CHAPTER - III
CHAPTER III

METHODOLOGY

The MADM vision proposed here is that of an "anarchic" collection of persistent, autonomous (but cooperating) KDD agents operating across the Internet. To investigate the nature of the envisaged MADM, its operation, its architecture, and the underlying research issues, identified above, the investigation was supported by establishing a working system.

The proposed system was termed as EMADS (Extendible Multi Agent Data mining System). Here we describe the under-pinning philosophy of MADM frameworks and, by extension, the view of the EMADS framework proposed. This chapter concentrates on the generic features of MADM and describes the EMADS framework to support the MADM vision.

Depending on context, EMADS is referred to both as a "system", since it is a Multi Agent System (MAS), and as a "framework" because EMADS defines a framework for achieving MADM. To realise the EMADS framework a general software development methodology was adopted that moved from problem definition and analysis to detailed design and implementation. This chapter provides a detailed description Existing Problem Definition of the framework in terms of its requirements, design and architecture.

3.1 MADM REQUIREMENTS ANALYSIS

Developing a data mining system that uses specialized agents with the ability to communicate with multiple information sources, as well as with other agents, requires a great deal of flexibility. For instance, adding a new information source should merely imply adding a new agent and advertising its capabilities; a process that should be facilitated in such a way that it is as simple as possible. As noted above the motivation for researching and implementing a fully operational MADM framework was to facilitate the investigation of the various MADM research challenges and issues outlined above.
3.1.1 Design Requirements

The realization of the desired MADM framework requires the consideration of a number of requirements. These requirements are itemized in the following way:

a) Multiple Data Mining Tasks: The MADM framework must be able to provide mechanisms to allow the coordination of data mining tasks. The number and nature of the data mining tasks that the framework should be able to address is not known a priori, and is expected to evolve over time. Consequently the framework should be designed in such a way as to anticipate future tasks.

b) Agent Coordination: The framework must be reactive since it must accommodate new agents as they are created in the environment. Careful consideration therefore needs to be directed at the communication mechanisms.

c) Agent Reuse: The framework must promote the opportunistic reuse of agent services by other agents. To this end, it must provide mechanisms by which agents may advertise their capabilities, and ways of finding agents supporting certain capabilities.

d) Scalability and Efficiency: The scalability of a data mining system refers to the ability of the system to operate effectively and without a substantial or discernible reduction in performance as the number of data sites increases. Efficiency, on the other hand, refers to the effective use of the available system resources. The former depends on the protocols that transfer and manage the intelligent agents to support the collaboration of the agents, while the latter depends upon the appropriate evaluation and filtering of the available agents to avoid targeting of irrelevant sources. Combining scalability and efficiency without sacrificing performance is, however, an intricate problem.

e) Portability: A distributed data mining system should be capable of operating across multiple environments with different hardware and software configurations (e.g. across the Internet), and be able to combine multiple models with (possibly) different representations. The framework should be able to operate on any major operating system. In some cases, it is possible that the data could be downloaded and stored on the same machine as the data mining software.
f) **Compatibility:** Combining multiple models of data mining results has been receiving increasing attention in the data mining research literature. In much of the prior work on combining multiple models, it is assumed that all models originate from the same database or from databases with identical schema. This is not always the case, and differences in the type and number of attributes among different data sets are not uncommon. The resulting model computed at a single database is directly dependent on the format of the underlying data. Minor differences, in the schema, between databases derive incompatible models, i.e. a classifier cannot be applied on data of different formats. Yet, these classifiers may target the same concept. The framework must be able to operate using several data sources located on various machines, and in any geographic location, using some method of network communication.

g) **Adaptivity and Extendibility:** Most data mining systems operate in environments that are likely to change, a phenomenon known as concept drift. For example, medical science evolves, and with it the types of medication, the dosages and treatments and of-course the data included in the various medical database. Alternatively lifestyles change over time and so do the profiles of customers included in credit card data; new security systems are introduced and new ways to commit fraud or to break into systems are devised.

It is not only patterns that change over time. Advances in machine learning and data mining will give rise to algorithms and tools that are not available at the present time. Unless the MADM system in use is flexible to accommodate existing as well as future data mining technology it will rapidly be rendered inadequate and obsolete.

The structure and operational requirements of the desired MADM framework are measured in detail. Given their significance in the context of MADM, the extendibility requirements are considered in detail.

### 3.1.2 Structural Requirements

The goal of the structural requirements analysis is to identify the flow of information through the envisioned MADM framework so as to clearly define the
expected system input and output streams. By breaking the framework into domain level concepts, it was possible to begin to identify the nature of the agents that MADM might require. Four main domain level components were identified: (i) user interface, (ii) planning and management, (iii) processing, and (iv) data interface. The interaction (information flow) between these four main modules is shown in Figure 3.1. The Figure should be read from left to right. The user interface component receives data mining requests. Once the request is received, it is processed (parsed) to determine the data mining algorithms and data sources required to respond to the request (this is the function of the Planning and management component). The identified data sources are then mined (the processing component), through access to the data interface component, and the results returned to the user via the user (interface) component.

