Swarm-based information retrieval

Automatic knowledge-acquisition using MAS, SI and the Internet

Håvard Rykkelid

Master of Science in Informatics
Submission date: June 2006
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Contents

Preface

This thesis was written by Håvard Rykkelid during the fall 2005 and spring 2006 at Institutt for Datateknikk og Informasjonsvitenskap (IDI), NTNU, Norway, with Keith Downing as teaching supervisor. The author has also received valuable help from Eric Monteiro, Pinar Öztürk from IDI and my close friend and fellow student Sveinung Monsen. This thesis is my finishing work for a M.Sc. degree in Artificial Intelligence and Learning. This thesis tries to combine interesting elements from multi-agent systems and swarm intelligence with automatic symbolic knowledge-acquisition using the Internet as source. A large part of the effort put into this thesis has been making a working system to prove the success of combining these elements in practice. This paper will therefor often refer to that program.

Abstract

In testing the viability of automatic knowledge-acquisition, using simple techniques and brute force on the Internet, a system was implemented in Java. Techniques from both multi-agent system and swarm intelligence paradigms were used to structure the system, improve searches, increase stability and increase modularity. The system presented relies on using existing search-engines to find texts on the World Wide Web, containing a user-specified key-word. Knowledge is identified in the texts using key-sentences, terms related to the key-word becomes new key-words in an incremental search. The result is expressed as sentences in a KR-language. The answers from a run were often interesting and surprising, and gave information beyond an encyclopedic scope, even if the answers often contained false information. The results of the implemented system verified the viability of both the designed framework and the theory behind it.

Acknowledgements

> Not everything that can be counted counts, and not everything that counts can be counted.
> - *Albert Einstein*

This thesis signals the end of my formal education (for now) and so it is natural that I look back on what have brought me here.
I am very grateful for my past, which has led me to this point.
Particularly I am grateful to all the people that have helped me - or made me - walk my own path through life. If you are reading this and wondering if that includes you, then it probably does. It also include a lot of people who will never read this. Whoever they are and wherever they are I am grateful to them, and I thank you all.

- Håvard Rykkelid

Chapter 1

Introduction

This paper will talk about the theories, reasons and results in connection with implementing an automatic knowledge-acquisition system relying on multi-agent architecture and swarm intelligence principles. In this stage the system is dependent on the Internet as source for all its information. This paper will discuss some theories and reasons for using the Internet as source, as well as discussing some pros and cons of all aspects of this type of system. The terms used above may be unknown to the reader. These terms will be explained in greater detail in later chapters, but here is a small description of the key-terms:

Automatic knowledge-acquisition is a term that describes some sort of automated process (a computer program) that will seek certain sources of information and extract knowledge from those sources. This knowledge will often be represented in such a way that it makes it possible for a computer program to reason over the knowledge gathered.

Multi-agent system (MAS) is a term that describes a computer program that consists of a lot of individual small programs that will run simultaneously and interact with each other. It is comparable with how an anthill work. The anthill may have a general purpose, but the execution of that general purpose is fulfilled by lots of small independent insects that each react to their surroundings and each other, without any form of centralized micro-management.

Swarm intelligence was first used by Beni, Hackwood and Wang [6, 8, 9, 7, 13, 14]. The term was then used strictly in the context of cellular robotic systems. It is however argued in the book "Swarm Intelligence: From Natural to Artificial Systems"[10] that this definition is unnecessarily restrictive, and a new definition is suggested. The new definition is suggested to include any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies.

Chapter 2

Motivation

'I wish life was not so short,' he thought. 'Languages take such a time, and so do all the things one wants to know about.'
- *J. R. R. Tolkien*

2.1 Why automatic knowledge-acquisition?

2.1.1 Background

The modern world is full of information, and so is the Internet. The modern man needs to access and digest a lot of information in his day-to-day business. The transformation of information to knowledge can be a painstaking operation. The problem today is not so much the absence of information, but the abundance and chaos of sources and interpretations. There is also a large amount of misinformation and information that is not so much targeted at enlightening people as it is targeted at making people believe what the sources of the information want you to believe. Within this jungle of sources, of facts and fictions we have to not only dig up information on our topic, but also translate this information into personal knowledge. Translating implicit(tacit) information into knowledge is very hard, and has a lot to do with experience. Explicit information is much faster to learn from, but not all information will necessarily exist in an explicit form [16, 12]. A human being cannot always spend enough time checking lots of different sources and verifying the content, especially if he needs to understand several related concepts.

The standard solution might be to browse the web, consult an encyclopedia or something similar (e.g. Wikipedia). Browsing the web is often a tiring exercise, which often yields poor or no results. The information is out there, but it's very hard to find. The problem with information found in encyclopedia-like sources is that it depends on updated information, and it expects the reader to have some basic knowledge of how the world works, it may even expect the reader to have pretty specific knowledge within the area of interest. In addition it may also state implicit information which is crucial for understanding the subject but not clear since it is not explicit. In most cases consulting an encyclopedia is better and more reliable, but in some cases it may not be sufficient. This may be because of outdated information, biased information, incomplete information, etc. So how do we make the implicit information explicit, and give an overview of an entire field of interest? How do we create information that will be easier to translate to knowledge for the user?

The automatic knowledge-acquisition solution One solution to getting information that is probably, with emphasis on probably, less biased (or at least not getting its bias from only one source), more up to date, more explicit and thus easier to digest, is to make a system that checks many different sources and extracts explicit information from both explicit and implicit information-statements. With the added function of organizing the information in a semantic-net so that the information can be reasoned over, the system should be able to give a comprehensive and current description of an area of interest. The system may also link to a page where you can find much of the information you seek, thus maybe linking to a page of an online encyclopedia.

2.1.2 Related work

There are several ongoing projects trying to use automatic knowledge-acquisition with several different methods on several different sources. Some of these projects try to improve results from database queries [23, 22]. Much effort has been put into trying to extract knowledge from medical texts and much of the ground-work for this field was laid as early as in the seventies by the

“Linguistic String Project” (LSP) [18]. These two first examples are typical for searching for knowledge within specific domains. Recently there have been a lot of interest surrounding the application of artificial neural networks (ANN) in the context of understanding natural language. There are several projects using artificial neural networks that are making good progress [21, 3].

2.1.3 What is new?

Precision The ambition of this project was not to create a perfect system that always gave perfect outputs, some precision would readily be sacrificed on the altar of simplifying the process.

The quality of the sources would not be important. While most other projects are very concerned with what sources to use, this project would rely solely on the massive amounts of information found on the Internet. Thus relying on “false” information being drowned out in the big picture.

Light and fast The algorithm used to recognize knowledge in this system was deliberately simple, so that it could hopefully run very fast. This system is not concerned with understanding and retrieving every little piece of knowledge from the texts, but relies on using its simple method on larger volumes of texts thus relying on quantity and not quality.

Modularity has been an important issue while developing this system. There are still some things that could have been even more modular, but this system is very modular and therefor very easy to modify. Most projects start with a lot of good intentions about modularity but not so many are able to see them through to the end.

2.2 Why multi-agent system?

2.2.1 Background

Conventional computer-programs are often very rigid structures which cannot tackle the unexpected, cannot easily be altered and follow a specific sequence of execution. They are however usually very fast if they do not have to wait on external resources.

Multi-agent systems When you create a program like the system described in this paper, you in essence start up a lot of small and, more or less, independent programs. If you would compare a multi-agent system with its single-thread counterpart, each of the agents in the multi-agent system would contain a small part of the code from the single-thread counterpart. And in many cases there would be several agents with the same code running at once. Some multi-agent systems even share the code between them, so any one of the agents can do any job in the system.

Comparison: It is easy to see that a multi-agent system will use a certain amount of resources to administer all the different agents, and so will use more resources than the single-thread system. That means that if all the information needed for each part of the program is always available to the program, then the single-thread system would be faster. With respect to this system that is not the case. This system is highly reliant on the Internet, which means that some parts of the system will be waiting for a response. In this case a multi-agent system could earn some time by simultaneously working on something else while waiting for a response. Additionally a multi-agent system can most of the time still keep working even if it encounters something unexpected, because if one of the agents die, usually you will have many other agents still doing the same job. You will however probably lose a small piece of data, which may or may not be important. The final advantage of a multi-agent system is that if you plan a system to be many small and individual agents, then the work of replacing or modifying some function should be much simpler, because it means that the tasks that your system will perform have been designed in such a way that they are very independent. It is possible to argue that by making a multi-agent system, you force a program that is very modular.

It has already been mentioned that one drawback of multi-agent systems is a greater overhead in monitoring all the threads. Other problems include:

How to send data and messages between agents, how to synchronize behavior (start, stop, etc) and sometimes more complex debugging.

2.2.2 Related work

There have been quite some effort put into researching and developing useful frameworks for implementing multi-agent systems. There are several frameworks already in use, like Mobile Agent System Interoperability Facility (MASIF) and the Foundation of Physical Intelligent Agents (FIPA). One system that have been implemented using the FIPA specification is the "Technology for a Realistic End-User Access Network Test-bed" (TORRENT) project [1]. FIPA specifications have also been used to develop the "Java Agent Development Environment" (JADE) [5] which is a framework that tries to simplify development while ensuring standard compliance. These examples does however relate to systems where heterogenous agents need to interact. Multi-agent systems does not have to be systems with competing and distributed systems.

