A Multi-agent System For Web Information Acquisition And Dissemination

By

Wanjun Sun, M. Eng.

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements of the degree of

Master of Science

Information and Systems Science

Department of Systems and Computer Engineering

Carleton University

Ottawa, Canada K1S 5B6

April 2004

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-93989-8

Our file Notre référence

ISBN: 0-612-93989-8

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

This thesis identifies the information overload problem as a result of the increasingly vast amounts of heterogeneous content available on the Web. As an attempt to meet the challenge, a multi-agent system is proposed to improve the efficiency and effectiveness of relevant knowledge acquisition and dissemination from various Web information sources. We analyze and design the agent system using pattern-driven approach. The system consists of a group of agents that work collaboratively to achieve goals that are beyond the scope of an individual agent. The multi-agent system locates and retrieves relevant information from various Web data sources according to users' profiles, performs opinion-based filtering on the information to improve its quality, integrates and presents the information to users.

A prototype of the multi-agent system is implemented to demonstrate the capabilities of the proposed system. AgentMom is adopted as the framework to develop the physical agent system.

Acknowledgement

I would like to take this opportunity to thank all of those who have helped me during my thesis work in many different ways, although some of their names are not mentioned here, due to limited space.

First and foremost, my grateful thanks go to Dr. Michael Weiss and Dr. Samuel A. Ajila, the supervisors of my thesis. Their wisdom, guidance, advice and support, which could not have been more prompt whenever I needed it, have helped me throughout my master's program. I also would like to thank the examiners who took their valuable time to review the thesis and attend my presentation: Dr. Chung-Horng Lung, Dr. Tony White, and Dr. Babak Esfandiari.

Thanks also go to the Department of Systems and Computer Engineering, Carleton University. Its well-designed Information and Systems Science program, as well as its strong support, have allowed me to significantly extend my knowledge and to prepare myself for the thesis work and my future career. I would like to thank Ms. Blazenska Power for her always warm and patient help.

My special thanks go to many of my friends who have supported and encouraged me during my graduate studies. I would especially like to thank Chen Wei, Yongwei Zhu, Haihui Chen, Jun Zhou, Chunsheng Liu, Wei Shi, Xiaofei Jia, Jinguo Liu, Xia Hou, Tony Han, Guangping Xie, Zhen Yuan, Yunbo Fang, Zhanhai Li, Peng Cai, and Yu Wang.

Above all, exceptional and affectionate thanks go to my family: my wife Hong Hao, my parents Jianying Chi and Yonghai Sun, my parents-in-law Bingru Ding and Juyou Hao, my brothers Wanzhong Sun and Wanli Sun, and my sister-in-law Bin Hao. It was the love, the understanding, the appreciation, the encouragement, and the concrete help from my FAMILY, which enabled me to accomplish this work. I feel indeed lucky that I have such a wonderful family.

Table of Contents

Abstract.....	iii
Acknowledgement	iv
CHAPTER 1 INTRODUCTION	1
1.1 Background.....	1
1.2 Related Work.....	2
1.3 Motivation and Objectives	6
1.4 Thesis Contributions	7
1.5 Thesis Organization	9
CHAPTER 2 DESIGN OF THE MULTI-AGENT SYSTEM	11
2.1 Agent Paradigm.....	11
2.1.1 What Is an Agent.....	11
2.1.2 Agent Classification	13
2.1.3 Collaborative Agents.....	14
2.1.4 Interface Agents	15
2.1.5 Information Agents	16
2.1.6 Multi-agent Systems	17
2.2 Multi-agent System Design.....	19
2.2.1 Agent System Design Using an Agent Pattern Approach	20
2.2.1.1 Base Pattern Language.....	20
2.2.1.2 Role Models	21
2.2.1.3 Applying the Patterns	24
2.2.2 System Architecture	25
2.2.3 Mediator Agent	27
2.2.4 User Agent	31
2.2.5 Extractor Agent	31
2.2.6 Notifier Agent	32
2.3 Information Filtering.....	32
2.3.1 Recommender System and Information filtering	32
2.3.2 Opinion-Based Information Filtering.....	33
2.3.3 Social Trust Model for Recommender Agents	34
2.3.3.1 User Profile Representation	34
2.3.3.2 Initial Profile Generation	36
2.3.3.3 Profile Exploitation for Recommendation	37
2.3.3.4 Relevance Feedback.....	39
2.3.3.5 Profile Adaptation	40
2.3.3.6 Discussions	42
CHAPTER 3 EXTRACTION OF WEB INFORMATION.....	44
3.1 Background.....	44

3.2 Technologies	45
3.3 Java Support For Regular Expressions	48
3.3.1 Regular Expressions.....	48
3.3.2 Java regex package	48
3.4 RSS	49
3.4.1 What Is RSS?	49
3.4.2 RSS Syntax	50
3.4.3 How Is RSS Used?	51
3.4.4 KlipFolio.....	52
3.4.5 Discussion.....	53
3.5 Detailed Extracting Procedure By an Example.....	54
3.6 Extraction From RSS Feeds.....	58
3.7 Retrieving Information that Matches User's Profile	60
3.8 Discussions	62
CHAPTER 4 THE MULTI-AGENT SYSTEM IMPLEMENTATION	64
4.1 Agent Development Environment.....	64
4.2 Operation Semantics	64
4.3 AgentMom Package.....	66
4.3.1 Agent Class	67
4.3.2 Conversation Class.....	67
4.3.3 MessageHandler Class	67
4.3.4 Message Class.....	68
4.3.5 Sorry Class.....	68
4.4 Conversation Diagrams.....	68
4.5 Conversation Scenarios.....	73
4.5.1 UserAgent Goes Online	73
4.5.2 UserAgent Goes Offline.....	74
4.5.3 RefreshData.....	74
4.5.4 GetLatestItemsOfOneDataSource.....	77
4.5.5 AskOpinion	78
4.5.6 GetOneDataSrcItemsRSS	79
4.5.7 Notify	81
4.5.8 CheckRecommendation	82
4.6 Graphic User Interface	85
4.6.1 Mediator Agent	85
4.6.2 User Agent	88
4.6.3 Extractor Agent.....	91
4.6.4 Notifier Agent	91
CHAPTER 5 NOTIFICATION OF UPDATES	93
5.1 Notification Process.....	93

5.2 Web Interface	95
CHAPTER 6 CONCLUSIONS AND OUTLOOK.....	99
6.1 Conclusions.....	99
6.2 Future Work	100
REFERENCES	102
APPENDIX A DATABASE SCHEMA	110
A.1 TABLE buddies.....	110
A.2 TABLE datasource	110
A.3 TABLE itemsource.....	111
A.4 TABLE recommendations.....	111
A.5 TABLE testratings.....	112
A.6 TABLE users	112
A.7 TABLE usersubscription.....	113
APPENDIX B PATTERN DEFINITION FOR EXTRACTOR AGENT AND NOTIFIER AGENT .	114
B.1 Extractor Agent	114
B.2 Notifier Agent.....	115

List of Figures

Figure 2.1 Structure for the Base Pattern Language	21
Figure 2.2 Role Diagram.....	22
Figure 2.3 Top-level Roles and Their Subtypes in This Pattern Language	22
Figure 2.4 Structure for the Domain-specific Pattern Language.....	24
Figure 2.5 Applying the Patterns	25
Figure 2.6 The Multi-agent System	26
Figure 2.7 Service Brokering and Matchmaking	29
Figure 3.1 An Extraction Process.....	46
Figure 3.2 Extraction Process Using J2SE Regular Expression Package	49
Figure 3.3 RssReader.....	51
Figure 3.4 KlipFolio	53
Figure 3.5 Web Page To Be Examined	55
Figure 3.6 HTML Source Code Extracted From the Web Page (Beautified by JTIty).....	56
Figure 3.7 Partial List of the Available RSS Feeds from Medicine Net	59
Figure 3.8 Steps to Extract Data from an RSS Feed	60
Figure 4.1 How AgentMom Works.....	65
Figure 4.2 AgentMom Package.....	66
Figure 4.3 Conversations Between UserAgent and ExtractorAgent.....	69
Figure 4.4 Other Conversations	70
Figure 4.5 State Diagram for Initiator.....	72
Figure 4.6 State Diagram for Responder.....	72
Figure 4.7 Activity Diagram for UserAgent's Going Online	73
Figure 4.8 Activity Diagram for UserAgent's Going Offline	74
Figure 4.9 Sequence Diagram for RefreshData	75
Figure 4.10 Sequence Diagram for getLatestItemsOfOneDataSource	78
Figure 4.11 Sequence Diagram for askOpinion	79
Figure 4.12 Sequence Diagram for getOneDataSrcItemsRSS	80
Figure 4.13 Sequence Diagram for notify.....	81
Figure 4.14 Activity Diagram for CheckRecommendation	82
Figure 4.15 Sequence Diagram for getOneItemRSS	84
Figure 4.16 The Mediator Agent: Agents Tab - All Agents Online.....	85
Figure 4.17 The Mediator Agent: Agents Tab - Agent John Offline	87
Figure 4.18 The Mediator Agent: Messages Tab	88
Figure 4.19 The User Agent: Subscriptions Tab	89
Figure 4.20 The User Agent: Friends Tab	89
Figure 4.21 The Extractor Agent: Messages Tab	91
Figure 4.22 The Notifier Agent: Messages Tab	92
Figure 5.1 Process of Updating User's Klip	94

Figure 5.2 User Clicking On a Delivered Item	96
--	----

List of Tables

Table 1.1 Differences between Systems.....	6
Table 2.1 Services in the System and their Requesters and Providers.....	30
Table 2.2 An Exemplar User Profile	36
Table 3.1 Predefined Character Classes	57
Table 3.2 Quantifiers.....	57
Table 4.1 Conversations and Their Goals	71
Table 4.2 Trust Weights For Three Categories.....	90

CHAPTER 1 INTRODUCTION

1.1 Background

The World Wide Web has become the dominating source for information distribution and acquisition. It is easy for information providers to publish information on the Web and for users to access information from the Web. However, the ever growing and changing nature of the Web makes it a challenge for users to find interesting information. A typical user faces repetitive tasks such as searching, browsing and filtering to acquire relevant information from the Web. Since the Web is designed for human browsing and information delivered is mainly in semi-structured documents [Baue00], it is difficult to retrieve and integrate relevant information from multiple sources [Musl01].

Data service providers deliver information to the users by offering special purpose or customized systems such as digital libraries, online communities, newsletters, search engines, etc. Different systems focus on specific information, and data is delivered in different structures, presented in various interfaces. Users usually need to learn to interact with these systems to get relevant information from multiple data sources for wider range of needs.

Due to these challenges, professionals in knowledge-intensive organizations like the healthcare sector usually spend 20% of their work time searching and acquiring information [Sere03b]. As the Internet grows exponentially and increasingly vast

amounts of heterogeneous content becomes available on the Web, the information acquisition and dissemination problem is likely to worsen at a significant rate.

Agent technology provides a promising solution to the information acquisition and dissemination problem. Cooperative, task-specific information agents will play an important role in actively searching, retrieving and filtering relevant information on behalf of their users. They will cope with the difficulties associated with the information overload of the users in a timely fashion, improving the efficiency and effectiveness of relevant information acquisition and dissemination [Delg98] [Mouk98] [Pazz02] [Pant02].

The quality of the service provided by such an information agent system will be improved by integrating opinion-based information filtering capability, which brings the benefit for users since they are only recommended with peer-reviewed information that best matches their tastes or preferences. Thus the multi-agent system becomes a powerful tool that searches, filters and recommends relevant information for users.

1.2 Related Work

Several efforts have addressed the use of agent systems for information acquisition and dissemination in recent years. Knoblock et al [Knob98] introduced an information agent system called Ariadne that is capable of extracting and integrating data from multiple semi-structured Web sources. Based on work from knowledge representation,

machine learning and automated planning, Ariadne developed methods to map a set of Web sources into a uniform representation, which can then be queried as if it was a single database. The query issued on the database-like model is decomposed into a series of simpler queries that are applied against the Web source set. The results are integrated to form a response to the original query. The system was designed to get results for database-like queries that require information from pre-specified sets of Web sites.

Syskill & Webert [Pazz98] is a software agent that develops a user's profile from the user's rating for visited Web links in an index page (i.e. a starting page containing a number of links) and its analysis of information delivered by these pages. The agent then uses the user's profile to predict the user's rating for the pages that are not explored by the user in the index page, thus deciding what pages might interest the user. The agent can also retrieve other pages that might be relevant to the user from the Web and rate these pages.

Amalthea [Mouk98] is an artificial ecosystem consisting of evolving agents. Two kinds of agents live in the system: the information agents that keep track of the user's interest and the information discovery agents that handles information sources, find and retrieve information of the user's interest. Each agent specializes in a narrow sector of a domain, and the agents in the system cooperate and compete with each other. The agents discover the Web and present the user with results that might interest the user. The user browses the delivered results and gives feedback on an item by rating it. The system

learns the user's preferences using machine learning techniques and adapts to the user's interest which may change over time. Based on the user's ratings, a positive or negative credit is assigned to the related filtering and discovery agents. Agents with good performance survive and reproduce while low-performance ones are eliminated from the system. In Amalthaea, each user has her own populations of the information agents and information discovery agents. Multiple users use independent instances of the system and no interaction between different users takes place in the system.

The WARREN financial portfolio management system [Syca96] [Syca97] [Syca98] is a multi-agent system that integrates information finding and filtering from the Internet in the context of supporting a user manages her financial portfolio. The system consists of agents that cooperatively organize themselves to monitor and track stock quotes, financial news, financial analyst reports, and company earnings reports to inform the portfolio owner of the evolving financial picture. The agents not only answer relevant queries but also continuously monitor Internet information sources for the occurrence of interesting events and alert the portfolio manager agent or the user. WARREN also includes agents that analyze user buy and sell decisions with respect to asset allocations and risk.

The Anticipator [Seva02] is an agent-based system for information dissemination in dynamic environments. The Anticipator employs two types of agents, a set of Profile Agents (PAs) and a set of Event Monitoring Agents (EMAs) to adapt both the content and

frequency of the information sent to the users. A data-driven EMA watches for an event that a user is interested in, and it reports the occurrence of this event to a particular Profile Agent. A Profile Agent maintains a user profile consisting of the user's changing information needs, and it adapts the user profile based on the dynamic changes. A time-driven EMA sends information request to the data sources and the requested result is delivered to the user.