![Figure 3.1: Main Domain Components](image)

### 3.1.3 Operational Requirements

Most current agent-based data mining frameworks share a similar high-level architecture, and provide common structural components, to that shown in Figure 3.1. Components of the form described above have become a template for most agent-based data mining and information retrieval systems. The structure illustrated in Figure 3.1 sets out three important elements of MADM systems: (i) agent technology, (ii) domain components, and (iii) information brokerage (middleware). Agent technology is a self-evident element of MADM. Multi Agent Systems (MAS) espouse the use of collaborative agents, operating across a network, and communicating by means of a high level query language such as KQML and FIPA ACL. Domain components or ontologies, give a concise, uniform description of semantic information, independent of the underlying syntactic representation of the data. Finally, information brokerage utilizes specialized facilitator agents to match information needs with currently available resources, so (for example) retrieval and update requests can be properly routed to the relevant resources.
Given the above considerations the general operation of EMADS (as suggested in Figure 3.1) is as follows:

a) **Source Identification**: When a request is received, select the appropriate information source or sources. One way to do this is using meta-data obtained at the time of the query to determine what sources to use. The advantage of this is that the knowledge sources are current at the time the query is made.

b) **Assignment**: Assign the appropriate data mining algorithm(s).

c) **Task Scheduling**: Plan and execute the required Task. Task Planning involves the coordination of data retrieval, and the ordering and assignment of processes to the appropriate agents. This is expressed in the form of a “plan”. The steps in the plan are partially ordered based on the structure of the query. This ordering is determined by the fact that some steps make use of data that is obtained by other steps, and thus must logically be considered after them.

d) **Result**: Return the results to the user. These steps are fairly generic and could be the foundation for any envisaged MADM.

### 3.2 Extendable Multi Agent Data Mining Agents and Users

The basic focus on the functionality of the agents, based on the design requirements identified, several types of EMADS agent were considered. However, regardless of “type”, EMADS’ agents adhere to the general agent definitions described in [126]. The EMADS agent types are: User, Management, Facilitator, Data Source, Data Mining (DM), and Registration. Each falls within the domain level concepts identified in structural requirements. The user and data source agents are all identified as interface agents because they all provide “interfaces” to either users, or data sources. User agents provide the interface between EMADS end users and the rest of the framework; at the same time as data source agents provide the interface between input data and the rest of the framework. Task agents and Data Mining (DM) agents are identified as processing agents because they carry out the required “processing” needed to respond to user requests, and possibly, to pre-process data within the system.

Figure 3.2 shows what agents will reside in each domain component identified in the structural requirements analysis in structural requirements. The task agent receives a
request and asks the broker agent to check all available databases (data agents) and DM agents to find: (i) which data to use, and (ii) which data mining algorithms (held by DM agents) are appropriate.

![Diagram showing System General Architecture](image)

**Figure 3.2: System General Architecture**

The task agent then passes the task to DM agents and monitors their progress. Each database must have an interface agent (data agent) to check the database for a matching schema and then report back to a DM agent and shows the system level inputs and outputs as they flow from agent to agent.

In general, it is convenient to think of an EMADS agent as formed by three interacting software modules: (i) the interface module, (ii) the process module and (iii) the knowledge module. Note that modules are only included when they are needed and these modules are included when they are necessary.

The agents are considered in the same order as they might be utilized to process an EMADS user request in order to illustrate how each agent’s function fits into the system’s overall operation and contributes to supporting the other agents. They are given below.

**a) User Agent**

There are four essential operations that a user agent must be able to undertake:
i) Receive and interpret user data mining requests.
ii) Communicate the user request to the processing agents.
iii) Return the generated results, in a suitable and easily understandable format, back to the user.
iv) Expect and respond to asynchronous events, such as "a stop mining" or "end operations" instructions that may be issued by the user.

A user agent is also required to have the knowledge needed to translate information from the user to a format that processing agents can understand and vice-versa.

The user agent is the only agent that interacts with the user. It asks the user to issue data mining requests, passes them to the system, and provides the user with results. The user agent's interface module contains methods for inter agent communication and for obtaining input from the user. The process module contains methods for capturing the user input and communicating it to the task agent. In the knowledge module, the agent might store the history of user interactions, and user profiles with their specific preferences. In addition, the interface module may provide access to (say) visualization software, that may be available to show results, or to other post-processing software.

The process module may contain methods to support ad-hoc and predefined reporting capabilities, generating visual representations, and facilitating user interaction. The knowledge module stores details about report templates and visualization primitives that can be used to present the result to the user.

b) Registration Agent

The general expected function of the registration is to inform all appropriate agents that are already in the system and interested of a new agent arrival. When any new agent is introduced into the system, it must first inform the registration agent that it has entered. Since the system is designed to be relatively static in terms of new data sources and data source types, the registration agent will be the least utilized agent. It should also be the first agent created and started in the system. Because the system cannot operate without a broker agent as well, the registration agent will initiate all methods, and then await a broker agent to enter the system.
Once notified a broker has entered the system, it completes initialization and waits for a registration request from any new agents. When a new agent enters the system, it sends a registration request message to the registration agent. When it receives notification of a new agent, the registration agent will determine the functions the agent can perform by the information transmitted in the registration message. The broker agent will be informed of all types of agents entering the system.

Once all appropriate existing system agents have been notified, the new agent will be informed it is active in the system. However, in the work described here is the registration agent was eliminated and its functionality was integrated with the broker agent. The agent’s interface module contains methods for inter-agent communication. The knowledge module contains meta-knowledge about all agents in the system.

c) Facilitator Agent

The facilitator agent/Broker agent serves as an advisor agent that facilitates the distribution of requests to agents that have expressed an ability to handle them. This is performed by accepting advertisements from supply agents and recommendation requests from request agents. The facilitator agent keeps track of the names and capabilities of all registered agents in the framework. It can reply to the query of an agent with the name and address of an appropriate agent that has the capabilities requested. Its knowledge module contains meta-knowledge about capabilities of other agents in the system. In general, any agent in EMADS can use the facilitator agent to advertise their capabilities in order to become a part of the agent system (what is known as a "yellow pages" service). When the facilitator agent receives notification of a new agent who wants to advertise its services; it must add this new agent’s identifier and its capabilities to the list of available system agents.