2.2.3 What is new?

In the system presented in this paper there will be some more detailed explanations on how the framework for the agents works and what thoughts lie behind those decisions. The framework presented in this paper is not competing with systems like JADE for heterogenous agent interoperability and the like. But the system presented in this paper should serve as a good foundation for building small and independent multi-agent systems and the system has an easily configurable and reconfigurable structure. This means that adding, removing or replacing agents or rerouting information is very easy with this system. This could be very useful features for prototyping, where you want to test different configurations and the like.

2.3 Why swarm intelligence?

2.3.1 Background

Swarm intelligence (SI) is an interesting field within artificial intelligence (AI). And the term as presented in the definition stated in the introduction to this paper defines a relatively new discipline within the AI community. Although the area is pretty new, it has already yielded some very interesting results. Especially when solving problems of combinatorial optimization [2]. There are however many tasks that swarms of insects solve in an effortless manner which seem almost inconceivable when considering the rather simple nature of the individuals in that swarm[10]. The capacity for such surprising levels of self-organization is very inspiring and interesting to work with. Some aspects of SI-inspired self-organization have been emulated in the system presented in this paper, both as a way of visualizing concepts and structures in the system, but also for improving searches.

2.3.2 Related work

Swarm intelligence is much studied in a lot of contexts all over the world. Listing all of the areas and all of the projects would be an enormous task. The areas are also so diverse that listing specific projects would be unfair to the diversity of the application of this paradigm. It may instead be a good idea to read about possible applications of this paradigm as stated in the book “Swarm Intelligence: From Natural to Artificial Systems” [10]. This does probably not list all possible applications for this paradigm, but it covers many of the most relevant topics.

2.3.3 What is new?

There is probably not much new in the system presented in this paper where it comes to swarm intelligence. It is just a reuse of already proven principles of positive feedback, where good sources of information gets rewarded and the bad sources of information will be ignored if there are several good sources to pursue instead.

2.4 Why use the Internet?

2.4.1 Background

The amount of information you can find on the Internet is huge. The information you can find there spans all known fields of interest. The information comes from vast numbers of independent and varying sources. In other words you should be able to find what you are looking for there, regardless of what field of interest you have. There are certainly limitations, but most limitations have to do with the amount of effort you have to invest to find the information you seek.

2.4.2 Related work

Using the Internet as a source of information is done every second, every day, for any number of reasons. There are not many projects with similar signature to the project presented in this paper that uses the Internet as a source though. Most such type of projects will be more selective with their sources. There are however examples of similar projects using the Internet as source [3].

2.4.3 What is new?

It is a bit hard to tell what is actually new in the use of Internet as a source. But there are probably not many projects using brute force with the Internet as source, and this would not have been a conceivable method until recently, because of the speed of the Internet. Today information flows much faster on the Internet and it is possible to use this speed increase and the increase in processing power on the end-user machines to use brute-force to get good results from simple methods. At least that is the theory.

Chapter 3

Goals & Task description

See first that the design is wise and just: that ascertained, pursue it resolutely; do not for one repulse forego the purpose that you resolved to effect.
- *William Shakespeare*

3.1 Task description

Based on the swarm-concept. Using key-sentences combined with key-words to form relational concepts in a semantic network using the Internet as source. Thus a key-word is given and used with key-sentences to build a symbolic representation of the meaning and context of the key-word. The swarm will signal paths and points of interest to each other, to facilitate efficient retrieval of information.

3.2 Automatic knowledge-acquisition

Testing the viability of using simple techniques for knowledge-recognition, with brute force, on a source as interesting - but also complex and difficult - as the Internet.

3.3 Multi-agent system and swarm intelligence

Using techniques from both multi-agent system and swarm intelligence paradigms to structure the system, improving searches, increasing stability and increasing modularity. Additionally it was a goal for this project to realize ideas for a simple framework with high reusability for a multi-agent system, which would facilitate communication while keeping the individual agents as independent as possible.

Chapter 4

Method

Though this be madness, yet there is method in 't.
- *William Shakespeare*

I hear and I forget. I see and I remember. I do and I understand.
- *Confucius*

This chapter describes which techniques/approaches have been used to investigate/solve the problems and meet the goals presented in this paper. This will hopefully give the reader an understanding of the foundation that the arguments and solutions in this paper have been built upon.

4.1 The methodological approach

There are different paths you can take to prove a point. In the case of scientific research you can use a theoretic/analytic approach, modeling/abstraction approach or a design/experimental approach. This paper will rely on putting theories out in practice and modeling and creating a framework for future study and utilization. That means that the research in this paper will use a combination of a modeling/abstraction approach and a design/experimental approach.

Chapter 5

Theory background

If the facts don't fit the theory, change the facts.
- *Albert Einstein*

This section explains the theoretical foundation for the system presented in this paper. This section was written before or in early stages of system development, so some of the references to the system are in future tense.

5.1 Information into knowledge

Information is not the same as knowledge When someone knows something, that person has an internal representation of that knowledge. Unfortunately no one can read this internal representation of the knowledge he possesses. Therefore he has to translate this knowledge into information representing this knowledge. This interpretation into information is not always easy, and it will not always be easy to understand this information. The interpretation of the knowledge can be in many different media; verbal, drawings, text, etc. The recipient of this information will have to observe the information, and interpret it back into internal knowledge. It becomes clear that information is the bridge between minds that lets us share knowledge with each other, and that much depend on the interpretations [16, 20].

Machine reasoning When you want a machine to know something and reason about it you usually use symbolic representation of the knowledge. This representation of the knowledge is just a kind of systemized information.

Ordinary text, which is what we will mostly get from the Internet, is also a kind of systemized information, but it is not optimal for reasoning over, because it is ambiguous and unstructured. Because of this, you need to extract the important informational bits of the text, and restructure it if you want to make it easy to reason over. Our system will not be interested in all the information in a document, it will only concern itself with information that has to do with the current search. The way that our system will get structured information out of the unstructured information it encounters, is by matching key-sentences with the information presented. Consider the sentence: "A chair is a kind of furniture". If you want to know what a chair is, and used the key-sentence "is a kind of" to establish what kind of category a chair belongs to, then the you could try and match this sentence with any occurrences: "chair is a kind of X", where X would be the category for the chair. This was a simplified example, but it shows the general idea. This information can then be represented in structured information that can more easily be reasoned over.

It is important to note that this kind of structured information is not knowledge, because it is not something that really has any meaning to a computer, the computer just has some rules to apply to symbols so that it can reason over them, but it does not know what the symbols represent, and thus it is merely information and not knowledge to the system.

5.2 The System

5.2.1 Mechanics

Using terms and definitions from "Complexity, coupling and catastrophe" by C. Perrow [17] to describe some features of the mechanics of the system.

Complexity

In a complex system , by Perrow's definition, it is not easy to exchange components, or change simple behavior of the system. This is because the components of the system are not easily distinguishable because of a high level of interdependency with lots of feedback-loops. Each component is highly dependent on several of the other components because of a complex and rigid communication structure. How does our system fit in with this definition?

Our system will be highly modular, with several very independent agents. These agents will be started by the “Conveyor”-class, but even though Conveyor keeps references to the agents, they will be completely autonomous after they have been started. The agents will be referring to the variables that Conveyor contains, and there will be no communication between agents other than through those variables. Even the names of the agents that should be started and what variables they will each refer to is stored in a configuration file that Conveyor will access. This means that Conveyor will not contain any code specifying names of other agents, and is thus completely independent of the other agents. The only dependencies are that every agent has to receive variable-references for input and output, and are dependent on these variables to function, and that data sent using those variables conforms to how the agents expect them to be. This means that while our system may seem very complex and daunting to understand, it has been designed to function with a very low level of complexity according to Perrow’s definition.

Coupling

A system that has tight coupling , by Perrow’s definition, is a system which has a very rigid flow of information, has set buffer-sizes for information and a strict order in which the tasks are to be performed.

Our system will be like a conveyor belt in that the information will flow in one direction although the last agent also will introduce new information in the “in-box” of the first agent. This means that no single agent can do any work before they receive new information in their “in-box”. Some agents will probably have to wait, while others will always have a full “in-box”. The agents of our system are expected to work at different speeds, and thus the buffers have to be pretty big. However, since the information that is processed comes from an information loop, no part of the system should go on a rampage. The agents will be “blind” of each other and work if they have information to process, and relax if there is nothing to do. So in a way there is a definite work- and information-flow, but the agents will often all be working in parallel with each other, signaling robustness and indifference towards a strict order of progress. This again signals a loose coupling according to Perrow’s definition.

Benefits and limitations

The benefits of the mechanics of our system have mainly to do with robustness and maintenance. The system is designed in such a way that it should be easy to update, replace and delete modules. This is an important aspect for this kind of system, relying on the Internet for information. It is also natural to expect that other aspects of this system will be necessary to change in time. This very modular approach can also theoretically be good for creating a distributed system.

The negative side to this approach is that it means a lower efficiency. The system will have an overhead in keeping track of the agents, and the agents will sometimes idle.

