1. Introduction

1.1 Motivation

Writing articles is an important part of work for a researcher at a university or a content provider working for a media company. While writing research or news articles, the author often conducts searches on the World Wide Web (WWW) and dig out relevant information that can be used to write the article. However, when an author is writing an article, it is often a distraction to stop writing, visit a few search engines, conduct keyword searches, retrieve relevant information and then use them in the article. This is not a very efficient especially when the author in writing an article under a deadline.

The motivation of the Multi-Agent Research Tool (MART) is to develop a research tool that helps the author to research while writing an article and save valuable time. This means that the research tool should not only be smart and efficient in conducting searches, it should also be able to work in the background fairly independently and at the same time be non-intrusive to the user. Moreover, since the Internet has now become a very common commodity and it is assumed that a person using MART has access to the Internet, it would be further more useful if the research tool could use distributed computing to retrieve research material and present them to the user whenever he/she decides to view or use them. MART is intended to be such a tool.

1.3 Multi-Agent Systems

Multi-Agent Systems are one of the recent programming paradigms used to build intelligent software agents. An intelligent agent can be defined as some object that is fairly autonomous, goal driven, pro-active and reactive. This means that intelligent agents should be able to “inhabit” an environment and not only should they be capable of responding to external stimuli, they should be fairly independent and have an end result in their “mind”, while going about doing small set of activities.

There are several methodologies for building Multi-Agent systems such as Gaia, Societies in Open and Distributed Agent (SODA), Methodology for Engineering Systems of Software Agents (MESSAGE) and Multi-Agent Software Engineering (MaSE).
MART will be built in the multi-agent programming paradigm and it will use MaSE as the Agent Oriented System Engineering (AOSE) methodology to construct the system.

1.4 Multi-Agent Software Engineering (MaSE)

Of all the Agent Oriented Systems Engineering (AOSE) methodologies, probably the simplest, the most logical and the most elaborate is the Multi-agent Systems Engineering (MaSE) methodology.

Like most of the other methodologies, the MaSE methodology also has an analysis phase and a design phase. The analysis phase yields goals, use cases and roles of the overall system while the design phase results in the development of agent classes, tasks and conversations. To build an agent’s system under the MaSE methodology, a developer must use the agentTool that not only helps in designing the system but also generates the stub code of the entire system in Java.

Under MaSE, the first step in the analysis phase is to break down the functional requirements of the overall system into a set of hierarchical goals. These goals are primarily of four types – summary, partitioned, combined and non-functional. Once the goals of the system have been identified, the next step in the analysis phase is to apply use cases and to develop sequence diagrams. At this state the aim is to develop as many event sequences of the system as possible without repeating any events.

The last step in the analysis phase is to develop roles and refine them from the goals and the sequence diagrams of the first two steps. Usually, the mapping of goals to roles is one-to-one, in the sense that one goal maps to one role. However, closely related goals may be combined into a single role. It is ultimately the overall system requirements and its characteristics that decide the goals, use cases and roles. After the roles are created, tasks associated with each role that describe the role’s behavior and that are required to achieve the goals associated are developed. The role diagram is the first step in MaSE which defines the interactions between different tasks of different roles. Here the tasks are also developed in the form of state diagrams. These tasks are either transient or persistent. These individual tasks are then developed into their concurrent task models which have an initial state, a termination state and transitions between the two and/or other states.

The design phase in MaSE starts after all the roles and tasks have been defined. Once this is done the first step in the design phase is now to develop agents by assigning them roles. Since the roles reflect the overall goals of the system and can be manipulated modularly, the authors refer to them as a bridge.
between the analysis and the design phase of this methodology. It is up to the system designer on how
many agents finally show up in the overall system and what roles each agents play. Once the agents have
been defined, the designer should now connect the agents using conversations. Each conversation is
designed from both the initiator as well as the responder’s perspective. These conversations are much like
the state diagrams that have initial and end states. A designer can also develop other states inside the
responder and the initiator keeping in mind the previous tasks and roles and finally verify to see whether
the conversations are deadlock free or not. The verification is done by the Spin tool which is embedded
inside the agentTool.

Another advantage of using MaSE is that while designing the individual agents, it is totally up to
the designer on what architecture to use for making the internal components of the agents. It is here that
Beliefs, Desires and Intentions (BDI) architecture can be used to design the agent architecture or the
designer can choose his/her own architecture and make the components accordingly.