To fulfill a task— the “task agent” must “talk to” the facilitator agent, asking which agents can fulfill the given request. The facilitator agent maintains all information on the capabilities of individual agents in the system and responds to queries from task agents as to where to route specific requests. By requesting only those agents who may have relevant information, the task agent can eliminate tasking any agents that could not possibly provide any useful information. However, it should be noted that the facilitator
agent does not maintain full information about the agents in the system, only their high
level functionality and where they are located.

d) Task Agent

A task agent / (or Management Agent) is responsible for the activation and
synchronisation of the various EMADS agents required to generate a response to a given
user request. Individual categories of task agent dedicated to different, but specific, data
mining operations have been identified; the various categories are considered in detail in
later chapters. A task agent performs its task by first generating a work plan, and then
monitoring the progress of the plan.

Task agents are designed to receive data mining requests from user agents and
seek the services of groups of agents to obtain and synthesize the final result to be
returned to the user agents. The agent interface module is responsible for inter-agent
communication; the process module contains methods for the control and coordination of
the various tasks. A task agent may be required, when generating a response to a request,
to: identify relevant data sources, request services from agents, generate queries, etc. The
knowledge module contains meta-knowledge about data mining tasks, i.e., what steps are
required for what type of task, input requirements for each of the data mining tasks, etc.

Once the user agent has received a user request, it passes it to the task agent. The
task agent then determines, according to the information passed to it and through contact
with the facilitator agent (see below), what other agents are required to generate a
response to the request. The nature of a received request can dictate one of two possible
types of action: (i) performance of a data mining task, or (ii) the cancellation of the
current operation. These user desires will be passed to the task agent in the form of a
request from the user agent.

In the first case the task agent will ask the facilitator agent for DM agents which
can fulfill the desired tasking. For instance, if the user wants all possible association rules
meeting a given minimum support and confidence thresholds, across all available data
sources, then the task agent will contact the appropriate DM agents with this request. The
task agent accepts the result from each individual DM agent and stores it in its knowledge
base. Once the DM task is completed, the task agent combines the results to provide a
single result before passing it to the user agent. Note that in some cases there may be only a single result in which case there will be no requirement to combine results. The second case may occur where the user feels (say) that the current operations are taking too long, or are no longer needed. In this case, the task agent must send "cancel" messages to all agents currently tasked and performing work.

**e) Data Mining Agent**

A Data Mining (DM) agent implements a specific DM technique or algorithm; as such a DM agent can be said to "hold" a DM algorithm. The interface module supports inter-agent communication. The process module contains methods for initiating and carrying out the DM activity, capturing the results of DM, and communicating it to a data agent or a task agent. The knowledge module contains meta-knowledge about DM tasks, i.e. what method is suitable for what type of problem, input requirements for each of the mining methods, format of input data, etc. This knowledge is used by the process module in initiating and executing a particular mining algorithm for the problem at hand.

A DM agent accepts a request, from a task agent, and initiates the mining algorithm using the values contained in the request. As the algorithm runs, the DM agent may make requests for data to data agents, or ask other DM agents to cooperate. The DM agent continues until it has completed its task, i.e. generated a result to the presented request, and then returns the results to the task agent to be passed on to the user agent and eventually to the EMADS end user who originated the request.

**f) Data Agent**

A data agent (or Resource Agent) is responsible for a data source and maintains meta-data information about the data source. There is a one-to-one relationship between a given data agent and a given data source. Data agents are responsible for forwarding their data, when requested to do so, to DM agents.

Data agents also take into account issues to do with the heterogeneity of data. The interface module for a data agent supports inter-agent communication as well as interfacing to the data source it is responsible for. The process module provides facilities for ad-hoc and predefined data retrieval. Based on the user request, appropriate queries are generated by DM agents and sent to data agents who then process the queries.
according to the nature of their data set. The results, either the entire data set or some specified sub-set, are then communicated back to the DM agents.

Once the facilitator agent has determined what agents can fulfill a given task, it passes this information back to the task agent. The task agent then tasks each “useful” data agent, passing it the relevant information, requesting it to provide appropriate data. The term “useful” refers to data agents that are responsible for a data source that includes some part of the required data. An individual data agent is not specific to any particular data mining task but rather is able to answer any query associated with its data. When a new data source is introduced into EMADS it must be “wrapped” (as described in the next chapter) so that a new data agent is created.

During this process the presence of the new data agent will be announced to the facilitator agent so that the new agent can be recognized by the system and so that the facilitator agent can add a reference for the new agent to the list of system agents. Once the new data agent has registered, it will query the user through its GUI interface for the domain of the data source (i.e. data file location) for which it is responsible.

g) EMADS End User Categories

EMADS has several different modes of operation according to the nature of the participant. Each mode of operation (participant) has a corresponding category of agent as described above and as shown in Figure 3.3. The figure also shows the JADE housekeeping agents (AMS and DF). The supported participants (EMADS end user categories) are as follows:

- **EMADS Developers**: Developers are EMADS participants, who have full access and may contribute DM algorithms and tasks.

- **EMADS Data Miners**: These are participants, with restricted access to EMADS, who may pose DM requests.

- **EMADS Data Contributors**: These are participants, again with restricted access, who are prepared to make data available to be used by EMADS data mining agents.

