An investigation into the issues of Multi-Agent Data Mining

Kamal Ali Albashiri, Frans Coenen, and Paul Leng
Department of Computer Science, The University of Liverpool
Ashton Building, Ashton Street, Liverpool L69 3BX, United Kingdom
{ali,frans,phl}@csc.liv.ac.uk

Abstract

Multi-agent systems (MAS) often deal with complex applications that require distributed problem solving. In many applications the individual and collective behaviour of the agents depends on the observed data from distributed sources. The field of Distributed Data Mining (DDM) deals with these challenges in analyzing distributed data and offers many algorithmic solutions to perform different data analysis and mining operations in a fundamentally distributed manner that pays careful attention to the resource constraints. Since multi-agent systems are often distributed and agents have proactive and reactive features, combining DM with MAS for data intensive applications is therefore appealing.

This Chapter discusses a number of research issues concerned with the use of Multi-Agent Systems for Data Mining (MADM), also known as agent-driven data mining. The Chapter also examines the issues affecting the design and implementation of a generic and extendible agent-based data mining framework. An Extendible Multi-Agent Data mining System (EMADS) Framework for integrating distributed data sources is presented. This framework achieves high-availability and high performance without compromising the data integrity and security.

Keywords: Meta Mining, Multi Agent Data Mining, Association Rule Mining, Vertical Partitioning, Classifier Generation.

1 INTRODUCTION

Knowledge discovery in databases (KDD) has evolved to become a well established technology that has many commercial applications. It encompasses subfields such as classification, clustering, and rule mining. However, it still poses many challenges to the research community. New methodologies are needed in order to mine more interesting and specific information from larger datasets. New frameworks are needed to harmonize more effectively all the steps of the KDD process. New solutions are required to manage the complex and heterogeneous sources of information that are today available for the analysts.

Research work in KDD continues to develop ideas, generate new algorithms and modify/extend existing algorithms. A diverse body of work therefore exists. KDD research groups and commercial enterprises are prepared (at least to some extent) to share their expertise. In addition, many KDD research groups have made software freely available for download [1]. This all serves to promote and enhance the current "state of the art" in KDD. However, although the free availability of data mining software is of a considerable benefit to the KDD community, it still requires users to have programming knowledge this means that for many potential end users the use of such free software is not a viable option.

The rest of the section is orgainzed as follows. The work motivation is presented in Subsection 1.1. The work objective is presented in Subsection 1.2. The work evaluation method is described in Subsection 1.3. EMADS Extendibility is described in Subsection 1.4. An overview of of EMADS implemented scenarios is presented in Subsection 1.5. The chapter organization is described in Subsection 1.6.

1.1 Motivation

There are a number of issues that DM researchers are currently addressing, these include: accuracy (especially in the context of classification), efficiency and effectiveness, privacy and security, and scalability. This last is significant as the amount of data currently available for DM is extensive and increasing rapidly year by year. One potential solution to the scalability issue is parallel or distributed DM although this often entails a significant communication overhead. Issues of privacy and security centre around legal issues and the desire of many holders of data to maintain the copyright they hold on that data.

Multi-agent systems (MAS) are communities of software entities, operating under decentralized control, designed to address (often complex) applications in a distributed problem solving manner. Multi-Agent Systems (MAS) offer a number of general advantages with respect to computer supported cooperative working, distributed computation and resource sharing. Well documented advantages include:

[1] Weka Tool Kit $http://www.cs.waikato.ac.nz/ml/weka/$, and the LUCS-KDD Software Library $http://www.csc.liv.ac.uk/\tilde{f}rans/KDD/Software/$

1. Autonomy
2. Decentralized control.
3. Robustness.
4. Simple extendibility.
5. Sharing of expertise.
6. Sharing of resources.
7. Process automation.
8. Data and task matching.
9. Result evaluation.

