fund NHG5-2211

A REPORT TO

THE NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

ON

Research & Development for an Operational Information Ecology

The User-System Interface Agent Project

Submitted By

Drs. Sadanand Srivastava & James de Lamadrid
Center for Research in Distributed Computing
Department of Computer Science
Bowie State University
Bowie, MD 20715

July 1998

The User System Interface Agent (USIA) is a special type of software agent which acts as the "middle man" between a human user and an information processing environment. USIA consists of a group of cooperating agents which are responsible for assisting users in obtaining information processing services intuitively and efficiently. Some of the main features of USIA include: (1) multiple interaction modes and (2) user-specific and stereotype modeling and adaptation. This prototype system provides us with a development platform towards the realization of an operational information ecology. This has been an exciting experience for us, involving undergraduate/graduate students, and faculty members of Bowie State University; graduate student and faculty member from University of Maryland at Baltimore County (UMBC), scientists from Loral Aerospace (now Lockheed Martin) and key personnel from Goddard Space Flight Center (GSFC).

PROGRESS REPORT

YEAR-ONE IN SUMMARY:

In the first phase of this project we focused on the design and implementation of a prototype system of the User-System Interface Agent (USIA). This system is to serve as a front-end to an Agent-based Information Processing environment, which is being developed by Loral Systems Corporation, and to be tested in the Flight Control Systems domain. Our USIA prototype encompasses these features: Graphical, icon-based, user interface; Restricted natural language for dialog; Initial measures for user modeling; and Mapping of user requests to system tasks.

YEAR-TWO IN SUMMARY:

The second phase of USIA allows user interaction via a restricted query language as well as through a taxonomy of windows that have been customized for the Report Generation application in the Flight Control Systems domain. The main activities are design and development of the following features: the Controller (which handles all communications between the Request Processor and the user and the User Modeling and the system interface); the domain specific dictionary; Semantic Analyzer (which does the context analysis and communicates with controller and dictionary); User-specific Knowledge Bases (user profile, History etc.);Command Interpreter; Interface module to UIA and RAA ; and User classification.

YEAR-THREE IN SUMMARY:

In this phase, the USIA System architecture was revised to include the following new modules: Result Manager Agent (RMA): responsible for collecting results from the domain specialist agent manager for presentation to the user. Local Request Server (LRS): provides an array of utilities for user services to manage objects locally, e.g., Print, Email, Display, Save, Delete, and List. Consequently, the USIA System software to reflect the new design by incorporating and/or enhancing the following features; an integrated user interface which combines windows-based selections, natural language text, and limited animated graphics, hence, providing a hybrid medium for user-system interaction; a user classification scheme which allows the system to adapt dynamically to user behavior by capturing repeated errors and ambiguities made by a user and correcting them once a "pattern of usage behavior" has been identified; system adaptation is influenced by the classification of a user's expertise level, which is dynamically adjusted by the system; expanded the grammar and the domain dictionary to support more complex requests in the Report Generation domain and limited requests in the Monitoring domain; features which allow variations in time specification have been enhanced; developed utilities which provide users with tools to manage (result) objects in their local environment; upgraded the inter-process communication software, which utilizes socket communication, to provide more robust and efficient "dialog" between agent-based on the client-server model.

FUTURE PLANS

As described above. we have accomplished much and solved many problems, but as we solved some problems many related research issues need to be resolved. In a three year plan, we propose the following activities to be explored, and solved.

YEAR- ONE:

- Integrate the USIA prototype system into the AFLOAT system in the NASA/Goddard Space Flight environment.
- Make the Parser powerful and flexible enough to handle various requests which are in natural language by enhancing the Grammar. Make the generic dictionary versatile and design faster dictionary searching algorithms.
- Apply Object oriented Programming to design a new User Modeling Agent which can create user models dynamically, and group user models into appropriate classes and coordinate as well as manage them.
- Enhance the Learning mechanism of the UMA and problem solving algorithms with forward and backward chaining for user model classes to enhance the reasoning capability.
- Investigate and implement the feasibility of using Java Programming Language as a way of providing user friendly screens and implementing new technologies.
- Modify the Result Management Agent enabling it to assemble and manage results from different computers in a distributed environment .

YEAR- TWO:

- Continue to upgrade UIA by incorporating user friendly features, which will make using the USIA system more intuitive for the user. Also more aesthetically appealing features will be incorporated into the UIA. Animation will be provided to UIA.

- Make the parser powerful and flexible enough to handle various types of queries entered by the user in the natural language. This will be enabled by using algorithms which will result in various ways of sub-string pattern matching. Adding to the parser, the capability to parse conjunctions.

- Make the generic dictionary versatile and design faster dictionary search algorithms.

