renata, Leonardo, Ricardo and Jaqueline were always there when I needed them.

Finally, I would like to thank my family in Brazil for the love and support and to deeply thank Flávia Figueiredo Machado, my future wife, for her infinite love, support and patience. Without her the completion of this thesis would never be possible.

I dedicate this thesis to Gabriel D’Annunzio Gomes.


Pablo Gomes Ludermir
CHAPTER 1

Introduction

This chapter presents the motivation, the objectives, and the structure of this thesis. In this thesis we present KARe, a system for artifact recommendation. Furthermore, it uses information retrieval techniques and a peer-to-peer network topology to solve knowledge management issues for nomadic users.

This chapter is structured as follows: Section 1.1 presents the motivation for this work. The section 1.2 states the thesis’ objectives, section 1.3 presents the approach adopted to develop this thesis and finally section 1.4 outlines the structure of this thesis by presenting an overview of the chapters.

1.1 Motivation

Knowledge has been recognized as one of the most important assets of competitive businesses [20]. However such an asset is usually highly distributed and hard to find because it is related with an individual’s actions and experience, ideals, values and emotions. In [20] knowledge is classified in two distinct types: tacit and explicit. Tacit knowledge is understood as the kind of knowledge the personal that is context-specific, hard to formalize and communicate. Explicit knowledge, on the other hand, is “codified” in formal and systematic languages.

To maintain its own knowledge, i.e. the organizational knowledge, an organization must give support to the individuals involved in it, to convert tacit knowledge into explicit. More specifically, knowledge can be spread throughout an organization by means of externalization, i.e. the conversion of tacit knowledge into explicit, and internalization,
i.e. the conversion of explicit knowledge into tacit.

However, even if an organization has the means to externalize and internalize knowledge, one still has to find who owns the knowledge itself. An effective process for finding knowledge usually lies among communities of practice (CoP). CoPs are groups of people that are formed based on similar interests, personal affinity and trust. In order to develop a CoP, one must support it by providing the necessary means, i.e. the technology and the social dimensions. In the social dimension, community members can, for instance, be rewarded and remembered. As for the technological support, it is important to note that an appropriate infrastructure needs to be provided to facilitate knowledge sharing.

This thesis aims at providing a technological infrastructure for members of a CoP to share knowledge in a distributed fashion. For that, we rely on a peer-to-peer network topology. Peer-to-peer networks (p2p) provide technological support to discover people, and thus CoPs.

Lately p2p networks are used as a means for interaction between software agents. Agents may represent our own interactions in the real world with more accuracy than objects, because agents are reactive entities, able to exchange messages and adapt their behaviour and state according to the communication with each other. Furthermore, we can represent organizations and CoPs by groups of agents.

A problem with p2p networks is to select peers to collaborate with in a scalable way. A user (represented by a peer in the p2p network) seeking recommendation would like to select as many peers as possible. However, he will not like to receive bad recommendations. Thus, there should be an approach for identifying and selecting suitable peers for collaboration which is not currently tackled by generic p2p networks. The approach taken by p2p networks is to spread a question as much as possible, preferably to all users that are connected.

In this thesis, we propose KARe (Knowledgeable Agent for Recommendations), a p2p recommender system that relies on the natural processes people use to exchange knowledge, providing recommendations to users based on their interest and expertise. We develop KARe as a nomadic mobile service. Nomadic mobile service provisioning is a promising new paradigm for service provisioning, where individuals can act as service providers, offering services on an ad-hoc basis to other users.

This paradigm opens up a new possibility for a knowledge management service. It is desirable to explore the possibilities of short-range network to propose a new architecture to offer new nomadic services that can go beyond the processor/memory limitations of mobile devices. In this way, such devices can be more vastly used in human activities such as knowledge searching, sharing and exchanging. In this sense we can create a knowledge management service that provides more tailored knowledge to user needs. We are based on the assumption that nearby users would be more suitable for answering his questions.
To provide recommendations, KARe relies on Information Retrieval (IR) techniques, based on the classical vector model [35]. Our recommendation algorithm is based on the vector model and uses semantic information about the recommendation artifacts to provide more precise recommendations to the user. Both the proposed IR technique and the use of mobile technology bring about a number of challenges. First, building the IR algorithm is not a trivial task and many problems may rise in the process, such as finding the right abstraction to represent the knowledge contained in the knowledge base, and guaranteeing a reasonable performance for the algorithm.

Here we also target the problems that result from the use of mobile devices and the targeted network technologies (Bluetooth, WiFi, etc.), such as: a) Finding service providers that offer a knowledge management service that are in the range of the users location, b) Using the service provider’s location information to retrieve knowledge from the different devices and c) To overcome resource limitations on mobile devices specially with data-intensive applications like a recommender system.

1.2 Objectives

The first goal of this work is to provide a tool to support knowledge management. This work aims to develop a distributed system for artifact recommendation named KARe [30, 4]. Our main contribution here is the development of a recommendation algorithm that includes semantic information about the recommended knowledge. With this algorithm, we aim at increasing the precision of current recommendation algorithms. Second, we are interested in exploring how mobile users may get recommendation (through questioning and answering) from their surrounding peers through portable devices. A third goal is to design an architecture that copes with the current resources (processing and communication) limitation of mobile devices, and that enables the communication of software agents on top of it in a decentralized approach.

1.3 Approach

To make possible the development of KARe the efforts on investigation include:

- To study available agents development frameworks;

- Design an agent architecture using the Agent-Object-Relationship Modeling Language (AORML). This architecture should be tailored to explore the autonomy and mobile characteristics of the agents to assist our KM process;

- Design and implement an agent-oriented mobile/nomadic recommender service;
• To design and implement an algorithm that uses IR vector model including semantic information;

• To collect data to perform an experiment on the algorithm and evaluate it according to its precision and recall.

1.4 Structure

The structure of this thesis reflects the order in which these issues have been dealt with throughout the research process. This thesis is structured as follows:

• Chapter 2 presents the literature study conducted to motivate this work. Knowledge Management (KM) issues and ways for supporting KM are addressed in this chapter;

• Chapter 3 identifies essential requirements to be satisfied by KARe to properly support knowledge management to nomadic users. These requirements are later referenced on the design phase;

• Chapter 4 describes a layered architecture of our system design. We present the agent-oriented design of KARe and the design of our vector model with semantic information and the nomadic service for artifact recommendation;

• Chapter 5 describes the implementation of the system;

• Chapter 6 shows the results of our experiment to evaluate the vector model with semantic information;

• Finally chapter 7 presents the final conclusions and remarks and indicates topics for future work.
CHAPTER 2

Supporting Knowledge Management

This chapter gives the motivation and background to our work. For that it is necessary to conduct some literature study about knowledge management, agent-mediated knowledge management, recommender systems and information retrieval.

This chapter is structured as follows: Section 2.1 provides definitions of knowledge and knowledge management. Section 2.2 describes intrinsic characteristics of knowledge management environments that motivate us to think of an agent-based solution to solve KM problems. Section 2.3 shows how agents can play a role in KM settings by explaining in which consists Agent-Mediated Knowledge Management. Section 2.4 gives a summary of the available classifications of recommender systems. Section 2.5 shows a general architecture for an information retrieval system. Finally, on section 2.6 we link these subjects in a discussion that serves as the motivation for our work.

2.1 Knowledge Management Definitions

The concepts of data, information and knowledge are often confused with each other. To understand knowledge and knowledge management we need to understand what data and information are, what their differences are and how they are related with knowledge.

Data, information and knowledge are not interchangeable concepts. Organizational success often depends on knowing which of them is needed and which of them the organization has [38]. It should be known also what the organization can and cannot do with each of these concepts to transform them into assets and create a competitive advantage [38 p.1]. These concepts are defined in the following sub-sections.

5
2. SUPPORTING KNOWLEDGE MANAGEMENT

2.1.1 Data

In an organizational context, data is considered the raw material in the production of information such as name, address, date and so on. Data does not say anything about the context where it is created and used having, thus, little relevance or purpose when left alone in a database.

Organizations, however, are very dependent on data and try to manage them with database systems to help decision making processes. Lately, people are often facing a problem of excessive amount of data which makes it harder to identify what actually is important to keep [38, p.3].

2.1.2 Information

Information is data that has been analyzed and interpreted, i.e. data are transformed into information by adding value in various ways. Information is data with relevance and purpose [38, p.4]. The way the receiver perceives the information can create an impact on his judgment and behavior. Thus, information provides a new point of view for interpreting events or objects, making visible previously invisible meanings [20, p.58].

Once data is transformed into information it can be used on decision making. For that, there are methods to interpret events or objects in which this transformation is enabled such as [38, p.4]:

- **Contextualization:** The purpose of data gathering is known;
- **Categorization:** The units of analysis or key components are known;
- **Calculation:** The data is analyzed mathematically or statically;
- **Correction:** Errors are removed from the data;
- **Condensation:** Data is summarized.

2.1.3 Knowledge

Knowledge is “a fluid mix of framed experience, values, contextual information, and expert insights that provides a framework for evaluating and incorporating new experiences and information.” [38, p.5].

Furthermore, knowledgeable people usually are specialized in their field. Knowledge is also related to belief and commitment, i.e. it is a function of a particular instance, perspective or intention. It is about specific action, context and meaning [20, p.58].

Using this point of view knowledge can be categorized into explicit and tacit [20, p.8]. Explicitly knowledge is something formal and systematic that can be expressed in words
and numbers, and thus easily communicated and shared in form of hard data, scientific formulae, codified procedures or universal principles. Tacit knowledge, on the other hand, is not easily visible and expressible. It is highly personal and hard to formalize, making it hard to communicate or to share with others. The tacit knowledge is deeply related with individual’s actions and experience, as well as ideals, values, or emotions.

Tacit knowledge can be further classified in two dimensions [20, p.8]: technical and cognitive. The first consists of informal skills a.k.a. “know-how”. It is when someone develops the ability to perform something but they cannot explain it with scientific or technical principles. The second dimension is called the cognitive dimension. It consists of mental models, beliefs and perceptions so integrated that people take them for granted. This dimension reflects people’s image of reality and their vision of the future, i.e. how they perceived the world.

2.1.4 Knowledge Management

With solid conceptualization of data, information and knowledge we can now define knowledge management. Knowledge management is about creating an environment where critical information is created, structured, shared, distributed and used [39]. Some authors specialize this definition saying that KM is an organizational process for acquiring, organizing, and communicating knowledge of employees [24].

Knowledge management (KM) is seen as an approach to maintain and handle the organization knowledge assets at different levels (individual, group, organizational and intra-organizational) [25]. Early KM attempts relied on centralized knowledge bases and have shown a number of drawbacks. For instance, centralized knowledge bases force users to agree in a common classification for the whole organizational knowledge, disrespecting its personal and distributed nature. To tackle these drawbacks, agents were used to represent the actors on KM scenarios and showed effectiveness due to their autonomy, proactive behavior and sociability [25, 29].

Knowledge management is considered an organizational asset because it can reduce the learning curve and/or increase the ability of people to assimilate new technologies, filling the gaps in a company’s knowledge [39].

The basic activities of knowledge management are the identification, acquisition, development, dissemination, use, and preservation of the enterprise knowledge [24, 39]. Further, a successful KM solution should consider not only the technological and management points of view, but also the combination of organizational culture and tools that support it. For instance, some aspects such as lack of trust and time, narrow view of productive work, intolerance to mistakes and so on influence the knowledge flow within the organization. For that they such aspects should be taken into account when adapting a KM solution.
2. SUPPORTING KNOWLEDGE MANAGEMENT

2.1.5 Knowledge Generation

Much has been written about the importance of knowledge in management, however little attention has been paid to how knowledge is created and how the knowledge-creation process is managed [20]. A healthy organization generates and uses knowledge all the time to keep and improve their business [38, p.52].

KM literature classifies knowledge generation in many ways. For instance, in [38, p.52], the process is classified in five: acquisition; dedicated resources; fusion; adaptation; and knowledge network. However, Nonaka and Takeuchi [20, p.57] claim that knowledge creation is based only on two dimensions: epistemological and ontological.

The epistemological dimension is concerned with the distinction between tacit and explicit knowledge and the ontological dimension is concerned with the levels of knowledge creation entities, i.e. whether the knowledge is in the individual, group, organizational or intra-organizational level. The interaction between tacit and explicit knowledge is the key for the knowledge-creating process and it reaches all the levels of ontological dimension. Further, four modes of interaction are possible, as shown in figure 2.1.
Combination: From Explicit to Explicit

Combination is the process of converting knowledge from different kinds of explicit knowledge. Explicit knowledge can be shared in meetings via documents, e-mails, education and so on [2]. The use of technology is important to manage and search collections of explicit knowledge [20, p.68]. To improve the collected information the employee could reconfigure it, i.e. making it more usable with the use of data warehouses [2].