In each case, before interaction with EMADS can commence, the participant must download and launch the appropriate EMADS software. Note that any individual participant may be a user as well as a contributor and/or developer at the same time. With respect to EMADS users it should be recalled that the nature of EMADS DM requests, that may be posted, is extensive.
3.3 Defining Interaction Protocols

Various protocols required to support the identified primary functions and high-level interactions are defined for each agent. EMADS employs efficient distributed protocols, to avoid the consequent potential use of many messages, while maintaining a generic architecture that can be extended with new DM tasks.

More generally, with respect to MAS, a protocol is some agreed message format for communication between agents that promotes some shared understanding. In MAS, protocols typically define how agents request information from one another, and how agents return responses to these requests. The agent initiating a message is called the sender, and the agent receiving it is the receiver. The response to a message usually requires the receiver to perform some action which will then generate the response to be returned to the sender. The message data is specified as content. The following syntax, expressed as a context free grammar in BackusNaur Form (BNF), is used to define the protocols.

![Figure 3.3: High Level View of EMADS Conceptual Framework](image)

< Protocol > ::= < Header > < Send - Message > < Receive - Message >
< Header > ::= < ProtocolName > < SenderAgent > < ReceiverAgent >

< Send - Message > ::= < Send - Message - Tag > |
  < Send - Message - Tag > < Send - Content > |
  < Send - Message - Tag > < Send - Action > |
  < Send - Message - Tag > < Send - Content >
  < Send - Action >

< Receive - Message > ::= < Receive - Message - Tag > |
  < Receive - Message - Tag > < Receive - Content > |
  < Receive - Message - Tag > < Receive - Action > |
The first three lines of each protocol (the header) identifies the protocol's name (label) and the intended message "sender" and "receiver". The sender or receiver can be one or more of the EMADS agent categories identified above. The next part of the protocol defines the nature of the message to be sent, this consists of at least a message tag, but may also define some message content and/or some action to be performed by the sender once the message has been sent. The message tag is one of a set of predefined tags (strings) that are understood by EMADS agents.

The final part of the protocol defines the nature of the message to be returned to the sender (i.e. the reply) and any consequent action. This part of the protocol also consists of at least a message tag, but again may also define some message and any expected action(s) to be performed by the sender on receipt of the message. In some cases there may be a number of alternative messages that can be received, in which case this will be indicated by an "OR" (and parentheses to enhance readability).

It is also possible for the sender to receive a sequence of replies of messages and actions. This format is used in the following protocols to define the seven main EMADS protocols that have been identified they are:

a) Find Other Agents Protocol

Prior to transmitting a message a sender agent needs to identify the subset of receiver agents appropriate to a particular requirement. A sender agent can first discover exactly what agents are interested in receiving its message by communicating with the facilitator agent. The ability of sender agents to do this is a general requirement that may
be repeated on many occasions, and the generic Find Other Agents protocol was therefore specifically created for this purpose.

**The protocol is as follows:**

- **Protocol Name:** Finding other Agents
- **Sender:** Any requesting agent
- **Receiver:** Facilitator Agent
- **send:** findAgents
- **content:** agent type
- **(receive:** agentsFound
  - **content:** agentList
  or
  **receive:** noAgents)

The receiver is always the facilitator agent. The sender agent sends a findAgents" tagged message, with the content defining the agent type that the sender wishes to communicate with, to the facilitator agent. The agent type indicates the nature of the DM task the sending agent is interested in (for example a DM agent may be interested in classification). The facilitator agent will then either return a message containing a list of appropriate agents identifiers or a “noAgent” tagged message indicating that no agents of the specified agent type were found.

**b) Agent Registration Protocol**

The Agent Registration protocol allows new agents to “register” their presence with the facilitator agent. The Agent Registration protocol is as follows:

- **Protocol Name:** Agent Registration
- **Sender:** New agent
- **Receiver:** Facilitator Agent
- **send:** register
- **content:** domain and agent meta-data (services, capabilities)
- **receive:** accepted or rejected

The sender agent sends a “register” tagged message, with the content describing the agent domain and a description of the new agent’s service and capabilities, to the facilitator agent. The sender will either receive an “accepted” tagged message, indicating that the registration was accepted and the agent is made public to other agents; or a “rejected” tagged message indicating that the registration failed.
c) User DM Request Protocol

User agents interact only with task agents. Once a request from the user is received the user agent initiates a request for some DM task to be performed based on the task type specified within the user DM request. The task agent will acknowledge receipt to the user agent and initiate the DM process. The user agent will then wait until either the result of the DM request is returned; or it is informed, by the associated EMADS end user, that the request has failed (for example because no appropriate data sources can be found). In the first case the user agent will forward the DM result to the EMADS end user. In the second case the user agent will confirm termination of the current DM task.

The DM Request protocol is as follows:

Protocol Name: Requesting DM
Sender: User Agent
Receiver: Task Agent
send: mining
receive: accept
(receive: resultsReady
content: results
or
receive: noData)

d) Start Data Mining Task Protocol

Task agents are central to the EMADS data mining process and interact with the user agents, the facilitator agent, and one or more DM agents. The task agent initially reacts to a request initiated by the user agent.

The user agent will request that some DM operation be undertaken and will pass the appropriate attributes (for example threshold values) dependent on the nature of the desired DM task. In response to a request to be undertaken, the task agent will determine the nature of the request according to the variables sent. Once the task agent has determined the type of the request (for example an ARM request), it informs the user agent it has all the information it needs with an “accept” message and initiates the required processing.