Koubarakis et al [Koub03] proposed P2P-DIET, a peer-to-peer system that enables P2P data sharing in scenario of ad-hoc query and that of selective information dissemination (SDI). SDI supports continuous query to the system to receive notifications whenever interesting information becomes available. SDI can be useful in advanced application such as alert system for digital libraries and e-commerce networks.

Table 1.1 summarizes the differences between these systems and our proposed system. Part of the dimensions used for comparison is taken from [Mont03].

System	Profile Learning	Feedback		Profile Adaptation	Filtering		Information Sharing
		Explicit	Implicit		Content-based	Opinion-based	
Ariadne	No	No	No	No	Yes	No	No
Syskill & Webert	Yes	Yes	No	Yes	Yes	No	No
Amalthaea	Yes	Yes	No	Yes	Yes	No	No
WARREN	Yes	Yes	No	Yes	Yes	No	No
Anticipator	Yes	Yes	No	Yes	Yes	No	No
P2P-DIET	No	No	No	No	Yes	No	Yes
Our System	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 1.1 Differences between Systems

1.3 Motivation and Objectives

Integrating the information agent technology and recommendation capability, it is possible to create a system to effectively deal with the information acquisition and dissemination problem described in section 1.1. The system will comprise a set of components that address the problems of information search, extraction, filtering, delivery and presentation in a unified interface from a wide variety of Web data sources.

The main objectives of this thesis are to investigate the various aspects of multi-agent systems that consist of multiple task-specific information agents capable of assisting the user to acquire relevant information from Web sources; and, to propose an

architecture and implement a prototype of such a system with an aim to improve the effectiveness and efficiency of information acquisition and dissemination from existing Web information sources. The aspects of the multi-agent system that need to be addressed include: system architecture, system components, agent patterns, task allocation between the agents, communication mechanisms between the components and various implementation issues. The system should be able to acquire information from a wide variety of data sources.

1.4 Thesis Contributions

This thesis identifies the information overload problem as a result of the increasingly vast amounts of heterogeneous content available on the Web. As an attempt to meet the challenge, a multi-agent system is proposed to improve the efficiency and effectiveness of relevant knowledge acquisition and dissemination from various Web information sources.

1. We analyze and design the agent system using pattern-driven approach, and two new agent patterns (extractor agent and notifier agent) are defined to complement the existing pattern language.

- (a) The system consists of user agents, the mediator agent, the extractor agent, and the notifier agent. (i) The mediator agent facilitates between agents in the agent society. (ii) The extractor agent locates and retrieves relevant information for user

agents. (iii) The notifier agent formats and delivers the results to the user's desktop.

(iv) User agents develop users' profiles and represent users to perform various tasks.

User agents instruct the extractor agent to retrieve interesting Web information according to users' profiles, request the notifier agent to deliver interesting items to users' desktops, offer opinions to other user agents and filter recommendations from other users.

(b) The system integrates content-based and opinion-based filtering capability to improve the quality and relevance of delivered information items.

2. A prototype of the multi-agent system is implemented to demonstrate the capabilities of the proposed system.

(a) AgentMom is adopted as the framework to develop the physical agent system and Multi-agent Systems Engineering (MaSE) philosophy is applied in the system development.

(b) RSS, a well-established XML format is used in the system to enable communication between agents and serve as a means to summarize information items.

(c) KlipFolio is adopted as an efficient presentation tool to integrate information from heterogeneous data sources and present it to the user in a unified fashion.

1.5 Thesis Organization

The remainder of the thesis is structured as follows:

Chapter 2 presents the design of the multi-agent system. First the underlying agent paradigm is introduced, then the design process guided by a pattern-driven approach. The system architecture, the components of the system, their functionality and how they fit in the system are discussed subsequently. Information filtering methods adopted in the system is then described in detail.

Chapter 3 describes the problem of retrieving and extracting interesting information from various Web sources. First, background and a review of existing approaches is presented. Then we describe the technology we used and illustrate the detailed procedure by some examples. RSS, an efficient and effective format for information dissemination, is adopted in the project as a means to enable unified presentation of information and message exchanges between agents. Therefore, a short introduction on RSS format is presented to clarify its usage in the project. The content-based filtering approach used to match retrieved items with user profile in the extraction process is also discussed.

Chapter 4 focuses on implementation aspects of the multi-agent system. First AgentMom, the multi-agent system development environment, is introduced. The operation semantics that describes how agents establish communication to perform tasks is illustrated, followed by the presentation of the agent package that makes up the AgentMom framework. In the subsequent sections, we describe conversations designed

between the agents to accomplish various tasks.

Chapter 5 discusses the notification step. The project is about finding relevant information for users, and notification is the last step to deliver the information to users. In this chapter, the procedure of how interesting items are delivered to the users' desktops is introduced and then, the mechanism of enabling users to recommend items to other users is described.

Chapter 6 concludes the thesis with the contribution of the thesis and suggestions for future work.

CHAPTER 2 DESIGN OF THE MULTI-AGENT SYSTEM

This chapter presents the design of the multi-agent system. We begin, in the following section, by introducing the underlying agent paradigm for the multi-agent system. Section 2.2 discusses the detailed design of the agent system, followed by a detailed description of information filtering capability integrated in the multi-agent system to improve the quality and relevance of items delivered to users.

2.1 Agent Paradigm

Agent technology appears to be a promising approach for developing many complex applications [Jenn01]. Application domains where agent solutions are being applied to or investigated include network management, air-traffic control, business process management, data mining, information retrieval and management, electronic commerce, education, personal assistants, digital libraries, smart databases, etc [Nwan96]. Agent technology gives us the tools with which to build applications that we were previously unable to build. It also provides a better means of conceptualizing and implementing a given application. In this section, a brief introduction of the agent paradigm is provided.

2.1.1 What Is an Agent

There is no single universally accepted definition for the term “agent” in the research community. Defining the term “agent” is challenging and diverse versions of definitions are in use in the literature [Weis98] [Karm98] [Gord00] [Wool97b]. Here we

give the definition by Shoham [Shoh97]:

An agent is “a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes”.

The behaviour of agents closely reflects that of the users whose work they are delegated to perform or support. Agents may perform their tasks collaboratively or individually, roam in the network or keep stationary, learn and adapt or work only according to pre-defined rules [Weis98] [Nwan96]. But despite the diversity, agents have four widely recognized characteristics [Wool97a]:

(1) Autonomy: agents act on behalf of their users, have a degree of control over their own actions and states, do not require user’s intervention on every step to perform the assigned tasks;

(2) Reactivity: agents sense the environment (which could be a user interface, other agents, the Web or many of these combined) and interact with the changes occurred in the environment in a timely manner;

(3) Pro-activeness: agents do not simply act in response to their environment, they also exhibit goal-directed behaviour to achieve their imposed goals;

(4) Social ability: agents interact with other agents as required in order to perform the more complex tasks than they themselves can handle.

There are similarities between object-oriented approaches and agent-based ones [Jenn01] [Wool97b]:

(1) They both encapsulate some internal state and behavior that is not directly accessible to the outside world;

(2) They both recognize the importance of interactions – they communicate via message passing.

There are also important differences between them [Jenn01] [Wool97b]:

(1) Objects are generally passive in nature, are activated only by messages passed to them while agents are autonomous and goal-directed;

(2) Although objects encapsulate some of its states and behaviors, they do not restrict behavior activation: any object can invoke the methods declared as public. The corresponding actions are performed once the methods are invoked. However, if an agent is requested to perform an action, it may or may not do it. It decides by itself on whether and how to react to a stimulus.

(3) Objects represent too fine a granularity of behavior and method invocation is too primitive for describing the interactions that take place in complex systems.

The motivations of adopting agent technology are: to simplify the complexity of distributed computing and to overcome user interface problems [Weis98].

2.1.2 Agent Classification

As is the case with the definition of the term “agent”, there are many typologies of agents due to different classifying dimensions adopted. Nwana [Nwan96] places existing agents into the following categories:

(1) Collaborative agents: Collaborative agents emphasize autonomy and cooperation in order to perform tasks for their owners.

(2) Interface agents: Interface agents emphasize autonomy and learning in order to perform tasks for their owners.

(3) Mobile agents: Mobile agents are capable of roaming in the network, interacting with foreign hosts, gathering information on behalf of their owners and coming “back home” having performed the duties set by their users.

(4) Information agents: Information agents perform the role of managing, manipulating or collating information from many distributed sources.

(5) Reactive agents: Reactive agents do not possess internal, symbolic models of their environments; instead they act in a stimulus-response manner to the present state of the environment in which they are embedded.

(6) Hybrid agents: Hybrid agents refer to those whose constitution is a combination of two or more agent philosophies within a singular agent.

Our multi-agent system consists of agents exhibiting characteristics of three different agent types: collaborative agents, interface agents and information agents. We next discuss in more detail about these three agent types.

2.1.3 Collaborative Agents

These agents work in a collaborative fashion, thus enabling the ensemble to perform tasks beyond the capabilities of any participating agent. They tend to be static and do not

perform any complex learning [Nwan96]. Our system is a system of collaborative agents. Each agent has its expertise in a particular domain and they need to collaborate with one another to achieve goals that are beyond a single agent's capability. For example, to get the update of a digital journal and present it to its user, a user agent need to work with the extractor agent and the notifier agent to achieve the goal. First the user agent asks the extractor agent to retrieve the relevant data from the Web and then it asks the notifier agent to deliver the update to the user's desktop. The user agent itself provides retrieving criteria to the extractor agent and filters the resulting items before it request the notifier agent to deliver the update. The collaborative agents paradigm enhances modularity, reliability, flexibility and reusability.

2.1.4 Interface Agents

An interface agent is an agent that actively assists a user in operating an interactive interface [Lieb97]. Interface agents emphasize autonomy and learning in order to perform tasks for their owners. The key metaphor underlying interface agents is that of a personal assistant who is collaborating with the user in the same work environment [Nwan96]. The interface agent learns from its user by monitoring the actions of the user over long period of time, and finding recurrent patterns in order to automate them [Maes94] [Lash94]. It is possible that the agent may need to interact with the interface while the user is also interacting with the interface. The user may or may not be aware of the agent's activities at any given moment.

The user agents in our system exhibit some characteristics of interface agents. They may not assist the user in interacting with the user interface, but they learn from the users' activities and dynamically develop and maintain users' profiles. More details about user agents and user profiles can be found in section 2.2.4 and 2.3.3.1.

2.1.5 Information Agents

Research on information agents was motivated by information explosion on the web. The Web can be viewed as a large, distributed information resource, with nodes on the network designed and implemented by different organizations and individuals. The growth of information delivered by the Web is astonishing. With the vast amounts of information on the Web however, people find themselves information rich but knowledge poor. There is an increasing demand for tools to manage such an information explosion.

Information agents have become useful tools that help us deal with the information overload problem. Information agents perform the role of managing, manipulating or collating information from a wide variety of distributed sources, especially the Web [Nwan96] [Weis03a]. Currently, agents on the Web mostly perform information retrieval and filtering [Syca98].

Information agents can be mobile or static. Current information agents are mostly static, but it was predicted [Nwan96] that the majority of future information agents will be able to navigate the Web, retrieve relevant information and store it in a home database, which will be queried using SQL by others for useful information. When retrieving Web

data, wrappers are needed to transform information from specific sources to a common format [Weis03a]. In the multi-agent system of our project, all the agents except for the mediator agent can be viewed as information agents to some extent. They work collaboratively to assist users to obtain relevant information. They extract information that may be of interest to the users from a wide variety of Web sources. They offer opinions to customize the information to make it more relevant and present it to the user in a unified interface.

2.1.6 Multi-agent Systems

The capacity of an agent is limited by its knowledge, its computing resources, and its perspective. Therefore more complex, realistic, and large-scale problems are beyond the capabilities of an individual agent. The only way this kind of problems can be reasonably addressed is to develop a number of agents, each of which is specialized at solving a particular aspect of the problem. This decomposition allows each agent to use the most appropriate capabilities for solving its particular problem. The agents in the system must coordinate with each other to achieve the goal. A multi-agent system can be defined as a loosely coupled network of agents that interact to solve problems that are beyond the individual capabilities or knowledge of each agent [Syca98]. These agents can be heterogeneous in nature. The characteristics of multi-agent systems are as follows [Syca98]:

- (1) Each agent has incomplete information or capabilities for solving the problem

and, thus, has a limited viewpoint;

(2) There is no system-level global control;

(3) Data is decentralized;

(4) Computation is asynchronous.

Multi-agent systems are ideally suited to representing problems that have multiple problem solving methods, multiple perspectives and/or multiple problem solving entities. Such systems have the traditional advantages of distributed and concurrent problem solving, but have the additional advantage of sophisticated patterns of interactions [Jenn98]. There are three important domain characteristics that are often cited as a rationale for adopting multi-agent technology [Syca98]:

(1) Data, control, expertise, or resources are inherently distributed;

(2) The system is naturally regarded as a society of autonomous cooperating components;

(3) The system contains legacy components, which must be made to interact with other, possibly new software components.

The first two categories of domain characteristics for adopting multi-agent technology fit our system perfectly. The problem domain requires a number of distinct problem solving entities and involves data sources that are physically or logically distributed, and the agents need to interact with one another in order to solve problems. Multi-agent system can often provide an effective solution in this setting.

2.2 Multi-agent System Design

As agent-oriented approach is rapidly becoming a new paradigm for modelling and developing complex distributed systems, many methodologies toward building multi-agent system have been proposed such as Gaia [Wool00], MASSIVE [Lind00] and MaSE [DeLo01]. Among them, Gaia [Wool00] [Zamb01] is a widely cited methodology specifically tailored to the analysis and design of multi-agent systems. It views a multi-agent system as a computational organization composed of a number of interacting agents. Each agent plays one or more specific roles in an organized society. The Gaia model identifies the roles that agents have to play within the multi-agent system and the interaction protocols in which the different roles are involved. MaSE is viewed as a further abstraction of the object-oriented paradigm where agents are a specialization of objects. Instead of simple objects with methods that can be invoked by other objects agents coordinate with one another via conversations and act proactively to accomplish individual and system-wide goals. Existing object-oriented techniques are applied to the specification and design of multi-agent systems.

Most of current agent system design methodologies are generally goal-driven [Weis03c]: an agent system is designed by iteratively decomposing system goals until they can be assigned to specific agents. One of the problems this led to is: developers concentrate on seeking solutions to common design problems without benefiting from how they were solved in the past. A pattern-driven design methodology however, appears

to be a more effective and less ad hoc approach for agent system design.