Externalization: From Tacit to Explicit

Externalization is the process of articulating tacit knowledge into explicit concepts [20, p.64]. The transformation of tacit knowledge into explicit is often full of gaps, however, it helps to promote reflection and interaction between individuals. Furthermore, externalization shapes metaphors, analogies, concepts, models’ hypotheses of each person’s tacit knowledge [20, p.65]. This process holds the key of knowledge creation because it creates new explicit knowledge from tacit knowledge.

Internalization: From Explicit to Tacit

Experiences such as socialization, externalization, and combination are internalized into individual’s tacit knowledge bases in the form of shared mental models or technical know-how [20, p.69]. Individuals have to experiment and to understand the knowledge acquired to internalize and create their own tacit knowledge. When acquiring knowledge from many sources, individuals have the opportunity to create new knowledge by combining their existing tacit knowledge with the knowledge from others [2]. However, this process is becoming more challenging since individuals have to deal with large amounts of information.

2.1.6 Codifying Knowledge

Social interaction has been recognized as the driving force behind the creation of knowledge [20]. Knowledge is created as a result of a transformation cycle between explicit and tacit knowledge and, for that, social interaction is not only necessary but essential, as can be noted in the previous section, where we described all four knowledge conversion processes.

According to Paulo Freire [31], a question is the first knowledge sparkle, as questioning is a means to explicitate one’s personal knowledge, starting with a reflection on what one knows and what one does not know [31]. In addition to that, questioning provides an opportunity for others to express their points of view, many times tacit. In other words, such process allows the explicitation of tacit knowledge, allowing it to be shared.

Codification makes knowledge available throughout the organization. The main
advantage in turning knowledge into some explicit form is the possibility to organize, categorize, search, and to facilitate its understanding for the members of the organization. However, the codification process is a challenging task.

Employees quickly create large amounts of information, and attempts to fully capture knowledge are often frustrated [9, p.16]. Even when only a subset of the generated knowledge is codified, when it becomes available normally it is already obsolete. Thus, collected information cannot be considered an asset if it is hard to find and is not leveraged to meet customers’ needs. To meet these principles, there are four basic principles to be followed to codify knowledge [38, p.68]:

1. The goals that the codified knowledge will serve should be defined;
2. Knowledge workers should be able to identify existing knowledge in various appropriate forms to reach these goals;
3. Knowledge managers should evaluate knowledge by its usefulness and appropriateness for codification;
4. Knowledge codifiers should identify an appropriate medium for codification and distribution.

Once the purpose of codification of knowledge is to put it in usable form, the organization needs to have some idea of the usage for such knowledge [38, p.69].

Tacit knowledge is the hardest to extract and codify due to its very personal nature, and to separate tacit knowledge from individual actions is a challenging task [38, p.70]. The main problems to capture and codify tacit knowledge are: 1) the access to tacit knowledge is available only when the owner has time to share it; and 2) when the knowledge owner leaves the organization, there is a threat of losing knowledge capital. Since there is no systemic solution for that, organizations usually overcome this situation by rewarding knowledge workers as an attempt to keep them.

2.2 Knowledge Management Environments

As an organization develops, its knowledge and expertise becomes increasingly distributed [38]. While this process promotes the growth of specialized knowledge communities, it also makes discovering relevant knowledge from these communities more difficult.

Organizations rely in two ways for seeking KM support. A common way to target this problem is to create a central repository to which all organizational members are asked to contribute with their own personal knowledge. These repositories may contain different documents used in the work routine, but also specialized records aimed at capturing the more tacit kind of knowledge, such as reports on lessons learned and best practices gathered
by workers’ experience in different projects [8]. One problem with this approach is to get users’ acceptance to spend extra hours on feeding the system, without knowing who will use his/her knowledge and for which purpose. Besides, employees have no guarantee of finding useful information in the repositories when they need it [7]. Thus, such approach presents some drawbacks for its acceptance such as [25]:

1. Centralized approaches ignore the distributed nature of knowledge within the organizations. Besides geographic distribution (e.g. knowledge that is sparsely distributed because it belongs to a global organization), knowledge could also be distributed by employees’ specializations. Each level of the organization implies some abstract level of specialized tacit knowledge.

2. The definition of a global classification scheme (such as a centralized database model or ontology) hides the particular classification of knowledge that employees have on the different levels of the organization, e.g. an employee usually has a more precise classification of a concept than the organization as a whole if he/she is a specialist on the topic. On the other hand, another employee could feel overwhelmed with some knowledge classification and needs a more shallow description when he is a layman.

3. KM does not serve the operational and business goals directly, i.e. KM practices are usually detached from the employees’ tasks. Thus, employees are not tempted to invest time to feed and extract knowledge from a repository. This makes knowledge bases obsolete and seldom used.

4. KM solutions do not give “ready-made recipes” to solve employees’ problems. There are usually few descriptions of how to execute or solve a task/problem that has not been seen before. Furthermore, when an employee faces a new problem he or she wants to be seen as the specialist on solving that rather than sharing it with others.

5. KM solutions should adapt to changing environments. Since they usually require a great investment, to pay off they must take into account the changes that occur in the organizations. In this sense, companies are encouraged to implement KM into their processes. Centralized solutions tend to have a very high cost when compared to small decentralized solutions being, thus, less adaptable to changes.

Workers’ dissatisfaction many times leads KM systems to be abandoned, while people continue relying on their natural ways of finding knowledge, such as asking for the help of colleagues that are part of their circle of trust.

Opposite to centralized approaches, the distributed KM paradigm [27], provides autonomous and locally managed knowledge sources organized in a peer-to-peer community. Peer-to-peer technology supports the horizontal relationship between people, seeing them as both consumers and providers of knowledge [3]. Each peer controls his own personal
knowledge artifacts and exchanges knowledge with other peers based on, for example, their common interests, roles, expertise, and trust. These same factors are the main motivators for the formation of communities of practices [13], which may strengthen the relationship between peers, leading to more effective knowledge creation and sharing.

2.3 Agents Support to Knowledge Management

Agents have been acknowledged as adequate metaphors to represent the actors involved in Knowledge Management settings (see for example [34], [25], [39] and [30]). They are well suited for such environments because of features as autonomy, social abilities, reactiveness and proactiveness (29, [19]).

Any KM system can be composed of multiple software agents to mediate the creation, integration and sharing of knowledge. The main services of KM (search, acquire, analyze, integrate, archive and present knowledge information from multiple sources) can be provided by agents, thus saving user’s time. Furthermore, agents could be used to model large and complex organizational settings because of their flexibility characteristics. Current agent-based systems for KM are classified as the types of services they provide such as [39]:

- **Personal Assistants**: These agents represent the user interests and are responsible for acting as an interface between the user and system. Personal assistants are used not only to provide what the user needs, but also to suggest knowledge sources and resources that are not directly requested by the user;

- **Cooperative Information Agents (CIAs)**: These agents are responsible for searching for information in different knowledge sources and to communicate with other agents about their own information;

- **Task analysts**: These agents monitor important business process tasks to determine the knowledge required to perform them. They communicate with other agents to provide the right knowledge to the user;

- **Source keepers**: These agents maintain, describe and extract information requested by a user or another agent;

- **Mediators**: These agents mediate services between agents, e.g. they may indicate users for collaboration, suggest tools, knowledge sources and documents.

2.4 Recommender Systems

One way to provide agent-mediated knowledge management (AMKM) solutions is through recommender agents systems. Here, we provide an overview and classification of such sys-
tems. Recommender systems aim at assisting users on searching, sorting, classifying, filtering and sharing information [28]. This directly relates to the objectives of KM systems, i.e. to create, structure, share and distribute knowledge [39].

In both KM and recommender systems, there are two common facts: 1) users are usually confronted with many options to choose from; and 2) both are getting support from the agent community. Recommender systems use agents to provide recommendations to users using personal agents/assistants (see 2.3). We should summarize the main methods used by recommender systems (c.f. [28]) to serve as a foundation to build our own solution. The kind of recommender system to be developed depends of the application requirements.

The initial step to begin using a recommender system is to create an initial user profile. This must be created so that the user’s agent will have information to provide recommendations. There are techniques to generate an initial user profile and update it by learning from the user’s behavior. When a profile is available, agents have to explore it to recommend items to a user. The exploitation process tries to match profile information with data about the items to be recommended.

2.4.1 Profile Generation and Maintenance

Five decisions must be taken for generating and maintaining a user profile [28]:

- The representation format of the user profile;
- The approach for creating the initial profile;
- The location where to get recommendation relevance feedback;
- The technique to learn from the profile and;
- The technique to update the profile over time.

The most common form of profile representation is to keep the user’s history. Most e-commerce systems use such representation to provide recommendations [28]. Another way to represent the profile is by using the information retrieval vector model, (see section 2.5.3). In this case, profile items are represented as vectors and the dimensions of the vectors correspond to presence or absence of certain features defined by the system.

During the creation of the initial profile, the system must balance two parameters: On one hand, the system needs as much initial information as possible to provide good recommendations from the very beginning. Thus, the personal agent must collect enough information before recommending any item; on the other hand, if such process takes a long time from the user he/she will not feel encouraged to spend time doing it. There are four possible ways to generate an initial user profile: 1) leaving it empty; 2) manually
assigning the user’s interests in form of keywords; 3) creating initial classes of users and thus stereotyping the users according to such classes and 4) training the system.

Once the initial profile is built, the system must find a source of information to update it. The choices usually are to update it with on-line information during transactions or to use other sources such as textual information or clustered data [28]. Either way the information is evaluated using information retrieval techniques (see section 2.5.1). The user might also provide relevance feedback on the recommendations given to maintain his profile. The feedback can be explicit, i.e. users are required to evaluate the recommended items; or implicit, i.e. the system monitors the user’s behavior and tries to infer his new preferences [28].

Besides the feedback, recommender systems should also balance old and new information. Since new information is most likely to represent the user’s preferences precisely, it is important to “forget” the old input to the user’s profile over time [28]. It can be a manual task, i.e. the user deletes the information he does not consider new anymore; or a gradual function to erase data, (e.g. using a “time windows” solution). The recommender system might also choose only to add new information and never forget the old.

2.4.2 Profile Exploitation

There are three methods to explore user profiles [28]: 1) demographic filtering; 2) content-based filtering and 3) collaborative filtering.

Demographic filtering tries to relate people with an item based on the characteristics of people who like it. The user profiles are created using classes of users (i.e. stereotypical descriptions) and the system tries to learn the “type” of people who liked one item to recommend it. Thus, it generalizes the users interests and as a consequence people from the same “type” will get very similar recommendations. Furthermore, there is very little individual adaptation to interest changes.

Content-based filtering recommends items based on what the user has evaluated before. So, the system will recommend more items similar to what the user has already liked and few different items will be recommended. When using this filtering method, an item-profile matching must be performed to recommend new items that could be interesting to the user. For that, the recommender agent could use 1) a list of keywords and match them against the user profile or 2) define the items as vectors and use the cosine similarity to match them against the profile (see section 2.5.4).

Finally, collaborative filtering matches users with similar interests and tries to recommend items based on the user’s preferences. For that it uses statistical analysis of items that user’s with similar preferences have liked. Thus, items with few ratings will be less recommended, since not many users have liked it yet. Furthermore, a user with different taste from the rest of the community will get bad recommendations since there
are not many other users with whom to compare him. The process consists of three parts: finding similar users; creating a neighborhood of users and making predictions based on the selected neighborhood.

To balance content-based and collaborative filtering drawbacks, recommender systems often rely on hybrid approaches.

2.5 Information Retrieval Systems

Information retrieval techniques are used to support recommendation agents. Retrieving information is an exploratory task done by a set of components that perform common processes. Such components have standard roles in information retrieval systems but can be implemented in different ways. Information is usually represented as a set of documents, which are defined as [35]:

A single unit of information, typically text in digital form, but it can also include other media. It can be a complete logical unit, like a research article, a book or a manual. It can also be part of a larger text, such as a paragraph or a sequence of paragraphs. A document can be any physical unit, for example a file, an email, or a web page.

Document construction is data input that obeys a certain syntax which together produces a semantically rich entity (the document) that is accessed via its presentation style. Thus, a document can be viewed in different perspectives and each of these perspectives is important to a group of people. For instance, someone could be interested on the document’s syntax for its automatic processing, when someone else would be concerned on its presentation for the users. A third perspective, the semantics, is what people perceive as the information that is embedded on the document.