Finally the structure of an instance of the entire system can be developed using the deployment
model in which several systems are designed, the agents are placed in the systems from the list of agents
and the agents are made to talk using conversations already developed from the previous step.
2. Project Overview

Figure 1. Overview of MART

2.1 Introduction

The MSE project involves building the Multi-Agent Research Tool (MART) software that will help researchers and authors to dynamically search for information and write articles. Put simply, MART is intended to be a software package that will be integrated with one of the most commonly used word processor program, MS Word® and help the user to conduct relevant searches in the background based on the words that have been typed in the article and then display the results to the user.

The primary goal of creating MART is develop a smart Multi-agent system that would intuitively and independently search and display information about an article while it is being typed by a researcher.

The main feature of the software is to do relevant searches and display the results dynamically even while the user is typing the article and the real challenge is to find results that are relevant to the article being written and are not redundant.
2.2 Goal

To develop a tool that will be integrated with MS Word and help an author in searching research information for an article based on the words that have already been typed in an article. The search results should be fairly accurate, relevant to the current article being typed and non-redundant.

2.3 Purpose

To build a multi-agent system with a collection of intelligent software agents that are fairly autonomous and use distributed computing to achieve a common goal.

2.4 Features

1. Integrated with MS Word.

2. There will be separate agents responsible for specific tasks. For example, one agent should be a specialist only in conducting searches in Google.

3. The search should be conducted only when the user want to use MART.

4. Allow the user to choose when he/she wants to view the results.
3. Requirement Specification

Figure 2. Overview of the MART structure

3.1 Main requirements

1. **Multi-Agent Systems based** architecture. Specific functions will be performed by individual agents that are fairly independent, goal driven, proactive and reactive.

2. Use of **Multi-Agent Software Engineering (MaSE) methodology and agentTool** in developing the system.

3. The final product, MART should be integrated with MS Word ® processor.

4. The individual agents should be able to **run on any machine** that has a Java virtual machine. This includes machines running on the following platforms – Windows, Linux, Macs.

5. Internet Explorer ®, a common web browser used by Windows users, will be used to display the search results.
6. For testing and demonstration purposes, the software should use the **Google API** to conduct searches on Google search engine.

7. Failure of one agent should not crash the whole software.

### 3.2 Specific Requirements

1. Allow a user to write an article in MS Word ®.

2. The user interface would include buttons embedded inside MS Word ® which will allow the user to search, save and display the results.

3. The program will generate HTML 4.0 result pages so that they can be viewed using a common web browser - Internet Explorer ®.

### 3.3 Critical Use Cases

![System (MS Word and MART) Diagram]

**Figure 3. Critical Use Case**
3.3.1 Use Case 1: Type article in MS Word

Description - The user should be able to type an article in MS Word with his/her normal settings and preferences.

Specific requirement -

3.3.1.1 Features - When MART is finally integrated with MS Word, none of the features, preferences and settings should either change or stop working.

3.3.1.2 Save - The user should also be able to save the articles as any other regular files.

3.3.2 Use Case 2: Search for research material

Description – The user should be able to click on a button somewhere in the MS Word menu that should trigger the MART search agents.

Specific requirements –

3.3.2.1 Filter – Once the search button is pressed, all the irrelevant words like “a”, “the,” “an”, etc should be removed and a list of keywords should be prepared for searching database.

3.3.2.2 Relevance – The list of keywords should be ranked according to the number of times each word has been used in the typed article.

3.3.2.3 Search – Using the list of relevant keywords, one Agent should crawl on an online search engine, conduct a search and return with html content that should be saved as a file named as results.html in My Documents folder of the user’s computer.

3.3.2.4 Tailor display – The content in results.html should be re-arranged by inserting appropriate html tags so that only the first 10 relevant links are shown in the web page. Clicking on these links should take the user directly to the information source.

3.3.3 Use Case 3: Display research material

Description – The user should be able to click on a button somewhere in the MS Word menu that should immediately open a window of Internet Explorer® browser that will have all links to web sites with information relevant to the current article being typed in MS Word.

Specific requirements –
3.3.3.1 Display – The results.html file should be opened in Internet Explorer browser.

3.4 Assumptions

1. The user has an internet connection whenever he/she is using MART.
2. Every agent knows the port numbers of the every other agent that they need to collaborate with.

3.5 Constraints

1. Since the Agent Tool generates stub code in Java, the program will be developed in Java.
2. The Google API only returns the first 10 search results per query.
3. There is a limit to the number of searches a program can conduct every day using the Google API.

3.6 Environment

1. The software package will be compiled using the JDK 1.4.0
2. agentTool will be used for the design and analysis part of building the system and to generate the stub code.
3. IBM Visual Age will be used for coding the software package.
4. USE 2.0 along with Object Constraint Language (OCL) will be used for formal specifications.
5. The software package will be tested on Windows XP and the individual agents will be tested on Windows XP, Red Hat Linux 9.0 and Unix Solaris 2.7.