- Enhance the learning mechanism of the UMA, by applying instance based, inductive and analytical learning.

- Modify the Result Management Agent by enabling it to assemble and manage results from different computers in a distributed environment.

- Add Voice Interface Agent which would provide a two-way voice communication between the system and the user.

- Modify the system so that the user could remotely access the system via telephone, telnet or other inter-computer communication devices.

- Providing the parser the capability to handle negations and induction in queries.

- Continue to develop machine learning mechanism for UMA by investigating abductive learning or multi -strategy learning.

YEAR - THREE:

- Continue to upgrade UIA, both functionally and aesthetically.

- To improve Voice Interface Agent, to handle more complex queries.

- Incorporate image processing techniques so that the results can be manipulated by the user to be stored according to his /her preferences.

- Add an error tracking mechanism that would be able to notify the user in different ways, when error has occurred.

- Assist Voice Interface Agent in processing the queries which will be in natural language.

- Continue to improve the parser to handle more types of queries.

- Applying Data Mining Techniques, to refine the knowledge that was learnt to be more concise and more effective.

- Applying better generalizations techniques to improve the preciseness of reasoning of the UMA

OVERVIEW OF THE USIA SYSTEM

An Agent is an entity that is *qualified* to perform information-processing tasks, which are delegated to it (with the possibility of *incomplete specifications* in a particular *domain).*

.

User System Interface Agent (USIA) is a community of cooperating agents directly responsible for assisting human users in obtaining services from an Information processing environment. Figure1 summarizes the design architecture.

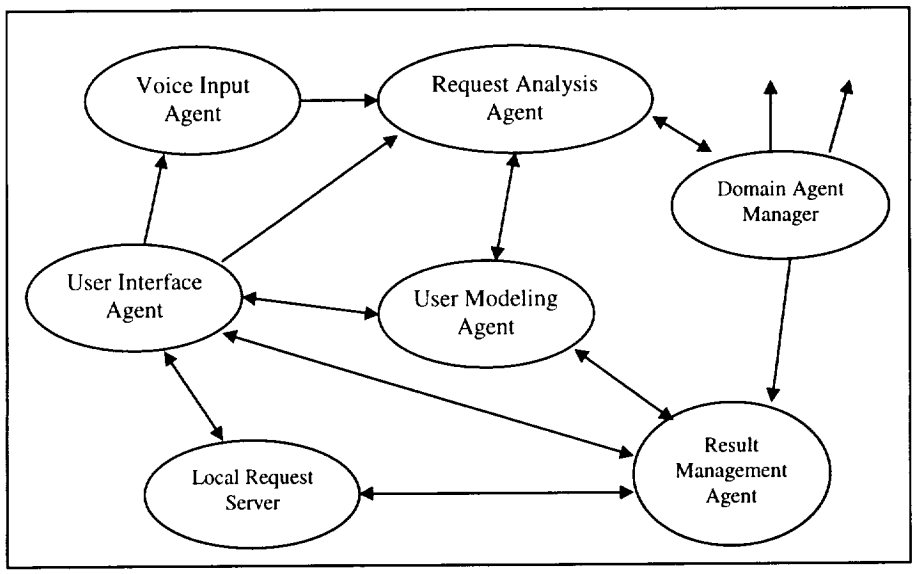


Figure 1. USIA Prototype Architecture

User Interface Agent: **UIA** is a special agent concerned with interaction/dialog with a human user. Its main responsibility is to facilitate the solicitation of input and the presentation of feedback and results back to the user.

Forms of input include:

- Voice - can be handled by a special Voice Input Agent (VIA) which assists the UIA to process spoken language
- Mouse - for a window-based graphical user interface
- Typed test Via keyboard
- Others?

Forms of output include:

- Displayed text
- Displayed images
- Synthesized voice

Request Analysis Agent: RAA is a special agent that is responsible for verifying the grammatical and semantic correctness of a service request which has been submitted through the UIA, possibly with incomplete specifications. The RAA is also responsible for resolving ambiguities and completing any missing specification based on known knowledge of the user and the domain. The output of RAA is the original request which has been transformed into a set of high-level actions and performatives for further analysis and decomposition by the Domain Agent Manager (DAM).

User Modeling Agent: UMA is a special agent that is responsible for providing user-modeling services to the USIA environment. Its primary role is to assist the RAA in understanding a user's request and, consequently better serve the user. The UMA gathers information from other agents (e.g., RAA) in order to maintain models of users and help the system adapt accordingly.

Domain Agent Manager: DAM is a special agent which is responsible for decomposing a request, which has been passed to it from RAA, into tasks, developing a request processing/execution plan and delegating tasks to domain specialist agents. It is also responsible for informing the UIA upon the completion of a request.

Note: DAM is not a part of USIA.