Information retrieval systems (IRS) usually follow a general structure, as seen in figure 2.2. The main components of an IRS are: 1) the user interface; 2) text pre-processor which consists of a set of operations over the text to prepare it for searching and indexing; 3) the searching mechanism, that retrieves documents from the index based on the user input; 4) the ranking function, which returns the results in the best order as possible; and the 5) indexer which processes, stores and maintains the documents in a database for later retrieval.

In the following sub-sections we discuss about the three main components: the text pre-processor (section 2.5.1), the indexer (section 2.5.2) and the searcher (section 2.5.3). Further, we summarize the classic modeling approaches for IRS (section 2.5.4) and their evaluation mechanisms (section 2.5.5).
2.5.1 Text Pre-processing

To create the vocabulary of a collection (i.e. the most representative terms of the collection), we must select all the index-terms that are relevant to represent our collection and include them in the vocabulary index. This implies some text operations to select the index terms. Mostly noun terms are selected because these usually carry the semantics on a sentence. Text pre-processing is thus a necessary step to improve the performance of any IRS. The most common operations performed over text are lexical analysis, elimination of stopwords, and stemming.

Lexical Analysis

The objective of the lexical analysis is to treat digits, hyphens, punctuation marks and the case of letters. This leads to the identification of individual words in the text, so that with the words in hands one can disregard the ones that are not good index terms for searching (such as numbers). This procedure includes often the removal of punctuation and hyphenation, and sometimes even the case of the letters.
2.5. INFORMATION RETRIEVAL SYSTEMS

Elimination of Stopwords

Stopwords are words too common in documents (e.g. articles, conjunctions, prepositions). They are usually filtered out because they are useless for information retrieval purposes, since they create very little differentiation among documents. An advantage of the elimination of stopwords is that it reduces the size of the index, so the system can perform the search faster.

Stemming

Stemming is the process of reducing a word to a common root, i.e. reduce word variants to a common concept. This could be achieved by removing prefixes and suffixes to a “stem”. So, if we have only stems instead of words (and their variants) in our index, the search can be more precise and faster. There are several types of stemming such as table lookup, successor variety and affix removals.

2.5.2 Indexing

It is very costly to handle a user’s query by searching the text in the whole document database, especially when collections tend to grow. Instead of using the actual text for searching a query, an IRS uses data structures. This is an appropriate choice when searching in static or semi-static collections. Only if the IRS is used for a dynamic collection one could think of combining the two approaches, i.e. a small database for searching the new documents (while they have not yet being indexed) and a large index for searching the old ones.

Almost every IRS uses the inverted file index data structure [35, ch.7] (see figure 2.3). There we see index terms on one side and document/occurrences tuples on the other side. The index terms consist of a set of words selected during the text processing. Next to each index term there is a list of tuples in the format \( \text{(Document; Occurrences)} \). For each word on the database we can directly retrieve the documents where it is located and know how many times it has appeared in each document.

2.5.3 Searching

As we have seen in Fig. 2.2, searching occurs in three steps: 1) query handling; 2) the search itself and 3) ranking of the results. The query input is processed the same way as the documents that are indexed on the system. The aim is to have the same representation of the query as we have on the index.

The searching and ranking mechanisms depend on the kind of information retrieval model the system was built upon. On section 2.5.4 we see the three classic approaches to
search and rank documents. Most search engines nowadays use full text search, meaning that documents are retrieved if they are “linked” to a word from the query on the index terms set. A more sophisticated approach is to model documents and queries as vectors and check the similarity amongst them using a cosine function (see section 2.5.4).

2.5.4 Information Retrieval Modeling

A model is a representation of real world set of entities and their relationships. With such representation one is able to simulate problems and solve them by analysis. Thus, for Information Retrieval Models we understand that on one hand we have “information” and on one hand we have the different approaches to retrieve it.

More specifically, “information” refers to the logical view of a collection of documents and the queries that form the input of a system. The approach to retrieve the information would be a framework that models the documents and queries into its logical forms and a ranking function that orders the document set according to queries. Thus, an Information Retrieval model is a quadruple \( \{D, Q, F, R(q_i, d_j)\} \) where:

- \( D \) is the document set. The documents are represented as the elements of the set \( D : \{d_1, d_2, \ldots, d_j\} \);
- \( Q \) is the user query;
- \( F \) is the framework to model documents and queries into a logical form;
- \( R(q_i, d_j) \) is a function to rank documents according to a particular query.

There are different ways to approach the modeling “framework” and the “ranking
2.5. INFORMATION RETRIEVAL SYSTEMS

function”. This results in different models for Information Retrieval such as: the boolean model, the vector model and the probabilistic model [35] [10]. In this work, we focused on the vector model and briefly described the other approaches.

**Boolean Model**

The boolean model is the most intuitive. The queries are represented by boolean expressions and the documents are classified as being relevant or not based on the query [35].

This approach thus does not consider documents that only match partially as relevant, since boolean expressions are not flexible. In systems that apply this kind of model, the creation of queries is usually simple but usually leads to large response document sets with lots of irrelevant documents.

There is then a balance between query simplicity and document relevancy on the response set because to find more relevant documents, the user must know more about the information to be extracted and refine the queries. This is usually an iterative process and its success depends on the user’s experience on the topic.

**Probabilistic Model**

Here there is an assumption that there exists an “ideal answer set” for a given query. Therefore the model aims to rank the documents by the probability of their relevance given a particular query. The query is composed of the properties of the documents on the ideal answer set, thus the model works iteratively. In each iteration the user is requested to evaluate the retrieved document set in order to refine the query. After that, the process is repeated with a refined query.

The ideal answer set is the document cluster when the relevance probability of the document set is maximized. Although it seems to be an effective model for information retrieval, some aspects are left unanswered such as how to compute the probabilities [35].

**Vector Model**

In the vector model for information retrieval [35], documents and queries are treated as real algebraic vectors where the dimensions of the vectors is determined by the dimension of the vocabulary (i.e., made of all terms for the given domain). Therefore, once the vocabulary has been determined (i.e. the text pre-processing determination of index-terms), all documents are represented by vectors. Each dimension of the vectors is calculated based on the frequency of each index term in each document itself. Having this vectorial representation, it is quite easy to determine the similarity between couples of documents or between a document and a query.
In figure 2.4, the depicted vectors are the abstraction of a query (\(Q\)) and any particular document (\(d\) from a set of documents \(D\)), and the angle \(\theta\) indicates how close these vectors are, thus indicating the similarity between the document and the query. Our algorithm calculates this similarity using a very common approach, i.e. the cosine of the angle \(\theta\). As a consequence of being a cosine, the result of the similarity function varies from 0 to 1 in an ascending order of vectors similarity. Equation 2.1 describes a cosine similarity between vectors which is a measure commonly used in information retrieval.

\[
similarity(\vec{d_j}, \vec{Q}) = \frac{\sum_{i=1}^{t} w_{i,j} \cdot w_{i,Q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{i,Q}^2}}
\]  

(2.1)

Here, \(w_{i,j}\) is the weight of the index terms of document \(\vec{d_j}\) and \(w_{i,Q}\) is the weight of the index terms of the query \(\vec{Q}\).

Figure 2.4: Similarity is given by the cosine between \(Q\) and \(d_j\)

Each dimension value of the vectors is computed based on the frequency of the index term in question on the document collection. There are several ways to compute the dimension’s weight and this process is called index term weighting. For this work, we chose the \(TF \times IDF\) method \[35\] (see Equation 2.2). This method computes the weight in two steps. First we calculate the term frequency (\(TF, \text{represented by } f_k\)) of a particular index term in each document. Secondly, we calculate the amount of documents that contain the term, i.e. inverse document frequency (\(IDF, \text{represented by } n_k\)). The aim here is on one hand, to increase the weight if a term is very popular in a document and on the other hand, to penalize the weight (i.e. to decrease its value) whether the term is present amongst many documents.

\[
w_k = f_k \times \log \frac{N}{n_k}
\]

(2.2)
2.5.5 Information Retrieval System Evaluation

There is no absolute measure that evaluates how good an information retrieval system or algorithm is. Instead, algorithms are measured according to their precision and recall and compared with each other. To find a number that can be compared, we calculate the F1 measure [35], which is an average of the precision and recall results. It is important to remember that to compare two systems one must use the same set of documents and queries.

**Precision**

Precision is the relevance measure to the searcher of the items that are retrieved, i.e. if a search returns ten documents of which nine are very relevant, that search has high precision. This measure is more important when the users of the system prefer an approach that retrieves a specific document very fast.

\[
\text{Precision} = \frac{\#\text{RelevantDocuments}}{\#\text{RetrievedDocuments}} \tag{2.3}
\]

The precision, as we see on equation 2.3 measures the ability of the system to refuse the non-relevant documents.

**Recall**

Recall is the proportion of relevant information that is retrieved by a search, i.e. if a search only retrieves one hundred relevant documents out of three thousand that are available (and relevant), that search has low recall. On the other hand, if it retrieves all the available documents on the topic of the search, it has high recall. This measure is, thus, dependent on the documents of the collection. This measure is more important when the users of the system prefer an approach that retrieves the highest number of documents around a specific topic, but are not concerned on finding one specific document quickly.

\[
\text{Recall} = \frac{\#\text{RelevantDocumentsRetrieved}}{\#\text{TotalRelevantDocuments}} \tag{2.4}
\]

As we can see on equation 2.4, the recall measures the ability of the system to retrieve the relevant documents.

2.6 Discussion

Knowledge is perceived as an asset to the organizations because it can bring competitive advantage [38, p.12]. To hire new employees, companies usually prefer the ones with more
experience and knowledge besides higher education because they understand the value of knowledge created over the time and experiences. Furthermore, knowledge creates competitive advantage because it is an asset difficult to copy and sustain since it considers ideals, values, emotions, and experience of the organizational members [20, p.10]. Also, this is the basis to generate new knowledge and consequently new assets.

A popular way of creating knowledge in the educational field is through questions and answers. Such process is the basis of the Question Pedagogy, proposed by a well-known constructivist named Paulo Freire. In a KM setting, the process of questioning and answering stimulates the pursuit for innovation. This can prevent the crystallization of knowledge and procedures, impelling people to seek for new and better ways of acting and performing. Moreover, this process forces both questioner and responder to find a common interpretative schema, by putting themselves in each other’s position and trying to grasp each of their individual and tacit view on the discussed topic.

In this work, we rely on questioning and answering as the process of creating and sharing knowledge. This is detailed in chapter 4 where we describe the main functionalities of our proposed system. By this, we aim at imitating the natural ways people share knowledge, i.e. by asking their doubts to their peers. This results in a dynamic knowledge sharing environment, where tacit knowledge is explicitated throughout the questioning and answering process.

As highlighted in this chapter, KM processes are crucial to large organizations even though they are not part of the organization’s primary business goals. However, to identify, acquire, develop, disseminate, use, and preserve organizational knowledge it is necessary to offer the proper technological support.

In this context, a recommender system may be beneficial. A recommender system aims at assisting the user to find items of interest based on his/her profile. For that, the system tries to match the profile with the description of the available items or tries to bring people with similar interests together and give recommendations based on that. In this sense it also encourages the formation of communities of practice.

To provide recommendations, it is necessary to explore a database of items, e.g. documents, descriptions. Besides, to match items against the user needs, it is necessary to use information retrieval techniques. These techniques can be used to reduce the vocabulary space to be searched, to rank documents according to the users’ wishes, and to search documents that fit to the user’s profile. However, much refinement is needed in the existing Information Retrieval (IR) algorithms to provide users with better recommendations.

On the next chapter we focus on the requirements and design of our recommender system. We have developed an agent based decentralized approach for recommending documents. Furthermore, we have implemented our own information retrieval technique, which aims at improving the precision and recall of standard approaches by taken into account semantic information about the location of documents.
CHAPTER 3

System Requirements

This chapter describes a set of requirements to be fulfilled. The chapter is structured as follows: Section 3.1 provides a high-level overview of KARe’s knowledge management requirements. Section 3.2 shows the requirements regarding the development of an information retrieval approach for supporting the system. Finally, section 3.3 shows the requirements of distribution of users and the relationship amongst them.

3.1 Knowledge Management Requirements

As seen in section 2.1.4 knowledge is a mix of experience, values, contextual information, and expert insights within an specialized domain [38]. Thus, within an organization knowledge and expertise become increasingly distributed, which promotes the growth of specialized knowledge communities. Such growth makes discovering of relevant knowledge from these communities a challenging task.