Autonomy and decentralized control are, arguably, the most significant features of MAS that serve to distinguish such systems from distributed or parallel approaches to computation. Autonomy and decentralized control [107] imply that individual agents, within MAS, operate in an autonomous manner and are (in some sense) self deterministic. Robustness, in turn, is a feature of the decentralized control, where the overall system continues to operate even though a number of individual agents have disconnected (crashed). Decentralized control also supports extendibility, in that additional functionality can be added simply by including further agents. The advantages of sharing expertise and resources are self evident.

The advantages offered by MAS are entirely applicable to Knowledge Discovery in Data (KDD) where a considerable collection of tools and techniques are current. MAS also have some particular advantages to offer with respect to KDD, and particularly data mining, in the context of sharing of resources and expertise. KDD is concerned with the extraction of hidden knowledge from data. Very often data relevant to one search is not located at a single site, it may be widely-distributed and in many different forms. There is a clear advantage to be gained from a software organization that can locate, evaluate, consolidate and mine data from these diverse sources.

The term Multi-Agent Data Mining (MADM) is used to describe Data Mining within a Multi-Agent Environment.

By its nature data mining is often applied to sensitive data. MAS allow data to be mined remotely. Similarly, with respect to data mining algorithms, MAS can make use of algorithms without necessitating their transfer to users, thus contributing to the preservation of intellectual property rights.

The motivation for the work described in this chapter is to examine whether, how, and to what extent MAS can be used to address the scalability and privacy and security issues, identified above, i.e. Multi-Agent Data Mining (MADM). The use of MAS will also provide a radical alternative approach to DM where collections of data mining agents (of various types) can be used to address traditional DM problems under decentralized control. The vision that the work espouses is that of an anarchic and dynamic agent community where agents interact with one another to address DM problems posted by users and where

data sources (agents) can be made available by individuals as desired by those individuals. The research issues that this entails are identified in the following section.

1.2 Objectives

The aim of the chapter is to evaluate the MADM vision, as described above, with the aim of establishing its efficacy and usability. The principal research question to be addressed is therefore: "does MADM provide the KDD community with any desirable advantage that would make the global establishment of such a system of genuine significance?" For example would it solve the scalability problem? Does it address privacy and security issues? Does it allow for simple extendibility and sharing of expertise?

The above research question encompasses a number of issues:

1. The negotiation and communication mechanism to be adopted to allow the envisaged agents to "talk" to one another.
2. The nature of the framework/platform in/on which the agents might operate.
3. The nature of individual DM algorithms that would gain full advantage of operating in a multi-agent setting.
4. Usage of such a system and the mechanisms for adding/removing agents.

The research presents the MADM vision the associated conceptualization and describes the Extendible Multi-Agent Data mining System (Framework), (EMADS). EMADS is a JADE implementation, which was used as a basis for investigating the issues identified above and evaluating approaches to dealing with them. Wrappers are used to incorporate existing software into EMADS. Experience indicates that, given an appropriate wrapper, existing data mining software can be very easily packaged to become an EMADS data mining agent.

Broadly EMADS is viewed as providing an "anarchic" environment to support MADM; more specifically it facilitates end-user participation and therefore allows for qualitative feedback. Each of the above research issues was investigated using a number of DM scenarios and applications commonly found in the domain of DM.

1.3 Evaluation

The following criteria were used to critically evaluate the work presented in later sections. The principal aim of the evaluation was to establish whether the EMADS vision satisfies the objectives described in Section 1.2. For this to be so, the EMADS should fulfill the following requirements:

- **Generality.** In order to be generic, the framework tasks need to be coordinated. The number of tasks is not known apriori, and may evolve over

time. The framework should also be reactive since it must accommodate new tasks as they are created in the environment.

- **Reusability.** The framework should promote the opportunistic reuse of agent services by other agents. To this end, it has to provide mechanisms by which agents may advertise their capabilities, and ways of finding agents supporting certain capabilities.