The task agent then interacts with the facilitator agent to determine what agents it should task for the given request. Then, the task agent awaits the result. It can receive one of two possible answers. It can receive an “agentsFound” message and a list of useful agents, or a “noAgents” message, indicating there are no data sources that could be mined.
for the variables given. If "noAgents" is received, the task agent sends a "noData" message to the user agent and ends the user protocol.

The task agent will first use the Finding Other Agents protocol, if agents are found, an "agentsFound" message tag will be returned to the sender together with a list of identifiers for the "found" DM agents. The task agent will interact with each of the identified DM agents. The task agent will request that the identified DM agents begin DM according to the values in the original request received from a user agent.

The task agent may pass any variables received from the user agent to the DM agents according to the nature of the DM request. Once the task agent receives the confirmation from the DM agent, it awaits either results or a request initiated by the user agent requesting a termination of the current DM operation. The Start Data Mining Task protocol is given below.

Protocol Name: *Start Data Mining Task*

Sender: Task Agent
Receiver: DM Agent
send: beginMining
receive: miningComplete
content: result
do: return result
or
do: terminate

A task agent may interact with more than one DM agent. In this latter case some post processing of the result is typically undertaken. After all necessary interactions with the DM agents have been completed; the task agent sends a "resultsReady" message to the user agent.

e) Advertise Agent Capabilities Protocol

The facilitator agent primarily interacts with task agents, DM agents, and data agents; but has the ability to respond to any properly formatted request for agents that can fulfill a given task. As noted above, the primary functions of the facilitator agent are to: (i) maintain a list of all agents in the system that want to advertise their services and the tasks they can perform, and (ii) answer queries requesting lists of agents that can fulfill any given task. To perform the first function, the facilitator must be able to communicate with any new agents entering EMADS and receive and process the new agent's
The advertising process begins with the new agent sending an “advAgent” message that contains the new agent’s full name and task list. The Advertise Agent Capabilities protocol is as follows:

- **Protocol Name:** Advertise agent capabilities
- **Sender:** New Agent
- **Receiver:** Facilitator Agent
- **send:** advAgent
- **content:** agentMetaData, serviceList
- **receive:** agentAdded

The facilitator agent will obtain the new agent's information from the message content, the global list of agents will be updated and an “agentAdded” message tag (i.e. an acknowledgement) sent to the new agent. Once this last message is sent the protocol is terminated.

**f) Perform Data Mining Protocol**

DM agents interact with task agents and data agents. A specific DM agent will await a “beginMining” message from the task agent. This message will contain the original request (from the user agent) and the name(s) of the data agent(s) to be used. Once this is received, the DM agent starts the mining algorithm associated with it by applying it to the data indicated by the specified data agent reference(s). When a DM agent’s algorithm completes, the DM agent sends a “miningCompleted” reply message, with the generated results as the content, to the sending task agent. The Perform Data Mining protocol is as follows:

- **Protocol Name:** Performing Data Mining
- **Sender:** Task Agent
- **Receiver:** DM Agent
- **send:** beginMining
- **content:** DM task type, DM and Data Agent lists
- **do:** wait
- **receive:** miningCompleted
- **content:** result
- **do:** if applicable process results and return result

The protocol includes an option to combine results where several DM agents or data agents may have been used.
g) Data Retrieval Protocol

Data agents hold the software required to interface with data sources. Data agents also have the capabilities to communicate with DM agents that will operate "over them". Data agents interact with DM agents using the following Data Retrieval protocol:

- **Protocol Name**: Data Retrieval
- **Sender**: DM Agent
- **Receiver**: Data Agent
- **send**: getData
- **content**: SQL
- **receive**: retrievedData
- **content**: data

3.4 Data Mining with Agents

In EMADS DM task, planning is realized by negotiation between EMADS agents through the message passing mechanism. The DM process begins with the user agent receiving notification from the end user describing a data mining request. The user agent picks up the user request and then starts a task agent. The task agent then asks the facilitator agent for the identifiers of all "useful" (DM and data) agents specified in the context of the request. An agent is deemed "useful" if it can potentially contribute to the resolution of the request.

The facilitator agent receives the request and compiles a list of all appropriate (interested) agents. The facilitator agent then returns the list of "useful" agents to the task agent. Once the task agent receives the list, it can commence the desired DM process. The nature of this DM process will depend on the nature of the request.

A number of different categories of task agent have been identified in the context of EMADS; these are discussed in further detail in later chapters. However, in general terms, the task agent sends a request to each identified DM agent in the list (there may only be one) to begin DM together with appropriate references to the identified data agents. Each DM agent accepts the request and begins mining their applicable data source. Once completed, the DM agents send the results back to the task agent. When the task agent has all the results, it processes the results (for example it may combine them), and notifies the user agent (in some cases there may of course only be one set off results). The user agent then displays the combined results to the user.
3.5 The Agent Development Toolkit

Building sophisticated software agents is a challenging task, requiring specialized skills and knowledge in a variety of areas including: agent architecture, communications technology, reasoning systems, knowledge representation, and agent communication languages and protocols. To this end a number of agent development toolkits are available which may be used to build MAS in an efficient and effective manner in that they reduce agent development complexity and enhance productivity. In general agent development toolkits provide a set of templates and software modules that facilitate and/or implement basic communication. Development toolkits may also provide templates for various types of agents, or constructs that agents can use. Basic communication can be as simple as direct communication among agents.