In section 2.6 we have seen that one approach to discover knowledge is through the support of the question pedagogy. Furthermore, in section 2.2 we realized that the distributed KM encourages more the organizational members to share their knowledge than the centralized approach. Besides, a centralized approach enforces users to classify knowledge all in the same way, inhibiting their own tacit knowledge to be revealed. Thus, we open a number of requirements to be fulfilled as follows:

R1: To simulate the question pedagogy. KARe aims at simulating the natural social processes involved in knowledge sharing. In real life, asking and answering to questions is

23
an interactive process. The questioner finds a suitable colleague and poses his questions. Usually, this choice is based on the questioner’s assumption that his colleague knows about the targeted subject, besides feelings of trust and comfort towards the responder (e.g. “he is not going to judge me for my question”). The responder, on his turn, is likely to provide the questioner’s with some help, provided that the trust between them is mutual. He will then use his own language and knowledge to provide the answer to the questioner. Besides solving the problem at hand, having the answer gives the questioner the ability to share this new knowledge with other colleagues.

The system should allow the user to pose natural language questions. It is not required that the system performs natural language processing. The question is made in natural language by the user, and the system should use this information as a set of keywords, searching in other users’ collection for answers among that user’s stored artifacts. These artifacts can be documents or messages sent by other peers responding previously to similar questions.

**R2: To allow users to control the contents of the knowledge bases.** KARe should support a distributed KM approach. Such an approach goes against a common KM solution: to create a central repository to which all organizational members are asked to contribute with their own personal knowledge. These repositories may contain different documents used in the work routine, but also specialized records aimed at capturing the more tacit kind of knowledge, such as reports on lessons learned and best practices gathered by workers’ experience in different projects [8]. One problem with this approach is to get users’ acceptance to spend extra hours on feeding the system, without knowing who will use his/her knowledge and for which purpose. Besides, employees have no guarantee of finding useful information in the repositories when they need it [7].

With KARe, users are allowed to publish their own knowledge on the network. Thus, the control over the knowledge assets will move from a centralize server to each user. Furthermore, the way users classify their own knowledge is particular, thus we do not require a common classification for their artifacts.

### 3.2 Information Retrieval Requirements

The requirement R2 denotes that each user might have their own particular knowledge classification. It implies that documents might be classified differently according to classification. So, to encourage organizational members to use our solution, the document recommendation algorithm in our system should have good precision and recall values. So, it is required to study and develop an information retrieval algorithm that can handle different document classifications, as follows:
3.3 General Requirements

Decentralized solutions, as proposed by the requirement R2, imply the necessity of a scalable solution to give recommendations. Assuming that our system should support communication amongst members in large organizations, we therefore should give the appropriate technological support.

We assumed that organizational members could be nomadic. Nomadic users are users that, with support of mobile computing such as laptops, handhelds and mobile phones, can use network applications as they change their location. Thus, as the location changes, the users’ environment (i.e. surrounding members) is likely to change over time. It is appropriate to take advantage of this nomadic behavior to select partners for asking recommendations. Furthermore, this approach should act autonomously on behalf of the users, so they will not be discouraged from using the system due to lack of time. For that we have proposed two requirements, as follows:

R4: To use location information to select responders. KARe provides the users a list of possible responders. One of the challenges of this work is to select suitable responders for a questioner. Our first assumption is that suitable responders can be selected based on their geographical proximity to the questioner. The argument is that we usually share spaces with individuals with some similar interests, e.g. our colleagues in the office, researchers in a conference, students in a classroom and even people at the sports center. Hence, by changing the user’s location, the recommendation is likely to change as well. In contrast, if you try to get a recommendation from a search engine, or any other “publishing” knowledge management solution, the result will be always the same until the document index is updated. This leads to more effort in finding knowledge. Taking on this approach, we enable intra-organizational knowledge management.

From the user’s point of view, the service will always be the same. The underlying provider will differ since the service providers are the other peers in the vicinity. Since the service provider will change with the location, it is crucial to have a common communication protocol amongst the parts. Agents provide such a protocol since their communication
is in the form of speech acts standardized in FIPA ACL (Agent Communication Language) [15].

R5: To provide an agent-mediated knowledge management solution. An agent approach has been chosen because agents can be viewed as autonomous and proactive technological building blocks, suitable for modeling social interactions and representing technological components at the same time. They can act on behalf of the organizational members in human activities such as knowledge searching, sharing and exchanging.

3.4 Conclusion

In this chapter, we have discussed some of the essential requirements to be satisfied by a nomadic recommender services platform. Having elaborated on the essential requirements we are able to propose an architecture to KARe. The design of the proposed architecture is reported in chapter 4.

We have classified the requirements in three categories. The first class of requirements are the ones related with KM. There we derived that our system should simulate the question pedagogy and allow the users to control their own knowledge by means of a personal knowledge base.

The second class of requirements relates to information retrieval techniques. There we stated that a recommendation algorithm that includes semantic information about the artifacts could improve the retrieval performance of a recommender system and thus should be developed.

Finally we gave general requirements to our architecture. The first architectural requirement is that the system could use location information to select responders. In this way the system will not be overwhelmed when selecting responders for getting recommendations. The last requirement is that our architecture should take advantage of agents characteristics (e.g. proactiveness and autonomy) to improve its usability.
This chapter describes KARe, a multi-agent system for artifacts recommendation. Here we present KARe in a component architecture design as follows: (1) an information retrieval component for indexing and searching knowledge artifacts; (2) agents components distributed in a peer-to-peer network providing question-answering functionality; and (3) an agent discoverer component to find potential peers based on proximity information.

This chapter is structured as follows: Section 4.1 gives an overview of the three architecture components of KARe and map these components to the requirements derived in chapter 3. Section 4.2 shows the physical distribution of components on the architecture. Section 4.3 gives the detailed design of our recommendation platform. Section 4.4 describes how our proposed recommendation algorithm works and finally in section 4.5 we derive some conclusions and closing remarks.

4.1 Component Architecture

To meet the requirements described in chapter 3 we approached the design in three components as we see in Figure 4.1. This component based architecture provides an overview of the recommendation service offered by KARe as follows:

1. Peer discovery component;
2. Recommendation agents component;
3. Information retrieval component.
Such component scheme aims at providing a pluggable architecture with replaceable system parts. The components communicate with each other via well defined interfaces facilitating the adaptability of new components into the architecture. For instance, the information retrieval component could be replaced by any “searching” mechanism, since it conforms with the interface with the agents component. More importantly, we could use any agent platform that conforms to the FIPA specifications to compose the recommender agents component.

In this section we provide an overview of each component and their details. In the section 4.3 we will present the design of each component in greater detail.

### 4.1.1 Peer Discovery

This component addresses the requirements R4 and R5 described in the section 3.3. These requirements discuss the need of an approach to select suitable peers for asking artifact recommendations. One proposal is that selecting nearby users to ask for recommendation could be an appropriate choice. Furthermore, it is important that the selection process happens autonomously, so the users will not have to face the hassle of choosing peers and asking them questions. Thus, we will use agents to give autonomous behavior to this part of the system.

The Peer Discovery component is a set of sub-components responsible to: 1) find users in the vicinity, and 2) forward the information of the user’s location to the recommendation agents triggering the recommendation process. In Figure 4.2 we see a UML sequence diagram showing this process and the components that compose this component. The stereotype << Agent >> identifies an agent.

There are four sub-components in this component:
1. **User:** The Graphical User Interface running in the user’s PDA (see figures 4.6 and 4.7);

2. **BluetoothAdaptor:** The API implementation for device discovery using bluetooth technology;

3. **KAResScanner:** Intermediary between the Peer Discovery and Recommendation Agents components in the mobile side. It translates the information discovered by the BluetoothAdaptor to ACL FIPA messages and sends it to the agents;

4. **PeerAssistant:** Agent that acts on the user behalf on the system. Upon reception of a message it triggers the recommendation process.

The discovery process happens in six steps. First, the device will scan the neighborhood for other devices (method `discoverDevices()`). When new bluetooth-enabled devices are found this information will be forwarded to the KAResScanner to check whether the device participates in the KARe platform or not (method `newDeviceFound()`). The KAResScanner will prepare a message and will send it to the PeerAssistant agent (method `sendMessage()`). Note that to the agent it will seem like an speech act from the application instead of a method call since agents do not expose messages but they exchange messages \[29\]. If the PeerAssistant has found the agent representing the device in the KARe platform, then it will send a message back to the KAResScanner (method `receiveMessage()`). Finally, the recommendation list will be presented to the user (method `notifyUser()`).

![Figure 4.2: Peer Discovery components interaction.](image-url)
4.1.2 Recommendation Agents

This component meets the requirements R1, R2 and R5 (see sections 3.1 and 3.3). The first requirement is to simulate the question question pedagogy [31]. For that, we have defined an ontology that enables agents to exchange messages with each other and exchange requests (questions) and recommendations (answers). Furthermore, the agents have to control the user’s knowledge base, i.e. whenever a recommendation arrives, the agent will store it in his user’s knowledge base, thus replicating knowledge and meeting requirement R2. The requirement R5 is an architectural requirement. We met this requirement by modeling the social interactions involved in recommendation processes as agents interactions.

To design our system, we applied the Agent-Object-Relationship Modeling Language [18], specifically tailored for designing agents. Figure 4.3 presents the AOR Agent Diagram, that shows all agents and objects involved in our system design.

![Diagram](image)

**Figure 4.3: KARe agent diagram.**

The main agents (represented by the UML-like packages with << Artificial >> stereotype) in KARe are the Peer Assistant and the Artifact Manager. The Peer Assistant acts on behalf of the human user (the peer) and is responsible for: a) submitting questions from his human peers, and b) searching for the answers to incoming requests sent by other peers. The Artifact Manager collaborates with the Peer Assistant on the process of searching for answers aforementioned, finding the appropriate Concept that may contain the answer to an incoming question. For that, this agent interacts with objects of the information retrieval component.
Questions and answers are represented by the Message class and a shared artifact is represented by the Document class. An Artifact Model is the meta-data representation of a Knowledge Artifact (either a Message or Document). Such representation is classified within a particular Concept that is part of the user’s Taxonomy.

4.1.3 Information Retrieval Component

In the Information Retrieval component, detailed in Figure 4.4, we meet the requirement R2 (see section 3.2). Since this is one of the main components of this work, we have left the description and design of the developed recommendation algorithm to the section 4.4. Here we present the other components that compose the information retrieval component, complying with the general architecture presented in section 2.5.

The Knowledge Artifact, Document, Message, Concept and Taxonomy classes have the same role as in Fig. 4.3. Here we introduce the classes that will operate in the previous classes, i.e. the classes that represent the vector model.

Each Knowledge Artifact will be parsed by a Text Pre-Processor. During this phase a number of operations might be performed on the artifact such as stemming and stopwords removal (see section 2.5.1). After the processing, the contents of the Knowledge Artifacts will be represented as a set of Index Terms. The Index Term class is the representation of a word from a document or message after being processed. The whole set of Index Terms consists of the user’s Vocabulary and this will be the basis for creating the Inverted File Index (see section 2.5.2).

The Vector class represents Documents, Messages or Concepts. We have already seen how to represent the first two as algebraic vectors in section 2.5.4. The representation of Concepts as vectors is part of the contribution of this work and is further defined in section 4.4.

The Indexer class is responsible for maintaining the Inverted File Index (see section 2.5.2), i.e. to add and remove artifacts to the index. The Searcher class is responsible to consult the index file and a particular query is asked.

4.2 Components Distribution

Figure 4.5 shows the distribution of components of KARe. The elements are physically distributed in three locations: handheld computers, desktop computers and a server. The server can be federated by many stations, but we show it as a single entity for simplicity. The dashed arrows show the dependency between components, and the numbers within the circles show the ordering of execution.

As we can see, each user has access to one handheld and one desktop computer which
are connected via a wireless connection (802.11x). Each handheld is responsible for sensing the presence of other devices in the vicinity and advertise its presence to its neighbors. The device discovery is performed using a bluetooth link. When another handheld is detected, the information about the new device is sent to the user’s desktop computer. Such information is forwarded to the server to verify whether the the found device is part of KARe peer-to-peer network or not.

When we discover the presence of another KARe peer on the neighborhood, the Peer Assistant Agent will contact the “answering” Peer Assistant to trigger the recommendation process. If the “answering” peer recommends any artifact, the user will be contacted on his handheld. Bellow we describe each step shown in Fig. 4.4.

- **Device Discovery (1):** In each handheld there is a module named “KARe scanner”. This module collects information about the devices found in the vicinity via bluetooth and triggers the device verification process.

- **Device Verification (2, 3, 4, 5, 6):** Once the handheld has detected and collected identification information about another device it sends this information to the user’s computer (step 2). The user’s Peer Assistant will receive the information and verify whether it corresponds to another peer or not. For that it forwards this information to the Directory Facilitator agent (step 3) that will consult the peers services database to check for the peer’s existence (steps 4 and 5). When it knows about the discovered device status, the Directory Facilitator will send its findings back to the
questioner Peer Assistant (step 6). If the discovered device is a peer in the KARe network the recommendation process is triggered, otherwise it is aborted. Figure 4.6 shows the resulting screen in the handheld after identifying the devices that run KARe.

