Architecture Design

for Multi-Agent Research Tool (MART)

Version 1.0

Submitted in partial fulfillment of the requirements for the degree MSE

Madhukar Kumar

CIS 895 – MSE Project

Kansas State University
1 Introduction

The purpose of the document is to describe the architecture design of the project Multi-Agent Research Tool (MART) as outlined in the requirements in section 3 of the vision document. The document will outline and describe the goals, the role model, the object diagram and the sequence diagrams of MART.

2. Goals

The overall goal of the system is to produce searched results. This goal is partitioned into four sub goals, mainly to rank and refine the keywords that will be used for search, searching the web for results, producing the result (output file) for the user and to manage the whole distributed system.

![Figure 1 – Goals of the system](image)

The “rank and refine search keywords” role is also partitioned and is divided into reading user inputted keywords and in ranking the keywords for search.

The “search” role is also partitioned and for the current version consists only of searching websites. In future versions, another role could be added into it that
would also conduct searches on database located on the host computer and/or local network.

The role “produce results” is partitioned into creating raw results, refining the results and producing the final result.

3. Use Cases

There are three primary use cases for the entire system namely “Read User Preferences,” “Search the Web” and “Generate Results.”

3.1 Read User Preferences

In this sequence diagram, the manager sends a message to the Keyword reader to read the user inputted keywords and preferences. The manager then takes this object and sends it to the Keyword ranker with a request to rank all the keywords. The Keywords ranker ranks the keywords based on the number of times they appear in the user typed article and returns the ranked keywords to the manager.
3.2 Search the Web

The following sequence diagram shows that the manager sends a message to the Sleuth, along with the keywords and requests it search the web. The sleuth conducts the search and returns the raw result to the manager.

Figure 3 – Search the Web
3.3 Generate results

The following diagram shows that the Manager, once it has the raw results, sends a message to the Result producer along with the raw results. The producer refines the results and then sends back the final result to the manager.

![Diagram](image)

Figure 4 – Generate results
4. Role diagram

The role diagram depicts how the different goals are mapped to the roles of the system. The following diagram shows that the controller has a bunch of tasks that collaborate with the other roles in order to read keywords, perform a search and generate the final result.

The dotted line in the Sleuth role (makeRaw) shows that it is an internal method call. The makeRawResults task is invoked by the rawResults task of the Sleuth once it receives the searchTheWeb request from the controller.

![Role Diagram](image-url)
5. Agent Template diagram

The agent template diagram shows that the current version of the software (MART) has five different agents namely AgentManager, AgentKey, AgentProducer, AgentGoogle and AgentTeoma. The red lines connecting between the agents represent the conversations that are initiated between the agents. For example searchTheWeb is a conversation that is initiated by AgentManager and it requests the sleuth agents (AgentGoogle and AgentTeoma) to conduct a search and return the raw results.

The different agents in the following diagram represent independent entities with their own separate main methods. These agents could be placed on different machines and still be able to talk to each other using the conversations defined in the system.

![Agent Template diagram](image-url)
6. Deployment diagram

The following diagram shows that since the agents are independent entities they can be placed on different systems (machines or networks) and yet talk to each other using pre-defined conversations.

The following diagram also shows that for the current version of the software the AgentManager and AgentKey will be on the same machine as the one which has the Microsoft Word running on it. This is because both the AgentManager and the AgentKey need to have access to the file that the user is typing the article on.

![Deployment diagram](image)

Figure 7 – Deployment diagram
Figure 8 – Classes generated after transformation (with components)
6. Object Model

When AgentTool generated code using “newAgentMom”, it made separate classes for each component. The object model for the code generated is shown in Figure 13. However, since it is simpler to have components as separate methods in the agents, the object model was redesigned as shown Figure 14.

Figure 9 – Object diagram
The object model shows each and every class of the software and the associations between them. The packages, “afit” and “KeyObjectPackage” are accessible to all the other packages in the system. The KeyObjectPackage hold the classes that store the keywords and the final result of the search. The primary packages (representing the agents and their component classes) are AgentKeyPackage, ManagerPackage, GooglePackage, TeomaPackage and ProducerPackage. Figure 15 shows the interaction between all the packages. ManagerPackage holds the AgentManager class which is the main controller of the whole system and initiates all the conversations with other Agents.

![Component diagram of the entire system](image)

Figure 10 – Component diagram of the entire system
**KeyObjectPackage**

The KeyObjectPackage is the data package that is used by all other packages to send results and keywords to each other. Class KeyObject is a simple class with three main attributes called name, rank and nooftime. The actual keyword is stored in the variable name while the integers rank and nooftime record the ranking of the keyword as compared to the other keywords and the frequency with which it appears in the typed article respectively.

The RawResult class is also a simple object that holds XML(html) string in its main attribute called “result.”

![Diagram of KeyObjectPackage](image.png)

*Figure 11 – The KeyObjectPackage*
AgentKeyPackage

AgentKeyPackage has three classes, the AgentKey, rankKeywords_AgentKey_R and readUserPreferences_AgentKey_R. The AgentKey is the main class of the package and has the main method as well as methods to read user keywords and rank keywords. When AgentKey receives a message from AgentManager (it does not respond to message from any other class), it creates the instances of conversation responder class based on the kind of message. The responder classes then invoke the methods that actually reads user inputted keywords and ranks them.