Result Manager Agent: RMA is a special agent that is responsible for assembling results of executed tasks of each service request and presenting them to the user, through the UIA. An RMA utilizes special display tools, which are capable of displaying the results based on their format.

Local Request Server (LRS) is a module, which provides local services in a particular environment. Such services include printing, email, delete, save and display a local object, e.g., a result file.

USER INTERFACE AGENT (UIA)

The User Interface Agent can be viewed as the front-end system that provides users with a transparent interface to a community of agents of which each agent may have a different type of expertise and therefore, a special protocol for using the interface. In this multiple-

agent -based system, the agents work in a cooperative manner to service a user's request. UIA is that special agent that handles this interaction. Currently the following features have been incorporated into the User Interface Agent:

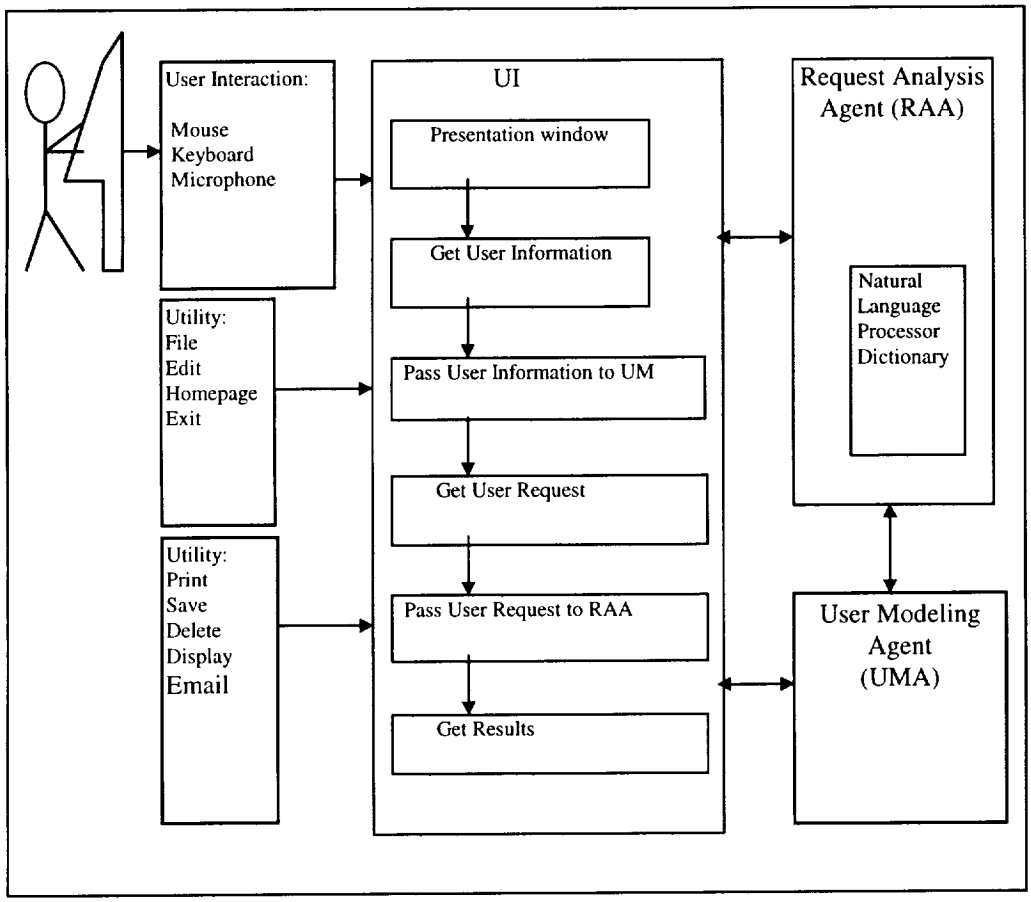


Figure 2. UIA Architecture

- **Type of development environment**:

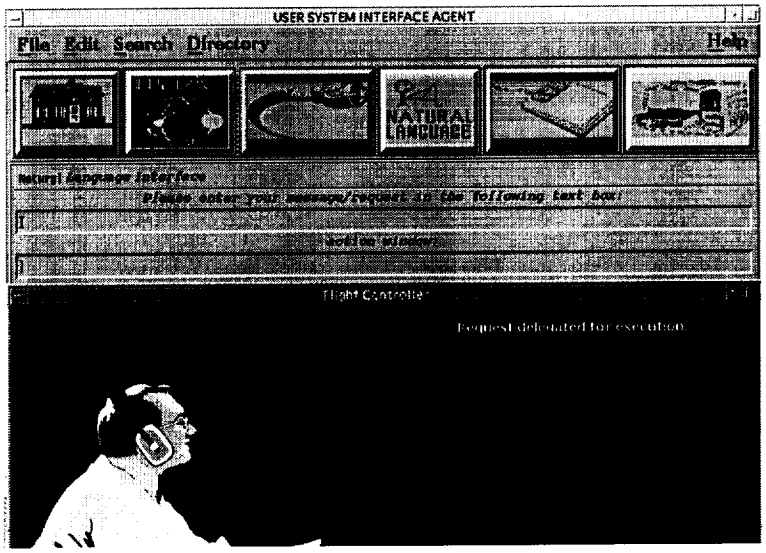
Combination of icon based graphical interface and restricted natural language. Tools such as CLIPs, CLIPsTOOL, and Motif toolkits for the X-Window system have been used in the implementation.