- **Recommendation process preparation (7, 8):** Before actually starting the recommendation process, the Peer Assistant agent will consult the User Questions database. If there are any questions, the Peer Assistant will put each question in the appropriate format for sending them over the network.

- **Recommendation process (9, 14):** This process is the core of the system. It is the actual simulation of the question-answering process. The first peer will send a question (step 9) which the second peer will try to answer as good as possible (step 14) by recommending an arbitrary number of artifacts (e.g. documents and messages).

- **Artifact search (10, 11, 12, 13):** These steps are the mechanism that enable the recommendation amongst peers. When the Peer Assistant receives a question it forwards it to the Artifact Manager agent, since the latter knows how to match the question against the User Knowledge Base. This matching is performed by an Information Retrieval algorithm (steps 11 and 12). Finally the retrieved artifacts are sent back to the Peer Assistant.
• **User notification (15, 16):** After receiving the answer for its question, the *Peer Assistant* will send a notification to the user’s handheld to warn him that new artifacts were recommended. Figure 4.7 shows the screen with the notification of new recommendation from another peer.

![Figure 4.6: Screen of the KARe scanner showing the users in the vicinity.](image)

Figure 4.6: Screen of the KARe scanner showing the users in the vicinity.

![Figure 4.7: Recommendation notification.](image)

Figure 4.7: Recommendation notification.

### 4.3 Design

In this section we derive the design phase diagrams of the KARe platform. We take a top-down approach to describe our platform, i.e. we begin describing KARe conceptually and we gradually add the design elements. We show here design elements in both AORML [18] and UML.

The model derivation begins with the static models: a conceptual agent diagram, a design agent diagram and the agent communication ontology. With that we aim at describing all components of our platform. After we have described each component, we will describe the dynamic parts of KARe. First we describe the interaction processes
amongst the agents. Finally we describe the mechanisms of our recommendation approach which is the main point of this work.

4.3.1 Conceptual Model

The conceptual model exhibited in figure 4.8 introduces the elements of the system and the dependencies amongst them. Here we use the AORML notation to create the model. On this notation, agents are shown like UML packages but designated with a stereotype identifying the type of the agent. An agent could get the following stereotypes: << Artificial >> (indicating that it is an artificial software agent), << Human >> (indicating that it is a human agent) and << Institutional >> (indicating that it is an institutional agent).

We have on the system two artificial agents (PeerAssistant and ArtifactManager), one institutional agent (CoP) and two human agent (CoP Member and Peer). The artificial agents have been already described in the section 4.1.2. The institutional agent CoP aggregates a number of agents (i.e. the CoP member agents) that have similar goals, roles and expertise. Further, CoP members trust each other. Finally, a Peer agent represents a CoP member that is present in the KARe platform. In this way, Peers can collaborate with each other and exchange Knowledge Artifacts.

The Knowledge Artifacts are specialized in two: Documents and Messages. Documents are any kind of artifact that can be shared to solve a CoP member question, e.g. books, tutorials, articles, magazines, papers and so on. Furthermore, the Documents are owned and shared by each Peer. When another Peer copies a Document then this will be duplicated and owned by the latter Peer as well. Messages represent the interaction amongst Peers.

The meta-data representation of the Knowledge Artifacts is depicted in the Artifact Model class. The metadata is used to classify indirectly the Knowledge Artifacts in Concepts. A Concept adds semantics to the location where the Knowledge Artifacts are stored, i.e. the artifacts are grouped according to a common concept classification designated by the Peer. The set of all interrelated Concepts form the user’s Taxonomy.

4.3.2 Platform Model

The model shown in figure 4.9 consists of the design specific to the JADE agent platform [14]. Here we combine the diagrams exhibited in Fig. 4.3 and 4.4 and include other platform specific elements.

To communicate in the JADE agent platform, our agents need to agree on the semantics of the messages exchanged by them. For that, the agents agree on an ontology that defines the conceptualization of the agents’ messages for a particular domain. In our
case we have developed the KareOntology (see section 4.3.3) as indicated by the respective class. The attributes of this class reflect the structure of the communication ontology as we will describe later on.

In JADE the agents are objectified in Java classes. However we left the AOR notation to differentiate them from the other elements and we added some of the agents attributes. Besides the domain-specific attributes (e.g. name, taxonomy, knowledgeBase, etc.) we have the platform specific attributes (e.g. myLocation, ontology). The remaining elements do not change since they are general-purpose classes and were implemented directly in Java.

### 4.3.3 Agent Communication Ontology

In agents and artificial intelligence domains, an ontology is [37].
"the specification of conceptualizations, used to help programs and humans share knowledge."

In other words, an ontology is a set of concepts (e.g. entities, events, relationships) that aim at creating an agreed-upon vocabulary for exchanging information. In figure 4.10, we see the definition of the communication ontology of KARe platform. This ontology is used so that agents will agree in a format to exchange messages and information.

We have five different entities specific to KARe and the relationships amongst them. The ArtifactManager and PeerAssistant entities are the specialization of the Agent entity. The agents play the central role on any ontology defined to the JADE platform since most events and other entities should be related to them.

Central to the communication process there are three entities: the Artifact, a Peer Question and a Peer Answer. An Artifact is owned by the Artifact Manager. Note that in our conceptual diagram the Peer owns the Knowledge Artifacts. We differ here because we are not modeling the human agents. Since the Artifact Manager controls the access to the artifacts through the recommendation algorithm, we modeled it as the owner of the Artifacts in the eyes of the system. Most importantly there are the Peer Question and Peer Answer entities. These are the reflection of the question pedagogy applied to KARe. These entities are generated and exchanged amongst Peer Assistants.
4.3.4 Agent interaction diagrams

An interaction diagram shows the sequence of action events and non-action events performed by agents. The diagram exhibits the protocol followed by the parts to communicate with each other [18]. The communication could be with the environment or with other agents, and this interaction is orderly showed in this type of diagrams.

In this section we explain two interactions by means of interaction diagrams. First we explain the local interaction between a Peer Assistant and an Artifact Manager upon the action event of the reception of a question (see figure 4.11).

The second diagram explains how the Peer poses a question to his Peer Assistant and how the latter propagates the question (see figure 4.12).

Interaction between Peer Assistant and Artifact Manager Agents

The interaction diagram in Fig. 4.11 shows the cooperation between the Peer Assistant and the Artifact Manager. Here we assume the situation where the Peer Assistant has received a question from another Peer Assistant on the environment and has forwarded the incoming question to the Artifact Manager (step 1) to search appropriate recommendations.

The Artifact Manager needs to receive, besides the request (i.e. the query), extra information to answer it. Such information consists of the concept related to the query, the vector that represents the concept, and the vocabulary spoken by the originating Peer Assistant as we see in the parameters of the message searchAnswer.

The Artifact Manager should perform an action (searchAnswer on the step 2) to
search for the artifact. Note that this action is different in notation to the message and denote different things in the model. The first searchAnswer is a communication act between the agents. The second searchAnswer is an action-event on response to an event (i.e. the incoming message).

When this action is complete it will send the suitable document(s) to the Peer Assistant that answered the originating question (step 3). The answer contains the artifact (or a list of artifacts) that the Artifact Manager recommends to answer the posed question. Further, the answer shows the concept associated with the artifacts and the similarity of the artifacts with the question. With that information the user can evaluate the quality of the answer in two aspects: 1) whether the concept relates or not to the question by checking the concept’s label and 2) whether the similarity of the artifacts are high enough to answer the question.

![Interaction Sequence Diagram: Search Answer.](image)

**Interaction between Peer and Peer Assistant Agents**

The interaction diagram in Fig. 4.12 shows the cooperation between the different Peer Assistants in the system. It begins with a request from the user to the Peer Assistant and it completes when other Peer Assistants are perceived in the vicinity.

The first step on the interaction is between the Peer and its Peer Assistant is to the latter what are the questions that the former wants to be answered. For that the Peer sends a message containing a query, a concept that is related to this query in the user’s taxonomy, the vocabulary spoken by the Peer (i.e. the set of index terms chosen during the artifact indexing process) and the vector associated with the concept. The first two parameters (the query and the concept) are chosen explicitly by the user. The last parameters (the vocabulary and the vector associated with the concept) are gathered by the system from the information retrieval component.

The communication between Peer and Peer Assistant is asynchronous and thus the
response to this message may not arrive. The *Peer Assistant* will respond the message when it perceives other *Peer Assistant*(s) in the vicinity. It means that the recommendation process is triggered by its neighborhood. We see that on the non-action event in step 2. There Pablo’s PA perceives that Renata’s PA is nearby. Thus it is now possible to send requests.

The requests between *Peer Assistants* are identical to the requests between *Peer* and *Peer Assistant* and the message (*requestExplanation* in step 1 and 3) has the same name and parameters, only the parts have changed.

If the targeted *Peer Assistant* has a suitable answer to the question it will respond by sending a *sendExplanation* message (step 4). The message contains the answer itself, the concept associated with it and the similarity between the question and answer. This message results from the interaction shown in the Fig. 4.11.

The *Peer Assistant* that has posed the question will forward the answer back to its *Peer* (step 5) and the latter might choose to save this answer or not. If the user decides to save the answer it will inform the *Peer Assistant* in which concept it should save the answer since the concept associated with the answer (step 6) might not exist in the questioning taxonomy. The *Peer Assistant* would then finally save the answer in the user’s taxonomy as indicated by the *saveExplanation* action on the step 7.

![Interaction Sequence Diagram: Request Explanation](image)

**Figure 4.12: Interaction Sequence Diagram: Request Explanation.**

### 4.3.5 Agent interaction pattern diagram

An interaction pattern diagram describes the possible reactions and executions performed by agents during the interaction process. This description includes reaction rules and
executable specification of a rule. Here we could use the rules derived from the diagram as a first attempt to implement the artificial agents.

In figure 4.13 we see the interaction pattern between the Peer Assistant and the Artifact Manager agents. Internally to the Artifact Manager agent we see the reaction rule R1 to the message searchAnswer. The rules states that the agent might react in three ways: 1) by sending back some documents (provideDocument message); 2) by sending back some explanation to the answer (provideExplanation message) or 3 not send anything in return to the message (noAvailableArtifact message).

The interaction will be independent of the Peer Assistant and Artifact Manager instances that we might have in our scenario. This allows us to to represent the reaction rules textually and thus allowing it to be implemented. We show the textual representation of the reaction rule R1 in the listing 4.1.

![Interaction Pattern Diagram: Search Answer.](image)

Listing 4.1: Textual description of the Rule R1 of the Artifact Manager

```plaintext
1 ON EVENT: RECEIVE searchAnswer (?keyQuest, ?vectorC) FROM ?PeerAssistant
2 IF Similar(?keyQuest,?vectorC,ArtifactModel(?Document))
3 THEN SEND provideDocument(?Document) TO ?PeerAssistant
4 ELSE IF Similar(?keyQuest,?vectorC,ArtifactModel(?Question))
5 THEN SEND provideExplanation(?Question,?Answer) TO ?PeerAssistant
6 ELSE SEND noAvailableArtifact TO ?PeerAssistant
```

4.3.6 Recommendation Algorithm UML sequence diagrams

The recommendation algorithm proposed in this work (see section 4.4) does not involve directly any of the KARe agents. For that we chose to design the process using UML
sequence diagrams instead of trying to model it as internal objects of some agent.

The algorithm is divided in two parts shown in the following diagrams. In the first, we show the process where user creates an index of the knowledge artifacts and concepts. In the second, we show the recommendation mechanism. The recommendation consists of a searching process for knowledge artifacts.

**Indexing sequence diagram**

The indexing process begins with the indication from a user. Such user could be an agent, an application or a human user. The Indexer is the class that receives the method call `createindex` from the user and is responsible for handling the process. The index file (see section 2.5.2) is kept here. The Indexer receives two parameters: 1) a list of documents to be indexed and 2) the taxonomy that classifies these documents. The first step towards the creation of the index is to parse each Concept of the Taxonomy.

During the Concept parsing, the Indexer must parse each Knowledge Artifact contained in it. This step is necessary to create the vocabulary and index terms on the system since they are contained within the Knowledge Artifacts.

After parsing the Concepts and Knowledge Artifacts the Indexer should perform some feature selection to reduce the index file to an appropriate size. Once the final index is determined, the Indexer can create the Vectors for the Concepts and Knowledge Artifacts. On this step the weight for each term in each document and concept is calculated and stored on the Vectors.