![Diagram of AgentKey package]

Figure 12 – AgentKey package
GooglePackage

GooglePackage has two classes AgentGoogle and searchTheWeb_AgentGoogle_R. AgentGoogle has one important method called search. When AgentGoogle receives a message from AgentManager, it creates an instance of the searchTheWeb_AgentGoogle_R that handles the conversation from there on. The conversation is closed once AgentGoogle conducts the search, serializes the result and sends back the serialized object (in the form of an instance of RawResult class in KeyObjectPackage) to the AgentManager.

![Diagram of GooglePackage classes and methods]

Figure 13 –GooglePackage
TeomaPackage

TeomaPackage is similar to the GooglePackage and consists of the following two classes AgentTeoma and searchTheWeb_AgentTeoma_R. These two classes perform the same functions like the classes in the GooglePackage. The main difference between the two packages lie in the way AgentGoogle and AgentTeoma conduct searches. AgentGoogle uses Google API to conduct keyword search on Google.com while AgentTeoma opens a URL connection with the website and gets back raw results.

![Diagram of TeomaPackage](image)

Figure 14 – TeomaPackage
ProducerPackage

ProducerPackage also has two classes called AgentProducer and generateResults_AgentProducer_R. The AgentProducer has a method called generateResult which creates the final HTML file and returns it to the AgentManager. The responder class is used for conversation with the AgentManager.

Figure 15 – ProducerPackage
ManagerPackage

ManagerPackage which acts as the controller for the whole system has several classes namely AgentManager, readUserPreferences_AgentManager_I, rankKeywords_AgentManager_I, searchTheWeb_AgentManager_I and generateResults_AgentManager_I.

The AgentManager creates instances of the conversation initiator classes when it has to get some job done from the other agents in the system. At first it initiates a conversation with the KeyPackage and gets back the keywords typed in the user and also the search preferences. It then initiates a conversation with the GooglePackage and TeomaPackage and requests them to conduct searches based on the keywords and preferences and return the raw results. Once the raw results arrive, the ManagerPackage calls the ProducerPackage and requests it create a final result in HTML based on all the raw results. The final result is then placed by the Manager in the same computer where the user is typing the article.
7. Formal Requirement Specification

A model to test and formally specify a primary property of the MART system was developed. The USE model has four different classes, the most important being the AgentManager. The AgentManager represents the AgentManager class from the system and has access to both the keywords that are generated from the user typed article and the results file created after a search has been conducted.

The KeyObject class in the USE model represents the KeyObject class of the KeyObjectPackage in the Object diagram. The class has three attributes – name, rank and nooftime. These attributes store the keyword itself, the number of times the word appears in the user typed article and a corresponding rank based in comparison to the other keywords.

The RawResult class in the USE model represents the RawResult class in the KeyObjectPackage of the object diagram. In actual implementation the entire result is stored as a string in the RawResult class. However, to show the property of the entire system that some of the original keywords are also present in the result, the USE model has ResultWord classes that are attached to the RawResult class. The ResultWord class represents each and every result words (String) that will be present in the RawResult class in actual implementation.

7.1 Constraints

The first constraint “keysalsoinresult” specifies that the search results (RawResult) will have some of the keywords that exist in the keywords (KeyObjects).

The second constraint “rankings” specifies that if a keyword appears more number of times in the user typed article then its rank will be higher than keywords that appear lesser number of times in the typed article.
-- MART.use in mad-cis\mse\presentation2 folder
-- The model demonstrates the primary property of
-- MART project being built for the MSE project.
-- The property it tries to state and check is that some
-- of the keywords originally typed by the user should also be
-- present in the final results file
-- Author - Madhukar Kumar
-- Date: July 25, 2003

model MART
--
-- CLASSES --
--
-- The AgentManager agent is the main controller
-- of the entire system
class AgentManager
end

-- Every instance of class KeyObject signifies
-- a single keyword read from a the User typed article (from MS Word)
class KeyObject
attributes
  name : String
  rank : Integer
nooftime : Integer
end

class RawResult
end

-- Every instance of ResultWord represents single words that are found in the RawResult file created by the Sleuth agent. The RawResult file is created by the Sleuth agent after it has conducted the search. In actual implementation RawResults will be one single flat file with several words (String) in it.
class ResultWord
  attributes
    word : String
  end

-- ASSOCIATIONS --

-- Keys: Manager has several keys attached to it
association Keys between
  AgentManager[1] role manager
  KeyObject[*] role keys
end

-- ResultWords: RawResult has several ResultWord attached to it
association ResultWords between
  RawResult[1] role resultfile
  ResultWord[*] role resultword
end

-- Result: KeyManager is attached to several Result File because each sleuth agent generates one RawResult file
association Result between
  AgentManager[1] role keymanager
  RawResult[1] role resultfile
end

-- CONSTRAINTS --

-- Some of the keywords in the KeyManager are also present in the RawResults.
context km: AgentManager
  inv keysalsoinresult:
(km.keys.name->asSet)->intersection(km.resultfile.resultword.word->asSet)->notEmpty

-- If a word appears more number of times in the user inputted article
-- its ranking will be higher (1 is higher than 2) than a word which
-- appears lesser number of times in the article
context key : AgentManager
    inv rankings :
        key.keys->forAll( r,s : KeyObject | r.nooftime > s.nooftime 
implies r.rank < s.rank)