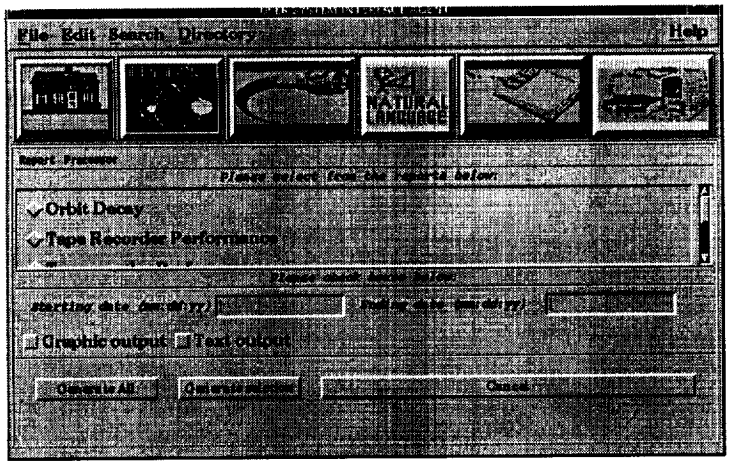
- **Types of dialog between the user and the UI A**:

Two Types of dialog:

1. Taxonomy –based 'select and combine' type of interface. This taxonomy is Based on information about the capabilities of the I.P.E.
2. A restricted natural language interface: that is simple enough for novice users to state their 'fuzzy', and often ambiguous requests, but powerful enough for expert users to state their specific and often complete requests.

Snapshot of User Interactions with UIA





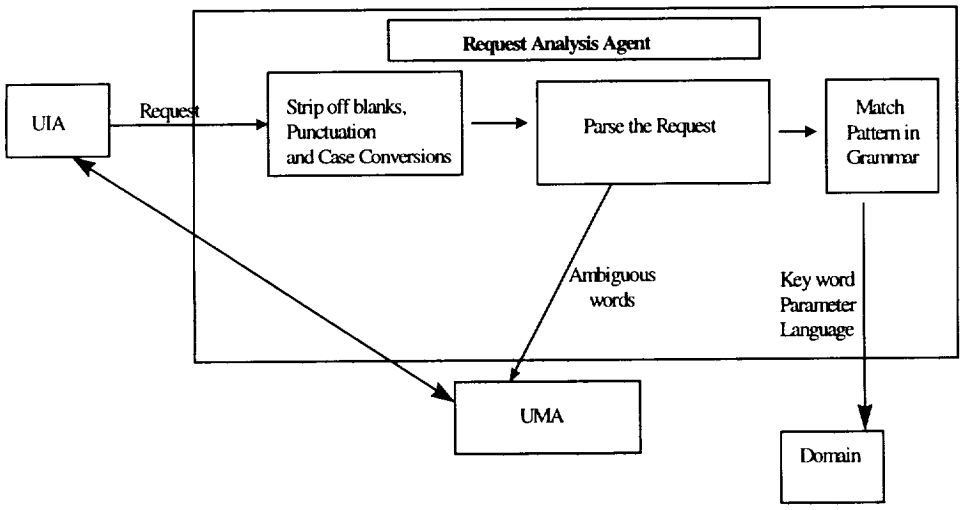
REQUEST ANALYSIS AGENT (RAA)

Request Analysis Agent is responsible for user request analysis and decomposition. The RAA parses and analyses the request of the user and converts it into a form that is recognizable by the domain agents. The main challenge of this agent is to provide the user with as much lexical latitude as possible, while at the same time sending exact requests to the agents.

The requests may be ambiguous. The RAA resolves the ambiguity with the help of User Modeling Agent and User Interface Agent. The request is parsed and analyzed with the help of the Generic and Domain dictionaries and by pattern matching in the Grammar maintained by the RAA.

The grammar and the domain dictionaries have been expanded to support more complex requests in the Report Generation domain. Features, which allow variations in time and date specification, have been provided.