5.2.2 Data-structures

Data-structures are often overlooked as an important aspect to consider when creating a system. The mentality is often that as long as it works that is all we need. However the data-structures are often very important to consider so that you chose data-structures that are easily interpretable and extendable.

Internal

The internal data-structures of our system will rely on the global variables in agent-supervisor for most of the internal communication. These variables will be vectors where one entry will be a bean with the necessary information. It is important to use a simple standard for these beans, so that it is simple and easy to send all required information with them. With this solution, the size of the data-structures will adapt to the needs of the system, and the beans can be passed along as object and modified on the way.

External

The data-structures for configuration, language packages, output and similar are maybe the most important data-structures to consider. These data-structures will decide how easy it will be for other programmers and programs to influence and use this system. There are many examples on how the interaction between programs and revisions of programs has been made difficult because of bad choices concerning data-structures. [19] But there are also examples where the problem is not the data-structures themselves, but cultural settings that dictate a different interpretation and usage of the data-structures. [11] It does however seem unlikely that culture will influence the use of data-structures in this system.

Our system will largely rely on the XML standard for settings and language packages, and will use both XML and Prolog for storing the results of the search. XML is a very well known and much used standard, and so is Prolog. The advantage of Prolog is that the information can then directly be used with a Prolog inference engine. Both the Prolog and the XML output should be easy to convert and use in another setting if desirable. However if you really want to streamline an output of another type it should not be very difficult to replace the existing encoding-agent in the system with another that suits your purposes. The choice of encoding redundant information was made to ensure a higher degree of compatibility. The operation of encoding the information in two different formats will not have a significant negative influence on the efficiency of the system.

Benefits and limitations

In the article “Integrating Health Information Systems: A Critical Appraisal” by Eric Monteiro [15], the point is illustrated that redundancy is not always a bad thing, and that it is wise to create information-systems that make translations of the created information easy by using standards as XML, Corba or the like. This article has inspired the firm belief in using such standards, even though it may be more work to implement.

The benefit of this approach have to do with the ease of which the information from this system can be transformed into a format useable by another system.

The disadvantage of using such an approach is that it takes elaborate structures to encode rather simple information in many cases. Besides the higher input for the programmer at the creation of the system, the parsing and encoding routines of these standards will also probably take more resources than custom-made data-structures. This last point is not very valid for our system though, since it rarely needs to access or create such information.

5.2.3 Information types

Using terms and definitions from "Knowledge management: The benefits and limitations of computer systems" by G. Walsham [20] to describe some features of the types of information the system will encounter.

Tacit

Tacit information is information that is not clearly defined. In a way it is information that is clearly present, but it is not discussed. A good example relating to our system would be semantic implicit information. A text may have an agenda and a theme, but even though the author of the text did not intend to explain relations between words and objects in the real word, outside of his agenda/theme, he will "by accident" do so. The sentences that he builds will contain lots of information about other stuff entirely. He may wish to convey that he has been treated badly on his visit to his neighbor, by stating something like: "The wooden chair I sat on was very uncomfortable." However this sentence also contain information about chairs. That they may be made of wood, that you can sit on them and that sitting on chairs may be very uncomfortable, maybe even more information that I did not think about.

All of this information may be very interesting for automatic knowledge acquisition, but it has nothing to do with the real message of the text. It is expected that our system will stumble onto much of its information from such tacit information on the web. We do not need to find statements that say exactly: “Chairs may be made of wood”. We can even use information that is even more obscure. If we find that chairs are a kind of furniture, and that furniture can be made of wood, then we may guess that chairs can be made of wood as well, by using a reasoning engine. It then quickly becomes apparent that with the power of the reasoning engine and tacit information, we can gather a lot of information from the right kinds of texts.

Explicit

Explicit information is the kind of information that has the purpose of stating the facts that we are looking for. This information is obvious. It is expected that a fair amount of the information that the system will gather will be explicit. This kind of information may be for example: “Chairs may be made of wood”.

5.2.4 Disaster and consequence

This section is inspired by the article “Complexity, coupling and catastrophe” by C. Perrow [17].

Potential problems

Our system is highly dependent on the structure and information on the Internet. That means that if the system is off-line for some reason, it can do next to nothing. At least that is the way that the system will be designed, but there would be no problem to make agents who would fetch information from other sources than the Internet. But the same would apply to all information sources; our system would be dependent on the structure and content of those sources. If the structure somehow changes and none of the agents can extract information from anywhere, then the system will suffer a major breakdown. No information will be processed and no information will be returned to the user. However the system should elegantly be able to handle it if not all source agents go down, although maybe with poorer results. Another major problem would be if the information from the source was faulty or the processing of the information was faulty, so that the results would be completely off target. The system would still do its job, but the results would not be correct. This last problem is a question of scale. In smaller versions of this last problem the system should be able to handle it. Since the Internet is not a highly reliable source of information, the system will keep track of the frequency of hits on differing information, and will present the results with some sort of indication to the probability that the information is correct. This will hopefully take care of small problems in the source-information.

Victims

The victim will naturally be the user of the system. This can either be an AI or a human user. The consequences of this failure will depend on the gravity of the failure and the goal of the user. The consequences will range from disappointment, failure to retrieve vital information or wrong conclusions drawn from the wrong information to an endless deadlock waiting for crucial information from the system.

5.3 The User

5.3.1 Types of users

The users will naturally play a very important role in the practical application of this system. It seems natural to divide them into two categories.

AI

It is this author's fond hope that this system or similar systems will be used by programs with some sort of intelligence to help them reason over subjects of interest. Even though this subsection has the ambitious name 'AI', it is also meant to include normal computer programs that are not intelligent in any way. If the program is intelligent then it may, for its own purposes or to help out a human that asks for help, try and find out information about a subject and build a small knowledge-base for that particular field. The information may be used and thrown away, or it may be stored for future reference. Often it will be impractical to reason about all the knowledge in the world, and it will then be much better to just select the area of interest and reason over this smaller domain.

Human

When referring to human users in this section, it will mean humans that directly interact with the system. These users will use the provided interface and possibly the provided reasoning engine to fulfill their goals. (Note: No reasoning engine will be included)

5.3.2 Goal of the user

AI

The goal of a program will almost always be linked with a goal that a human end-user might have. The exception I can think of is an intelligent program that has been programmed to pursue its own personal agenda, independent of any agenda the creator or owner of the AI might have had. If the user is a non-intelligent program, it will not be searching for information to satisfy its own curiosity, but it may be an actor for a human just as the AI might. The goal of the program will probably most often involve building temporary knowledge bases and reason over them for specific purposes, and throw them away when they have outlived their usefulness. Another possible goal could be to gather statistics from the web, where the program would probably build a lot of knowledge-bases and extract some useful statistical data from them. There are probably a lot of other motivations that these users also could have, and many of them will probably be linked to a human end-user.

Human

The goal of the human user may be: To know more about a certain subject, to experiment with the system, to reason over a subject, etc.

5.4 The Internet

5.4.1 Information

The Internet contains a lot of diverse information. To understand how our system will work, and what kind of information one can expect our system to find on the Internet, it is prudent to take a closer look at what kind of information you can find on the Internet, and why.

What?

The easy answer to "what information can you find on the Internet?" is "Everything!", but that is not very informative. The Internet is full of information from all kinds of people and organizations. There are blogs, advertisements, pornography, tutorials, software, general information, search engines, hoaxes, humor, personal adds, gossip, racist statements, encyclopedias, etc.

Why?

When you consider whether to trust a source or not, you should always consider why the source is offering information. Is it an altruistic source, or is there a hidden agenda of some sort. Some sources are more reliable than others, on its worst the Internet is totally worthless as a source of information, and on its best it is priceless. The fact you can exploit is that often the sources of false information do not agree with each other, and therefore the correct information will be more frequent. But it is probably nevertheless smart to have in mind that many sources on the Internet have an agenda, and such is not trustworthy. Identifying some of the main drive-forces for seeding misinformation is probably a good idea.

5.4.2 Structure

The Internet is highly distributed and uses a set of very reliable protocols. A history of the protocol-disputes is described in: "The Internet Challenge: Conflict and Compromise in Computer Networking" [4]. There is no strict pathways for information flow, or any very rigid protocols that dictate firmly how the information needs to be packaged and routed through the network.

The Internet is just a number of connections that try as best it can to transport information from one end to another.

5.4.3 Benefits and limitations

Information

The benefits of the information on the Internet is that it has not been written by any one author, but by very many. That way much of the information will be biased, but at least the bias will not be the same for every source of information. As such it may yield more neutral information if you collect and compare information from lots of different sources. If the information you get from the Internet has any bias it should then be because many authors has this bias, and therefore may be a legit bias to present. Another advantage is the content of the Internet, which is very varied and up to date. Of course you may find outdated information there, but if you are searching for information about new subjects that may not have been included in encyclopedias yet, then you would still often be able to find it on the Internet. The Internet will offer information from almost all kinds of people all around the world, and this may be a great advantage. It is tempting to draw a parallel with "A Theory of the Firm's Knowledge-Creation Dynamics" [16]. That article points out that one possible reason why Japanese industries have been so successful is that people from all kinds of position within the firms communicate a lot with each other. The mix of people from very different positions allegedly creates much more interesting information and ideas. This concept can be applied in connection with the Internet, and one can argue that the Internet will have the same effect on information and ideas on a world-wide scale. The same line of argumentation can be applied in connection with the "Mechanisms for producing working knowledge: Enacting, orchestrating and organizing" article [12]. This last article claims that dynamic information is important to create interesting information, and the Internet is indeed a very dynamic information source.