In figure 4.14 we can see the mentioned process. The process takes the following steps:

1. **createindex**: indication from the user to create an index. The parameters provided are the documents to be indexed and the taxonomy where they are classified;
2. **parseConcept**: first step towards the creation of the index. Each concept of the taxonomy is parsed. This will trigger the next method call: **parseArtifact**;
3. **parseArtifact**: called by the **parseConcept** method. It begins the process of reading the Knowledge Artifact to put its contents on the index file;
4. **parseToken**: pre-processing of some word on the documents. This step verify the presence of stopwords to be removed and performs the stemming on the terms;
5. **createIndexElement** and **create**: after the words are processed they can be inserted in the vocabulary. For that we create an IndexTerm instance to represent the processed word;
6. **performFeatureSelection**: after all documents and concepts are parsed and their contents are inserted in the index file the Indexer should reduce the index file size.
This is performed by the feature selection process that is responsible to select the
appropriate index terms to remain in the index;

7. createVector, getTerm and calculateWeights: after the Indexer has reduced the
index file then we can create the vectors that represent the documents and concepts.
The vectors dimension size will be the size of the vocabulary (i.e. number of index
terms). To create the vectors it is necessary to read all index terms and count their
occurrences in the documents.

Figure 4.14: UML Sequence Diagram: Indexing Process.

Searching sequence diagram

The searching process is the main point of this work. It describes how the system finds
suitable knowledge artifacts according to a question made by an user. Furthermore, we
show here how the system finds the appropriate concept by taking into account the infor-
mation from the concept where the question has been made.

The main component here is the Searcher. It centralizes the query handling process
and interacts with the user. When a question is made, the first step is to pre-process it.
In this way, the question will have the same format as the contents of a document, i.e.
it will have the stopwords removed and the stemming will be performed. After parsing
the query it is important to normalize the information arriving from the questioning side to the responding side. It means that the concept vector from the questioner should be aligned to the vectors and vocabulary of the responder side.

Having the questioning concept vector aligned with the responder’s now we can begin the searching process. The search begins by finding a suitable concept on the responder’s taxonomy that fits the questioning concept. We use a similarity function (see sections 2.5.4 and 4.4) to assess that.

After the Searcher has the knowledge of the similarity of all concepts to the questioning concept it is possible then to choose the best matching concepts to search for artifacts. In this step the Searcher will create a Vector containing the question terms and will compare it with the Vectors of the Knowledge artifacts within the best matching Concepts. Finally the result of this similarity assessment is sent back to the user.

In figure 4.15 we can see the mentioned process. The process takes the following steps:

1. query: indication from the user that a question has been made. The parameters are the question itself, the Vocabulary spoken by the questioning peer and the Concept Vector associated with the question;

2. parseQuery: process of pre-processing the question. It performs stemming and remove the stopwords from the query;

3. normalizeConceptVector: process of aligning the questioning concept vector with the targeted vectors and vocabulary;

4. getVector, storeSimilarity and getBestConcepts: the questioning concept vector should be compared to each concept vector on the destination taxonomy. For that we must retrieve each vector and check its similarity with the questioning concept vector. On the end of the process we will be able to retrieve the best matching concepts with the questioning concept;

5. getConcept and getArtifacts: once we have a good concept in hands we should retrieve its artifacts to begin the comparison with the question itself;

6. createQueryVector: to compare the question with the knowledge artifacts we must create a vector representation of the question itself;

7. storeSimilarity: the last step is to check the similarity amongst the documents from the selected concept and the question vector. It serves as a ranking function of the documents according to the query.
4.4 Recommendation algorithm

As we have seen in the section 2.5.4, the vector model for information retrieval [35, p.27], documents and queries are treated as real algebraic vectors. The standard vector model approach for the retrieval of information in a given knowledge base is to compute the similarity between the query and all the documents in the given base and then select the most similar vector as the winner document. However, this approach does not consider any knowledge that users may have about the structure and concepts involved with the artifacts being searched. This can lead to increased noise on the search results. For instance, trying to search on Google the word “agents” would result in several different kinds of agents (e.g. chemical, real state and travel agents).

The solution we propose is to include a user context in the searching mechanism, providing semantics to the artifact (i.e. relating it to the concept it is associated). Similar documents are grouped by the user under the same concept in the context tree, which facilitates artifacts’ retrieval. Besides, before submitting the question, the user contextualizes the query, assigning it to a specific concept. Doing this, the user is giving to the system an extra hint on the query’s content (besides the keywords contained in the query itself), leading to more accurate results.

As an example, consider two users A and B willing to share their contextualized
knowledge (see Figure 4.11). User B submits the following question: “What is an agent?”, assigning it to the “Agents” concept in the context in the left side of Fig. 4.11. Taking the context of the user B in the right side of the Fig. 4.11 it is most likely that the concept being searched is within “Computer Science → Software Engineering → Agent Oriented”. But how can the algorithm be aware of that?

Figure 4.16: The knowledge of users A and B structured in their contexts

Essentially, each of the concepts of a user context has a vectorial representation of the words associated to it, which measures the relevancy of the words according to the documents under that given concept. The relational information among concepts (i.e., parent and/or children relationship) could help to determine reference vectors that also exploit this relational knowledge [17]. The determination of the concept reference vectors follows the equation 4.1.

$$w(term_i, concept_j) = \frac{\sum_{k=1}^{n} w_{i,k}}{n} \quad (4.1)$$

Here, $w(term_i, concept_j)$ stands for the weight of the term “i” on the concept “j”. Such an approach was based on the $TF \times IDF$ measure, i.e. the weight calculation for the concept is an average of all weight values within that concept for each word in each document. The variable $w_{i,k}$ represents the weight of the term “i” on the document “k” and this document belongs to the document set of the concept.

Once the concept reference vectors are determined, the searching can take place. The first step of the search is to find the concept in the taxonomy of user B that best matches the concept selected in context A for the given query. This again is simply determined by computing the cosine distance between the selected concept of A and all concepts of B. Then the most similar concept of B (i.e., with the greatest cosine value) is selected. We call this process a query scope reduction, the actual novelty of our approach. In summary, the query scope reduction can be seen as a reduction in the searched document set before we retrieve information from it, based on the fact that the required information is more likely to be found in a specific region of this set (in our case, within the concept that is
more similar to the one selected by user A to contextualize his query). We hope that adding this process prior to the execution of the query, we will increase the quality of our search, resulting in a result of less noise, thus recommending only pertinent documents to our users.

In our example, we would calculate the similarity function among the vectorial representation of the concept “Agents” of user A and all concepts of the user B. Each concept is represented by a vector created with the basis on the documents stored under that particular concept. User A and user B have different global indexes, i.e. the vectors of each taxonomy are created based on different sets of keywords (index terms). Consequently, the first step in the query scope reduction is to project the concept vector coming from A in the new space of B. This is made with an intersection between the vector coming from A and the index of B. Then, the projected vector is compared to each of the vectors representing the concepts of taxonomy B. Finally, the concept in user B’s context that has the highest similarity will be chosen for performing the query.

Then, in order to retrieve the answer to user A’s question, the system searches all artifacts within the selected concept of user B. In this phase, all keywords of the user’s query are taken into account to select the artifacts of the given concept. The documents are then ranked in a result set that is finally sent to user A. We try to show our recommendation approach by means of a pseudo-code in the listing 4.2.

Listing 4.2: An excerpt of KARe’s recommendation algorithm

```plaintext
procedure answer(conceptVectorA, peerQuestion, questioner) {

  // step 1: search the best matching concept for the scope reduction
  projConceptVectorA := intersect(conceptVectorA, indexB)
  for each (concept on the user B context) {
    s := cosine(currentConceptVectorB, projConceptVectorA)
    if (s > maxSimilarity) {
      bestConcept := currentConceptB
      maxSimilarity := s
    }
  }

  // step 2: search among the documents in the bestConcept
  queryVector := createQueryVector(peerQuestion, indexB)
  for each (document in bestConcept) {
    documentList.add(document, cosine(queryVector, documentVector))
  }
  documentList.sortBySimilarity()

  // step 3: send the answer back to the questioner
  sendAnswer(documentList, questioner)
}
```
4.5 Conclusion

In this chapter, we have discussed some of the design of the platform proposed in this work. We begin by dividing our architecture in three distinct components. These components are then mapped to the requirements derived from the chapter. After presenting the component based architecture we divide the components of our architecture by their physical location. In this way we have components on a handheld computer, components on a desktop computer and components on a server.

Having introduced the components and their physical distribution we derive some design models using AOR modeling language and UML. The use of AOR is to model the agents defined in the KARe platform. Our diagrams were able to demonstrate the social interactions of the system agents which was one of the main goals of using AORML.

Finally we formalize our recommendation approach by giving some scenarios that describe its capabilities. During the formalization of our approach we show how it relates to the vector model for information retrieval and how a taxonomy of concepts can be used to enhance the it.
CHAPTER 5

Implementation

This chapter presents the implementation of KARe system. The implementation validates the concepts presented in the design phase in chapter 4.

The chapter is structured as follows: Section 5.1 presents our implementation approach. Section 5.2 describes the details of each implemented component. Finally section 5.3 presents the concluding remarks.

5.1 Approach

The main objectives of the prototype are to demonstrate the concepts with respect to 1) the information retrieval approach proposed to this system and 2) the integration of the software agents to assist the users in their KM-related tasks. For that, we we have implemented the following parts of the system:

- An information retrieval component that takes into account information from a taxonomy to retrieve documents;

- A tool that feeds the IR component. This tool assists the user to create his/her document index and to load the taxonomy into the system;

- The system agents that enable the communication amongst the users;

- A simulation of the JSR-82 discovery agent specification [23]. This simulation is used to give KARe information about bluetooth enabled devices in the user’s vicinity;
• A GUI that enables interaction between the user and his Peer Assistant agent. This tool is the means for providing recommendation to the user. Furthermore, the GUI will present to the user a list of peers in his vicinity after discovered by the handheld’s bluetooth adaptor;

• A component that bridges the gap between the system agents and the GUI. This component transforms the information read and presented at the GUI to a format understandable by the agents. After the agents exchange recommendations this component will transform recommendation information to a format that can be presented to the user in the GUI.

We have used Java language to implement the prototype. Furthermore, we used several software libraries to support our implementation. The JADE, Java Agent Development Framework [14], was used to implement and deploy the system agents. JADE works as a middleware for the agents communication. The agents are implemented via Java classes that communicate with each other via Java RMI.

We have developed an ontology in section 4.3.3 that enables the communication of our system agents which we named “KARe Ontology”. The KARe Ontology was designed using the Protégé Ontology Editor [32] and implemented in Java classes using the Beangenerator Protégé plug-in [1].

We based the implementation of our information retrieval approach in Lucene [6, 12]. Lucene is a search engine library and it contains implementations of well-known algorithm and components used in our system such as: the inverted file index (see section 2.5.2), a stopword remmover component and the “Porter Stemmer” algorithm [26] to perform stemming.

The GUI was implemented with Java language, more specifically with Java 2 Micro Edition version (J2ME). We used the Personal Profile [22] API implementation to run the GUI on an iPAQ handheld with the CVM virtual machine.

The communication between the handheld and the user’s knowledge base server was implemented using the Jini Surrogate Architecture [36] and the surrogates represent the nomadic users in the system.

In section 5.2 we describe each of the mentioned components in greater level of detail.

5.2 Modules

The implementation of our system took place in three phases. The first component developed was the information retrieval approach and its functionalities are described in section 4.3.4. With the algorithm developed and evaluated we moved to the implementation of the
system agents. The developed agents made use of our IR component to retrieve documents for recommendation based on the users' needs.

Our agents implementation runs in a peer-to-peer fashion and is by itself a recommender system running on desktop computers. However, we aimed at porting our recommender system to handheld devices. Thus, we developed a GUI that runs in an iPAQ handheld device. To overcome problems with the limited resources on such devices we have "bridged" the information gathered by the iPAQ to our agents, i.e. this component works as a GUI. We have also implemented the device discovery simulation within the GUI.

In the sub-section 5.2.1 we will describe our information retrieval component implementation. In the sub-section 5.2.2 we describe the system agents implementation and finally in the sub-section 5.2.3 we describe the peer discovery component and its GUI that are running in the iPAQ.

### 5.2.1 Information Retrieval Module

The first part to be developed in any information retrieval component is the function responsible for document indexing. To develop this module we used Lucene [6] as a support tool to perform fast document indexing. It is able to index over 20MB of text per minute and to reduce greatly the size of the index file when compared to the size of the files being indexed. We chose Lucene because it is being already used by a large community and the indexing processing feature comes completely “off-the-shelf” with it.