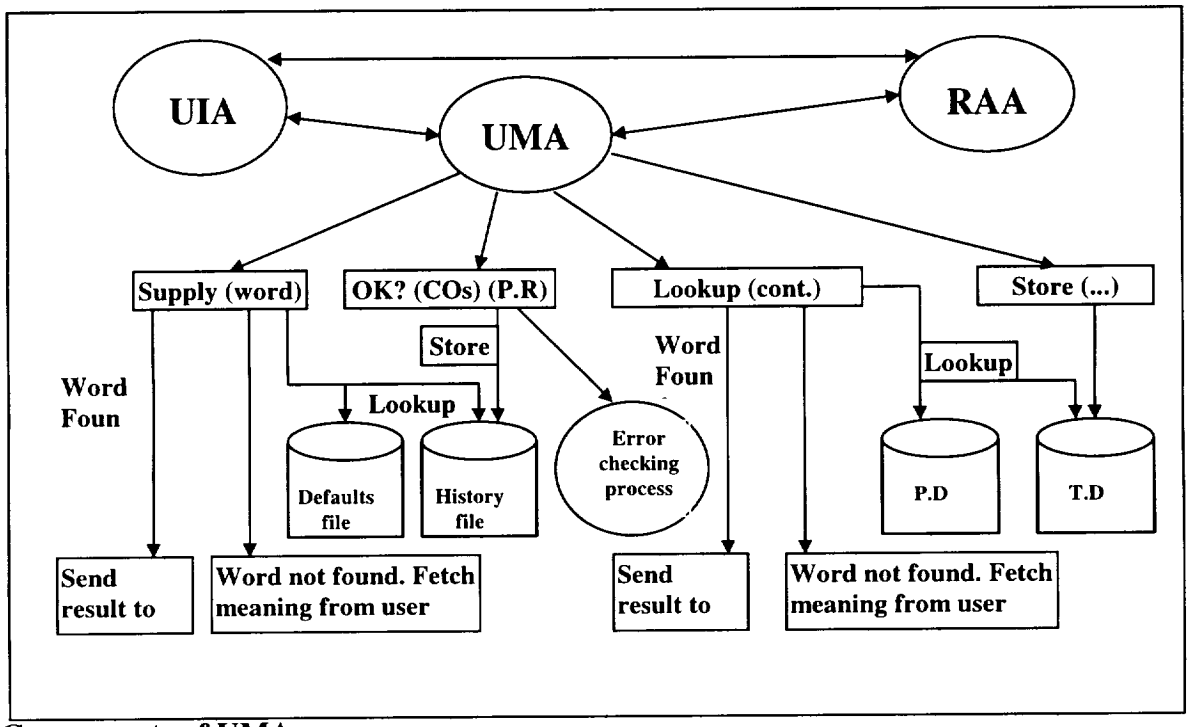
Future work will involve making the parser more powerful and flexible, making the Generic dictionary versatile, evolving faster dictionary searching algorithms and adapting the grammar to handle various types of requests.



THE USER MODELING AGENT

Purpose

The objective of User Modeling (UM) is to make USIA appear more like a trained personal assistant to each user of the system. Like a personal assistant, initially, this User Modeling Agent is not very helpful since it only has default parameters of human users in the user profile when a user model is created, before it becomes familiar with a human user's behavior. However, as interactions between man and machine increase, this agent becomes familiar with the work habits and preferences of human users. Every experience the agent learns, by observing how, when and why a user performs tasks, by learning from other more experienced agents through communication within the environment, is added to a knowledge base and enhances UMA's capability. Gradually, it can recognize the pattern of which user behaves. Through pattern matching and reasoning, it can solve ambiguity predict and offer to automate user actions. Architecture of UMA is shown below:



Components of UMA

- ***User-Specific Knowledge Bases***

  1. User Profile: A file that contains user's default parameters such as, user id, preferred interface (Natural, Taxonomy or Voice), type of user (Mouse or

keyboard), preferred output format (Graphical or text), method of presenting report (Display automatically whenever they are ready or save them in a file and displayed later), preferred printer, preferred color for background and foreground and aliases.

2. History base: A file that contains all valid requests issued by each user.

3. Temporary User Dictionary: A file that contains frequently used words by the user up to a threshold value to monitor their frequency. The dictionary has the following format:
<user word> <context> <meaning> <current threshold value>

4. Permanent User Dictionary: A file that contains frequently user words by the user, which have past the threshold test in the temporary dictionary. The dictionary has the following format:
<user word> <context> <meaning>
Note: A word can have more than one mapping depending on the context.

- ***Message-handler***

This module monitors and coordinates the activities of UMA. It accepts incoming message/request from client agents, interprets and delegates tasks to expert modules for processing, collects and assembles result back to client agents.