The arguments against using the Internet as an information source are much the same. The information will probably be biased, and the danger of getting results that also show this strong bias (racist, advertisements, etc.) is therefore high. The information may be a series of very conflicting statements as the system picks up information from lots of different sources. Another argument why you should not use the Internet as a source is because it can be very hard to find the useful information you seek.

Structure

The benefit of the structure is that it is a very reliable and robust structure, the absence of set pathways means that the information will have many different routes open to travel, and thus will get there even if some parts of the Internet is down. The packaging, routing and packet-loss monitor does not rely much on the network itself.

The down-side of this, is that it means more work for the end-user of the network, which means a higher demand on computer resources. This is however not a very significant demand, and is a most acceptable sacrifice to gain the reliability and robust structure of the Internet.

5.5 Putting it all together

The system presented in this article has two different users in mind.

The computer-user , often referred to as the AI, needs to use a program such as this to be able to extract information from the Internet, or another program with the same goal, but different approach. A computer-user cannot get any knowledge from the Internet without some sort of interpretation routine, so such a system is dependent on a system like this or similar to gain anything from the Internet as an information source.

The human user , can get information from the Internet just by reading the content of different pages. How can our system compete with ordinary browsing and search-engines? One answer could be to offer what the user already has, in form of referring to relevant pages for information, and at the same time offer something on top of that, namely structured information on how the key-word relates to other words, and how those words again relate to other words, etc. But to create results that are useful there are many things to consider.

Things to bear in mind Throughout this chapter there has been presented a lot of points on things to avoid, things to use and things to bear in mind when designing a system like this. These are all points that have to be carefully considered when designing this system, so that it is possible to avoid the worst pit-traps. The system should be designed with a low level of complexity and low coupling [17]. It is important to use data-structures that are easy to translate, like XML and maybe Prolog-code [15]. We should try to anticipate what kind of accidents may occur and try to handle those potential problems [17]. We should think carefully about what kind of information can be found on the Internet, and how we can possibly screen out some of the information, so that the Information we retrieve will be more relevant, current, less biased and at the same time from a wide range of sources. If all of these points are taken into consideration when building this system we should hopefully be able to build a very useful system.

What now? The system needs to be written and tested and rewritten. Future embellishments of this system, if it is worthy of life, would maybe include: Search in different languages, formatting the information as a textual presentation of the subject, use of the Conveyor concept and class for other exciting projects and maybe this system as an additional search-option when using search-engines.

Note: (Today) The system works and produces very interesting results. The embellishments listed above are still good suggestions for what could be done in the future.

Chapter 6

Implementation

> One describes a tale best by telling the tale. You see? The way one describes a story, to oneself or to the world, is by telling the story. It is a balancing act and it is a dream. The more accurate the map, the more it resembles the territory. The most accurate map possible would be the territory, and thus would be perfectly accurate and perfectly useless.
> - *Neil Gaiman*

It is a balancing act to figure out what should and should not be included in a retelling of the tale. This chapter will start with showing a class-diagram of the system presented in this paper. Use this class-diagram as a reference. No class will be discussed in absolute detail, but most will be described in some detail. Some important principles and features of parts of the program will be explained in greater detail, but with a focus on generalizing the principles and not on this specific implementation. This implementation is not meant to be a commercial product, and so there are many things that could have been improved upon. But this implementation serves to prove that this works to some extent. And it may be a good system to build upon or refer to in future research in this area or maybe even in a commercial solution. Since this system is implemented to prove what can be done, some of the agents described does not represent only one type of agent, but rather a family of closely related agents with similar functions.

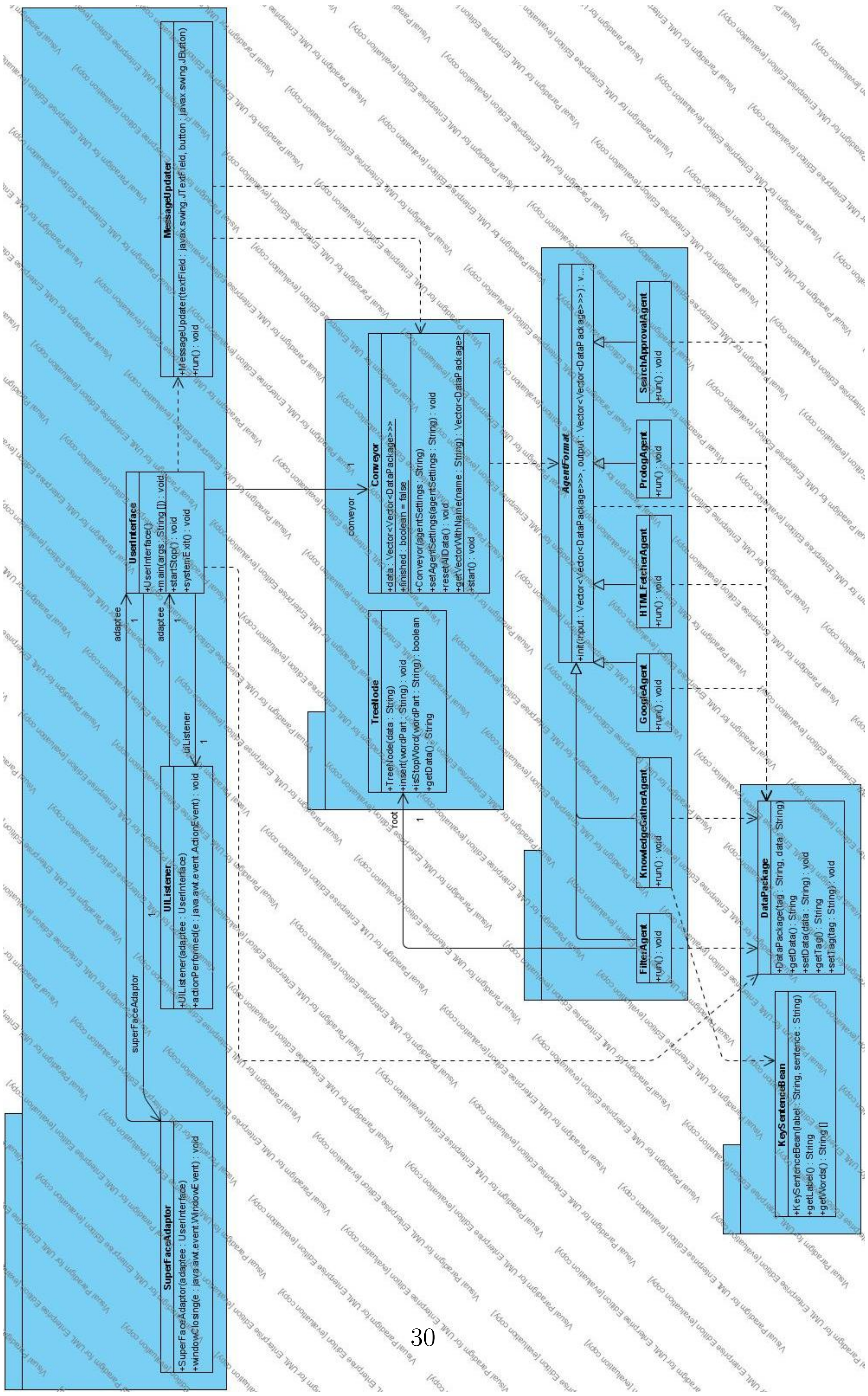

MessageUpdater
+MessageUpdater(textField : javax.swing.JTextField, button : javax.swing.JButton)
+run() : void
UserInterface
+UserInterface()
+main(args : String []) : void
+startStop() : void
+systemExit() : void
adaptee
1
UIListener
+UIListener(adaptee : UserInterface)
+actionPerformed(e : java.awt.event.ActionEvent) : void
uiListener
superFaceAdaptor
SuperFaceAdaptor
+SuperFaceAdaptor(adaptee : UserInterface)
+windowClosing(e : java.awt.event.WindowEvent) : void
conveyor
Conveyor
+data : Vector<Vector<DataPackage>>
+finished : boolean = false
+Conveyor(agentSettings : String)
+setAgentSettings(agentSettings : String) : void
+resetAllData() : void
+getVectorWithName(name : String) : Vector<DataPackage>
+start() : void
TreeNode
+TreeNode(data : String)
+insert(wordPart : String) : void
+isStopWord(wordPart : String) : boolean
+getData() : String
root
AgentFormat
+init(input : Vector<Vector<DataPackage>>, output : Vector<Vector<DataPackage>>) : v...
SearchApprovalAgent
+run() : void
PrologAgent
+run() : void
HTMLFetcherAgent
+run() : void
GoogleAgent
+run() : void
KnowledgeGatherAgent
+run() : void
FilterAgent
+run() : void
DataPackage
+DataPackage(tag : String, data : String)
+getData() : String
+setData(data : String) : void
+getTag() : String
+setTag(tag : String) : void
KeySentenceBean
+KeySentenceBean(label : String, sentence : String)
+getLabel() : String
+getWords() : String []

Figure 6.1: Class Diagram

6.1 The Metaphors

When working with this system, it was natural to start envisioning certain ways to view the system. This section tries to convey the image of these metaphors. On the next page is a graphical illustration of how this can be envisioned.