- **Extendibility and scalability.**

  Extendibility and scalability can be evaluated by considering the following questions:

  1. Is the framework complaint with the standard agent communication language i.e. FIPA ACL [34]?
  2. Ease of use and extendibility are achieved through the ease of adding and removing mechanisms of agents. Can new DM techniques be added to the system and out-of-date techniques be deleted from the system dynamically?
  3. Can DM technique agents interact at run-time with ease under this framework? In other non-agent based systems, these interactions must be determined at design-time.

Overall the aim of the evaluation is to show that EMADS can be used to perform data mining by considering a sequence of data mining scenarios designed to determine whether the above listed requirements are achieved. More details on the scenarios that were considered are given in Section 1.5.

1.4 Extendibility

One of the principal objectives of EMADS is to provide an easily extendible framework that could easily accept new data sources and new data mining techniques. In general, extendibility can be defined as the ease with which software can be modified to adapt to new requirements or changes in existing requirements. Adding a new data source or data mining techniques should be as easy as adding new agents to the system. The desired extendibility is achieved by the EMADS Architecture as implemented in JADE using a system of wrappers. The agent wrapper agentifies an existing application by encapsulating an implementation of the wrapping application. It manages the states of the application, invoking the application when necessary. EMADS wrappers are used to wrap up data mining artifacts so that they become EMADS agents and can communicate with other EMADS agents [6]. Two broad categories of wrapper have been defined: (i) data wrappers and (ii) tool wrappers. Each is described in further detail in Section 4.

1.5 Overview of EMADS Implemented Scenarios

The demonstration scenarios used to evaluate EMADS, and referred to throughout this chapter are considered in this subsection.

1.5.1 Meta Association Rule Mining (ARM)

The first scenario [5, 7] comprises a novel extension of ARM where a meta set of frequent itemsets are constructed from a collection of component sets which have been generated in an autonomous manner without centralized control. This type of conglomerate has been termed Meta ARM to distinguish it from a number of other related approaches such as incremental and distributed ARM. A number of MADM meta ARM algorithms were developed and evaluated: (i) Bench Mark, (ii) Apriori, (iii) Brute Force, (iv) Hybrid 1 and (v) Hybrid 2. This demonstrator is discussed in more detail in Section 3.

1.5.2 Vertical Partitioning and Distributed/Parallel ARM

The second demonstration scenario comprises an approach to distributed/parallel Association Rule Mining (ARM), DATA-VP, which makes use of a vertical partitioning approach to distributing the input data as described in [27]. Using this approach each partition can be mined in isolation while at the same time taking into account the possibility of the existence of large itemsets dispersed across two or more partitions. This scenario is discussed in more detail in Section 6.

1.5.3 Generation of Classifiers

Classification Association Rule Mining (CARM) aims to discover a small set of rules in the database that forms a classifier. The third demonstration scenario was a CARM scenario. This scenario [6] illustrates the operation of EMADS in the context of a classifier generation task. The scenario is that of an end user who wishes to obtain a best classifier founded on a given, pre-labeled, data set; which can then be applied to further unlabelled data.. This demonstrator is discussed in more detail in Section 7.

1.6 Chapter Organization

Section 2 of this chapter provides background a brief overview of the field of MADM. The concepts of association rules are presented in detail as it is central to three scenarios considered. To make the overview broader, other techniques such as decision tree induction and classification are also covered briefly. Section 3 gives an overview of the design and implementation of EMADS. Section 4 provides a detailed description of the EMADS extendibility feature in terms of its requirements, design and implementation. The three evaluation scenarios are considered in Sections 5, 6, and 7 respectively. The last Section presents some conclusion.

2 BACKGROUND AND LITERATURE REVIEW

This section presents a discussion and critical review of the current research relating to multi-agent data mining (MADM). It provides an overview of the theoretical background of the research discipline, identifying the approaches adopted, and discusses the benefits and challenges posed. It also highlights the principal areas in which current solutions fall short of requirements.