As we have seen in section 2.5 we have some tasks that precede the indexing like lexical analysis, stemming and stopwords removal. These tasks are all supported by Lucene as well. For stemming it uses the Porter Stemmer algorithm [26]. The stopwords are possible to be configured, and we have merged several stopwords lists that are available on the web (including 530 words). Lucene has configurable pre-processing options. For that we must subclass the “Analyzer” class and implement the method `tokenStream(String fieldName, Reader reader)` as we can see in the listing 5.1.

```
package kare.IR;

import org.apache.lucene.analysis.*;
import java.io.*;

public class PorterAnalyzer extends Analyzer {
    private static java.util.Hashtable _stopTable;

    // this field contains a list of all the stopwords to be removed by the system
    private static final String SMART_STOP_WORDS[] = { /* stopwords list here */ };
```
public PorterAnalyzer() {
    this(SMART_STOP_WORDS);
}

public PorterAnalyzer(String[] stopWords) {
    _stopTable = StopFilter.makeStopTable(stopWords);
}

public final TokenStream tokenStream(String fieldName, Reader reader) {
    // setup of the stemming filter. it includes:
    // stopword remover and lowercase tokenization process
    return new PorterStemFilter(
        new StopFilter(new LowerCaseTokenizer(reader), _stopTable));
}

To perform the document indexing itself we have wrapped the Lucene implementation in a class called Indexer. The class has a public method createIndex(File f) that receives the user’s taxonomy as a parameter from the Artifact Manager (see listing 5.2).

The creation of the index is divided in three parts. First the taxonomy file is read by the Indexer using the XML Beans classes that were generated by an XML Schema that defines the taxonomy format. The XML Beans classes are shown is listing as the instances of the classes PeerKnowledgeDocument and ConceptDocument. The method loadConcept visits each document of a particular concept in the given taxonomy and indexes it. The physical indexing using the Lucene objects is performed in the indexDocument method.

Listing 5.2: Indexing documents with Lucene

```java
public void createIndex(File _contextFile) {
    try {
        // 1. Load Context XML File
        PeerKnowledgeDocument myKnowledge = PeerKnowledgeDocument.Factory.parse(_contextFile);
        ConceptDocument.Concept[] concepts = myKnowledge.getPeerKnowledge().getConceptArray();

        // 2. Load Concepts and creating the index
        IndexWriter writer = this.getIndexWriter();
        ArrayList conceptTerms = new ArrayList();
        for (int i = 0; i < concepts.length; i++) {
            loadConcept(writer, concepts[i], this.forceLabel, conceptTerms);
        }
        if (writer != null) writer.close();

        // 3. Create the concept vectors and its artifact vectors
        index.parseIndex(this.vocabularySize, conceptTerms, this.forceLabel);
    } catch (XmlException e) {
        e.printStackTrace();
    }
}
```
5.2. MODULES

```java
// this method reads the taxonomy file and load each concept and document
// in it to the index.
public void loadConcept(IndexWriter writer, ConceptDocument.Concept concept,
    boolean forceLabels, ArrayList conceptTerms) throws IOException {

    // 1. get the first concept
    String conceptName = concept.getName();

    // 2. retrieve the concept's artifacts
    ConceptDocument.Concept.ArtifactModel[] artifacts = concept
        .getArtifactModelArray();
    int size = artifacts.length;

    try {
        if (size > 0) {
            // 3. include the concept information in the index data structure
            this.conceptCount++;
            this.indexes.add(conceptName);
            // 4. performs stemming in the concept's label
            QueryTermVector qtv = new QueryTermVector(conceptName,
                this.analyzer);
            int qtvSize = qtv.getTerms().length;
            for (int k = 0; k < qtvSize; k++) {
                String t = qtv.getTerms()[k];
                if (conceptTerms.indexOf(t) < 0) {
                    conceptTerms.add(t);
                }
            }

            // 5. reads all artifacts and include them in the index
            for (int i = 0; i < size; i++) {
                String artifactModelFile = artifacts[i].getFile();
                ArtifactModelDocument artifactModelDocument;
                artifactModelDocument = ArtifactModelDocument.Factory.parse(new File(
                    artifactModelFile));
                ArtifactModelDocument.ArtifactModel am = artifactModelDocument
                    .getArtifactModel();
                File txtFile = new File(am.getDocument().getFile() + "\".txt\"");
                Document doc = Indexer.indexDocument(txtFile, conceptName, am
                    .getDocument().getTitle());
                if (writer != null)
                    writer.addDocument(doc);
            }
        }
    }

    // 6. get sub concepts
```
The last step in the preparation of our documents is to create the document and concept vectors. The concept vectors will be used to verify which is the most suitable concept to search for a particular question (see section 4.4). The document vectors will be used to rank the documents according to a given question, which is also represented by a vector.

As we have seen in section 2.5.3 the $TF \times IDF$ \cite{35} is a common way to calculate the weights for the vector’s dimension values. We have implemented it in the method \texttt{calculateDocumentVectors} in listing 5.3. The method \texttt{calculateConceptVectors} calculates the weight for the concept vectors as defined in section 4.4.

**Listing 5.3: Creating Document and Concept Vectors**

```java
public void parseIndex(int vocabularySize, ArrayList conceptTerms, boolean forceConceptTerms) throws Exception {
    IndexReader ir = getIndexReader();
    int documentCount = ir.numDocs();

    // 1. Calculate the maximum frequency of each document
    int[] maxFrequencies = new int[documentCount];

    for (int i = 0; i < documentCount; i++) {
        TermFreqVector tfv = ir.getTermFreqVector(i, UserContext.CONTENT_FIELD);
        int[] f = tfv.getTermFrequencies();
        Arrays.sort(f);
        maxFrequencies[i] = f[f.length - 1];
    }

    // 2. Calculate the document vectors
```
The last implemented part is the searching mechanism. This mechanism is connected to the Artifact Manager via a well defined interface defined in listing 5.4. The method askQuestion is used by the Artifact Manager to request the search for some artifact. The method answerQuestion is used for callback by the search component to inform the Artifact Manager that an answer for its request was found.

The “questioning” function takes place in four steps: (1) The user chooses a concept related to his question followed by (2) the question itself. At this point, the question is...
broadcasted among the peers, and their local agents (3) find appropriate concepts within their knowledge context that matches the question’s concept, and then finally (4) search for answers within the chosen concepts.

Any information retrieval mechanism that implement the interface in listing 5.4 is able to interact with our system agents.

Listing 5.4: Interface for asking and answering questions
```java
public interface SearchInterface {
    public void askQuestion(String concept, String query, boolean extendedModel);
    public void answerQuestion(PeerAnswer answer, Agent artifactManager);
}
```

The method `answerQuestion` implements the cosine similarity measure [35]. We show our implementation in the listing 5.5. The method `createAnswer` is called by the `answerQuestion` implementation to create the answer. In this method we normalize the question’s vectors to the answering vectors (see section 2.5.4), create the query vector, compare the questioning concept vector to the answering concept vectors and finally rank the document vectors according to the query vector. The method uses Lucene to access the index files.

Listing 5.5: Process of finding the suitable concept and documents for a particular question.
```java
public PeerAnswer createAnswer(Asks question, boolean extended) throws IOException, ParseException, Exception {
    String query = question.getQuestion().getQuery();
    PeerAnswer answer = new PeerAnswer();
    int vocabularySize = getUserContext().vocabularySize;
    // 1. Create the query vector
    double[] queryDocumentVector = createQueryVector(query);
    // 2. normalize the concept vector
    double[] normalizedConceptVector = normalizeVector(question.getQuestion().getTerm(), question.getQuestion().getWeight(), vocabularySize);
    // 3. check similarity
    IndexReader ir = getIndexReader();
    IndexSearcher searcher = new IndexSearcher(ir);
    Indexer index = getUserContext().index;
    Query q = null;
    BooleanQuery bq = new BooleanQuery();
    QueryFilter conceptFilter = null;
    ResultSet rs = new ResultSet(20);
    if (extended) {
        ConceptSimilarityQueue sq = selectMatchingConcept(normalizedConceptVector, 5);
        while (sq.size() > 0) {
            ConceptSimilarityEntry entry = sq.get(entryIndex);
            conceptFilter = QueryUtil.generateBooleanQueryFromConcept(entry.getConcept(), entry.getTerm(), entry.getWeight());
            bq.add(entryFilter);
            bq.add(conceptFilter);
            Query q = QueryUtil.createBooleanQuery(bq);
            searcher.search(index, q, rs);
Figure 5.1 shows a screenshot of the tool developed to index the documents and load the taxonomy information into the KARe system. On the left part of the window there is the user context, showing in a tree of concepts how the user has structured his knowledge. The tool can be used also as a desktop version of KARe recommender service.

On the right side we divided the screen in two parts. The “Details” screen shows information (metadata) of the selected item (concept or artifact) on the concepts tree. The “Results” screen shows the answers to the questions performed, classified by peer and similarity with the query as we can see in Figure 5.2.
5.2.2 Recommender Agents Module

We developed the system agents using the JADE framework for agents [14] and the LEAP add-on [21]. JADE is a framework to develop multi-agent systems that comply with the FIPA standard. Recently the FIPA standard has been recognized as the IEEE standard for agents systems [16].

We used the 3.3. version of the framework and the LEAP add-on to run our system agents to both J2SE and J2ME java editions. When running in a LEAP enabled agent platform, the system agents require less resources from the communication system, i.e. it compresses the data sent over the network between the agents.

To create an agent we have to subclass the `jade.core.Agent` class and include our processing there. JADE agents exchange information via messages, therefore it was necessary to define these messages in a well-known format. Here we developed an ontology representing the basic elements exchanged by the agents only. This ontology is defined in section 4.3.3.

The `Artifact Manager` is shown in the listing 5.6. There we see that every agent
must implement the methods `setup()` and `takeDown()`. The first method is invoked when the agent is created. In the case of the Artifact Manager we tell it what is the ontology that it understands and then we register it as a service in the Directory Facilitator agent in the JADE platform. The Directory Facilitator agent stores knows the location of every agent and the services offered by it within its platform. The second method is responsible for releasing the resources allocated by the agent and to unregister from the Directory Facilitator.

An agent could also implement several “behaviours”. Each behaviour is the definition of a particular action to be taken upon reception of a specific message. In the listing 5.6 we see that the agent adds a behaviour called ArtifactManagerListener during the setup phase. This behaviour is responsible for continuously listen incoming messages of any type. Outgoing messages of the Artifact Manager are defined by the ArtifactManagerResponder behaviour. The last is created during the question answering.

Listing 5.6: Artifact Manager agent implementation

```
public class ArtifactManager extends Agent {
    private ContentManager manager = (ContentManager) getContentManager();
    private Codec codec = new SLCodec();
    private Ontology ontology = KareOntology.getInstance();

    protected void setup() {
        manager.registerLanguage(codec);
        manager.registerOntology(ontology);

        // Register the service in the yellow pages
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());

        ServiceDescription sd = new ServiceDescription();
        sd.setType('ArtifactManager');
        sd.setName('KARe Artifact Manager');
        dfd.addServices(sd);

        try {
            DFService.register(this, dfd);
        } catch (FIPAException fe) {
            fe.printStackTrace();
        }

        addBehaviour(new ArtifactManager_Listener(this));
    }

    public void takeDown() {
        // Deregister from the yellow pages
        try {
            DFService.deregister(this);
        } catch (FIPAException fe) {
            fe.printStackTrace();
        }
    }
}
```
The system agents are connected to the peer discovery module and the surrogate architecture via a lightweight component that includes an agent that reports to the Peer Assistant the information gathered by the mobile application. The description of such component is deferred to the section 5.2.3.

### 5.2.3 Peer Discovery Module

This component is a nomadic service for artifact recommendation. It is built upon the interconnect architecture (see Figure 5.3) to enable the communication between the service running in the handheld device and a surrogate object running in a desktop computer.

![Interconnect Architecture](image)

Figure 5.3: The Interconnect Architecture.

This interconnect architecture (Fig. 5.3) was implemented in [11]. The components developed there were reused here to enable the communication between handheld devices and the surrogate object that is connected to our agent platform. This architecture uses HTTP to communicate between the parts.

In figure 5.4 we see the objects’ distribution in the nomadic service side of our implementation. The KARePDAGUI is the implementation of the GUI running in the iPAQ. It instantiates a KAReNomadicService object as we can see in the listing 5.7. The Bluetooth class is the simulation of a JSR-82 [23] implementation.

Listing 5.7: Nomadic Service Initialization

```java
private void initService(){
    try {
        service = new KAReNomadicService('http://localhost:8080/kare.jar',
                                        'kare');
```
SurrogateConnection is a class that holds the connection between the nomadic service and the surrogate object. Furthermore, the class is responsible to handle the message exchange between these two components. The messages are encoded in HTTP and sent over the surrogate connection object as seen in the listing 5.8.

Listing 5.8: Message exchange example between the Nomadic Service and the Surrogate