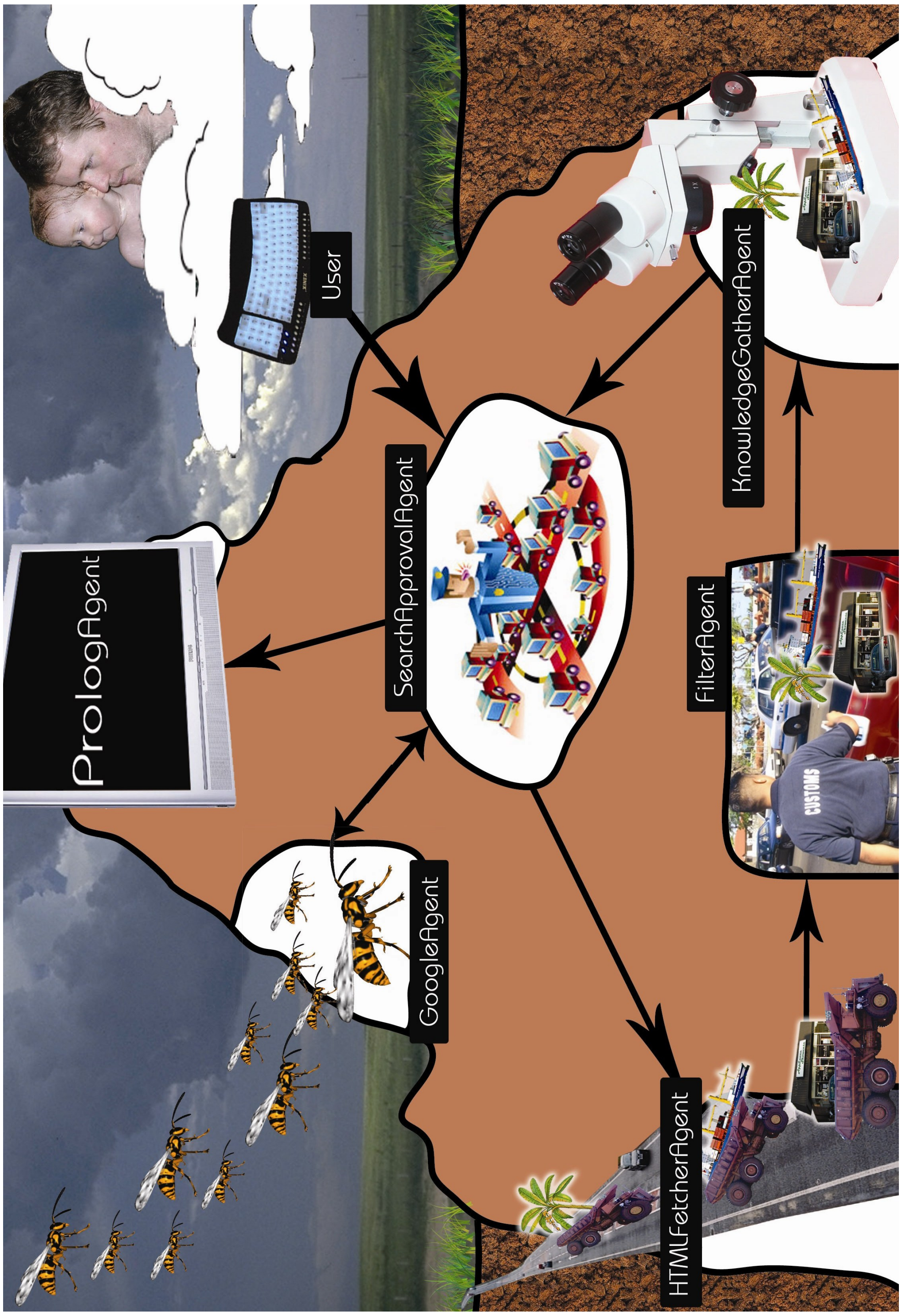


Figure 6.2: The hive and swarm

6.1.1 The Conveyor

The Metaphor for the Conveyor class can be that of a conveyor belt, but it may be more natural to compare it with the anthill. In other words Conveyor is the infrastructure of the hive of the swarm. When Conveyor is started, it will be given the location of the blueprint. Conveyor will then start to construct the anthill according to the description in the blueprint. It will create rooms for all the different types of agents and will set up path-ways for communication, between those rooms that are going to communicate. Each room will be made for a certain number of agents of that type. So Conveyor decides not only which agents should be invited into the anthill, but also how many there are room for and where their communications will go to and come from.

6.1.2 The Swarm

The agents in this system was meant to act a bit like a swarm, and so it is natural to think of them as a swarm. The swarm may consist of strange insects indeed, but that is not the point. The point is that they all focus on individual tasks and communicate together in order to solve a much bigger and more complex task. The agents will be presented in order, according to the work-flow when trying to solve a task. Let us imagine that the system gets a request to find out more about bananas. The first agent to receive this request is:

1. SearchApprovalAgent

SearchApprovalAgent can be compared to some sort of automated traffic-flow surveillance. Much of the data sent between agents will go through SearchApprovalAgent. The data sent contains special fields that SearchApprovalAgent will register for future reference. SearchApprovalAgent will however not only register data, it will also act upon it. It can be viewed also as a traffic-warden. It will not forward data that contains a request for a search that has already been requested. SearchApprovalAgent also has some more related functions, but they will be discussed later in the work-flow. For now banana is sent to GoogleAgent or similar.

2. GoogleAgent

GoogleAgent is the only implemented agent of its kind. But the system was designed for having multiple agent-types of this kind. GoogleAgent and its kind can be compared to fast and agile scouts, maybe they have wings and can quickly find points of interest based on the search-request received. These points of interest are sent back to SearchApprovalAgent. GoogleAgent has one more function later in the work-flow, but for now we return to SearchApprovalAgent.

3. SearchApprovalAgent

This time SearchApprovalAgent will register information about the URLs that are sent as points of interest by the scouts. If SearchApprovalAgent has registered poor results from any of the URLs, then those URLs are not forwarded, thus the system will concentrate on the URLs that yield interesting results. This is an effort to save resources which is a good idea when using brute force on the Internet. The approved URLs are sent on to HTMLFetcherAgent.

4. HTMLFetcherAgent

HTMLFetcherAgent can be compared to some sort of heavy-duty lifter. HTMLFetcherAgent is not as fast as the scouts, but he can carry a huge load. Let us imagine that the scouts have identified 3 places you can find bananas: A banana tree, a freight-ship and a grocery store. The heavy lifter will then go out and fetch the tree, the freight-ship and the grocery store. These will then be passed on to FilterAgent.

5. FilterAgent

FilterAgent can be compared with the toll-agent whose job it is to stop illegal goods from entering the system. FilterAgent will remove unwanted material before sending the modified data on to KnowledgeGatherAgent.

6. KnowledgeGatherAgent

KnowledgeGatherAgent is the meticulous scientist. He will sit down and analyze the banana carefully and try and figure out what connections he can discover between bananas and other objects. With a good set of key-sentences maybe he can find out that bananas grow on trees, that they are shipped on freighters and that they are sold in grocery stores. Whenever KnowledgeGatherAgent finds something out, it will send data back to SearchApprovalAgent.

7. SearchApprovalAgent

A hit is associated with the URL it was found at, thus making it less likely that that particular URL will not be pursued in the future. If a new object has been discovered in connection with the hit, that hasn't been investigated already, then there will be sent a new request to the scouts, to find out more about that object. SearchApprovalAgent will check if this knowledge have already been sent to verification. If the information has not already been verified, then it will be sent to the scouts for verification.

8. GoogleAgent

When a request comes in for verification of some knowledge found, the scouts will fly out to try and recognize how many bananas grow on trees, how many are shipped with freighter and how many are sold in a grocery store. The number from this count is sent back to SearchApprovalAgent.

9. SearchApprovalAgent

SearchApprovalAgent now receives the result from the verification and stores the result with the knowledge. This knowledge with the verification data is stored in a queue that SearchApprovalAgent shares with PrologAgent and possibly other agents that want to format the knowledge into some form. When one of the SearchApprovalAgents finds out that the system is ready to finish, then that agent will go through all the results in that queue and remove the knowledge that have few hits on the verification, relative to the other knowledge found. That way the knowledge that remains is the data that is most likely relevant and correct.

10. PrologAgent

PrologAgent and other agent of the same kind will only start up after all others have finished. PrologAgent can be compared with a computer screen, that takes the information and present it in that way instead of displaying the information in a printout or something similar. PrologAgent will format the knowledge in sentences in the knowledge-representation language called Prolog, and stores the information in a file that can be read by a Prolog interpreter. That way the information is presented in a way that can be used to reason over in another program.

6.2 Technically

The implementation of this system relies on the new Java 1.5 (Also known as Java 5) specifications. This section of the paper is easier to understand if you have some familiarity with java.