2.2.1 Agent System Design Using an Agent Pattern Approach

Patterns are reusable solutions to recurring design problems that occur during software development, and provide a vocabulary for communicating these solutions to others. Each agent pattern documents a proven assignment of roles to agents, and their interaction. It also documents the system qualities achieved by the application of this pattern [Weis03c]. A number of closely connected patterns form a pattern language, which guides the developer of agent system through the process of building a system. Patterns in the same pattern language are organized in a refined structure from high-level to low-level patterns [Weis03d].

2.2.1.1 Base Pattern Language

The pattern language used in the system design is based on the pattern language from [Weis03e] which is shown in figure 2.1. The base pattern language can be viewed as a starting point for more specific agent pattern languages. It documents the forces - the drivers, and design trade-offs - involved in agent-based design, and a set of generic agent patterns that apply to all agent systems. These generic agent patterns can therefore be used as a conceptual framework and context for documenting more specific agent patterns [Weis03e]. Agent Society is the highest-level pattern that depends on the following three patterns: Agent as Delegate, Agent as Mediator and Common Vocabulary. Pattern Agent as Delegate can be refined into User Agent and Task Agent and Agent as

Mediator can be refined into Resource Agent. The application is modeled as a society of agents and interactions between agents are defined in terms of roles and policies. We next give a brief introduction on role models and then identify the roles in our system.

Figure 2.1 Structure for the Base Pattern Language

2.2.1.2 Role Models

Roles are abstract loci of control. Protocols describe the way the roles interact. Policies define constraints imposed by the society on the roles [Weis03d]. Role diagrams (figure 2.2) are used to describe roles and their interactions. Roles and their subtypes can be represented in a role diagram. A role has other roles to interact with. Dynamic interactions between roles can be described by arrows. Subtyping can be represented in the similar means as inheritance relationship in class diagrams. A pattern documents how the roles interact in a given design context. The designer's task is to assign roles to agents by selecting patterns [Weis02].

Figure 2.2 Role Diagram

The base pattern language introduced in section 2.2.1.1 defined four types of generic roles: User, Task, Service, and Resource [Weis03d]. In the domain of agent-based application for Web information acquisition and dissemination, we identified all these four roles as top-level roles, from which more specialized roles can be derived by subtyping. Figure 2.3 depicts these roles and their subtypes we used in a role diagram.

Figure 2.3 Top-level Roles and Their Subtypes in This Pattern Language

The User role encapsulates the behavior of managing a user's task agents, providing a presentation interface to the user, developing and maintaining the user's profile, providing opinions for other user agents and filtering recommendations from other users.

Task roles represent users in a specific task. This is typically a long-lived, rather than a one-shot transaction [Weis01]. In the context of Web information acquisition and dissemination, we create Notifier role as a subtype of Task. The Notifier role integrates

information items extracted from various sources into a unified format and notifies the user.

A Service role typically provides a service to a group of users. It mediates the interaction between two or more agents through this service [Weis01]. In an open system where agents in the system are not statically predefined but can appear and disappear unexpectedly, a Mediator role help agents find others by name, or locate another agent that possesses a particular capability.

The Resource role abstracts information sources. These can be legacy data sources wrapped by “glue” code that converts generic requests from other agents to the API of the data source. These can be agents that process more complex queries by first breaking them down into subqueries and then collating the results [Weis01]. To meet the challenge of extracting information from heterogeneous Web data sources, we create concrete Resource role named Extractor. The Extractor role assists users in locating distributed sources and retrieving relevant information from heterogeneous Web data sources. It applies source-specific extracting rules on various Web data repositories to extract information that might be of interest to users and wrap it in a unified format.

To satisfy the responsibilities of all the roles, the base agent pattern language (described in figure 2.1) is extended by introducing two domain-specific patterns: Extractor Agent and Notifier Agent, refined from Task Agent and Resource Agent respectively. The definition for these two new agent patterns is described in Appendix B.

Figure 2.4 shows the structure of the extended pattern language.

Figure 2.4 Structure for the Domain-specific Pattern Language

2.2.1.3 Applying the Patterns

The global view of the agent patterns applying to our system can be described as: User Agents act on behalf of real users, develop and maintain their profiles, offer opinions to other User Agents, and filter recommendations from other User Agents. The Extractor Agent locates and extracts relevant information from various Web data sources according to users' profiles. The Notifier Agent carries out specific tasks (i.e. delivering updates to users in a unified fashion). Agent as Mediator is required for mediating communications between these agents. The agents coordinate and form an Agent Society where a Common Vocabulary is the agreed format for message exchange. Figure 2.5 shows the role diagram that results from applying the patterns to the system design. The

role diagram shows the roles agents need to fill in the pattern. When the patterns are applied, one agent may fill several roles. In our system however, roles and agents exhibit one-to-one correspondence.

Figure 2.5 Applying the Patterns

2.2.2 System Architecture

Figure 2.6 illustrates the system architecture which is the result of the pattern-driven design. The application is comprised of a set of distributed agents. Each agent can be thought of as a component achieving one or more objectives and the constituent components work together to deliver the overall functionality. The agent system is an open system, which means agents in the system are not statically predefined but can dynamically change over time. They may appear and disappear unexpectedly. The mediator agent, the extractor agent and the notifier agent are static, existing in the system all the time providing services to the user agents. But user agents may join or leave the

agent society at any time, which makes it an open system. Unlike the more complex open system that consist of entities that a designer had no knowledge of at design-time, our system has only one type of agent - user agent that might be appearing and disappearing.

Figure 2.6 The Multi-agent System

DBManager in figure 2.6 is the database access class that implements all of the database related operations required by the agents and other classes (e.g. JavaBeans used in JSP pages). The *DBManager* separates the application from the database and provides interfaces to the application, which eliminates the concerns about database-specific

implementation details in the application. The `extractWorker` is another helper class that can be viewed as part of the extractor agent since all the low level operation dealing with Web information sources are provided by this class and its methods are only invoked by the extractor agent. `KlipFolio` and user's `Klip` is the unified presentation tool to deliver relevant information to an end user. More about this tool will be introduced in section 3.4.4.

The applications can serve many users independently accessing the shared environment. Each user using the system has a user agent running on her behalf. The user instructs her user agent via the GUI, and the user agent monitors the user's activities to collect information to develop and maintain her profile. Agents within the multi-agent system have different capabilities therefore they need to cooperate in order to perform tasks that are beyond their scopes. The mediator agent helps them find each other before they can inter-operate. Agents communicate via conversations and the lines connecting the agents represent the conversations between them.

The system is designed to be generic, modular and extensible in order to ensure compatibility and applicability to varying environments, and data services.

We next describe the capabilities of each agent in the agent society. The implementation of the physical agent system will be discussed in Chapter 4.

2.2.3 Mediator Agent

One of the basic problems for open, multi-agent systems is finding the other agents

who might have the information or other capabilities that one agent needs [Deck97]. Like middlemen in physical world, mediator agents (also known as middle agents) support the flow of information in the agent community, assisting in locating and connecting the ultimate service providers and the ultimate service requesters. For the purpose of studying mediator agent, we categorize the agents in an agent system into three general types [Klus01]: (1) Provider agents that provide capabilities, e.g. extractor agent retrieving information from various Web sources; (2) Requester agents that consume information and services offered by provider agents in the system, e.g. a user agent requesting opinions from other user agents; (3) Mediator agents that mediate among both, requesters and providers, for some mutually beneficial collaboration. When a provider agent is instantiated, it advertises its capabilities to a mediator agent by sending a message describing the set of services it can offer. Researchers have identified different kinds of mediator agent that help agents find others in multi-agent systems [Jenn98] including matchmakers or yellow page agents (that match advertisements to requests for advertised capabilities), blackboard agents (that collect requests), and brokers (that process both).

(a) Service Brokering

(b) Service Matchmaking

Figure 2.7 Service Brokering and Matchmaking

Figure 2.7 illustrates the basic principle of service brokering and matchmaking [Klus01]. Requests for any provider agent's capabilities have to be sent to a mediator agent. Every request a matchmaker or broker agent receives will be matched against its advertisement set. If a match is found the matchmaker agent returns a ranked set of appropriate service provider agents together with the relevant advertisements to the requester. From this point on, the requester and provider agent communicate directly for service request and result reply. In contrast, when a match is found by the broker agent, it involves in the task of contacting the relevant providers itself by means of forwarding the service request to the service provider and communicating the respective results to the requester. A blackboard is a mediator agent that keeps track of requests [Deck97].

Requesters post their problems; providers can then query the blackboard agent for events they are capable of handling. Mediator agents are advantageous since they allow a system to operate robustly in the face of agent appearance and disappearance, and intermittent communications [Jenn98].

We define a system where agents already know what services they are asking for and know there are particular agents providing such services (e.g. extraction of Web information, and delivery of updates to users). Table 2.1 shows the service that will be needed and offered in the system, and the requester and provider agent involved in these services. For a multi-agent system with a limited number of agent types and interactions like ours, matchmaking can be designed in a way that agents are matched by hand according to their request and capabilities [Ogst01]. This is the simplest solution of the matchmaking approach. A service requesters do not query the mediator agent to look for another agent that possesses a particular capability (it already knows whom to resort to), rather, it just wants to know the provider's address (residing host and port) since the agents in the system are distributed across the network and their locations may change over time. After the requester obtains the address info from the mediator, it communicates directly with the provider.

Service	Requester(s)	Provider(s)
Web information extraction	User Agents	Extractor Agent
Update notification	User Agents	Notifier Agent
Opinion request	User Agent	User Agents

Table 2.1 Services in the System and their Requesters and Providers

2.2.4 User Agent

The user agents are capable of assisting the users in managing their profiles, automating relevant information acquisition from interesting Web sources, and representing the users in offering opinions to other user agents and filtering recommendations from other user agents. They exhibit the characteristics of collaborative agents, interface agents and information agents. The user agents act as collaborative agents when cooperating with other agents (the extractor agent, the notifier agent and other user agents) to perform specific tasks. The user agents act as interface agents when monitoring users' activities in order to develop and maintain their profiles. Also, the user agents act as an information agent in automating the process of information acquisition and dissemination.

The user profile plays a key role in identifying relevant items from large amount of information. We will elaborate on it in section 2.3. More about user agent's role in automating the process of Web information acquisition and dissemination will be discussed in Chapter 4.

2.2.5 Extractor Agent

The extractor agent is responsible for retrieving Web information from various distributed sources. To deal with heterogeneous data repositories, the extractor agent uses specific extracting wrappers to perform the assigned tasks. Provided with the users' profiles, the extractor agents also need to find the documents that best match the users'

interest. More detail about extractor agent and extraction procedures is discussed in Chapter 3.

The system will be able to acquire information from a wide variety of data sources. In this thesis however, we restrict our attention simply to Web-accessible digital libraries such as PubMed [PubM04] and Web sites that provide RSS feeds.

2.2.6 Notifier Agent

The notifier agent notifies users of interesting updates by integrating the items retrieved from various information sources, wrapping it in the unified format, and delivering it to users' desktops. More details about notification process will be discussed in Chapter 5.

2.3 Information Filtering

The multi-agent system integrates information filtering capability to improve the quality and relevance of items delivered to users. In this section we give a detailed description of this feature.

2.3.1 Recommender System and Information filtering

Recommender Systems help users identify particular items that best match their tastes or preferences. They make recommendations to users according to available information including data on items as well as different profiles of other users. Information filtering methods are essential in selecting the most appropriate information

with which to make decisions in recommendation. There are three widely recognized information filtering approaches for making recommendations [Mont03]: demographic filtering, content-based filtering, and collaborative filtering. Demographic filtering uses descriptions of people to learn about the relationship between a particular item and the type of people who like it. Content-based filtering uses descriptions of the content of the items to learn the relationship between a single user and the description of the items. User profile-item matching methods are used to compare the user's interests and the items. Collaborative filtering uses the feedback of a set of people on a set of items in order to make recommendations, but ignores the content of the items. Hybrid systems that combine features of content-based and collaborative filtering appear to be useful since the two methods involved prove to be complementary [Mont03].

2.3.2 Opinion-Based Information Filtering

When agent-based approach is applied to the information filtering domain, a centralized recommender system becomes a distributed world of recommender agents. Therefore a new information filtering approach called the opinion-based filtering method appears [Mont02]. The basic idea of this approach is to consider other user agents as personal entities in whom the agent trusts at different levels. A user agent asks its trusted friends for an opinion about a particular item and filters large amount of items based on the collected opinions. Trust level is expressed using a trust weight with which each user agent labels other user agents. Just as in the real world people ask their friends who have

similar tastes and interests for advice about interesting items, a user agent asks only trusted agents. Agents are not considered trustworthy either because their honesty or their trustworthy information but because of similar preferences and interests.

Traditional collaborative recommender systems employ a simple approach that recommends items based on similarity between user profiles. When a similar user provides unsatisfied advice, there is no way to ignore it and over and over again this agent causes a descent in performance. In the opinion-based approach however, when a user agent gives unsuccessful advice, it loses trust by the other agent to some degree. Its opinion or recommendation will be ignored if the agent provides irrelevant advices over and over again and trust in this agent is decreased below a threshold.

2.3.3 Social Trust Model for Recommender Agents

The opinion-based information filtering approach is based on a social trust model described in [Mont03]. The following dimensions of recommender agents are identified: user profile representation, initial profile generation, profile exploitation, relevance feedback, and profile adaptation.

2.3.3.1 User Profile Representation

The development of accurate profiles is a key task since the success of the information filtering system will depend to a large extent on the accuracy of users' profiles [Mont02]. A user's profile stores user-specific information such as her interest, preference, and trust relationships with other users. It determines how the user agent acts

in decision-making scenarios such as deciding whether or not to accept another user's recommendation.

The user's profile is comprised of two parts: (1) Interest aspect: attributes describing the user's subscription of Web information sources and associated keywords, represented by a vector of subscription-keywords pairs; (2) Trust relationships aspect: attributes describing the user's trust relationships with other users, represented by a vector of name-trust weight pairs.

In a user's profile, a group of keywords are associated with each of the subscribed information sources. The keywords are assigned with respective weights that describe the importance level of the keywords. The importance weights vary from 0 to 10, where 10 means the most important and 0 the least. Keywords and associated importance weights are the key factors that determine which items will be delivered to the user.