- ***Expert modules***

1. Knowledge acquisiter: A module that dialogues with UIA (user) to capture more knowledge about user such as confirm with users for a parsed request or ask user to supply more information.

2. User profiler: A module that creates and maintains user's profile.

3. User classifier: A module that has capability of identifying and classifying users.

User classification

By default a first time user is classified as a beginner and is provided an interview questionnaire to set up user's default profile; however, classification is based on three criteria: (1) number of times the user uses the help, (2) number of unsuccessful requests and (3) number of interactions between the UIA (user) and the UMA. Based on these criteria, a mathematical formula has been develop to carryout user classification:

User class = (x * 0.3) + (y * 0.5) + (z * 0.2) where

X = number of help a user has requested

Y = number of unsuccessful request a user attempted to make

Z = number of interactions has made in between UIA and UMA

The above is based on 50 request per cycle which means that upon reach 50 request limit, user will be reclassified by *user classifier* as beginner, intermediate or expert based on the above formula.

User class:

- The Novice user group
    a. First time users or not frequent users.
    b. Simple request statements(according to specific threshold test)
    c. Use only a selected number of the system utilities(ratio)
    d. More frequent use of the help system, tutorial (threshold test)
    e. High rate of mistakes (repeated, new...etc.) (ratio)

- The Intermediate user group
    a. Just been promoted from Novice (Past the threshold test)
    b. More frequent users
    c. Make use of some automation tools
    d. Less frequent use of the help system, tutorial (threshold test)
    e. Moderate rate of mistakes (ratio)

- The Advanced user group
    a. Just been promoted from Intermediate (Past the threshold test)
    b. Very sophisticated users
    c. Little or no use of the help system
    d. Low rate of mistakes (ratio)

Spell check

Misspelled word can be detected and corrected in this version of UMA by classifying a user as a good typist or a poor typist. The assumption is made when a user makes 10 or more mistakes out of 60 words, he/she will be classified as a poor typist and therefore the spell checker is turned on against the user.

Performatives:

UMA communicates with other agent using a set of performatives. In the following is our current set that has been implemented:

Between **UMA** and **RAA**

(1) **Lookup** :
Usage: Look up a word
Syntax: (a) Lookup (context) (word)
(b) Lookup (word)

From RAA to UMA
(a) 1 2 3 4 5 6 request lookup "email Ed"
(b) 1 2 3 4 5 6 request lookup "Ed"

From UMA to RAA
(a) 1 2 3 4 5 6 respond lookup "Ed Miller.bsu.umd.edu"
(b) 1 2 3 4 5 6 respond lookup "Ed Miller"

(2) **Supply**:
Usage: Used by RAA to ask UMA to supply a device name or directory name
Syntax: Supply (word)

(a) From RAA to UMA
1 2 3 4 5 6 request supply "printer"
(b) From UMA to RAA
1 2 3 4 5 6 respond supply "Laser Jet III P"

(3) **Ok? :**
Usage: Used to confirm the correctness of a request
Syntax: Ok? (Corrected Original Statement) (Parsed Request) (# Of Spellerrors)

(a) From RAA to UMA
1 2 3 4 5 6 request ok? (C.O.S) (P.R) (#)
(b) From UMA to RAA
1 2 3 4 5 6 respond ok? OK

(4) **Spellcheck :**
Usage: Used to turn on or off the Spell check
Syntax: spellcheck on/off

(a) From UMA to RAA
1 2 3 4 5 6 FYI spellcheck "on"

(5) **Store :**
Usage: Used by RAA to inform UMA to store data
Syntax: store message

(a) From RAA to UMA
1 2 3 4 5 6 FYI store "dolly phone 123"

Between **UMA** and **UIA**

(1) **Find:**
Usage: Used by UIA to find user classification
Syntax: Find (userid) ----------(a) From UIA to UMA
1 2 3 4 5 6 request find "ra205"
Syntax: Find (word) ----------(b) From UMA to UIA
1 2 3 4 5 6 respond find "new"

(2) **Ask:**
Usage: Used by UMA to ask user to supply more information through UIA
Syntax: Ask (word)

(a) From UMA to UIA
1 2 3 4 5 6 request ask, "email Chris"

(b) From UIA to UMA
1 2 3 4 5 6 respond ask, "email chris104@cosc"

(3) **Choose**:
Usage: Used by UMA to ask user to identify a right choice when UMA finds several
Option correspond to a request
Syntax: choose <word +>

(a) From UMA to UIA
1 2 3 4 5 6 request choose "Ed" "Ed Miller" "Ed Ruberton"
(b) From UIA to UMA
1 2 3 4 5 6 respond choose "Ed Miller"

(4) **Confirm**:
Usage: Used by UMA gets confirmation from through UIA once a parsed request has
Been generated
Syntax: confirm (context) (word)