6.2.1 Descriptions of simple building-blocks

This section contains a description of some simple elements of the system that is not really very important in themselves but may be important for understanding some of the more important elements.

DataPackage

DataPackage is a simple class for storing information. DataPackage is meant to simply be overwritten by a new implementation of it if it does not fit the needs of the system. It contains some variables and some methods to read and write to those variables

AgentFormat

AgentFormat is also a pretty simple class. It is an abstract class which means that it is not possible to make an instance of this class. This class is only meant to be extended by all the agents in the system. AgentFormat implements “Runnable”, which means that all the agents that extend AgentFormat will have to include a method called “run”. AgentFormat will have some general methods that all agents should have, one of them is called “init” and will receive two Vector<Vector<DataPackage>>. These will be stored in two variables that each agent will have. One is called “input” and the other is called “output”. That means that all the agents will have an unspecified number of input and output variables, where one variable will be Vector<DataPackage>. That means that each variable contains an unspecified number of DataPackages. In the system presented in this paper, these variables will be treated as queues.

6.2.2 The Conveyor

The Conveyor-class is really a pretty simple class in terms of lines of code, but conceptually it is not so simple.

Variables

finished The only public variable is a boolean variable named “finished” which is also a static variable, which means that the value of this variable can be set and read by any thread in the system. finished is meant to be a control variable that should be checked by all agents, to see if they should terminate, or keep running.

agentSettings Conveyor will receive and store the location of the settings file. This file will tell Conveyor what agents to start, and how many of each type should be started and their priority. It will also tell Conveyor about communication between agents. But this will be discussed in more detail further down.

agents is a Vector<Thread>, which means that it can contain an unspecified number of Thread-instances. Each instance of an agent will be wrapped in its own Thread-instance. Everyone of these Thread-instances are stored in this variable. This makes it possible to initialize all variables before the agents are started.

data is a Vector<Vector<DataPackage>>. data will contain all the variables used by the agents to communicate. Since all variables used by the agents to communicate will possibly contain multiple elements, each of these variables will be a Vector<DataPackage>. That means that each variable can contain an unspecified number of DataPackages. (See description of DataPackage under “Description of simple building-blocks”) So each variable is a Vector<DataPackage>, and since there is an unspecified number of variables in this system, we will create a Vector containing all of these variables. This means that the final structure for keeping all the variables will be: Vector<Vector<DataPackage>>

index is a Vector<String>. index is what links the names of the variables with the variables themselves. Whenever a new variable is created, there will be created an entry in index as well, with the name of the variable. Thus the name of the variable in “index” will have the same index as the variable in “data”.

Methods

All the methods in Conveyor are public, but only one of them is static so that it can be used without making an instance of Conveyor and referring to that instance. And even that method needs Conveyor to be already instantiated.

Conveyor is the constructor-class which will make an instance of Conveyor. It requires a reference to where the settings-file can be located, but will not do anything except store that reference.

setAgentSettings will enable the system to refer to a new location for the settings-file.

getVectorWithName is the only static method in Conveyor. This means that this method can be called from any class within the system. getVectorWithName will take a String as input. It will use this String to try and find a match in “index”. If “index” contains a variable name that equals the name in the String, then the variable in “data” with the same index as the name in “index” will be returned. If there is no match between the name in the String and any of the variable names in “index”, then this method will create a new variable in “data” and put the name in the String in “index”. After the name in “index” and the new variable in “data” have been given places with same index, the new variable is returned. This means that this method will always return a variable. It will either return one it has an old reference to or a newly created one.

resetAllData will throw away all old information stored in the system. resetAllData will also read the settings-file and create new fresh variables and agents based on the information in the settings-file. This method will use initiate all the agents with the variables specified in the settings-file, and thus will decide what agents will share variables. One will have a variable as output while another agent will have the same variable as input. Each agent may have any number of input and output variables.

start sets the variable finished to false, and starts all the agents.

Summary

Conveyor makes up a framework for starting up agents and assigning variables. All the specifics on how it will all fit together is found in a settings-file that Conveyor will read. The location of this settings-file is sent in to Conveyor, and can be changed during run-time. This means that you can make many different configurations of your system in different settings-files and make sure that Conveyor will refer to the settings-file of your choice. In other words, this framework will make it easy to replace agents, reroute information, skip agents in the work flow, etc. Conveyor depends on two other classes. One is called “AgentFormat” and the other is called “DataPackage”. Both these classes are intended to be expendable, and to be overwritten if you want to change their function.

Comment The intention of explaining Conveyor in this level of detail is not to set in stone that this is how it has to be, nor to bore the reader with details. The reason for this thorough explanation is to illustrate one approach to this kind of framework for agents. Hopefully it will also help to know how this has been implemented if someone wants to use this concept in their own work.

6.2.3 The Swarm

Technically the swarm is not that interesting. You receive a request for information about a word that is sent into the system. This request is forwarded by SearchApprovalAgent to GoogleAgent. GoogleAgent finds URLs and sends them back to SearchApprovalAgent. SearchApprovalAgent forwards the approved URLs to HTMLFetcherAgent. HTMLFetcherAgent downloads the text from the URLs and sends the text to FilterAgent. FilterAgent reads what elements are unwanted from a language-specific portion of its settings-file. FilterAgent filters away elements that will distract, and will send the filtered text to KnowledgeGatherAgent. KnowledgeGatherAgent will read the language-specific portion of its settings-file and try to find matches for the key-sentences in the text. The key-sentences have reserved spaces for key-words. These key-words will be the words that the sentence will signal a relation between. If there is a match for a key-sentence, then Knowledge-GatherAgent will see if one of the key-words match the word the system is searching for. If there is a match, then some knowledge has been found and data concerning that knowledge is sent to SearchApprovalAgent along with sentences that can be used to verify the data found. SearchApprovalA-gent will extract different parts of the data received. SearchApprovalAgent will send queries for information about new words to GoogleAgent, and it will send verification-sentences for verification to GoogleAgent. GoogleAgent uses the verification-sentences to find how many hits there are using that specific sentence. Thus giving an indication on how probable the information is. The number of hits are sent back to SearchApprovalAgent, which will send store the information in the variable it shares with PrologAgent. When the system is about to finish, SearchApprovalAgent will go through all the verification data and remove the low-probability knowledge. When that is done, the global variable called “finished” in “Conveyor” is set to “true”. When PrologAgent reads that the system is finishing, PrologAgent will start converting the knowledge into Prolog-statements. These statements are written out to a file which can be read by a Prolog interpreter.

6.2.4 Swarm Intelligence

Swarm intelligence was used as a paradigm for modeling the system. In addition to this, there were two particular concepts from swarm intelligence as defined in “Swarm Intelligence: From Natural to Artificial Systems” [10] implemented in this system.

Positive Feedback

In the system in this paper there were a kind of positive feedback implemented. The usual form of positive feedback will mark certain sources or paths, so that they are preferred over other paths or sources. The best path or source get marked more often and so will receive more visits. This is implemented a bit differently in the system presented in this paper. In our system marks will be given to those URLs that yield knowledge, but this is not used to prioritize sources. The system uses a formula to determine which sources should be ignored. What we want to accomplish is to ignore sources that often appear but does not give any knowledge. Instead of focusing on the whole URL, the host name of the URL is extracted and used for the purpose of figuring out which URLs should be pursued. The algorithm used in this system is:
h = [The number of hits found using that host]
s = [The number of searches that have been made with that host]
H = [The total number of results returned]
S = [The total number of searches made]

$$\frac{5h}{s} + S - H$$

If the result of this equation is greater than -10, then the URL is approved.

There are some reasons why the equation was made this way, but it is probably not very hard to find better solutions. The first section of the equation is 5 times the number of hits with that host, divided by the number of searches made with that host. This gives an average amount of hits per search using this host. But we want to reward hits more than just one point, so we give 5 points for each hit. Then we add the total number of searches to that number, so that we have sort of a buffer building up to not exclude hosts before they have had a chance to prove themselves. Finally we subtract the total number of hits, which really serves to exclude low-yielders. This means that if the hit-ratio is low, every source will be approved, but if there is a high hit-ratio it will be a harder competition to be approved.

Division of Labor

Another feature typically associated with social insects is the concept of division of labor and specializing on certain tasks. Each of the agents in this system is specialized to perform its specific task. The advantages of multiple agents working in parallel have been discussed in the previous chapter.