The organization of this section is as follows. An overview of data mining and distributed data mining is presented in Subsections 2.1 and 2.2 along with three specific data mining techniques: Association Rule Mining (ARM), Classification Rule Mining (CRM), and Decision Tree (DT) generation; which are used for demonstration purposes in this Chapter. In Subsection 2.3 a general description of agent and multi-agent systems is provided together with a review of multi-agent systems platforms. Finally, the use of agent and multi-agent systems for data mining, as an emerging field of data mining, is reviewed in Subsection 2.4, where overall research in this field is summarized and some well established approaches are presented.

2.1 Data Mining

During the last two decades, our ability to collect and store data has significantly outpaced our ability to analyze, summarize and extract "knowledge" from this continuous stream of input. Knowledge discovery in databases (KDD) denotes the complex process of identifying valid, novel, potentially useful and ultimately understandable patterns in data [33]. Data mining refers to a particular step in the KDD process. It consists of particular algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns (models) over the data. Data mining [44, 46, 48, 106] deals with the problem of analyzing data in scalable manner.

A considerable number of algorithms have been developed to perform data mining tasks, from many fields of science [108]. Typical DM tasks are classification (assign each record of a database to one of a predefined set of classes), clustering (find groups of records that are close according to some user defined metrics) or association rules (determine implication rules for a subset of record attributes).

The next three subsections briefly review the data mining techniques that are used (to varying degrees) for evaluation purposes in this chapter: the task of discovery of association rules (Association Rule Mining) and the task of classification using ARM and suing DTS.

2.1.1 Association Rule Mining

The most popular task of DM is to find trends in data that show associations between domain elements. This is generally focused on transactional data such as a database of purchases at a store. This task is known as Association Rule

Mining (ARM). It was first introduced in Agrawal et al. [1]. Association rules identify collections of data attributes that are statistically related in the underlying data.

An association rule is of the form X → Y where X and Y are disjoint conjunctions of attribute-value pairs. The *confidence* of the rule is the conditional probability of Y given X, Pr(Y|X), and the *support* of the rule is the prior probability of X and Y, Pr(X and Y). Here probability is taken to be the observed frequency in the data set.

The traditional ARM problem can be described as follows. Given a database of transactions, a minimal confidence threshold and a minimal support threshold, find all association rules whose confidence and support are above the corresponding thresholds.

An example of this type of rule is the statement that 90% of transactions in which cereal and milk were purchased, jam was also purchased, and 5% of all transactions contain all three items. The antecedent of this rule (X) consists of cereal and milk and the consequent (Y) is jam. The 90% represents the confidence factor of this rule and the 5% is the support for the rule. The rule can then be specified as cereal ∩ milk → jam. Both the antecedent and consequent can have sets of items, or can be a single item.

The most computationally demanding aspect of Association Rule Mining is identifying the frequent sets of attribute-values, or items, whose support (occurrence in the data) exceeds some threshold. The problem arises because the number of possible sets is exponential in the number of items. For this reason, almost all methods attempt to count the support only of candidate itemsets that are identified as possible frequent sets. It is, of course, not possible to completely determine the candidate itemsets in advance, so it will be necessary to consider many itemsets that are not in fact frequent.

Most algorithms involve several passes of the source data, in each of which the support for some set of candidate itemsets is counted. The performance of these methods, clearly, depends both on the size of the original database and on the number of candidates being considered. The number of possible candidates increases with increasing density of data (greater number of items present in a record) and with decreasing support thresholds. In applications such as medical epidemiology, where we may be searching for rules that associate rather rare items within quite densely populated data, the low support-thresholds required may lead to very large candidate sets. These factors motivate a continuing search for efficient algorithms.