A user agent models its relationships with other user agents by trust weights. Trust weights reflect the degree to which an agent is willing to trust another agent's opinions or recommendations. Trust weights vary between -1 and 1. If user agent A's trust weight against B is 1, that means B is one of the most trusted user agents of A, -1 means the least trusted and 0 irrelevant. Each agent has a list of peer user agents with associated trust weights. Trust weights are solely related to evaluations on opinions and recommendations one user agent give another, and have nothing to do with anything else. For example, Mary and Bob may be very good friends in real world, but they work on very different

fields and do not share the same opinions on same articles. Therefore Mary's opinion of an article may be totally irrelevant to Bob, so Bob's trust weight against Mary could be 0 or even negative. In another example, Alice and Peter do not get along very well in real life but they are both in the field of bird flu research. As Alice is the expert in this area, her opinions are valued very much by many peers including Peter. Therefore, Peter's trust value for Alice could be 1.

Table 2.2 shows an example of the attributes in a user's profile.

Interest		Trust Relationship	
Information Source Name	Keywords	User Name	Trust Weight
Biological Procedures Online	DNA:10; HIV:8; Cancer:9	Peter	1
MedicineNet Daily News	Obesity:10	Mary	0.8
		Alice	0

Table 2.2 An Exemplar User Profile

2.3.3.2 Initial Profile Generation

It is desirable to learn as much as possible from the user in order to start making satisfactory recommendations from the beginning. There are several methods for initial user profile construction [Mont02]: manual input, stereotyping, training sets, and automatic recognition. We consider manual input for interest aspect of the profile construction since this approach appears to best suit the information acquisition problem that involves a wide variety of Web information sources. In our system, a user can browse and edit the interest aspect of her profile using the user agent GUI. The user can

add/remove subscribed data sources and edit keywords and their corresponding importance weights associated with each source. More details about browsing and editing user's profile via agent GUI can be found in section 4.6.2. One of the advantages of this method is the transparency of the system behaviour. When an item has been delivered to a user, she can easily guess why it was delivered. One problem with this method is that it requires some user's effort.

For the construction of trust relationship aspect of user profile, [Mont02] introduced the playing agent procedure to compute initial trust weight. It is essentially about using a training set for an experimental evaluation. In the interactive procedure, each agent asks the other agents about a set of known items and gathers their opinions on them. According to the similarity between the provided opinion and their own, each agent is able to infer a trust weight for each of other agents. We simply employed a method similar to the empty technique, but instead assigned an initial value, e.g. 0.5, as the starting trust weight due to its nature. The value will be updated when the user begins interaction with the system.

2.3.3.3 Profile Exploitation for Recommendation

To recommend items to a user, the user agent makes recommendations to users according to available information including data on items as well as different profiles of other users. Information filtering methods are based on user profile-item matching techniques and user profile matching techniques. In this agent-based Web information

acquisition and dissemination system, we consider the content-based filtering (item-profile matching) and the opinion-based filtering (based on trust model).

Content-based filtering is the first step to filter the vast amounts of items. During the process of extracting items for a user from Web data sources, the user's profile (the interest aspect) is used to check against the items to decide which of the items are likely to be of interest to the user. The user profile contains a list of weighted keywords that are used in deciding the relevance of the item in question. A more detailed description of the procedure can be found in section 3.7.

When the information extraction process is finished, the extractor agent provides a user agent with a set of retrieved items each associated with a certain relevance degree (calculated by comparing the content of the item with the user's profile, i.e. through content-based filtering). The user agent then decides whether to recommend the items to the user. If the relevance degree of an item is above a given threshold t^+ , the user agent is confident to recommend the item to the user. If the relevance degree is below a threshold t^- , that means the user has no interest in it and therefore the user agent skips the item. If the relevance degree is in $[t^-, t^+]$, the user agent turns to opinion-based filtering to make decision.

The opinion-based filtering method involves the following two steps:

- (1) Ask the trusted user agents for their opinions on particular items.

An user agent receiving a opinion request may choose to ignore it for a variety of

reasons: The user may not have read the requested item therefore the agent have no opinion for it; the requesting user agent may not have been very helpful in the past or the requesting agent may not be important enough. If however, an agent decides to respond to an opinion request, it sends back a response containing its opinion combining user's explicit as well as implicit rating about the requested items.

(2) Compute the estimated rating for the item with the opinions collected from all the queried agents. Since we are dealing with opinions from different agents with various trust weights, the appropriate combination function is the weighted average where weights are the trust weights of the queried agents. So

$$r_e = \frac{\sum_{i=1}^n r_i \times w_i}{\sum_{i=1}^n w_i}$$

where r_e is the estimated rating, w_i is the trust weight the requesting agent has against agent i , r_i is the opinion from agent i , and n is the number of the queried agents.

Combining the relevance degree obtained through content-based filtering and the estimated rating obtained through opinion-based filtering, the agent can decide whether or not to recommend each of the items to its user respectively. The combined value is represented as a recommendation level and will be included in the item delivered to the user as a means to explain to the user why the item is recommended.

2.3.3.4 Relevance Feedback

The user's profile evolves over time since her interest and trust in other users may

change as time goes on. To maintain the user profile, the user agent needs to be able to catch user's direct or indirect feedback by observing the user's actions, and update the user's profile continuously.

Direct feedback comes from user's explicit ratings for specific items. The user is given an opportunity to rate an item when she browses it (see section 5.2 for more details). The ratings range from 1 (not interesting) to 5 (very interesting). For an item delivered to the user through the automated extracting process, a high rating strengthens the weights of the set of keywords associated with the item and a low rating weakens them. For an item recommended by another user, high rating gains credit for trust in that particular user and low rating decrease the trust in that user.

Indirect feedbacks are caught from user's implicit evaluations on delivered items. Positive evaluations could be perceived from user's following activities: longer time spent on reading an item, saving an item on user's computer, recurrent visits to the same site, recommending an item to another user, saving the link of an item, etc. Negative evaluations could be reflected by: shorter time spent on reading an item, neglecting the recommendations of a particular agent, deleting an item from the delivered item list without reading, or dismissing (changing the state of an item to read state without actually reading it) an item, etc.

2.3.3.5 Profile Adaptation

A user's interest in information topics and trust in other users change over time.

User agents need to adapt to these changes to provide the most appropriate information to their users at any given time. Therefore, the update of user profile is required. The key issue in profile adaptation is the relevance feedback (explicit as well as implicit) from previous recommendations (i.e. delivered items).

(1) The interest aspect of a user profile

During the process of Web information extraction, each item in question is represented by a set of keywords that are used to compare against user defined keywords (more details can be found in section 3.7). The user's feedback on the item affects the importance weights on these keywords. For example, if an user agent notices that its user almost always read articles with "bird flu" in the title no matter what recommendation level the item is rated, then it develops the user's profile by putting more weight on this keyword and delivers to the user all items regarding this topic in the future. New keywords may be added to the user's profile in this way as well.

(2) The trust relationship aspect of a user profile

An agent's trust weight against another user agent also evolves. Trust is developed and updated over time. The user agent interprets events that can influence the degree of trust either as negative or positive. If the event is interpreted as negative, the agent will lose its trust to some degree and if it is interpreted as positive, the agent will gain its trust to some degree. Let us suppose that user agent P is the last user agent that joins the agent society. At the time it joins, every other user agent's trust weight against P is set to an

initial value, say 0.5. As time passes, user agent P offers opinions and recommendations to other user agents, gains credits for good ones and loses credits for the ones that are not that relevant. Therefore, every other user agent's trust weight against agent P evolves. User agent P may share similar opinions with some of the user agents, hence their trust weights against P increase. And some of them share different opinions with P, their corresponding trust weights decrease.

The initial trust weight should not be set to 0 or negative, otherwise the new agent's opinions and recommendations will be disregarded and therefore it has no opportunity to gain credit for trust by other user agents. The initial value should not be set to 1 either since it may seriously affect the combined opinion and recommendation of an item and cause misleading results.

The user is able to set her interest (keywords and associated weights) and trust weight against any particular user explicitly via the user interface. More about user agent GUI can be found in section 4.6.2.

2.3.3.6 Discussions

As the user agent dynamically develops and maintains the user's profile and affects the items delivered to the user's desktop accordingly, it would be very useful to give the user the opportunity to know what exactly the user agent is doing for her. This could be accomplished in the following ways [Lash94]:

- (1) Activity monitor: it shows the agent's activities such as catching a user's specific

action (e.g. saving an item, recommending an item to another user, deleting an unread item, etc.) and modifying a particular attribute of the user's profile.

(2) Explanation facility: it will be very helpful for the user to understand how her actions are perceived and translated to affect her profile attributes by the user agent.

(3) Interface that enables the user to browse and edit the profile: there are occasions that user's accidentally mistaken actions are caught and translated to some attribute changes in her profile by the agent; or simply some of the attributes in her profile become outdated, do not reflect her current preference and therefore need to be updated. In these situations, the user should be given the opportunity to browse and explicitly edit her profile. In our system, a user can browse and edit her profile using the user agent GUI. More details about browsing and editing user's profile via agent GUI can be found in section 4.6.2.

The social trust model discussed in section 2.3.3 is not fully implemented in current development. The keywords in interest aspect of user profile is not used, therefore content-based filtering is not applied either. Trust relationship aspect of user profile is implemented but profile exploitation and adaptation is left for future work due to its workload.

CHAPTER 3 EXTRACTION OF WEB INFORMATION

The objective of this multi-agent system is to find information that might be of interest to the users from diverse Web sources, extract the information, and present it to the users. In this chapter, we discuss the problem of Web information extraction. Section 3.1 gives a brief introduction of the background, followed by the discussion of various technologies used in the literature (section 3.2). In section 3.3, we introduce the Java support for regular expressions, the underlying technology of our approach. Section 3.4 gives an introduction to RSS format that is closely related to the extraction, presentation, and agent message exchanges. In section 3.5 and 3.6, detailed extraction procedures for Web sites without and with RSS feeds are described respectively. Section 3.7 discusses the issues about how to match the user's profile while retrieving Web information, followed by a short discussion that concludes this chapter.

3.1 Background

The Internet is now undisputedly the richest source for public information. Its structure, however, makes it a difficult source from which to extract data programmatically [Myll00]. HTML, the major carrier of the data of Web pages, was “designed to display data and focus on how data looks” rather than “to describe data and focus on what data is” [W3Sc04]. The majority of an HTML file describes the formatting of the Web page, which is irrelevant to the data it is meant to deliver. This property of

HTML poses a challenge for the developers who need to extract information from Web pages for data driven applications.

However, many information sources on the Web are semi-structured, which makes the data extraction from the Web possible. “A web page is semi-structured if information on the page can be located using a concise formal grammar, such as a context-free grammar. Given such a grammar, the information can be extracted from the source without recourse to sophisticated natural language understanding techniques” [Knob98]. Compared with unstructured data, the advantageous aspect of HTML is that interesting data is always wrapped in specific tags that can be identified by analyzing the source code. This feature of HTML is the foundation that enables various existing approaches in extracting information from Web pages. In the meantime, the information sources (e.g. online journals) we are going to work with are produced routinely and tend to have a degree of structure, which also makes some extent of analysis and extraction feasible.

3.2 Technologies

The simplest implementations take advantage of the built in String methods to extract Web-delivered information. In the approaches introduced in [Blok01] and [Leah99], string processing method `indexOf()` is used to locate the tags wrapping interesting information and `substring()` to extract the needed data. This method however often leads to complex or messy code when the source code of Web page gets very

complex.

Myllymaki et al [Myll00] proposed another approach to extract Web-delivered information. Illustrated in figure 3.1, the approach first retrieves the Web page content from the Web. Then the source is converted to XHTML by JTidy, a Java class library for cleaning up malformed and faulty HTML [Tidy04]. After that, the Web page and the resulting XHTML tree is examined manually to find the reference points for interesting information within the data. XPath expressions matching the anchors are then created and saved in an XSL file, which is used later to convert the XHTML into an XML file that contains only the concerned information. With all data stored in well-structured XML file, we can conveniently manipulate them using available XML class packages.

Figure 3.1 An Extraction Process

However, the more widely adopted approach of implementing the Web data extraction involves the use of regular expression.

WebSPHINX (Website-Specific Processors for HTML Information eXtraction)

[SPHI02], developed at Carnegie Mellon University, is a Java class library that enables automatic Web-page browsing, processing and Web information extraction. The extraction capability was accomplished by utilizing the Apache Regular Expression package [Apac03].

ShopBot [Door97] is a comparison shopping agent that helps the user to find superior prices at online vendors and reduce user's shopping time. Regular expressions are used to extract product attributes from the vendor pages.

ActiveRefresh [Acti03], an off-the-shelf software provided by SisyphSoft, is a "Web news monitor" that can not only aggregate available RSS feeds (a brief introduction on RSS can be found in section 3.4) but also create customized sources for Web sites that do not yet provide RSS feeds. User creates a new customized feed by defining a search pattern (in the form of a regular expression) to specify which part of information will be extracted from a Web page.

The approach we adopted uses Java support for regular expressions through the standard `java.util.regex` package [Weis03a] [Homm03] [J2SE03]. Although there are already many third party regular expression packages available for Java before Sun Microsystems introduced its own, the quality of the new package from Sun is excellent and surpasses most of the third party ones [RegE04].

3.3 Java Support For Regular Expressions

3.3.1 Regular Expressions

“A regular expression is simply a string of characters that defines a pattern used to search for a matching string” [Chng01]. The pattern-matching capability of regular expressions can be used as a tool to search, edit or manipulate text or data.

Regular expressions are often used in applications that require text processing for features like word search, search-and-replace functionality, email validation, or XML document integrity. These tasks often involve pattern matching.

3.3.2 Java regex package

The Java 2 Platform, Standard Edition (J2SE) starts to support regular expression for text processing from version 1.4 [J2SE03]. Developers can use the `java.util.regex` package to find, display, or modify some or all of the occurrences of a pattern in an input sequence. The package consists of three classes: `Pattern`, `Matcher` and `PatternSyntaxException`.

Figure 3.2 illustrates the extraction process using the Java support for regular expressions. The Web page content is first retrieved from the Web. It is then matched against the defined regular expression pattern created by analyzing structure of the Web page and the source code. The matching results are usually a group of strings if we need to extract multiple sets of data. Very often we need to further match the results against new patterns to get desired information.

Figure 3.2 Extraction Process Using J2SE Regular Expression Package

Before moving to the detailed description of the extraction procedure, we give an introduction to RSS, the format extensively used in this project.

3.4 RSS

RSS format is used in the project as a means to pass messages between agents and as a data carrier in various scenarios in extraction, recommendation and notification processes. Following is a brief introduction on RSS basics and how it is used in our project.