(a) From UMA to UIA
1 2 3 4 5 6 request confirm "name Jim"

(b) From UIA to UMA
1 2 3 4 5 6 respond confirm "yes"

(5) **Pushedhelp :**
Usage: Used by UIA to inform UMA that user has requested a particular help

Syntax: pushedhelp

(a) From UIA to UMA
1 2 3 4 5 6 FYI pushedhelp

(6) **Pushedinterface :**

Usage: Used by UIA to inform UMA that a particular interface has been used by a User

Syntax: pushedinterface interface

(a) From UIA to UMA
1 2 3 4 5 6 FYI pushedinterface "NL"

(7) **Changelevel :**

Usage: Used by UMA to inform user that his/her level has been changed when he/her Has reached a particular expertise level

Syntax: changelevel word

(a) From UMA to UIA
1 2 3 4 5 6 FYI changelevel "beginner"

Appendix

Classification of the user:

For the UM to be able to assist users in performing their various tasks, it must first learn about certain *application-relevant* characteristics each user might possess. So for this the user is classified based on four criteria:

(I) Frequency of using the system
(ii) Typing skills
(iii) Decisiveness
(iv) Using the same sequence of requests

For any classification three aspects have to be considered
- How to classify i.e., what criterion should be considered for classification
- How to implement i.e., how do we implement the classification in CLIPS
- How is it useful i.e., how this type of classification is useful for the user and For the system

(I) Frequency of usage: Beginner, Intermediate or Expert

The classification of user as a beginner, intermediate or expert is done based on how frequently the user is using the system.
Using the 'last' command to know how many times a particular user used the system can capture this information.

If once the user is classified as a beginner, intermediate or expert the system can help the user in different ways:

If the user is a beginner/novice the system can guide the user how to use the system and can suggest him/her the sequential order he/she should follow. While giving the results it can tell the user about the different modes of displaying and ask him/her to choose one. So the system guides and helps a novice user in using the system.

Once when the user is thorough about the system facilities and if the user is using the system more frequently the user will be reclassified as an expert or intermediate user and the system can turn off the verbose help mode and can suggest him/her more direct / advanced ways of using it.

(ii) Typing skills: Poor or Good typist

If the user is making more mistakes in typing i.e., if the ratio of number of errors to numbers of words keyed in is considerable then we can classify him/her as a poor typist.

The mistakes could be of the nature - mistyping a key, transposing of letters, missing one letter or typing extra letter etc.

RAA will be sending the original statement keyed in by the user and number of mistakes it detected to the UM. The UM will again check its own word-dictionary and detect if any more mistakes were there. Once it detects all mistakes the ratio of number of mistakes to number of words keyed in will be computed. Here we are making an assumption that a poor typist will make 10 mistakes out of 60 words. So the computed ratio is checked and based on that the user is classified.

When once the system knows that a particular user is a poor typist it can decide the commands for him/her - like if the user types NMAE for NAME the system will not ask the user for the meaning of NMAE instead it will supply the correct word NAME, which it will have in its word-dictionary, to the RAA i.e., the system *tolerates* the user's simple typing mistakes. When once if this user is reclassified as a *good typist* this option can be turned off because this searching through the word-dictionary with *tolerance* is expensive.
The system can also help the user (by suggesting and by reminding) to make aliases or shorthand commands so that the user can be more comfortable.

(iii) Decisiveness: Ambiguous or Concise

The user who is not clear in his requests is classified as an ambiguous user and the one who is clear in his requests is classified as a concise user.

To classify the user under this category we will have certain ratio of ambiguity and if it exceeds that, then this user will be classified as an ambiguous user.

When once the system knows that a particular user is ambiguous then the ambiguity-resolution process is activated and the system resolves the ambiguity for the user.

When once this user is reclassified as a concise user then the ambiguity-resolution process is deactivated because this is expensive.

(iv) Using the same sequence of requests: Stereotypical

If the user is entering the same sequence of commands everytime he/she logs on then he/she can be classified as a stereotypical user.