Chapter 7

Results

I may not have gone where I intended to go, but I think I have ended up where I needed to be.
- *Douglas Adams*

7.1 Results

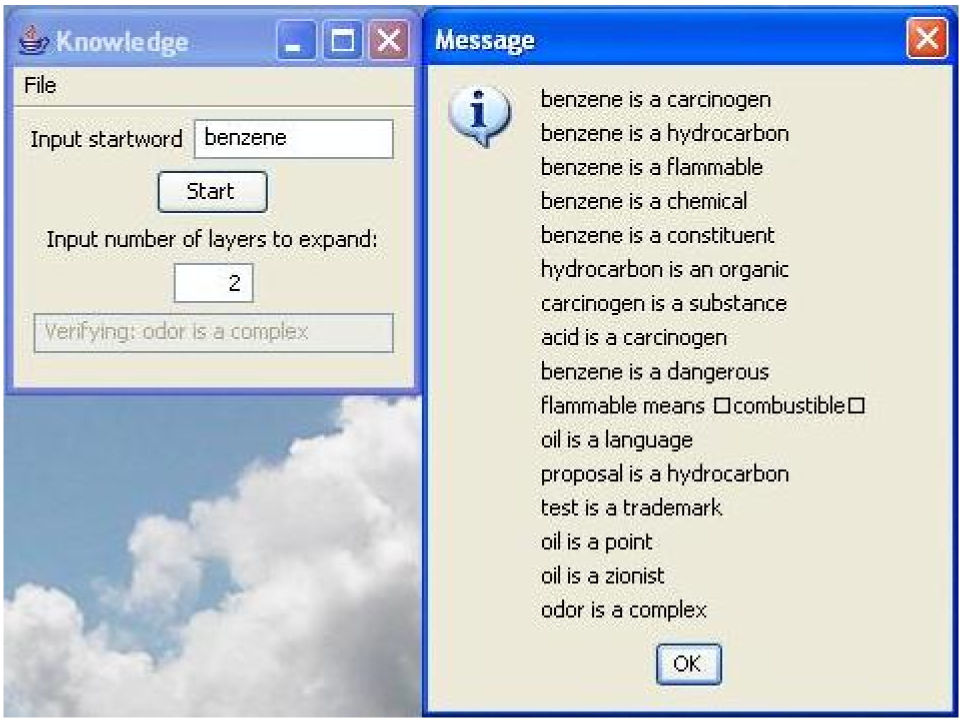
7.1.1 Framework

The success and results of this project can be as much measured in the framework created as in actual answers. Creation of a functioning framework, well equipped to solve the problem of extracting knowledge in this way, is in a way a major success of this whole project. The framework presented in this paper shows the way to one possible solution to implementing such a system as is described. The solution presented solves the set task and must therefor be considered a success.

7.1.2 Some answers

In this section there will be presented some screen-shots of answers the program has come up with. The user interface should be pretty self-explanatory. There is one input field, where you can input a word you want to know more about. There is a field where it says: "Input number of layers to expand:". In this field you enter the number of "generations" that you want the program to run. In other words, how many layers of new words should be investigated.

The minimum number of layers that make any sense is one. Then there is the start button. There is also a field where you are not allowed to write anything. It is only for delivering messages to the user. When the system has finished, it will show a message-box with its findings to the user. Some examples are included in the next few pages.



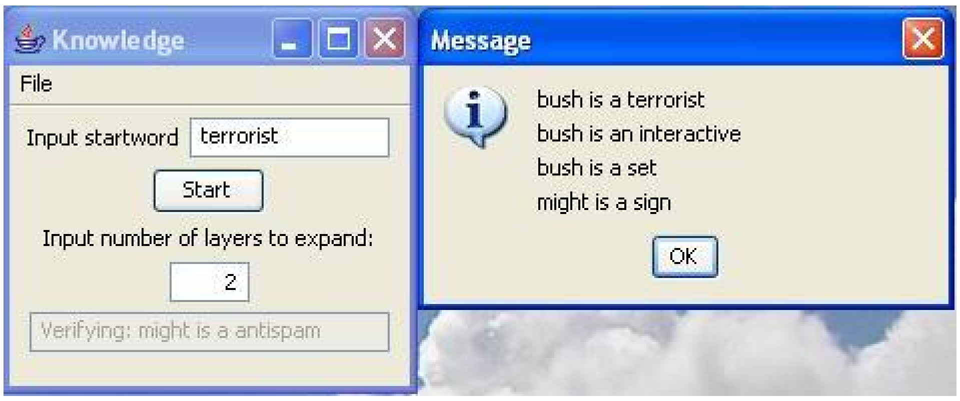
Input startword: benzene

Input number of layers to expand: 2

Verifying: odor is a complex

```
benzene is a carcinogen
benzene is a hydrocarbon
benzene is a flammable
benzene is a chemical
benzene is a constituent
hydrocarbon is an organic
carcinogen is a substance
acid is a carcinogen
benzene is a dangerous
flammable means □combustible□
oil is a language
proposal is a hydrocarbon
test is a trademark
oil is a point
oil is a zionist
odor is a complex
```

Figure 7.1: Benzene



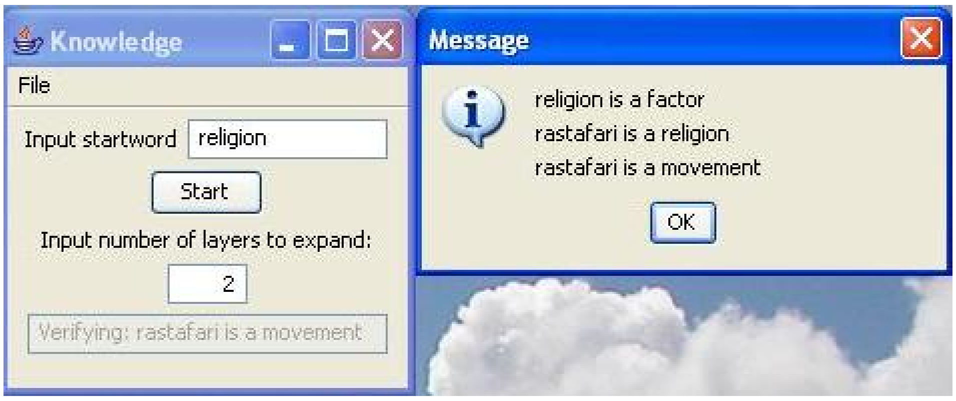
Input startword: terrorist

Input number of layers to expand: 2

Verifying: might is a antispam

```
bush is a terrorist
bush is an interactive
bush is a set
might is a sign
```

Figure 7.2: Terrorist



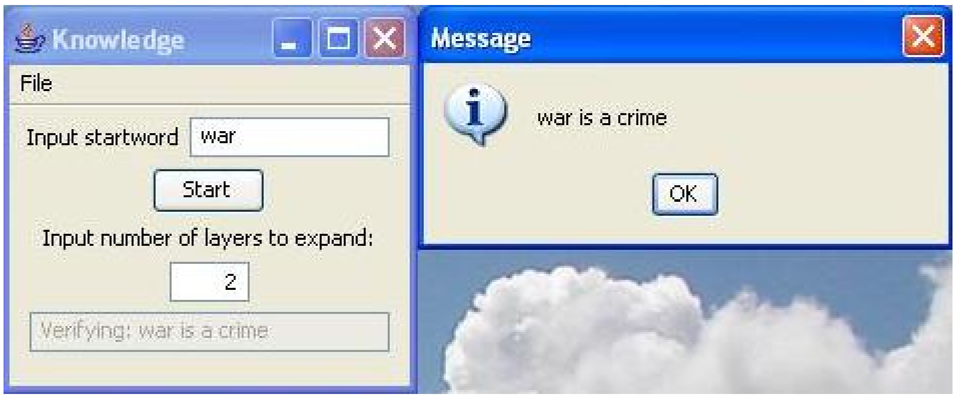
religion

2

Verifying: rastafari is a movement

religion is a factor
rastafari is a religion
rastafari is a movement

Figure 7.3: Religion



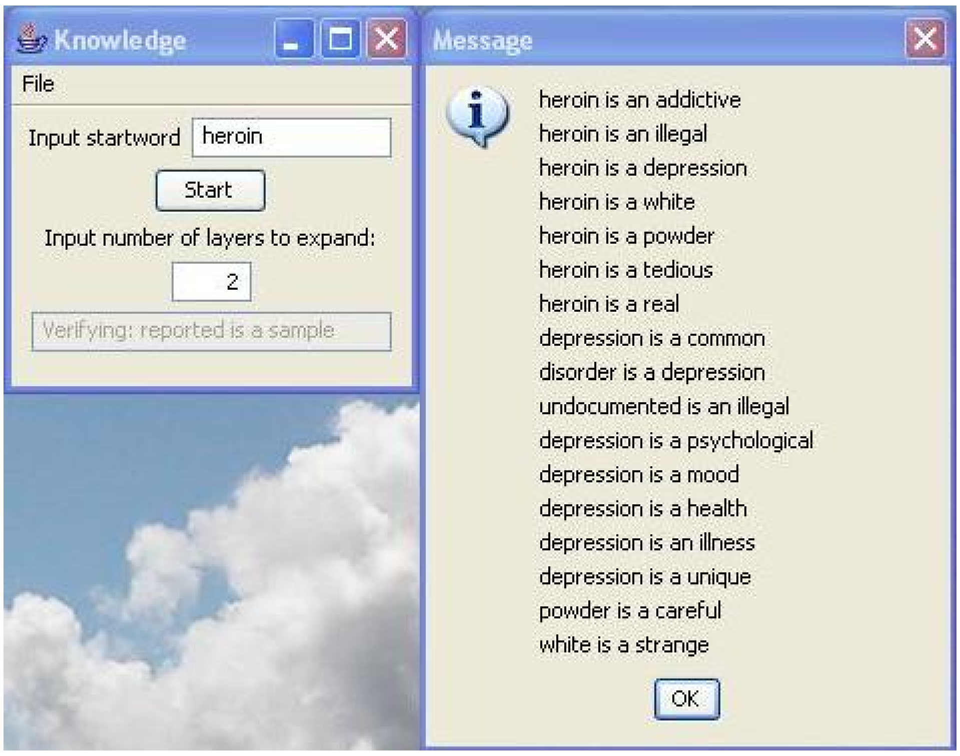
war

2

Verifying: war is a crime

war is a crime

Figure 7.4: War



Input startword heroin

Input number of layers to expand: 2

Verifying: reported is a sample

```
heroin is an addictive
heroin is an illegal
heroin is a depression
heroin is a white
heroin is a powder
heroin is a tedious
heroin is a real
depression is a common
disorder is a depression
undocumented is an illegal
depression is a psychological
depression is a mood
depression is a health
depression is an illness
depression is a unique
powder is a careful
white is a strange
```

Figure 7.5: heroin

7.2 Conclusion

It is obvious that there are a lot of unexpected information from these searches. Some answers are wrong, many are heavily biased, a lot of information is missing. On the other hand there is a lot of information that is found, and some of it seem surprisingly insightful. Information that would not be found in an encyclopedia, but which most people would agree on e.g. "war is a crime". Often a lot of information is checked but found too unlikely after verification. Sometimes things that are true are not included in the final result and sometimes things that are false are included. There are probably a lot of tweaks that could be made which would improve the results of this program.