3.4.1 What Is RSS?

RSS is interpreted as Rich Site Summary, RDF Site Summary or Really Simple Syndication at different times. “RSS is the name given to a simple and well-established XML format used to syndicate headlines. Once a website creates an RSS file they have created a means to allow others to syndicate their headlines” [Purp01]. RSS is an efficient

and effective way for Web sites to distribute news-like content and for users to keep up with their favourite information channels. It has evolved into a popular means to broadcast information and been adopted by many Web sites featuring news, entertainment, health and so forth.

3.4.2 RSS Syntax

There are currently many RSS versions due to its complicated evolution history. The version we have been using is 0.91, which is an old but still quite popular version. Following is a sample RSS 0.91 feed (adapted from the sample feed of [Sere03a]):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="0.91">
  <channel>
    <title>Serence Sample Klip</title>
    <link>http://www.serence.com/sample_site/index.html</link>
    <description>Serence sample Klip using RSS</description>
    <language>us-en</language>
    <item>
      <title>Building a Klip is Easy</title>
      <link>http://www.serence.com/sample_site/sample1.html</link>
      <description>Sample Article</description>
    </item>
    <item>
      <title>How it all Works</title>
      <link>http://www.serence.com/sample_site/sample2.html</link>
      <description>Sample Article</description>
    </item>
  </channel>
</rss>
```

The structure of a RSS feed is quite simple. It comprises a channel, which has a title, link, description, and language (optional), followed by a series of items, each consisting

of a title, link, and description. In this project, what we are really concerned is data wrapped in element item.

3.4.3 How Is RSS Used?

RSS feeds readers (also known as RSS-aware news aggregators) are used to check RSS feeds. RSS feeds readers can pull RSS feeds from the Web sites regularly and display new items to the user, thus the news comes to the user instead of the user going to the news. A short list of available readers can be found in [Medi04].

Figure 3.3 RssReader

Figure 3.3 is the screen shot of one of the free RSS feed readers: RssReader [RssR04]. We can see from the figure that the user subscribed RSS feeds from various

sources: latest bargains, software updates and several channels related to medicine and health care.

3.4.4 KlipFolio

One of the purposes of this project is to disseminate information, i.e., to deliver the items that might interest the user to the user's desktop. The information is in the form of RSS and the RSS-aware news aggregator being used is Serence Inc.'s KlipFolio [Sere04] shown in figure 3.4. KlipFolio is a desktop application that allows a user to easily view and manage live, automated and customizable channels to unique information sources. These channels are called Klips [Sere04].

A Klip is an XML file with special tags and directive that instruct KlipFolio where to get data and how to display it. The data source for a Klip is called KlipFood, meaning that it is the food for a Klip. Various formats are supported as KlipFood such as .food, .rss, .rdf, .xml, .html, etc. RSS is chosen as the format of KlipFood in our project because it is the widely accepted format for syndication. The syntax of RSS was described in section 3.4.2.

The KlipFolio shown in figure 3.4 integrates channels of various information sources: CNET news, local weather condition, stock indices, medical news, tech bargains, etc. The expanded Klip on the bottom is the one created by our multi-agent system. This customized Klip delivers to the user the latest updates extracted from various Web sources according to her interests and the updates recommended by her trusted peers.

Figure 3.4 KlipFolio

The simplified version of our task can be described as constructing and updating the users' Klips, i.e., extracting interesting information from various sources on the Web, constructing RSS feeds based on the retrieved information and other users' opinion about it, and modifying users' Klips accordingly.

3.4.5 Discussion

Semantic Web [W3C04] can be used as an alternative output format. The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. The idea is to have data on the web defined and linked in a way that it can be used by machines not just for display purposes,

but for automation, integration and reuse of data across various applications. It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming. RDF is data about web data - or metadata that defines extra information (e.g. other users' opinions) about a page and allows other tools to process the data. Similarly, as a variation of XML format, RSS can include same metadata and support program processing.

3.5 Detailed Extracting Procedure By an Example

We explain the method of data extraction we used by means of an example. In this example, we are going to extract the title and abstract information of an article from an online medical journal: “BioMedical Engineering OnLine Volume 2 Number 1, 2003”. The title of the article is “Dynamics of a disabled population in Morocco” and the website is “<http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=151273>”. The screen shot of the Web page is shown in figure 3.5. This extraction is the real case taken place when the extractorAgent gets the detailed information of a recommended item, which is described in section 4.5.8.

Figure 3.5 Web Page To Be Examined

First the content of the Web page is retrieved and saved as a String object. The JTidy beautified result is shown in figure 3.6. In the figure, the document is divided by a splitter bar to show the two portions (title and abstract) that we are concerned.

Note the original extracted result is usually represented as a single line long string, which makes the analysis of source code extremely difficult. Therefore JTidy becomes a very handy tool to make the job easier.

Now we examine the original Web page and the result source string to locate the specific regions in the source code where we can extract the data we are interested without concerning ourselves with the irrelevant information. The two parts that we are

used in the pattern string. The Pattern API contains a group of useful predefined character classes, which offer convenient shorthands for commonly used regular expressions:

Predefined Character Classes	
.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

Table 3.1 Predefined Character Classes

Quantifiers are used to specify the number of occurrences to match against. There are three types of quantifiers defined in the Pattern API specification: greedy, reluctant, and possessive. The subtle implementation differences are explained in [Homm03].

Quantifiers			Meaning
Greedy	Reluctant	Possessive	
X?	X??	X?+	X, once or not at all
X*	X*?	X*+	X, zero or more times
X+	X+?	X++	X, one or more times
X{n}	X{n}?	X{n}+	X, exactly n times
X{n,}	X{n,}?	X{n,}+	X, at least n times
X{n,m}	X{n,m}?	X{n,m}+	X, at least n but not more than m times

Table 3.2 Quantifiers

Next, we need to extract the abstract of the article from the HTML source. By examining the source and the rendered Web page, we can see that the abstract is composed of different parts: background, methods, results and conclusions. First we use

the following regular expression pattern to fetch the chunk of data enclosing all parts of the abstract:

`“</a>Abstract.*?</td>(.*?)</a>”`

And then the extracted result is matched against the following pattern to remove the formatting tags and extract the information delivered:

`“<p><span class=\"abs-head2\">(.*?)<br></span>(.*?)</p>”`

At this point, we get all data we need from the extraction procedure. We then reorganize the extracted data and wrap them in RSS format which will be used in subsequent processes.

3.6 Extraction From RSS Feeds

Many news-like websites now provide RSS feeds as a means to deliver updated content posted on the website to users’ desktop [Medi04] [Ecto04] [XPBa04]. Take MedicineNet for example, it has general and specific medical and health channels covering the latest medical news, diseases and conditions, procedures and tests, medications, and the MedTerms medical word of the day channel. A partial list of the available channels from MedicineNet is shown in figure 3.7. By subscribing to a channel, users can have important headlines delivered to their desktops on health and medical topics of interest to them.

General Health and Medical Channels	
Allergies <i>General</i> XML	High Blood Pressure <i>General</i> XML
Alzheimer's Disease <i>General</i> XML	HIV <i>General</i> XML
Arthritis <i>General</i> XML	Liver <i>General</i> XML
Asthma <i>General</i> XML	Lungs <i>General</i> XML
Cancer <i>General</i> XML	Medications <i>General</i> XML
Cholesterol <i>General</i> XML	Menopause <i>General</i> XML
Chronic Pain <i>General</i> XML	Men's Health <i>General</i> XML
Cold & Flu <i>General</i> XML	Migraine Headaches <i>General</i> XML

Figure 3.7 Partial List of the Available RSS Feeds from Medicine Net

The task of extracting Web-delivered information becomes much easier when dealing with RSS feeds. Since RSS is essentially an XML format that was designed to describe data, structure data and has rich Java package available to manipulate the data, we can extract the information we need by the steps illustrated in figure 3.8.

Similar as approaches described earlier, the RSS feed is first retrieved from the Web. Then we obtain an instance of the DocumentBuilderFactory that can give us a document builder. After an instance of the builder is obtained, it is used to parse the fetched RSS String to construct a DOM tree. Then we can use the methods provided by org.w3c.dom package [J2SE03] to locate the nodes and extract the delivered information we care about. For RSS feeds, we are interested in three kinds of node of the items: title, link and

description.

Figure 3.8 Steps to Extract Data from an RSS Feed

3.7 Retrieving Information that Matches User's Profile

To retrieve only the items that best match users interests, content-based filtering technique is used in the Web information extraction process. Content-based filtering uses descriptions of the content of the items to learn the relationship between a single user and the items. User profile-item matching methods are used in order to compare the user's interests and the items.

When requested by a user agent to retrieve Web information from a specific source, the extractor agent is provided with that particular user's profile (user profile is discussed in section 2.3.3.1). The extractor is required to retrieve only the pages that best match the user's interest that is represented by a set of weighted keywords associated with the

specific source.

The vector space model [Salt94] [Sing95] is used to determine how close a page match the user's profile during the information retrieval. The basic idea is to represent each document as a vector in a vector space so that documents with similar content have similar vectors. The agent first performs a full text analysis on the page in question. Then both the online document as well as the search keywords are represented by sets of terms (term vectors). The two vectors are compared with each other and vector similarity coefficients are obtained to identify whether the document is sufficiently similar to the keywords defined in the user's profile. If the page matches a user's profile with a score above a certain threshold, it will be accepted and further processed. Otherwise, the page will be skipped.

The vector space model of text processing is the most widely used information retrieval model [Sing95]. In the model, information item is stored as a set (or vector) of terms and non-content words like "the", "of", "an", ... are removed from the vector. Term weight is used to describe the importance of a term, usually defined by the number of occurrence of the term, also called "term frequency" or *tf*. Higher term weights are assigned to more important terms. Term frequencies would be larger for longer documents because they tend to contain more assigned terms. To make the situation fairer, normalization of term weights is often used. Following is the formula that calculate normalized weight for the *i-th* term:

$$\frac{tf_i}{\sqrt{tf_1^2 + tf_2^2 + \dots + tf_n^2}}$$

Cosine normalization is adopted in the calculation.

A vector inner product is used to find how close the term vector of a page matches the keyword vector from the user's profile. If $\mathbf{K}$ is the keyword vector and $\mathbf{D}$ is the vector representing for the document, the similarity between the keyword and the document is computed as

$$Sim(\mathbf{K}, \mathbf{D}) = \sum_{\text{common terms } t_j} k_j \times d_j$$

where t_j is a term present in both the keyword and document vector, k_j is the weight of a term t_j in the keyword and d_j is its weight in the document. The resulting quantified similarity is then compared with a predefined threshold to see whether the document is good enough to be fetched for the user.

3.8 Discussions

When extracting data from an online information sources, the following factors have to be taken into consideration:

- (1) Will the source be available persistently?
- (2) Will the structure of the source be stable?

The Internet keeps changing ceaselessly. It is often the case that a link saved in the favourite folder leads to a page telling you "Page Not Found" or to a page with very

unfamiliar structure even though the content is still familiar. Take the dynamic nature of the Internet into account, it is not easy to find robust solutions that will work all the time. The extraction of Web-delivered information will be easiest when the sources are reliable and stable. Structure change of the Web source means changes in extraction patterns.

In future work of this project, it is possible to create a subsystem to monitor the changes of concerned Web sources and automate the modification of corresponding extraction patterns. Bauer [Baue00] describes a possible solution to this problem. Information extraction wrappers (for Web information extraction) are developed based on the paradigm of programming by demonstration. A human trainer teaches a wrapper interactively so it can locate and extract certain pieces of information from online documents. These would to some extent eliminate the tedious task of hand-coding regular expression patterns for information extraction and updating them whenever the corresponding Web sites have been changed.

CHAPTER 4 THE MULTI-AGENT SYSTEM IMPLEMENTATION

This chapter focuses on the implementation aspects of the agent system. Section 4.1, 4.2 and 4.3 introduce agentMom, the agent framework used in the project. The conversations modeled for the system and the scenarios where the conversations take place are then described in section 4.4 and 4.5. Section 4.6 introduces the graphic user interfaces for the agents.

4.1 Agent Development Environment

An agent development framework called AgentMom [Agen04] is being used in the project. AgentMom is a Java based framework for the development of multi-agent system. It provides basic building blocks for creating agents, conversations between agents and messages passed in the conversations.

AgentMom is a simple and easy to understand implementation of an agent framework, consisting of only a handful of classes based on which the developer can build their own agent systems. The lightweight feature of the framework is one of the important reasons we adopted it.

4.2 Operation Semantics

Figure 4.1 shows how agentMom works [DeLo00a] [Weis03a]. An agent gets ready for communication by starting a message handler that listens to a port for messages.

Agents communicate via conversations, which define valid sequence of messages that the participating agents can understand. A normal conversation has an initiator and a responder agent with each agent has its own side of conversation.

Figure 4.1 How AgentMom Works

One agent starts its half conversation as a new Java thread to initiate the communication. The conversation establishes a socket connection with the other agent's message handler and sends the initial message. When the responder agent's message handler receives a message, it passes the message to the agent's receiveMessage method where the message is compared against its known list of accepted messages to see whether the message is a valid start of a conversation. If the conversation is valid, the agent starts its side of the corresponding conversation, also as a new Java thread. From that point on, all communication is controlled by the two conversation threads.

Conversation exchange messages using built in readMessage and sendMessage methods.

While conversations handle the messages passing between agents, they also need to invoke methods on their parent agents.

4.3 AgentMom Package

AgentMom Package shown in figure 4.2 is the basic that makes up the agentMom framework. The original package consists of three abstract classes MomObject, Agent and Conversation and three concrete classes Message, MessageHandler and Sorry.

Figure 4.2 AgentMom Package

We expand the package by adding the following sub-classes: abstract class

AgentOne and ConversationOne, concrete classes MessageHandlerOne, MediatorAgent, ExtractorAgent, NotifierAgent, UsesrAgent and several conversation classes that are not shown in the figure for the sake of clarity. Following we give a short introduction of the main classes in the package.

4.3.1 Agent Class

This abstract class defines the minimum requirements for an agent. It must be runnable in a thread of its own, which requires a run method. It must have two parameters agent name and port number on which its message handler monitors for incoming messages. Also the agent must implement the receiveMessage method that will be called by the message handler when the agent receives a request for a new conversation.

4.3.2 Conversation Class

This is an abstract class that carries out the actual message passing between agents. There are two methods sendMessage() and receiveMessage() that pass message back and forth over the socket connection. There are two types of conversation classes that implement the two halves of the conversation: initiator and responder. The basic difference lies in the constructor and details of run method.

4.3.3 MessageHandler Class

This is a concrete class that will be used directly by agent. Agent create a message handler as a new Java thread. When started, the message handler starts a socket server and waits for a connection from another agent. When it receives a message, the message

handler calls its parent agent's `receiveMessage` method and the agent does the rest as described above.

4.3.4 Message Class

Similar to KQML (Knowledge Query and Manipulation Language) [UMBC04] and FIPA's ACL (Agent Communication Language) messages [FIPA02], the message class defines the fields used in the message passed between agents. The most important fields are host, port, performative and content. Fields host and port will be filled automatically by `sendMessage` method using parent agent's attributes. The performative field describe the action the message intends and the content field carries the content passed between agents. The content can be any Java object as long as it implements `Serializable` interface. In this project, we use `String` object to pass simple message and `Hashtable` or `Vector` objects for complex messages.

4.3.5 Sorry Class

The sorry class is the default conversation class that sends a sorry message when a performative is received and not expected by the parent agent's `receiveMessage` method or within the flow of a conversation.

4.4 Conversation Diagrams

Figure 4.3 Conversations Between UserAgent and ExtractorAgent

We can use AgentTool to model agents and conversations between agents graphically [Agen04]. The AgentTool is an interactive software engineering tool based on the Multi-agent Systems Engineering (MaSE) methodology proposed by S. DeLoach et al [DeLo01]. Figure 4.3 is the snapshot of the agent template diagram panel for modeling conversations between UserAgent and ExtractorAgent. We can see from the diagram that three conversations are defined between these two agents. For all these conversations, UserAgent is the initiator and ExtractorAgent is the responder. These conversations will take place in different scenarios discussed in later sections.

The naming convention for conversations is as follows: the name starts with “Conv_”, meaning that it is a conversation. Followed are the two participating agents’ names divided by “2”, which means “to”. The last portion of the name states the goal of the conversation. Take the first conversation in figure 4.3 for example: Conv_UserAg2ExtractorAg_GetLatest is the conversation between a UserAgent and the ExtractorAgent with the UserAgent as the initiator and ExtractorAgent the responder. The Goal of the conversation is to get the latest items of a data source, which is represented as GetLatest. As described earlier, each conversation is implemented by two types of

conversation classes: initiator side (identified by “_I”) and responder side (identified by “_R”).

Figure 4.4 Other Conversations

Figure 4.4 illustrates all other conversations defined in the project and table 4.1 lists their Goals.

Conversation Name	Goal of the Conversation
Conv_AnyAg2MediatorAg_Register	Agent uses this conversation to register with mediatorAgent when it goes online.
Conv_AnyAg2MediatorAg_DeRegister	Agent uses this conversation to deregister with mediatorAgent when it goes offline.
Conv_AnyAg2MediatorAg_Lookup	Agent uses this conversation to get another agent's host and port before initiating a conversation with it, therefore it does not need to know where the responder agent resides.
Conv_MediatorAg2AnyAg_UpdateOnlineAgentList	MediatorAgent uses this conversation to notify all online userAgents to update their online agent lists. It takes place when an agent goes online or offline.
Conv_MediatorAg2AnyAg_Bye	The mediatorAgent starts this conversation with a deregistering agent to ask its message handler to close the server socket on which it monitors incoming messages. The port can therefore be released and used again afterwards.
Conv_UserAg2ExtractorAg_GetLatest	UserAgent uses this conversation to ask the extractorAgent to get the latest update list from a particular data source.
Conv_UserAg2ExtractorAg_getOneDataSrcItemsRSS	UserAgent uses this conversation to ask the extractorAgent to get the detailed information about items in one data source.
Conv_UserAg2ExtractorAg_getOneItemRSS	This conversation is used in userAgent's CheckRecommendation scenario when the userAgent asks the extractorAgent to get the detailed information about one recommended item.
Conv_UserAg2UserAg_AskOpinion	UserAgent uses this conversation to ask other userAgents' opinion about a set of items extracted from a data source.
Conv_UserAg2NotifierAg_Notify	UserAgent uses this conversation to notify the real user of the updates it thinks valuable.

Table 4.1 Conversations and Their Goals

State diagrams can be used to model the dynamic aspects of conversations [DeLo00b]. Every conversation has two diagrams that describe it: the initiator state

diagram and the responder state diagram.

Figure 4.5 State Diagram for Initiator

Figure 4.6 State Diagram for Responder

Figure 4.5 and 4.6 illustrate the two state diagrams for conversation Conv_UserAg2ExtractorAg_GetLatest. Format for state transitions is as follows [Weis03a]:

recv-msg(args1) [guard] / action ^send-msg(args2)

The statement can be interpreted as: if the message `recv-msg` is received with the content `args1` and the condition guard holds, the action will be invoked and the message `send-msg` will be sent with content `args2`. Note all elements are optional. Also defined in the states below the state name are the actions of a state which is also optional. Every conversation can be modeled using the state diagram. Since the logics for the conversation are quite similar and straightforward, we do not show the rest state diagrams here.

4.5 Conversation Scenarios

In this section, we describe the scenarios when the major conversations take place.

4.5.1 UserAgent Goes Online

Figure 4.7 Activity Diagram for UserAgent's Going Online

4.5.2 UserAgent Goes Offline

Figure 4.8 Activity Diagram for UserAgent's Going Offline

4.5.3 RefreshData

When userAgent starts running, a thread called RefreshDataThread starts as well. Its responsibility is to check updates from the user's subscribed data sources and refresh user's displaying items based on the user's designated refreshing schedule. The dynamic aspects of the procedure are illustrated in sequence diagrams. Because the complete trace of the flow of control is complex, we break it up into separate diagrams shown in figure 4.9, 4.10, 4.11, 4.12 and 4.13 respectively.

Figure 4.9 Sequence Diagram for RefreshData

Figure 4.9 shows the overall flow of control by time ordering. After the instance of UserAgent starts, it creates and starts an instance of RefreshDataThread. The thread RefreshDataThread first gets the user's subscription titles and respective keywords from the database, table usersubscription, to be specific. For each of the subscribed data sources, it goes through the following steps: It gets the latest items of this particular data source by communicating with the extractorAgent (ref getLatestItemsOfOneDataSource). Then it obtains the user's buddy list from the database with the help of DBManager. After that, each userAgent in the buddy list is contacted to get their opinions about the fresh items just extracted from the data source (ref getOpinion). With all opinions from its trustworthy "buddies", the userAgent now starts calculating the estimated rating for each of the items using collaborative algorithm. The result ratings are then checked against the user specified threshold. The threshold is used to decide whether or not an item is good enough to be recommended to the user. If the calculated rating is lower than the threshold, the userAgent thinks the item is not valuable enough to notify the user and therefore skips the item; otherwise, the conversation with extractorAgent is established again to obtain the details of the saved items of that particular data source (ref getOneDataSrcItemsRSS). The reasons why detailed information was not collected in the first time userAgent communicated with extractorAgent are as follows:

- (1) The extractorAgent does not extract information from the same website for these two conversations, therefore different extracting rules and processes are used. The first

conversation obtains information from website containing only directory-like entries and the second one obtains items' detailed information from the particular items' Web pages showing the details of the items. The first conversation usually returns RSS string with only title and link of an item while the second conversation gets description (its abstract in most cases) of the item.

(2) Getting all the information at once seems to save the second conversation, but the large amount of bytes transmitted in the userAgent-extractorAgent extracting conversation and userAgent-userAgent opinion-seeking conversations may cause significant overhead. And many items may be discarded later due to low estimated ratings after all these transmissions, which means a lot of byte transmissions were wasted. When the detailed information about all the kept items of the current data source is collected, it is the time to notify the user of these updates (ref notify).

A complete cycle finishes when the procedure described above go through all the user's subscribed data sources once. And the RefreshDataThread runs such cycle regularly based on user's schedule. The user can modify the schedule by setting the time interval value between consecutive data refresh cycles and the RefreshDataThread check the value every time before it starts the cycle.

4.5.4 GetLatestItemsOfOneDataSource

Figure 4.10 is the sequence diagram illustrating the flow of control in the process of extracting the latest items of a particular data source. It is accomplished by

RefreshDataThread communicating with the extractorAgent. Thread RefreshDataThread initiates the conversation Conv_UserAg2ExtractorAg_GetLatest_I with extractorAgent, requesting the latest items of the particular data source. The extractorAgent responds with its corresponding half of conversation to fulfil the task. When the conversation ends, the userAgent gets the latest items of the particular data source.

Figure 4.10 Sequence Diagram for getLatestItemsOfOneDataSource

4.5.5 AskOpinion

Figure 4.11 shows the Sequence Diagram for process askOpinion. The userAgent initiates the conversation Conv_UserAg2UserAg_AskOpinion_I with a userAgent in the buddy list to get its opinion on a set of items. The contacted userAgent responds by

starting its corresponding half of conversation. When the conversation ends, the userAgent gets the other useAgent's opinions about all newly extracted items of the current data source.

Figure 4.11 Sequence Diagram for askOpinion

4.5.6 GetOneDataSrcItemsRSS

Figure 4.12 shows the Sequence Diagram for process `getOneDataSrcItemsRSS`. The `RefreshDataThread` initiates the conversation `Conv_UserAg2ExtractorAg_getOneDataSrcItemsRSS_I` with `extractorAgent` to get detailed information about the items in one data source. The `extractorAgent` responds by starting its corresponding half of conversation that calls its parent's extracting methods to

fulfil the task. When the conversation ends, the detailed information of the selected items of the particular data source is returned in complete RSS format. The information also includes the calculated rating data. The items in the result RSS string are in the following format (only partial information of each element is shown for clarity):