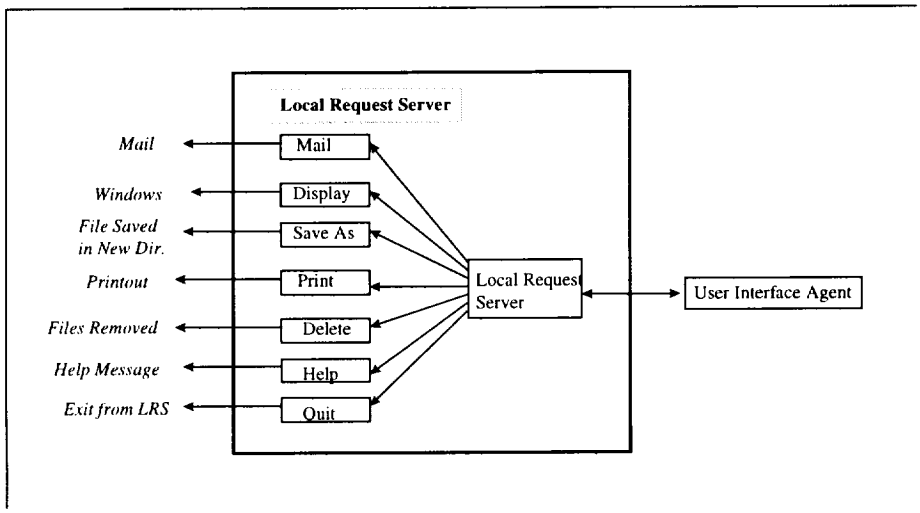
The UM will be storing all the commands entered by the user in a request server for that particular user i.e., it will have the command history of each user. So if the user is using the same type of commands everytime then he/she can be classified as a stereotypical user.

If once the system knows that a particular user is a stereotypical, when next time he/she logs on, the system will do all the jobs for him/her instead of the user entering all the commands again.

LOCAL REQUEST SERVER (LRS)

Local Request Server (LRS) is a small module, which is responsible for all Local requests. The services provided by LRS are mail, display, save as, Delete, print, quit, and help. LRS mainly interacts with the User Interface Agent.

The existing graphical user interface for LRS was created with the Motif Toolkit. The features that have been implemented for LRS include mail, Display, save as, delete, print, quit, and help.



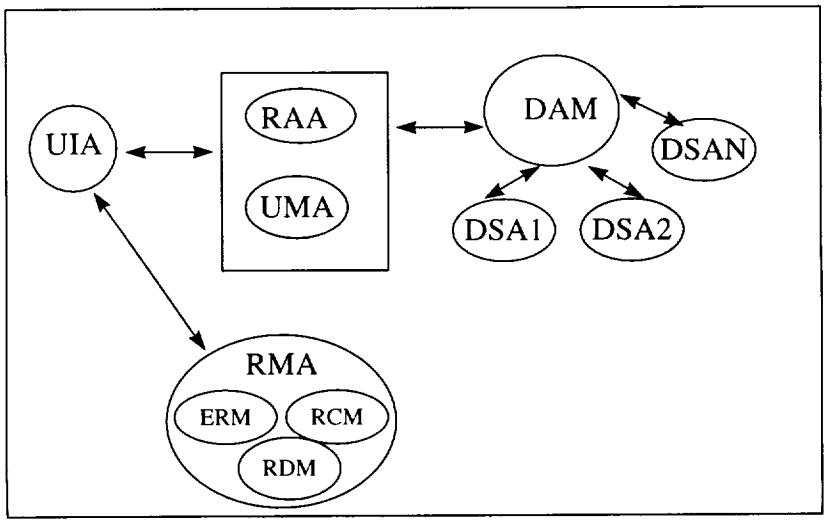
RESULT MANAGER AGENT (RMA)

Result Manager Agent (RMA) is a module which is mainly responsible for collecting results from the Domain Agent Manager (DMA), and for interacting with User Interface Agent in presenting the results to the user. The two agents that RMA has to collaborate with in the USIA system are UMA and UIA.

* RCM (Results Collecting Module) is responsible for interacting with DAM and for interpreting messages from DAM.

* RDM (Results Distributing Module) is responsible for interacting with UIA so that the results can be sent to UIA for display purposes.

* As for ERM (Error Reporting Module), it is used to handle various error problems that happen when RMA is receiving results, interpreting message headers, or when it is interacting with UIA.



FUTURE PLANS

- The feasibility of using Java Programming Language as a way of providing user-friendly screens and implementing new technologies is being researched.
- The addition of voice interface agent that would have the responsibility of relying information back to the user in the form of voice communication is being explored.
- Addition of an error tracking mechanism that would be able to notify the user in different ways, when an erroneous situation has occurred is under consideration.
- To adapt the system so that the user could remotely access the system via telephones telnet or other inter-computer communication devices.
- Make the parser powerful and flexible enough to handle various requests, which are in natural language by enhancing the Grammar. Also the generic dictionary will be made versatile and faster dictionary searching algorithms will be evolved.
- Object oriented Programming will be used to design a new User Modeling Agent who can create user models dynamically, and group user models into appropriate classes and coordinate as well as manage them. Learning mechanism of the UMA will be enhanced and problem solving algorithms with forward and backward chaining for user model classes to enhance the reasoning capability will be evolved.
- Local Request Server will be developed in Java.
- Result Manager Agent who manages and assembles results from different machines will be implemented.