The key difference between most development toolkits lie in the implementation and architecture of the provided communication and agent functionality. When selecting a toolkit to build some desired MAS developers should make their decision based on the MAS goals and services that are desired. Any potential toolkit should also be evaluated for potential problems related to the toolkit’s strengths and weaknesses prior to any decision being made. JADE was chosen for the proposed EMADS framework development. JADE was selected for a variety of reasons as follows:

- JADE is both popular and regularly maintained (for bug fixes and extra features).
- It was developed with industry quality standards.
- The tool kit covers many aspects of MAS, including agent models, interaction, coordination, organisation, etc.
- It is simple to set-up and to evaluate. This includes good documentation, download availability, simple installation procedure, and multi-platform support.
- It is FIPA-compliant.

Some of the reasons that these other platforms were avoided included:

- That they were still in an experimental state, abandoned, or confidentially distributed, Very little documentation was associated with them,

- They covered only one aspect, or a limited number of aspects, of MASs; such as single agent platforms, mobile agent platforms, interaction infrastructures toolkits, They were found to be too short on some construction stages, for example purely methodological models.
3.5.1 JADE

JADE is a software environment, fully implemented in the JAVA programming language, directed at the development of MAS. JADE is a FIPA-compliant middle-ware that enables development of peer to peer applications based on the agent paradigm. JADE defines an agent platform that comprises a set of containers, which may be distributed across a network (as desired in the case of EMADS).

The goal of JADE is to simplify the development of MAS while at the same time ensuring FIPA compliance through a comprehensive set of system services and agents. While appearing as a single entity to the outside observer, a JADE agent platform can be distributed over several hosts each running an instance of the JADE runtime environment. A single instance of a JADE environment is called a container which can “contain” several agents as shown in Figure 3.4. The set of one or more “active” containers is collectively referred to as a platform (as indicated by the dashed perimeter line in Figure). For a platform to be active it must comprise at least one active container; further containers may be added (as they become active) through a registration process with the initial (main) container. A JADE platform includes a main container (the middle container in Figure), in which is held a number of mandatory agent services. These are the Agent Management System (AMS) and Directory Facilitator (DF) agents.

The AMS agent is used to control the life-cycles of other agents on the platform, while the DF agent provides a lookup service by means of which agents can find other agents. When an agent is created, upon entry into the system, it announces itself to the DF agent after which it can be recognised and found by other agents.

Further “house-keeping” agents are the Remote Monitoring Agent (RMA) and The Sniffer Agent (SA). The first keeps track of all registered agents, while the second monitors all message communications between agents.
Within JADE, agents are identified by name and communicate using the FIPA Agent Communication Language (ACL). More specifically, agents communicate by formulating and sending individual messages to each other and can have “conversations” using interaction protocols that range from query request protocols to negotiation protocols. JADE supports three types of message communication as follows:

a) **Intra-container**: ACL message communication between agents within the same container using event dispatching.

b) **Intra-platform**: Message communication between agents in the same JADE platform, but in different containers, founded on RMI.

c) **Inter-platform**: Message communication between agents in different platforms uses the IIOP (Internet Inter-ORB Protocol).

The latter is facilitated by a special Agent Communication Channel (ACC) agent also located in the JADE platform main containers. The JADE communication architecture is intended to offer (agent transparent) flexible and efficient messaging by choosing, on an “as needed” basis, the most appropriate of the FIPA-compliant Message Transport Protocols (MTP) that are activated at platform run time. Basically, each container has a table containing details of its local agents, called the Local-Agent Descriptor Table (LADT), and also maintains a Global-Agent Descriptor Table (GADT), mapping every agent into the RMI object reference of its container. The main container has an (additional) Container Table (CT) which is the registry of the object-references and transport addresses of all container nodes. Each agent is equipped with an incoming message box and message polling can be blocking or non-blocking.
JADE uses LADTs and GADTs for its address caching technique so as to avoid querying continuously the main-container for address information and thus avoiding a potential system "bottleneck". However, although the main-container is not a "bottleneck", it is still a single potential point of failure within the platform. This is a recognised issue within the JADE user community.

Research has been reported seeks to address this issue, for example Bellifemine et al. (2007) who described a mechanism whereby a JADE main-container replication service was used to deploy a fault-tolerant JADE platform. However, most users simply "live with the problem"; a strategy that has also been adopted with respect to the EMADS framework.

FIPA specifies a set of standard interaction protocols, such as FIPA-requests and FIPA queries. These protocols can be used to build agent "conversations" (sequences of agent interactions). In JADE, agent tasks or agent intentions are implemented through the use of behaviours.

![Figure 3.5: ACL message Example](image)

**Figure 3.5: ACL message Example**

All agent communications are performed through message passing using the FIPA ACL. Figure 3.5 shows an example of a FIPA ACL message. Each agent is equipped with an incoming message box, and message polling can be blocking or non-blocking with an optional time-out.

According to the structure of the JADE protocols, which are FIPA-compliant, the sender sends a message and the receiver can subsequently reply by sending either: (i) a
not-understood or a refuse message indicating the inability to achieve the rational effect of the communicative act; or (ii) an agree message indicating the agreement to perform the communicative act. When the receiver performs the action it must send an inform message. A failure message indicates that the action was not successful. JADE provides ready-made classes for most of the FIPA specified interaction protocols. Table 3.1 shows a list of FIPA predefined message types, while Table 3.2 shows a list of FIPA pre-defined message parameters.