```
<item>
  <description>Recommend Level: *4* Abstract: The provision of ...</description>
  <link>http://....</link>
  <title>Multi-purpose HealthCare Telemedicine Systems with ...</title>
</item>
```

Figure 4.12 Sequence Diagram for `getOneDataSrcItemsRSS`

We can see in the description portion, the calculated rating is included as part of the text content, which provides the user with a hint of explanation for the recommended items. This content customization is also one of the reasons why the extraction cannot be accomplished in a single conversation with the `extractorAgent`. In the first conversation,

the estimated rating is not available yet.

During the getOneDataSrcItemsRSS process, the extractorAgent insert new records into the database (table itemsource) where data source name and item link pairs are stored. These records are used in the recommendation processing scenario.

4.5.7 Notify

Figure 4.13 Sequence Diagram for notify

Figure 4.13 shows the Sequence Diagram for process notify. This process takes place when selected items from a particular data source are ready to be delivered to the user. To fulfil this task, the thread RefreshDataThread initiates the conversation Conv_UserAg2NotifierAg_Notify_I with notifierAgent and the notifierAgent responds with its half corresponding conversation to customize the items and update the user's

Klip, therefore notifies the user of the new updates.

At the end of each conversation illustrated in figure 4.10, 4.11, 4.12 and 4.13, the initiator conversation and the responder conversation threads are destroyed, meaning that the process is finished and the related resources are released.

4.5.8 CheckRecommendation

Figure 4.14 Activity Diagram for CheckRecommendation

When userAgent starts running, another thread called

CheckRecommendationThread starts as well. Its responsibility is to check other users' recommendations to the user and update the user's Klip regularly. The activity diagram in figure 4.14 illustrates the procedure. The thread CheckRecommendationThread first checks the database, i.e. table recommendations, for new recommendations other users made to its owner user. If there is no recommendation, the process finishes. Otherwise, for each recommendation, it gets the recommender's user ID and the recommended item ID that is identified by a link. The userAgent then checks the database (table buddies) for its trust weight against this recommending user. If the trust weight is lower than the trust threshold, the recommendation is ignored because the user does not trust any recommendations this particular user sends to her. If the trust weight is higher than the specified threshold, the recommendation will be further processed. First the data source name of the link is retrieved from database (table itemsource), and then the userAgent initiates the conversation Conv_UserAg2ExtractorAg_getOneItemRSS_I with the extractorAgent to get the detailed information about the item. The extractorAgent starts its half conversation to extract the desired information of the item for the userAgent. Sequence diagram in figure 4.15 illustrates the process. In this process, the extractroAgent also adds the recommendation-specific messages to the result RSS string: It puts the recommender's trust weight against the description portion of the RSS result. The format of the description part is therefore in the following format:

<description>

Recommended by: *Bob(trustWeight=0.6)* Abstract: Bird flu is...

</description>

This is also a way to explain to the user from whom and why she gets the recommendation. In the above example, the user will learn from the Klip item that Bob recommends this item to her and she receives his recommendation because her trust weight against Bob is high enough.

Figure 4.15 Sequence Diagram for getOneItemRSS

When the conversation getOneItemRSS finishes, the userAgent initiates the conversation Conv_UserAg2NotifierAg_Notify_I with the notifierAgent to notify the user of the recommended item. The process is same as discussed in the previous section (notify and figure 4.13).

4.6 Graphic User Interface

4.6.1 Mediator Agent

The mediator agent mediates the interaction between members in the agent society. It assists in locating and connecting the service providers and the service requesters. For the mediator agent in our multi-agent system, its key task is matchmaking. Although the mediator agent can function at its full capacity without a user interface, a graphical user interface (GUI) can definitely provide a better view of the multi-agent system to the developer/user.

Figure 4.16 The Mediator Agent: Agents Tab - All Agents Online

The collaborative agent society can be formed by a group of agents that are

distributed across the network. From the mediator agent GUI (figure 4.16 shows the Agent information tab), the developer/user can check the status of the multi-agent system: Who is online? From which host and which port? An agent's identification shown in the tab is in the following format: "agentID@hostName:portNumber". From the figure we can see that there are agents currently running in the system, located in three different computers: c900, c600 and nb (They are my three networked home computers: two desktops with CPU Celeron 900 and 600, and a notebook). The three service agents, namely, the mediator agent, the extractor agent and the notifier agent reside on c900 at port 5555, 5001 and 5002 respectively. User agent Mary also runs on c900 while other five user agents are distributed over c600 and nb. All the agents shown in figure 4.16 have coloured awake face icons, displayed with agent name, residing computer name and port, indicating that they are online. When an agent goes down (e.g. leaving the society permanently) the icon representing the agent becomes grey-coloured sleeping face icon to indicate that the agent is inactive. Also the inactive agent's host name and port number are removed. In figure 4.17, John's user agent is shown in inactive state.

Figure 4.17 The Mediator Agent: Agents Tab - Agent John Offline

Figure 4.18 shows the message tab of the mediator agent GUI. The top two panels display messages received and sent by the mediator agent respectively, and the bottom panel displays the internal state changes of the mediator agent under working condition. The message tab is not strictly necessary, but is often useful for demonstration and debugging purposes.

Figure 4.18 The Mediator Agent: Messages Tab

4.6.2 User Agent

User agents assist the users in managing their profiles, automating relevant information acquisition from interesting Web sources, and representing the users in offering opinions to other user agents and filtering recommendations from other user agents.

Figure 4.19 illustrates the subscription tab of the user agent GUI (user Mary's agent shown, size of subscription tab reduced for clarity). The user can browse and edit the interest aspect of her profile via the subscription tab. She can add or delete her subscribed Web information sources, edit keywords, and modify the weights of the keywords for her

subscriptions.

Figure 4.19 The User Agent: Subscriptions Tab

Figure 4.20 The User Agent: Friends Tab

Figure 4.20 shows the friend tab of a user agent GUI where the user can explicitly modify the trust relationships aspect of her profile. She can move the icon representing other users between Blocked (Untrusted), Regular and Trusted categories to modify her trust relationships with other users. The user's trusts in other users evolve according to the credit gaining and losing rules described in section 2.3.3, therefore any particular user's icon may move within these three categories. There are trust weight thresholds for the three groups, which is defined by table 4.2:

Category	Blocked	Regular	Trusted
Weight Range	[-1, 0]	(0, 0.8)	[0.8, 1]

Table 4.2 Trust Weights For Three Categories

Blocked represents a trust relationship where the other user's opinions and recommendations are irrelevant. This category may be useful in scenarios such as two users conduct research in totally different areas and do not share similar opinion in same articles. Trusted means user agents in this list are the most trusted ones and their opinions and recommendations are highly valued. This category may be used by a user to include peers working within a group or a very close area. Category Regular is used to put all other user agents against whom the trust weights evolve regularly by gaining and losing credits for their opinions and recommendations. The trust weight in this category should be between 0 and 1.

4.6.3 Extractor Agent

The extractor agent locates and extracts relevant information from heterogeneous Web data sources using source-specific wrappers. Figure 4.21 displays the message tab of the extractor agent GUI. Similar to those in the mediator agent and the user agent GUI, it shows the message exchanges between the extractor agent and other agents and the state transitions under working condition.

Figure 4.21 The Extractor Agent: Messages Tab

4.6.4 Notifier Agent

The notifier agent notifies users of interesting updates by integrating the items retrieved from various information sources, wrapping it in the unified format, and delivering it to users' desktops. The GUI of the notifier agent (figure 4.22) also contains the message tab that illustrates message exchanges between the notifier agent and other agents and the state transitions under working condition.

Figure 4.22 The Notifier Agent: Messages Tab

CHAPTER 5 NOTIFICATION OF UPDATES

The final stop of the data flow in the system is the user's desktop. As described in section 3.4.4, information is presented to the user via a Klip incorporated in the user's KlipFolio. The KlipFood, the feed for a Klip, is in the form of RSS. The notification process involves delivering the items that might interest the user to the user's desktop, i.e. we need to update the user's KlipFood with the delivered information, both of which are in RSS format.

5.1 Notification Process

Figure 5.1 demonstrates the notification process. First the user's KlipFood file is read in and converted into a string. The KlipFood string and the delivered string, both in RSS, are then parsed by an instance of a DocumentBuilder which is created by a DocumentBuilderFactory instance. The parsed results are two DOM objects: the KlipFood DOM tree and the Update DOM tree. Each element in the Update DOM tree is then compared against the elements in the original KlipFood DOM tree. The elements are identified by the link nodes since every link is unique. If the element from the Update DOM tree already exists in the KlipFood DOM tree, the element in the KlipFood DOM tree is replaced; otherwise, the element is inserted into the KlipFood DOM tree.

The reason why we replace the existing element with the element having same link node from the Update DOM tree is that the description node of the element may contain

different texts. The text node of the description includes other users' recommendation about this particular item (see section 4.5.6 and 4.5.8 for example items). And the recommendation level (represented by a number) is calculated each time the userAgent gets the item from the extractorAgent and asks for opinions from other user agents. The opinion about an item one user agent provide for another user agent changes over time and the trust weight at which level one user agent trust another user agent also changes. Therefore the recommendation level calculated using the collaborative algorithm might produce different results at different times even for the same item. In this way, the recommendation level representing the latest opinions from other user agents will be presented to the user.

Figure 5.1 Process of Updating User's Klip

After the KlipFood DOM tree is updated with every item in the Update DOM tree, it is written back into the user's KlipFood file again. Because the KlipFolio refresh the

Klips by reading KlipFoods regularly, the items in a Klip gets updated and the user get notified of new items. In current version of the system, a KlipFood file is simply named by the user's id. For example, mary.rss is the KlipFood file for user Mary. Similarly, user's Klip file that instruct KlipFolio where to get data and how to display the data is also named by the user's id but with extension ".klip". Again, take Mary for example, her Klip file is mary.klip.

The nodes manipulation in the notification process is accomplished by methods provided in org.w3c.dom package [J2SE03].

5.2 Web Interface

When the user clicks on an item displayed in the Klip, the default Web browser will launch and open the page that the link points to. We developed the JSP files that introduce new features to the Web pages, allowing the user to rate the displayed item explicitly and recommend the displayed item to other users.

Figure 5.2 shows the customized JSP page. The page is comprised of two frames. The top frame displays the original Web page related to the item. The bottom frame displays the user-specific information: a greeting to the user, the user's rating for the item, and the user's available recommending users list.

If the user already rated the item, the option corresponding the rating would be pre-selected when the page is displayed. The user can change the rating by selecting a

different option and pressing the button “Rate It”. The rating is stored in database table ratings that has three columns: userID, itemID and rating [Appendix A.5]. The Web link of the item is stored as the itemID.

Figure 5.2 User Clicking On a Delivered Item

The users listed in the drop down box are the available users that the viewing user can recommend an item to. The list contains all users other than the ones blocked by the viewing user no matter they are online or not. The list is constructed on the fly each time the JSP page is displayed. It is created based on the trust weight at which degree one user trusts another user. The data is stored in database table buddies that has userID1, userID2 and trustWeight columns [Appendix A.1]. When a user browses an item and find that it

may be helpful for a peer user, she can recommend the item to the peer user through this page. A record is inserted into the database table recommendations that has userID1 for recommender, userID2 for the recommended, and itemID [Appendix A.4]. The table is checked regularly by all user agents for new recommendations to their owner users. The recommendation checking process was discussed in section 4.5.8.

The database access operations are accomplished by invoking methods of a Java Bean that is developed to fulfil a series of database manipulations with the help of the DBManager.

Another important point is that we need to customize the links stored in each item element in the KlipFood file to direct the user to the desired JSP page when the item is clicked. This is accomplished by formatting the link in a KlipFood in the following manner:

```
<item>
  <description>Recommend Level: *4* Abstract: We describe the use of atomic force
  microscopy (AFM) in studies of...
</description>
  <link>http://localhost:8080/cf/testLatestPapers.jsp?userID=Mary&itemID=http://
  www.pubmedcentral.nih.gov/articlerender.fcgi?artid=315478
</link>
  <title>Force and Compliance Measurements on Living Cells Using Atomic Force
  Microscopy (AFM)
</title>
</item>
```

Examining the link node of the above item element, we can see the text of a link is composed of three parts. The first part is “http://localhost:8080/cf/testLatestPapers.jsp?”, the link to run the JSP page, which is same for every item for every user. The second part

is the userID passed to the JSP page, which provides the parameter to customize the bottom frame. The last part is the itemID, which is the actual link for the item to display in the top frame. The first two portions are added as a prefix during the process of updating the KlipFood DOM tree with the Update DOM tree described in the previous section. In the current prototype version of the system, the link for each item is represented by a plain unencrypted form for simplicity. In future development, it can be constructed in an encrypted form using an encryption algorithm if necessary.

CHAPTER 6 CONCLUSIONS AND OUTLOOK

6.1 Conclusions

The agent-based approach appears to be a suitable solution to overcome the information overload problem. Information filtering techniques play a key role in filtering increasingly vast amounts of information available on the Web. Particularly, the opinion-based filtering method is well suited for applying agent technology to the information filtering domain. Our approach to designing agent-based systems is based on a pattern-driven design methodology.

In this thesis, we have developed a multi-agent system to meet the challenge of Web information acquisition and dissemination. The design of the system was guided by the agent pattern approach. Content-based and opinion-based filtering techniques are applied to improve the relevance of the items. The key benefits of the system are:

(1) Information that might be of interest to the user is gathered from a wide variety of Web resources, integrated into a structured format, and presented to the user via a unified interface in a timely manner.

(2) Interesting information (that is, relevant to the user) is pulled from the Web and pushed onto the user's desktop. This saves the user effort and time otherwise spent searching, browsing and filtering for relevant information.

(3) Information gathered is filtered by the user's interest profile (using content-based

filtering) to ensure it is relevant to the user, and reviewed by peers (using opinion-based filtering), thus improving the quality and relevance of the delivered information.

(4) A user agent develops and dynamically maintains the profile of its user by observing the user's behaviour, and extracting the user's preference and interest from it. Therefore, the items that are extracted, reviewed, and ultimately presented to the user best suit the user's profile.

(5) The system is designed and implemented in a way that encourages expansion so that integrating new and modifying the use of existing data sources as well as introducing user agents for new users will be straightforward.

6.2 Future Work

Future work should include the following aspects:

(1) Empirical evaluation of the system with different types of users with different interest profiles. Performance testing of the of the system with real users needs to be carried out to see how well the system could actually perform the task of finding and presenting relevant information for users. One important aspect of the testing phase is to select a set of metrics for evaluating the performance of the system. These metrics should represent overall user satisfaction from using the system.

(2) Research on finding a better approach to develop more robust extraction agent for dealing with the ever-changing nature of Web sources.

(3) More information source-specific extraction capabilities need to be implemented so current extractor agents can handle a wide variety of information sources (e.g. search engines, newsletters, newsgroups, internal emails, etc.). For a system with a large number of users and/or Web information sources, a system with multiple cooperative extractor agents might be a better solution than the current solution.

(4) The patterns described in the current pattern language are just the beginnings of a pattern language for agent-based web information acquisition and dissemination system design. More patterns need to be created and added to the language to make it fledged pattern language.

(5) More functionality can be integrated into the graphic user interfaces. For example, we can add capabilities to the mediator agent GUI to display registered advertisements of provider agents, and monitor the mediator's current matchmaking and/or brokering activities.

REFERENCES

- [Acti03] SisyphSoft, *ActiveRefresh*, <http://www.activerefresh.com>, 2003.
- [Agen04] The Multiagent & Cooperative Robotics Lab, Department of Computing & Information Sciences, Kansas State University, *The agentTool Project*, <http://www.cis.ksu.edu/~sdeloach/ai/projects/agentTool/agentool.htm>, 2004.
- [Apac03] Apache Software Foundation, *Apache Regular Expression package*, <http://jakarta.apache.org/regexp/index.html>, 2003.
- [Baue00] M. Bauer, D. Dengler, and G. Paul, *Instructible Agents for Web Mining*, IUI2000, pp. 21-28, 2000.
- [Blok01] H. Blok, *Exchange rate calculator*, http://www.rememberjava.com/2001_nov_48.shtml, 2001.
- [Booc98] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1998.
- [Chng01] Benedict Chng, *Matchmaking with regular expressions - Use the power of regular expressions to ease text parsing and processing*, <http://www.javaworld.com/javaworld/jw-07-2001/jw-0713-regex.html>, 2001.
- [Deck97] K. Decker, K. Sycara and M. Williamson, *Middle-Agents for the Internet*, Proc. Of the 15th Int. Joint Conference on Artificial Intelligence, Aug. 1997.
- [Delg98] J. Delgado, N. ISHII and T. URA, *Intelligent Collaborative Information Retrieval Actively Learning to Classify and Recommend Documents*,

IBERAMIA, 1998.

- [DeLo00a] S. DeLoach, *AgentMom User's Manual*, 2000.
- [DeLo00b] S. DeLoach, *AgentTool User's Manual*, 2001.
- [DeLo01] S. DeLoach, M. Wood and C. Sparkman, *Multiagent Systems Engineering*,
The International Journal of Software Engineering and Knowledge
Engineering, Vol. 11 No. 3, pp.231-258, June 2001.
- [Deug01] D. Deugo, M. Weiss and E. Kendall, *Reusable Patterns for Agent
Coordination*, in: Omicini, A., *Coordination of Internet Agents*, Springer,
2001.
- [Door97] R. Doorenbos, O. Etzioni, D. Weld, *A Scalable Comparison-Shopping Agent
for the World-Wide Web*, Proceedings of the First International Conference
on Autonomous Agents (Agents'97), 1997.
- [Ecto04] Ectopic Brain, *What's New*, <http://pbrain.hypermart.net>, 2004.
- [FIPA02] FIPA, Agent Communication Language,
<http://www.fipa.org/repository/aclspecs.html>, 2002.
- [Gord00] G. Gu, *Agent Policy Patterns*, Master's Thesis, School of Computer Science,
Carleton University, 2000.
- [Homm03] S. Hommel, *Lesson: Regular Expressions*,
<http://java.sun.com/docs/books/tutorial/extra/regex>, 2003.
- [J2SE03] Java 2 Platform, Standard Edition (J2SE),

<http://java.sun.com/j2se/1.4.2/docs/api>, 2003.

- [Jenn98] N. Jennings, K. Sycara and M. Wooldridge, *A Roadmap of Agent Research and Development*, Autonomous Agents and Multi-Agent Systems, Vol.1, pp.7-38, 1998.
- [Jenn01] N. Jennings, *An agent-based approach for building complex software systems*, Communications of the ACM, Vol.44, No.4 pp.35-41, 2001.
- [Karm98] Ahmed Karmouch, *Mobile Software Agents for Telecommunications*, IEEE Communications Magazine, Vol.36, No.7, pp.24-25, July 1998.
- [Klus01] M. Klusch, *Information Agent Technology for the Internet: A Survey*, Journal on Data and Knowledge Engineering, Vol.36 No.3 pp.337-372, Mar. 2001.
- [Knob98] C. Knoblock, S. Minton, J. Ambite, N. Ashish, P. Modi, I. Muslea, A. Philpot and S. Tejada, *Modeling Web Sources for Information Integration*, Proc. Fifteenth National Conference on Artificial Intelligence, 1998.
- [Koub03] M. Koubarakis, C. Tryfonopoulos, S. Idreos, and Y. Drougas, *Selective information dissemination in P2P networks: problems and solutions*, ACM SIGMOD Record archive, Vol. 32, No. 3, pp.71-76, September 2003.
- [Lash94] Y. Lashkari, M. Metral, and P. Maes, *Collaborative Interface Agents*, Proceedings of the 12th National Conference on Artificial Intelligence 1, Seattle, WA, AAAI Press, pp.444-449, 1994.
- [Leah99] F. Leahy, *Extending IIS Page 5 - The Stock Grabber*

http://hotwired.lycos.com/webmonkey/99/29/index2a_page5.html?tw=programming, 1999.

- [Lieb97] H. Lieberman, *Autonomous Interface Agents*, Proceedings of the ACM Conference on Computers and Human Interface, CHI-97, 1997.
- [Lind00] J. Lind, *MASSIVE: Software Engineering for Multiagent Systems*, Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000), 2000.
- [Maes94] P. Maes, *Agents that Reduce Work and Information Overload*, Communications of the ACM Vol.37, No.7, pp.31-40, 1994.
- [Medi04] <http://www.medicinenet.com/rss/article.htm>, 2004.
- [Mont02] M. Montaner, B. Lopez and J.L. de la Rosa, *Opinion-based Filtering through Trust*, CIA'02, Madrid, Spain, 2002.
- [Mont03] M. Montaner, B. Lopez, J.L. de la Rosa, *A Taxonomy of Personalized Agents on the Internet*, Artificial Intelligence Review, Vol. 19, pp.285-330, June 2003.
- [Mouk98] A. Moukas and P. Maes, *Amalthaea: An Evolving Multi-Agent Information Filtering and Discovery System for the WWW*, Autonomous Agents and Multi-Agent Systems, Vol. 1, pp.59-88, 1998.
- [Musl01] I. Muslea, S. Minton AND C. Knoblock, *Hierarchical Wrapper Induction for Semistructured Information Sources*, Autonomous Agents and

Multi-Agent Systems, Vol. 4, pp.93-114, 2001.

- [Myl100] J. Myllymaki, J. Jackson, *Web-based data mining - Automatically extract information with HTML, XML, and Java*,
<http://www-106.ibm.com/developerworks/web/library/wa-wbdlm>, 2000.
- [MySQ04] MySQL 4.0, <http://www.mysql.com/downloads/mysql-4.0.html>, 2004.
- [Nwan96] H. Nwana, *Software Agents: An Overview*, Knowledge Engineering Review, Vol.11, No.3, pp.205-244, Oct./Nov. 1996.
- [Ogst01] E. Ogston and S. Vassiliadis, *Matchmaking Among Minimal Agents Without a Facilitator*, Proceedings of the Fifth International Conference on Autonomous Agents, 2001.
- [Pant02] G. Pant and F. Menczer, *MySpiders: Evolve Your Own Intelligent Web Crawlers*, Autonomous Agents and Multi-Agent Systems, Vol.5, pp.221-229, 2002.
- [Pazz98] M. Pazzani, J. Muramatsu and D. Billsus, *Syskill & Webert: Identifying Interesting Web Sites*, AAAI/IAAI, Vol. 1, 1998.
- [Pazz02] M. Pazzani and D. Billsus, *Adaptive Web Site Agents*, Autonomous Agents and Multi-Agent Systems, Vol. 5, pp.205-218, 2002.
- [PubM04] PubMed Central (PMC), <http://www.pubmedcentral.nih.gov>, 2004.
- [Purp01] RSS Headline Syndication, <http://www.purplepages.ie/rss>, 2001.
- [RegE04] Using Regular Expressions in Java,

- <http://www.regular-expressions.info/java.html>, 2004.
- [RssR04] RssReader, <http://www.rssreader.com>, 2004.
- [Salt94] G. Salton, J. Allan and C. Buckley, *Automatic Structuring and Retrieval of Large Text Files*, Communications of the ACM Vol.37, No.2, pp.97-108, Feb. 1994.
- [Sere04] <http://www.serence.com>, 2004.
- [Sere03a] Serence Inc., *Klip Development Kit*,
http://www.serence.com/provider.php?page=dl_home, Dec. 2003.
- [Sere03b] Serence Inc., *Serence Project Proposal*, Jan. 2003.
- [Seva02] H. Sevay, and C. Tsatsoulis, *Agent-based intelligent information dissemination in dynamically changing environments*, Intelligent agents and their applications, pp.1-26, Physica-Verlag GmbH Heidelberg, Germany 2002.
- [Shoh97] Y. Shoham, *An Overview of Agent-Oriented Programming*, Software Agents. AAAI Press/The MIT Press, pp.271 - 290, 1997.
- [Sing95] A. Singhal, G. Salton, *Automatic Text Browsing Using Vector Space Model*, Proceedings of the Dual-Use Technologies and Applications Conference, pp.318-324, 1995.
- [SPHI02] *WebSPHINX: A Personal, Customizable Web Crawler*,
<http://www-2.cs.cmu.edu/~rcm/websphinx>, 2002.

- [Syc96] K. Sycara, K. Decker, A. Pannu, M. Williamson and D. Zeng, *Distributed Intelligent Agents*, IEEE Expert, Vol.11, No.6, pp.36-46, 1996.
- [Syc97] K. Sycara, K. Decker and D. Zeng, *Intelligent Agents in Portfolio Management*, Agent Technology: Foundations, Applications, and Markets, 1997.
- [Syc98] K. Sycara, *MultiAgent Systems*, AI Magazine Vol.19, No.2, 1998.
- [Tidy04] JTidy, <http://jtidy.sourceforge.net/index.html>, 2004.
- [UMBC04] UMBC, Knowledge Query and Manipulation Language, <http://www.cs.umbc.edu/kqml>, 2004.
- [W3C04] W3C, *Semantic Web*, <http://www.w3.org/2001/sw>, 2004.
- [W3Sc04] W3Schools, *XML Tutorial*, http://www.w3schools.com/xml/xml_what_is.asp, 2004.
- [Weis98] M. Weiss, *A Gentle Introduction to Agents*, Carleton Computer Science Conference (COSACS), 1998.
- [Weis01] M. Weiss, *Patterns for e-Commerce Agent Architectures: Using Agents as Delegates*, Pattern Languages of Programming (PloP), 2001.
- [Weis02] M. Weiss, *On the Use of Patterns for Agent System Design*, Agent-Oriented Information Systems (AOIS@AAMAS'02), 2002.
- [Weis03a] M. Weiss, *Electronic Commerce Technologies course notes*, 2003.
- [Weis03b] M. Weiss, *Patterns for Motivating an Agent-Based Approach*,

Agent-Oriented Information Systems (AOIS@ER'03), 2003.

- [Weis03c] M. Weiss, *Pattern-Driven Design of Agent Systems: Approach and Case Study*, Conference on Advanced Information Systems Engineering (CAISE), LNCS 2681, Springer, 2003.
- [Weis03d] M. Weiss, *Patterns for Motivating an Agent-Based Approach*, Agent-Oriented Information Systems (AOIS@ER'03), 2003.
- [Weis03e] M. Weiss, *A Pattern Language for Motivating the Use of Agents*, special LNCS on the AOIS 2003 workshop, 2003.
- [Wool97a] M. Wooldridge and P.Ciancarini, *Agent-Oriented Software Engineering: The State of the Art*, Springer-Verlag Lecture Notes in AI Volume 1957, 2001.
- [Wool97b] M. Wooldridge, *Agent-based Software Engineering*, In IEE Proceedings on Software Engineering, Vol.144, No.1, pp.26-37, Feb. 1997.
- [Wool00] M. Wooldridge, N. Jennings and D. Kinny, *The Gaia Methodology for Agent-Oriented Analysis and Design*, Journal of Autonomous Agents and Multi-Agent Systems, Vol.3, No.3, pp.285-312, 2000.
- [XPBa04] XPBargains, <http://www.xpbargains.com/index.php?link=35>, 2004.
- [Zamb01] F. Zambonelli, N. Jennings and M. Wooldridge, *Organisational rules as an abstraction for the analysis and design of multi-agent systems*, Int J. of Software Engineering and Knowledge Engineering, Vol.11, No.3, pp.303-328, 2001.

APPENDIX A DATABASE SCHEMA

The database system used in the project is MySQL 4.0 [MySQ04]. Following are the tables' schema definitions for the project.

A.1 TABLE buddies

Table buddies stores the data about one user's trust weights against other users. The trust weights values range between -1 and 1 . -1 means user1 (identified by userID1) trusts user2 (identified by userID2) the least, 1 means the most and 0 means that the two users are not relevant.