Chapter 8

Evaluation & Discussion

What was, was. The past defines itself. Historians refuse to accept that definition and instead superimpose their analysis of the past through the eyes of the present, while the true past is lost behind the reflected image presented by historians who would have us see what they believe, rather than what was.
- *Neil Gaiman*

This project has been a journey. When this project first was conceived of, it was at best a vague idea. A desire to make a system that relied on simple logic and the vast amount of information on the Internet to extract knowledge. It was early on decided that the system would be designed using the swarm-concept.

8.1 What has been accomplished?

The goals that were set have been accomplished. The system is not a a highly polished product ready for the end-user, but the principles and theory show much promise. The system produces very interesting answers to some questions. It is apparent that this program gives much better answers to questions concerning technical terms. The answers to technical terms seem to be similar to answers one could find in an encyclopedia, in that they are non-biased and factual. In the author's opinion the answers to other questions are even more interesting though. Those answers are not so interesting for building a relevant knowledge-network maybe, but they are very interesting for sampling opinions. Something that was never considered before the answers themselves came in.

8.2 Problems

8.2.1 Problem-description

Frameworks

When implementing this system the point was not to reinvent the wheel, so instead of designing a framework for extracting the text from html-documents a completed framework was found and used instead. There have however been a lot of trouble with that framework. There have also been some problems with the framework from Google. On the other hand it would have taken a lot of time to design and implement a html-parser, but in hindsight a better framework might have been found.

Answers

Answers are often heavily biased, lacking information and contain false information.

8.2.2 Handled problems

Some of the worst problems with the external frameworks have been handled, though not in an optimal way.

HTMLParser

This external framework will sometimes hang for several minutes when trying to retrieve the text from certain URLs. This sometimes turned this part of the system into a terrible bottleneck, but after setting up several agents of this type to work in parallel, this problem was not so apparent.

Google

The framework from Google made many things a lot easier, but it seems that quite often it is very hard to get any queries through. And particular search-words are harder to get through than others. Some times it will work fine, but some times it is impossible to make a search at all. The solution used in our system was to take a small time-out and try again if any problems occurred. This will unfortunately not always solve the problem, because sometimes the system will not, in reasonable time, get the query processed.

8.2.3 What can be done?

Given time and resources, what could have been done to better solve the problems.

Frameworks

HTMLParser To improve the function of the HTMLParser one could either find a better framework, or create one.

Google is but one search-engine, and it would be a good idea not to rely on just one search-engine. Future improvements could be to implement several agents for interfacing with different search-engines.

Answers

- One way to improve the answers of a search would be to investigate more sources. This would however increase the processing time of a query.

- Include more unwanted words in the filter-file, thus increasing the quality of the data sent to KnowledgeGatherAgent.

- Adding more key-sentences to the settings-file of KnowledgeGather-Agent, for recognizing additional types of knowledge and improving recognition of types already included.

- Revising verification-sentences in the settings-file of KnowledgeGather-Agent.

All these measures, except the first, could benefit from application of formal linguistic theory. The settings-files have all been created without thorough consideration, from a linguistic perspective.

Chapter 9

Summary

9.1 Goal

- Test the viability of using simple techniques for knowledge-acquisition, with brute force on the Internet.
- Using techniques from both multi-agent system and swarm intelligence paradigms to structure the system, improving searches, increasing stability and increasing modularity.
- Building a framework for easy and modular implementation of multi-agent systems.

9.2 Method

To solve the goal, a combination of a modeling/abstraction approach and a design/experimental approach was used. First a theoretical background was built, together with a framework model. A functional system was then built to verify the theory and the framework.

9.3 Result

A functional framework was built, thus fulfilling one of the goals in itself. Additionally this framework verified that the theory could be realized in practice. Consequently it was possible to run the program and view the theory in light of the answers given. The answers were often interesting and surprising, and gave information beyond an encyclopedic scope. However the answers were often biased, lacking or false. Much of the faulty knowledge can be attributed to imperfections in the implementation of the system. It is hard to estimate to what extent the theoretical foundation is to blame for the faulty knowledge, when it is clear that adjustments to the implementation could improve results.

Therefor one must consider the state of the implementation when evaluating the answers.

Bibliography

[1] *Technology for a Realistic End-User Access Network Test-bed (TORRENT)*, http://cordis.europa.eu/ist/rn/torrent.htm.

[2] *Swarm intelligence*, http://en.wikipedia.org/wiki/Swarm_intelligence.

[3] *The Artificial Intelligence Group*, http://www.site.uottawa.ca/∼szpak/AI_Group.html.

[4] J. Abbate. The internet challenge: Conflict and compromise in computer networking. In *Changing large technical systems*. Westview Press, 1994.

[5] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi-agent systems with a fipa-compliant agent framework. In *Software: Practice and Experience*, volume 31, pages 103–128. John Wiley & Sons, Ltd., 2000.

[6] G. Beni. The concept of cellular robotic system. In *Proceedings 1988 IEEE Int. Symp. on Intelligent Control*, pages 57–62. IEEE Computer Society Press, 1988.

[7] G. Beni and S. Hackwood. Stationary waves in cyclic swarms. In *Proceedings 1992 IEEE Int. Symp. on Intelligent Control*, pages 234–242. IEEE Computer Society Press, 1992.

[8] G. Beni and J. Wang. Swarm intelligence. In *Proceedings Seventh Annual Meeting of the Robotics Society of Japan*, pages 425–428. RSJ Press, 1989.

[9] G. Beni and J. Wang. Theoretical problems for the realization of distributed robotic systems. In *Proceedings 1991 IEEE International Conference on Robotic and Automation*, pages 1914–1920. IEEE Computer Society Press, 1991.

[10] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, 1999.

[11] G. Bowker and S.L. Star. Knowledge and infrastructure in international information management: Problems of classification and coding. In *Information acumen: The understanding and use of knowledge in modern business.* Routledge, London, 1994.

[12] Gunnar Ellingsen and Eric Monteiro. Mechanisms for producing working knowledge: Enacting, orchestrating and organizing. *Information and Organization*, 13(3):203–229, 2003.

[13] S. Hackwood and G. Beni. Self-organizing sensors by deterministic anealing. In *Proceedings 1991 IEEE/RSJ International Conference on Intelligent Robot and Systems, IROS'91*, pages 1177–1183. IEEE Computer Society Press, 1991.

[14] S. Hackwood and G. Beni. Self-organization of sensors for swarm intelligence. In *Proceedings IEEE 1992 International Conference on Robotics and Automation*, pages 819–829. IEEE Computer Society Press, 1992.

[15] Eric Monteiro. Integrating health information systems: A critical appraisal. *Methods of Information in Medicine: a critical perspective*, 42:428–432, 2003.

[16] Nonaka and H. Takeuchi. A theory of the firms knowledge-creating dynamics. In *The dynamic firm. The role of technology, strategy, organization and regions.* Oxford University Press, 1998.

[17] C. Perrow. Complexity, coupling and catastrophe. In *Normal accidents*, chapter 3, pages 62–100. Princeton University Press, 1984.

[18] N. Sager, C. Friedman, and M. Lyman. *Medical Language Processing. Computer Management of Narrative Text.* Addison-Wesley, 1987.

[19] C. Soh, S.S. Kien, and J. Tay-Yap. Cultural fits and misfits: Is ERP a universal solution.

[20] G. Walsham. Knowledge management: The benefits and limitations of computer systems. *European Management Journal*, 19(6):599–608, 2001.

[21] S. Wermter and V. Weber. Artificial neural networks for automatic knowledge acquisition in multiple real-world language domains. In *Proceedings of the International Conference on Neural Networks and their Applications*, pages 289–296, Marseille, 1995.

[22] Xindong Wu. Building intelligent learning database systems. *The AI Magazine*, 21(3):61–67, 2000.

[23] C. T. Yu and W. Sun. Automatic knowledge acquisition and maintenance for semantic query optimization. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):362–375, September 1989.