3.5.2 JADE AGENT INTERACTION

In JADE, agent tasks or agent intentions are implemented through the use of behaviours. Behaviours are logical execution threads that can be composed in various ways to achieve complex execution patterns and can be initialised, suspended and spawned at any given time. Behaviours are implemented in terms of fields and methods contained in one or more sub-classes of a parent Behaviour class provided with JADE. Any given JADE agent keeps a task list that contains the active behaviours. JADE uses one thread per agent, instead of one thread per behaviour, to limit the number of threads running on the agent platform. A behaviour can release the execution control with the use of blocking mechanisms, or it can permanently remove itself from the queue during runtime. Each behaviour performs its designated operation by executing the method “action()”. The Behaviour class is the root class of the behaviour hierarchy that defines several core methods and sets the basis for behaviour scheduling as it allows state transitions (starting, blocking and restarting).

3.6 EMADS ARCHITECTURE AS IMPLEMENTED IN JADE

EMADS is implemented using the JADE MAS development framework; Broadly speaking JADE defines an agent platform that comprises a set of containers, which may (as in the case of EMADS) be distributed across a network. Here we start with an overview of the EMAD JADE implementation. More specific details of the JADE implementation focuses on: Agent “Behaviours”, agent interaction, mechanisms for cooperation, user request handling, and extendibility.
Message Type: Meaning:

Accept Proposal: The action of accepting a previously submitted proposal to perform an action.

Agree: The action of agreeing to perform some action, possibly in the future.

Cancel: The action of one agent informing another agent that the first agent no longer has the intention that the second agent performs some action.

Call for Proposal: The action of calling for proposals to perform a given action.

Confirm: The sender informs the receiver that a given proposition is true, where the receiver is known to be uncertain about the proposition.

Dis-confirm: The sender informs the receiver that a given proposition is false, where the receiver is known to believe, or believe it likely that, the proposition is true.

Failure: The action of telling another agent that an action was attempted but the attempt failed.

Inform: The sender informs the receiver that a given proposition is true.

Inform If: A macro action for the agent of the action to inform the recipient whether or not a proposition is true.

Inform Ref: A macro action for sender to inform the receiver the object which corresponds to a descriptor, for example, a name.

Not Understood: The sender of the act informs the receiver that it did not understand the action the receiver just performed.

Request: The sender requests the receiver to perform some action.

Request When: The sender wants the receiver to perform some action when some given proposition becomes true.

Propose: The action of submitting a proposal to perform a certain action, given certain preconditions.

Refuse: The action of refusing to perform a given action, and explaining the reason for the refusal.

---

**Table 3.1: FIPA Communicative Acts (pre-defined message types)**

<table>
<thead>
<tr>
<th>Message Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>:sender</td>
<td>Denotes the identity of the sender of the message, i.e. the name of the agent of the communicative act.</td>
</tr>
<tr>
<td>:receiver</td>
<td>Denotes the identity of the intended recipient of the message. The recipient may be a single agent name, or a tuple of agent names.</td>
</tr>
<tr>
<td>:content</td>
<td>Denotes the content of the message; equivalently denotes the object of the action.</td>
</tr>
<tr>
<td>:reply-with</td>
<td>Introduces an expression which will be used by the agent responding to this message to identify the original message.</td>
</tr>
<tr>
<td>:envelope</td>
<td>Denotes an expression that provides useful information about the message as seen by the message transport service.</td>
</tr>
<tr>
<td>:language</td>
<td>Denotes the encoding scheme of the content of the action.</td>
</tr>
<tr>
<td>:ontology</td>
<td>Denotes the ontology which is used to give a meaning to the symbols in the content expression.</td>
</tr>
<tr>
<td>:reply-by</td>
<td>Denotes a time and/or date expression which indicates a guideline on the latest time by which the sending agent would like a reply.</td>
</tr>
<tr>
<td>:protocol</td>
<td>Introduces an identifier which denotes the protocol which the sending agent is employing. The protocol serves to give additional context for the interpretation of the message.</td>
</tr>
<tr>
<td>:conversation-id</td>
<td>Introduces an expression which is used to identify an ongoing sequence of communicative acts which together form a conversation. A conversation may be used by an agent to manage its communication strategies and activities.</td>
</tr>
</tbody>
</table>

---

**Table 3.2: FIPA pre-defined message parameters**
Within JADE, agents can have “conversations” using interaction protocols, in the case of the EMADS implementation; agents may be created and contributed by any EMADS user/contributor. One of the containers, the main container, holds the house-keeping agents and the Agent Management System (AMS). Both the main container and the remaining containers can hold various DM agents. The EMADS main container should be located on the EMADS host organisation site, while the other containers may be held at any other sites worldwide.

Other than the house-keeping agents, held in the main container, EMADS currently supports the four categories of agents identified in EMADS agents and users i.e. User agents, task agents, DM agents and data agents. Registration and facilitator functionalities are provided by JADE house-keeping agents (AMS and DF agents respectively). It should be noted that EMADS containers may contain both DM agents and data agents simultaneously as well as user agents. DM agents and data agents are persistent, i.e. they continue to exist indefinitely and are not created for a specific DM exercise as in the case of task agents. Communication between agents is facilitated by the EMADS network.

Figure 3.6 gives an overview of the implementation of EMADS using JADE. The figure is divided into three parts: at the top are listed N user sites. In the middle is the JADE platform holding the main container and N other containers. At the bottom a sample collection of agents is included. The solid arrows indicate a “belongs to” (or “is held by”) relationship, while the dotted arrows indicate a “communicates with” relationship. So the data agent at the bottom right belongs to container 1 which in turn belongs to User Site 1; and communicates with the AMS agent and (in this example) a single DM agent.