```
--  
-- Table structure for table 'buddies'  
--  
CREATE TABLE 'buddies' (  
  'userID1' varchar(100) NOT NULL default '',  
  'userID2' varchar(100) NOT NULL default '',  
  'trustWeight' double(10,6) NOT NULL default '0.000000',  
  PRIMARY KEY  ('userID1','userID2')  
) TYPE=MyISAM;
```

A.2 TABLE datasource

Table datasource stores the data about data source names and their respective Web sites. Column isExtractingRuleDone indicates whether the extracting rule for the data source is available.

```
--  
-- Table structure for table 'datasource'  
--
```



```
CREATE TABLE 'datasource' (
  'dataSrcName' varchar(255) NOT NULL default '',
  'dataSrcWebsite' varchar(255) NOT NULL default '',
  'isExtractingRuleDone' varchar(6) default 'n',
  PRIMARY KEY ('dataSrcName')
) TYPE=MyISAM;
```

A.3 TABLE itemsource

Table itemsource stores the data about itemIDs identified by links and their data sources which is represented as a string.

```
--
-- Table structure for table 'itemsource'
--
CREATE TABLE 'itemsource' (
  'itemID' varchar(255) NOT NULL default '',
  'dataSrcName' varchar(255) NOT NULL default '',
  PRIMARY KEY ('itemID')
) TYPE=MyISAM;
```

A.4 TABLE recommendations

Table recommendations stores the users' recommendations to other users, in which itemIDs are identified by links. The status of whether a recommendation is checked is represented by the value of column isChecked.

```
-- Table structure for table 'recommendations'
--
CREATE TABLE 'recommendations' (
  'userID1' varchar(100) NOT NULL default '',
  'userID2' varchar(100) NOT NULL default '',
  'itemID' varchar(200) NOT NULL default '',
  'isChecked' varchar(6) NOT NULL default 'n'
) TYPE=MyISAM;
```

A.5 TABLE testratings

Table testratings stores the data about one user's explicit ratings on items.

```
--  
-- Table structure for table 'testratings'  
--  
CREATE TABLE 'testratings' (  
  'userID' varchar(100) NOT NULL default '',  
  'itemID' varchar(100) NOT NULL default '',  
  'rating' double(10,6) NOT NULL default '0.000000'  
) TYPE=MyISAM;
```

A.6 TABLE users

Table users stores the data about one users' personal information, including userID, password, host and port where the userAgent is running (port = 0 and host = '' means offline), the time interval set between consecutive subscription refresh. Column isServiceAgent is used to distinguish whether the agent is a service agent because information of all agents (user agents as well as service agents) is stored in this table.

```
--  
-- Table structure for table 'users'  
--  
CREATE TABLE 'users' (  
  'userNo' int(11) NOT NULL auto_increment,  
  'userID' varchar(100) NOT NULL default '',  
  'password' varchar(100) NOT NULL default '',  
  'host' varchar(255) NOT NULL default '',  
  'port' int(11) default '0',  
  'isServiceAgent' int(2) default '0',  
  'refreshIntervalInHour' float(10,6) default '1.000000',  
  PRIMARY KEY ('userNo', 'userID')  
) TYPE=MyISAM;
```

A.7 TABLE usersubscription

Table usersubscription stores the data about one users' subscription identified by subscribed data source names and keywords for each of the data sources.

```
--  
-- Table structure for table 'usersubscription'  
--  
CREATE TABLE 'usersubscription' (  
  'userID' varchar(100) NOT NULL default '',  
  'dataSrcName' varchar(255) NOT NULL default '',  
  'keyword' varchar(100) NOT NULL default '',  
  PRIMARY KEY ('userID','dataSrcName')  
) TYPE=MyISAM;
```

APPENDIX B PATTERN DEFINITION FOR EXTRACTOR AGENT AND NOTIFIER AGENT

The definition of agent pattern follows the format described in [Weis03e]. The patterns of the pattern language have the following structure: pattern name, context, problem, forces, solution and resulting context.

B.1 Extractor Agent

Context

Your application needs to extract interesting information from various heterogeneous Web data sources. You need an agent to get commands from other agents that request the extracting service and return the results in an agreed format.

Problem

How do you instruct the agent where to extract information and what to extract? In what format should the agent return the extracted results?

Forces

- Great diversity of Web data sources in terms of structure, data format, and presentation style.

Solution

An extractor agent can locate interesting Web data repositories, apply source-specific extracting rules on them to extract information that might be of interest to

users, and wrap it in a unified format. With user profile provided by the requesting agents the extractor agent can match the Web data against the profile and extract only the relevant information. RSS, a well-established XML format can be used as a means to format extracted results.

For a simple system involving a small number of users and/or Web information sources, a single extractor agent should be enough for request processing. When the number of users and/or Web information sources increases, multiple cooperative extractor agents might be a better solution.

Resulting Context

- For extraction requester and an extractor agent to understand each other, they need to agree on a common exchange format, as described in Common Vocabulary.
- Extraction requester and an extractor agent find each other through Agent As Mediator.

B.2 Notifier Agent

Context

You need an agent to get commands from user agents and notify their users of updates by modifying contents that will be presented to the users.

Problem

What information should a requesting user agent send to a notifier agent? How does

the notifier agent understand the information passed to it and converts it into the required format for notification?

Forces

- Autonomy

Solution

The notifier agent understands and translates the information passed from user agents, extracts and wraps interesting contents into the formats required for presentation and updates the file that enables presentation. For example, the notifier agent may convert a news item wrapped in RSS, make appropriate modifications (e.g. constructing http address based on the information passed to it), and save it as a KlipFood (could be a new RSS) file. The requesting user agents need to send to the extractor agent the information that required for above processing in an agreed format.

Resulting Context

- For user agents and the notifier agent to understand each other, they need to agree on a common exchange format, as described in Common Vocabulary.
- User agents and the notifier agent find each other through Agent As Mediator.