2.1.2 The Apriori Algorithm

Since its introduction in 1994, the Apriori algorithm developed by Agrawal and Srikant [4] has been the basis of many subsequent ARM and/or ARM-related algorithms. In [4], it was observed that ARs can be straightforwardly generated from a set of frequent itemsets (FIs). Thus, efficiently and effectively mining FIs from data is the key to ARM. The Apriori algorithm iteratively identifies FIs in data by employing the "closure property" of itemsets in the generation of

Algorithm 2.1: Apriori

Input: (a) A transactional database Dt;
(b) A support threshold s;
Output: A set of frequent itemsets S;
1: begin:
2: $k \leftarrow 1$;
3: $S \leftarrow$ an empty set for holding the identified frequent itemsets;
4: generate all candidate 1-itemsets from Dt;
5: while (candidate k-itemsets exist) do
6: determine support for candidate k-itemsets fromDt;
7: add frequent k-itemsets into S;
8: remove all candidate k-itemsets that are not sufficiently supported to give frequent k-itemsets;
9: generate candidate $(k + 1)$-itemsets from frequent k-itemsets using closure property;
10: $k \leftarrow k + 1$;
11: end while
12: return (S);
13: end Algorithm
Note: A k-itemset represents a set of k items.

Table 1: Apriori Algorithm

candidate itemsets, where a candidate (possibly frequent) itemset is confirmed as frequent only when all its subsets are identified as frequent in the previous pass. The closure property of itemsets can be described as follows: if an itemset is frequent then all its subsets will also be frequent; conversely if an itemset is infrequent then all its supersets will also be infrequent. The Apriori algorithm is shown in Table 1.

Apriori performs repeated passes of the database, successively computing support-counts for sets of single items, pairs, triplets, and so on. At the end of each pass, sets that fail to reach the required support threshold are eliminated, and candidates for the next pass are constructed as supersets of the remaining (frequent) sets. Since no set can be frequent which has an infrequent subset, this procedure guarantees that all frequent sets will be found.

2.1.3 Classification Rule Mining

Classification Rule Mining is probably the most studied data mining task. In this task the goal is to predict the value (the class) of a user-specified goal attribute based on the values of other attributes, called the predicting attributes. For instance, the goal attribute might be the credit of a bank customer, taking on the values (classes) "good or "bad", while the predicting attributes might be the customer's Age, Salary, Account Balance, whether or not the customer has

an unpaid loan, etc.

Classification rules can be considered to be a particular kind of prediction rule where the rule antecedent ("IF part") contains a combination - typically, a conjunction - of conditions on predicting attribute values, and the rule consequent (THEN part) contains a predicted value for the goal attribute. Examples of classification rules are:

IF (paid-loan? = "yes" and (Account-balance > £3,000)
THEN (Credit = "good")
IF (paid-loan? = "no") THEN (Credit = "bad")

In the classifier generation task the data being mined is divided into two mutually exclusive data sets, the training set and the test set. The data mining algorithm has to discover rules by accessing the training set only. In order to do this, the algorithm has access to the values of both the predicting attributes and the goal attribute of each example (record) in the training set. Once the training process is finished and the algorithm has found a set of classification rules, the predictive performance of these rules is evaluated on the test set, which was not seen during training. For a comprehensive discussion about how to measure the predictive accuracy of classification rules readers should refer to [47].

2.1.4 Classification by Decision Trees

A decision tree is a tree structure in which each internal node denotes a test on an attribute, each branch represents an outcome of the test and leaf nodes represent classes. Decision tree induction methods are used to build such a tree from a training set of examples. The tree can then be used (following a path from the root to a leaf) to classify new examples given their attribute values. Because of their structure, it is natural to transform decision trees into classification rules, that can be easily inserted into a reasoning framework. Notice that some machine learning tools, such as C4.5 [84], already include a class rulesets generator.

Let us consider a very well known example, taken from [84]. Given a training set of examples which represent some situations, in terms of weather conditions, in which it is or it is not the case that playing tennis is a good idea, a decision tree is built which can be used to classify further examples as good candidates for playing tennis (class Yes) and bad candidates to play tennis (class No). Table 2 shows the original training set, given as a relational table over the attributes Overlook, Temperature, Humidity, Wind. The last column (class or target attribute) of the table represents the classification of each row.