```java
public void sendDeviceAddress(SurrogateConnection surrogateConn,
        String deviceName,
        String blueToothAddress) throws IOException {
    if (surrogateConn == null)
        throw new IOException('Surrogate connection is null!');

    OnewayMessage reqm = (OnewayMessage) surrogateConn.
        createMessage(Message.T_ONEWAY, (byte)7);
    Encoder enc = reqm.getEncoder();
    enc.encodeString(deviceName);
    enc.encodeString(blueToothAddress);
    surrogateConn.invokeMessage(reqm);
}
```

In figure 5.5 we present the objects on the surrogate side. The KAReSurrogate class is responsible to act on behalf of the device, i.e. it handles the service messages that are requested to the device. The handheld devices are connected to the surrogate via an interconnect and exchange. Further, the surrogate is loaded into the device via a webserver. Usually the surrogates are used to enable devices to participate in a Jini network. However, here we replaced the Jini services by the agent services. In our case, the recommendation services provided by the Peer Assistant and Artifact Manager agents.

The messages received by the surrogate are actually handled by its servant. In
Fig. 5.5 the servant is represented by the \texttt{KAReClientServant} class. The servant has an instance of the intermediary between the surrogate and the system agents, i.e. the \texttt{Surrogate2AgentCommunication} class.

This class bridges the components by translating information to an understandable format for each of connected components. The surrogate object and the system agents exchange two kinds of messages: 1) information about the bluetooth enabled devices in the vicinity (which has been collected by the nomadic service and transmitted to the surrogate) and 2) recommendation gathered by the user’s \textit{Peer Assistant}. So, when the surrogate object receives a message from the recommendation service it should translate it to a format understandable by the \textit{Peer Assistant}, i.e. FIPA ACL Messages [15]. Furthermore, when the \textit{Peer Assistant} receives the answer to a question it should translate it to a human-readable format.

Since the \textit{Peer Assistant} is a GUI-enabled device, and thus a heavy component, we chose to put another agent in between the surrogate and the \textit{Peer Assistant} itself. We called this agent the \texttt{SurrogateAgent}. This agent holds an instance of the class \texttt{Surrogate2AgentCommunication} and thus is able to detect when messages arrive to the surrogate. Furthermore, when the \textit{Peer Assistant} receives a question answer it forwards it to the \texttt{SurrogateAgent} and the latter passes it directly to the surrogate via its \texttt{Surrogate2AgentCommunication} instance as we can see in the listing 5.9.

```
Listing 5.9: Integration between the surrogate object and the \textit{Peer Assistant}

public class Surrogate2AgentCommunication {

```

5.3. Conclusion

In this chapter, we have discussed the implementation of this work. We begin by showing what has been implemented from the design and the approach taken to implement the software. Furthermore, we briefly introduce the technologies used to implement the KARe system.

We used Java as the main language to implement the prototype. The JADE agent...
framework was used to develop our system agents and J2ME Personal Profile was used to implement the nomadic service for artifact recommendation. Furthermore, we have used Apache Lucene to support the document indexing and searching. XML was used to represent the document taxonomy and Apache XML Beans was used represent the instances of xml documents within the system.

To develop the ontology used for the agent’s communication we used Protégé Ontology Editor [32] and the Beangenerator Protégé plug-in. The interconnect architecture was used to provide communication between the nomadic service and the surrogate objects that provided the agents’ services. Finally, we introduced each of the implemented components in detail and showed the most important methods that compose our system.
This chapter presents the methodology to evaluate the proposed recommendation algorithm. The results of our evaluation are also shown here.

This chapter is structured as follows: Section 6.1 describes the methodology of the experiment taken to perform our evaluation. And section 6.2 shows the results of our experiment when comparing the standard approach based on the vector model with the proposed approach of this work. Finally in section 6.3 we derive some conclusions from the experiment.

6.1 Methodology

The contexts in our experiment are given by two taxonomies classifying scientific papers. Taxonomy A has been created by a PhD student to collect papers of her interest, prior to the development of the system. Taxonomy B is taken from the ACM Computing Classification System. Our idea is to simulate the questions and answers using the title and the body of the scientific papers respectively. We tested if the algorithm was able to retrieve a paper giving its title or keywords from the abstract as a query. This approach is not in fact the ideal scenario.

The best situation would be to compare the approaches using a dataset composed of real questions and answers classified by a taxonomy. However, since this dataset is not available at the moment, we chose to simulate the questions and answers using scientific papers. The experiment has two distinct phases as follows:

\[\text{http://www.acm.org/class/}\]
1. Preparation of the taxonomies:
   - Select the papers that are used as queries. These papers should be classified by both taxonomies so that we know which is the contextualizing concept in the questioner’s taxonomy and the concept the algorithm should find in the responder’s taxonomy;
   - Subtract the selected papers from the questioner’s taxonomy to avoid bias (i.e. the keywords of the selected papers should not be used to compute the concept vectors);
   - Subtract the titles of all papers from both taxonomies to avoid bias (i.e. the title keyword should not be used to compute the concept and document vectors).

2. Experiment execution:
   - Take the title of a document selected as query;
   - Choose the concept associated with this paper in taxonomy A (questioner’s taxonomy) to contextualize the query;
   - Submit the contextualized query to the algorithm, which searches for the answer in taxonomy B (responder’s taxonomy);
   - Examine the result set, seeking for: a) the right concept in the responder’s taxonomy, i.e. the concept in which that given paper has been classified in taxonomy B; and b) the given paper itself.

We have performed the experiment using the standard approach based on the vector model and the approach proposed in the section 4.4. In our approach we have considered two options: 1) to have on the result set only the best matching concept with the questioning concept; and 2) to have a small subset of concepts that best match the questioning concept. In figure 6.1 we describe the scenario of our experiment:

The steps exhibited in Fig. 6.1 are described bellow:

1. To distribute documents about a common domain in the taxonomies extracting their titles;
2. To select one paper from the user questions database and submit its title for questioning;
3. To send the paper title and the vector of the concept associated to it in ACM’s taxonomy to a destination taxonomy, i.e. Renata’s Taxonomy. This forms the “contextualized query”;
4. To examine the result set, seeking for the paper with corresponding title and other papers that are related with the correct concept from the originating taxonomy.
After repeating these steps several times, we are able to evaluate the algorithm in terms of recall (i.e., the fraction of relevant documents retrieved) and precision (i.e., the fraction of retrieved documents that are relevant) and the harmonic mean “F1” of recall and precision [35, p. 82]. The relevant documents here are the ones belonging to the concept of the “targeted” document. Finally, we compared these measures with the ones resulting from the search not using the query scope reduction approach to evaluate our algorithm.

### 6.2 Results

We have performed 77 queries over our taxonomies, and the results are shown in the table 6.1 and in the figure 6.2.

<table>
<thead>
<tr>
<th></th>
<th>Standard Approach</th>
<th>Proposed Approach (1 concept)</th>
<th>Proposed Approach (5 concepts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Queries</td>
<td>77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Relevant Documents</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relevant Documents Found (RDF)</td>
<td>264</td>
<td>500</td>
<td>395</td>
</tr>
<tr>
<td>Documents Found (DF)</td>
<td>62</td>
<td>25</td>
<td>57</td>
</tr>
<tr>
<td>F1 (DF)</td>
<td>0.80</td>
<td>0.32</td>
<td>0.74</td>
</tr>
<tr>
<td>F1 (RDF)</td>
<td>0.17</td>
<td>0.32</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 6.1: Experiment Evaluation Results

The first column of the table shows the results of the vector model. The column at the center shows the results of our approach when returning documents from only
one concept on the result set. The last column shows the results of our approach when returning documents from up to five concepts on the result set.

We have calculated the F1 measure based on two measures: 1) the number of times that the algorithm found the specific document which title was being searched (which we call DF) and 2) the number of related documents to the one being searched, i.e. the number of documents that are classified under the same concept as the one being searched (which we call RDF).

For the F1(DF) value, the vector model has a better F1 measure. However, it performed poorly for the F1(RDF) value. Our approach returned more documents from the same concept as the query than the vector model did. However, initially the F1(DF) value of our approach was very low compared to the vector model, so we have changed the experiment. Instead of returning only documents from a single best matching concept, we tried to improve the number of specific documents found by increasing the number of concepts on the result set.

Taking this approach, we have increased the number of specific documents found to almost as many as in the vector model. Our “increased” approach had an advantage over the vector model. We could retrieve many more documents related to the one being searched, which gave us also a good F1(RDF) value. On average we found that our approach with increased number of concepts performed best during the experiment. It was slightly better than the vector model but it performed well in the two kinds of measures we were interested.

The vector model produced almost the same result on average but the number of related documents was very low on the result sets. The disadvantage with that is that the vector model will produce result sets that have many unrelated documents, or noisy result sets. Finally our initial approach has produced constant F1 values during the experiment. Furthermore, it showed to be the best approach when searching for related documents.
only (instead of specific documents as we did here).

6.3 Conclusion

In this chapter, we have discussed the methodology and results of an experiment to evaluate the recommendation algorithm proposed in chapter 4. We begin our discussion here showing the phases of the experiment (its preparation and execution) and we provide a small scenario to describe it.

Further, we exhibit the results of the experiment. On the results we compare our proposed approach with the vector model. After some initial results we have increased the number of concepts on the result set of our approach to evaluate its retrieval performance.

We concluded that on average our approach with increased number of concepts on the result set has performed better than the others. However, if the user is searching for documents that are related to a specific concept, rather than with some specific document, our initial approach has performed better as we can see by its F1(RDF) value. On the other hand, if one is interested on finding very specific documents more than finding documents about a specific concept, then the vector model is the most appropriate choice.
CHAPTER 7

Conclusion

This chapter presents the main contributions of this thesis, draws some relevant conclusions and identifies points where further work is necessary.

This chapter is further structured as follows: Section 7.1 presents our general conclusions and main contributions of this thesis, and Section 7.2 identifies the future work.

7.1 General Conclusions

We have proposed KARe, a multi-agent recommender system that simulates knowledge sharing social behaviors amongst nomadic users in a peer-to-peer environment. The main idea behind it is to promote interaction through questions and answers, aiming at facilitating the exchange of both explicit and tacit knowledge. The core of the system is an information retrieval algorithm that has been the focus of this work. Here are the main differences of KARe compared to other recommender systems:

(a) We provide a question/answer function, instead of a simple keyword search. The user needs are indicated through an imitation of the human social behavior of asking and answering questions to colleagues, i.e. stimulating the user’s interaction by supporting the question pedagogy [31];

(b) The users structure their knowledge according to their own conceptualization of a domain. This gives autonomy to the users to organize knowledge following their own points of view and language;

(c) The users only share documents that they want to share. This kind of knowledge
is decentralized, i.e. it is present in each user’s computer, not on a central server host. Furthermore, this results in the artifacts being available locally, rather than captured by a crawler;

(d) We support knowledge exchange amongst nomadic users. The users insert their questions in the system and it will search for recommendations in the network of users that are nearby.

The results of the performed experiment have shown a gain in the recommendation quality using the proposed approach. One interesting issue investigated in our retrieval approach is the possibility of finding not only one but a few similar concepts in the responder’s side (e.g. in our experiment we tried with 5 concepts). This allowed us to get better results considering that the responder’s taxonomy can be more refined than the questioner’s context.

Furthermore, we designed an architecture for nomadic recommender systems, extending KARe towards a mobile platform. Such a service seeks service providers at the user range and exchanges knowledge with them via the recommender agents. The goal of this architecture is to take advantage of current wireless technologies, while at the same time, coping with current resource limitation of mobile devices.

7.2 Future Work

The current KARe system does not explore all the identified challenging issues. The following list presents the topics which are indicated for further investigation:

- To confirm our conclusions about the information retrieval approach by experimenting the algorithm against different and larger datasets;

- To use different measures for calculating the weights of the vectors’ dimension values especially in the concept vectors. A possible approach is to smooth the values taking into account the neighbor concepts. We could also evaluate the algorithm with different index sizes;

- We can verify the possibility of allowing the questioner to contextualize the query in more than one concept;

- To investigate the use of the relations between nodes (parents and siblings) to enhance the retrieval approach.

- To support the message exchange amongst users and finally to explore the possibility of answering questions with stored messages instead of artifacts;
• To integrate the surrogate architecture with our agents’ architecture instead of bridging the gap between them. The implementation of an agent framework that supports the interconnect as its underlying message transport protocol would bring novelty to agent platforms for mobile devices with limited resources.


[33] Renata S. S. Guizzardi and Anna Perini and Virginia Dignum. Providing Knowledge Management Support to Communities of Practice through Agent-oriented Analysis. In Proceedings of the 4th International Conference on Knowledge Management, Graz, Austria (June 2004).