The principal advantage of this JADE architecture is that it does not overload a single host machine, but distributes the processing load among multiple machines. The results obtained can be correlated with one another in order to achieve computationally efficient analysis at a distributed global level.
Many important design issues were considered while implementing EMADS within the JADE framework; including:

a) ACL Messages (protocols, content).
b) Data structures.
c) Algorithms and software components.

The ACL messages were defined with respect to the JADE ACL Message class fields [9] and the FIPA ACL Message Structure Specification [20]. The procedure to map the EMADS agent interaction protocols, and JADE behaviors was found to be relatively straightforward. EMADS agent activities and protocols were translated to a number of predefined JADE behaviours (defined in terms of methods contained in the JADE Behaviours class) to either action methods or to simple methods of behaviours. Predefined JADE behaviours that were found to be useful to EMADS were:

- **OneShotBehaviour**: Implements a task that runs once and terminates immediately.
- **CyclicBehaviour**: Implements a task that is always active, and performs the same operations each time it is scheduled.

- **TickerBehaviour**: Implements a task that periodically executes the same operations.

- **WakeBehaviour**: Implements an atomic task that runs once after a certain amount of time, and then terminates.

When dealing with complex responsibilities, it was found to be better to split the responsibilities into a combination of a number of simpler tasks and adopt one of the composite behaviour classes provided by JADE. These composite behaviour classes include:

- **Sequential Behaviour**: Implementing a composite task that schedules its sub-tasks sequentially.

- **FSM Behaviour**: Implementing a composite task that schedules its subtasks according to a Finite State Machine (FSM) model.

Composite behaviour can be nested and therefore there can be, for instance, a subtask of a Sequential Behaviour that is in turn a FSM Behaviour and so on. In particular, all complex responsibilities that can be modelled as Finite State Machines can be effectively implemented as FSM Behaviour instances.

Furthermore, the behaviours that start their execution when a message arrives, can receive this message either at the beginning of the action method (simple behaviours) or by spawning an additional behaviour whose purpose is the continuous polling of the message box (complex behaviours). For behaviours that start by a message from a Graphical User Interface (GUI), a GUI event receiver method should be implemented on the agent that starts the corresponding behaviour. Finally, those behaviours that start by querying a data source, or by a calculation, should be explicitly added by their upper level behaviour.

### 3.6.2 Agent Interactions

A user agent, as shown in Figure 5.8, runs on the user's local host and is responsible for accepting user input, launching the appropriate task agent that serves the user request, and displaying the results of the distributed computation. The user expresses a task to be executed with a standard (GUI) interface dialog mechanisms by...
clicking on active areas in the interface, and in some cases by entering some thresholds attributes; the user does not need to specify which agent or agents should perform the task.

For instance, if the question “What is the best classifier for my data?” is posed in the user interface, this request will trigger (create and start) a task agent (in this case a classifier generation task agent). The task agent requests the facilitator to match the action part of the request to capabilities published by other agents. The request is then routed by the task agent to appropriate agents (in this case, involving communicate on among all classifier generator agents in the system) to execute the request. On completion the results are sent back to the user agent for display.

The key elements of the operation of EMADS in the context of agent interaction that should be noted are:

a) The mechanism whereby a collection of agents can be harnessed to identify a “best solution”.

b) The process whereby new agents connect to the facilitator and register their capability specifications.

c) The interpretation and execution of a task is a distributed process, with no one agent defining the set of possible inputs to the system.

d) That a single request can produce cooperation and flexible communication among many agents spread across multiple machines.

3.6.3 Mechanisms of Cooperation

Cooperation among the various EMADS agents is achieved via messages expressed in FIPA ACL and is normally structured around a three-stage process:

a) Service Registration: Where providers (agents who wish to provide services) register their capability specifications with a facilitator.

b) Request Posting: Where user agents (requesters of services) construct requests and relay them to a task agent.
c) **Processing:** Where the task agent coordinates the efforts of the appropriate service providers (data agents and DM agents) to satisfy the request. Stage 1 (service registration) is not necessarily immediately followed by stage 2 and 3; it is possible that a provider’s services may never be used. Note also that the facilitator (the DF and AMS agents) maintains a knowledge base that records the capabilities of the various EMADS agents, and uses this knowledge to assist requesters and providers of services in making contact. When a service provider (i.e. data agent or DM agent) is created, it makes a connection to the facilitator.

Upon connection, the new agent informs its parent facilitator of the services it can provide. When the agent is needed, the facilitator sends its address to the requester agent. An important element of the desired EMADS agent cooperation model is the function of the task agent.

### 3.6.4 User Request Handling

A task agent is designed to handle a user request. This involves a three step process:

a) **Determination:** Determination of whom (which specific agents) will execute a request.

b) **Optimization:** Optimization of the complete task, including parallelization where appropriate.

c) **Interpretation:** Interpretation of the optimized task.

Thus determination (step 1) involves the selection of one or more agents to handle each sub-task given a particular request. In doing this, the task agent uses the facilitator’s knowledge of the capabilities of the available EMADS agents (and possibly of other facilitators, in a multi-facilitator system). In processing a request, an agent can also make use of a variety of capabilities provided by other agents. For example, an agent can request data from data agents that maintain data.

The optimisation step results in a request whose interpretation will require as few communication exchanges as possible, between the task agent and the satisfying agents (typically DM agents and data agents), and can exploit the parallel processing capabilities of the satisfying agents. Thus, in summary, the interpretation of a task by a task agent involves: (i) the coordination of requests directed at the satisfying agents, and (ii) assembling the responses into a coherent whole, for return to the user agent.