Several algorithms have been developed to mine a decision tree from datasets such as the one in Table 2. Almost all of them rely on the the basic recursive schema used in the ID3 algorithm [83] (see Algorithm 3). The ID3 algorithm is used to build a decision tree, given a set of non-categorical attributes C1, C2, .., Cn, the categorical attribute C, and a training set T of records.

Overlook	Temperature	Humidity	Wind	Class
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcas	t Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Lowv	Weak	Yes
Rainy	Cool	Low	Strong	No
Overcast	Cool	Low	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Low	Weak	Yes
Rainy	Mild	Low	Weak	Yes
Sunny	Mild	Low	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Low	Weak	Yes
Rainy	Mild	High	Strong	No

Table 2: Training set of examples on weather attributes

Differences between algorithms usually depend on the splitting criteria used to identify the (local) best attribute to use as a node. Using a standard decision tree inductive algorithm, we may obtain the decision tree in Figure. 1 from the training set in Table 2. As we have already pointed out, each internal node represents a test on a single attribute and each branch represents the outcome of the test.

A path in the decision tree represents the set of attribute/value pairs that an example should exhibit in order to be classified as an example of the class labeled by the leaf node. For instance, given the above tree, the example Overlook = Sunny, Humidity = Low is classified as Yes, whereas the example Overlook = Sunny, Humidity = High is classified as No.

Notice that not all the attribute values have to be specified in order to find the classification of an example. On the other hand, if an example is too under-specified, it may lead to different, possibly incompatible, classifications. For instance, the example Overlook = Sunny can be classified both as Yes or No, following the two left-most branches of the tree: in this case many decision tree based classifiers make a choice by using probabilities assigned to each leaf. It is also worth noticing that the decision tree may not consider all the attributes given in the training set. For instance, the attribute Temperature is not taken into account at all in this decision tree.

2.2 Distributed Data Mining

The ever growing amount of data that are stored in distributed form over networks of heterogeneous and autonomous sources poses several problems to re-

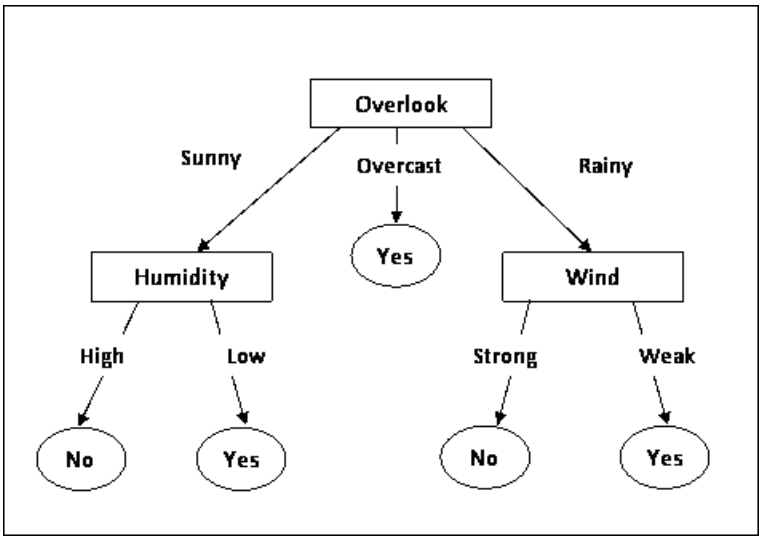


Figure 1: Decision Tree Example

Algorithm 2.2 ID3

function ID3 (R: a set of non-categorical attributes,
C: the categorical attribute, S: a training set) returns a decision tree;
1: begin
2: If S is empty, return a single node with value Failure;
3: If S consists of records all with the same value for the categorical attribute,
4: Return a single node with that value;
5: If R is empty, then
6: Return a single node with as value the most frequent of the values of the categorical attribute that are found in records of S; (note that then there will be errors, that is, records that will be improperly classified)
7: Let D be the attribute with largest Gain(D,S) among attributes in R;
8: Let {dj| j=1,2, .., m} be the values of attribute D;
9: Let {Sj| j=1,2, .., m} be the subsets of S consisting respectively of records with value dj for attribute D;
10: Return a tree with root labeled D and arcs labeled d1, d2, .., dm going respectively to the trees
11: ID3(R-D, C, S1), ID3(R-D, C, S2), .., ID3(R-D, C, Sm);
12: end ID3;

Table 3: ID3 Algorithm

search in knowledge discovery and data mining, such as communication minimization, autonomy preservation, scalability, and privacy protection.

DDM is a branch of the field of data mining that offers a framework to mine distributed data paying careful attention to the distributed data and computing resources.

As pointed out in [82], building a monolithic database, in order to perform non-distributed data mining, may be infeasible or simply impossible in many applications. The cost of transferring large blocks of data may be prohibitive and result in very inefficient implementations.

Surveys [56] and [77] provide a broad, up-to-date overview of DDM, touching on issues such as: clustering, association rule mining, basic statistics computation, Bayesian network learning, classification, and the historical roots of DDM. The collection of papers in Kargupta and Chan [54] describes a variety of DDM algorithms (ARM, clustering, classification, preprocessing, etc.), systems issues in DDM (security, architecture, etc.), and some topics in parallel data mining. Survey [110] discusses parallel and distributed association rule mining in DDM. Survey [111] discusses a broad spectrum of issues in DDM and parallel data mining and provides a survey of distributed and parallel association rule mining and clustering. Other DDM applications [52, 89] deal with continuous data streams. An overview of the data stream mining literature can be found in [10].

The bulk of DDM methods in the literature operate over an abstract architecture which includes multiple sites having independent computing power and storage capability. Local computation is done on each of the sites and either a central site communicates with each distributed site to compute the global models or a peer-to-peer architecture is used [87]. In the latter case, individual nodes might communicate with a resource rich centralized node, but they perform most of the tasks by communicating with neighboring nodes by message passing over an asynchronous network. For example, the sites may represent independent sensor nodes which connect to each other in an ad-hoc fashion.

Typically centralized control communication is a bottleneck. Since communication is assumed to be carried out exclusively by message passing, a primary goal of many DDM methods in the literature is to minimize the number of messages sent. Some methods also attempt to load-balance across sites to prevent performance from being dominated by the time and space usage of any individual site.

2.3 Agents and Multi-Agent Systems

Agents and multi-agent systems are an emergent technology that is expected to have a significant impact in realizing the vision of a global and informational rich services network to support dynamic discovery and interaction of digital enterprises. Significant work has already been done for more than a decade since agents have been claimed to be the next breakthrough in software development, resulting in powerful multi-agent platforms and innovative e-business applications.

2.3.1 Agents

Agents are defined by Wooldridge [107] as computer systems that are situated in some environment and are capable of autonomous action in this environment in order to meet their design objectives. Intelligent agents [86, 107] are defined as agents that can react to changes in their environment, have social ability (communication) and the ability to use computational intelligence to reach their goals by being proactive. Agents are active, task-oriented, modeled to perform specific tasks and are capable of autonomous action and decision making. The agent modeling paradigm can be looked at as a stronger encapsulation of localized computational units that perform specific tasks. This can be paraphrased as follows: an object, i.e. a software component in (say) a distributed system, does things because it has to; an agent does things because it decides it wants to (i.e. it does not have to).

2.3.2 Multi-Agent Systems

By combining multiple agents in one system to solve a problem, the resultant system is a multi-agent system (MAS) citewood03. These systems are comprised of agents that individually solve problems that are simpler than the overall system problem. They can communicate with each other and assist each other in achieving larger and more complex goals. Thus problems that software developers had previously thought of as being too complex [71] can now be solved, by localising the problem solving [101]. For example, Multi-agent systems have been used in predicting the stock market [69], industrial automation [104] and e-learning systems [36].

In general, MAS adhere to the following three primitives. First, MAS must specify appropriate communication and interaction protocols. Despite agents being the building blocks of a problem solving architecture, no individual problem could be effectively solved if no common communication ground and no action protocol exists. Secondly, MAS must be open and decentralized, with no prior knowledge of, for example, the number of participants or behaviors. In a running MAS, new agents may join at any time having only to conform to the communication protocol, being able to act on the way they choose, often in unpredictable manner. Finally, MAS may consist of possibly heterogeneous agents that are scattered around the environment and act autonomously or in collaboration.

2.3.3 MAS Development Platforms

To develop a multi-agent system effectively, the developers have to deal with several issues such as agent characteristics, agent functionalities, protocols, communication, coordination and co-operation. Furthermore agent-based systems should be robust, scalable and secure. To achieve this, the development of open, stable, scalable, and reliable architectures that allow agents to discover each other, communicate and offer services to one another is required.

To date several agent development platforms and toolkits have been produced [18]. AgentBuilder [97] is a tool for building Java agent systems based on two components: the Toolkit and the Run-Time System. The Toolkit includes tools for managing the agent software development process, while the Run-Time System provides an agent engine, that is, an interpreter, used as execution environment for agent software. AgentBuilder agents are based on a model derived by the Agent-0 [92] and PLACA [100] agent models.

AgentTool [31] is a graphical environment to build heterogeneous multi-agent systems. It is a kind of CASE tool, specifically oriented towards agent-oriented software engineering, whose major advantages are the complete support for the MaSE methodology (developed by the same authors together with the tool) and the independence from agent internal architecture (with MaSE and agentTool it is possible to build multi agent systems made of agents with different internal architectures). ASL [60] is an agent platform that supports the development in C/C++, Java, JESS, CLIPS and Prolog. ASL is built upon the OMGs CORBA 2.0 specifications. The use of CORBA technology facilitates seamless agent distribution and allows adding to the platform the language bindings supported by the used CORBA implementations. Initially, ASL agents used to communicate through KQML messages, now the platform is FIPA compliant supporting FIPA ACL.

Bee-gent [57] is a software framework to develop agent systems compliant to FIPA specification that has been realised by Toshiba. Such a framework provides two types of agents: wrapper agents used to agentify existing applications and mediation agents supporting the wrappers coordination by handling all their communications. Bee-gent also offers a graphic RAD tool to describe agents through state transition diagrams and a directory facility to locate agents, databases and applications.

Grasshopper-2 [14] is a pure Java based Mobile Agent platform, conformant to existing agent standards, as defined by the OMG - MASIF (Mobile Agent System Interoperability Facility) [73] and FIPA specifications. Thus Grasshopper-2 is an open platform, enabling maximum interoperability and easy integration with other mobile and intelligent agent systems. The Grasshopper-2 environment consists of several Agencies and a Region Registry, remotely connected via a selectable communication protocol. Several interfaces are specified to enable remote interactions between the distinguished distributed components. Moreover, Grasshopper-2 provides a Graphical User Interface (GUI) for user-friendly access to all the functionality of an agent system.

MOLE [13] is an agent system developed in Java whose agents do not have a sufficient set of features to be considered truly agent systems [37, 93]. However, MOLE is important because it offers one of the best supports for agent mobility. Mole agents are multi-thread entities identified by a globally unique agent identifier. Agents interact through two types of communication: through RMI for client/server interactions and through message exchanges for peer-to-peer interactions.

The Open Agent Architecture [70] is a truly open architecture to realise distributed agent systems in a number of languages, namely C, Java, Prolog,

Lisp, Visual Basic and Delphi. Its main feature is its powerful facilitator that coordinates all the other agents in their tasks. The facilitator can receive tasks from agents, decompose them and award them to other agents.

RETSINA [95] offers reusable agents to realise applications. Each agent has four modules for communicating, planning, scheduling and monitoring the execution of tasks and requests from other agents. RETSINA agents communicate through KQML messages.

Zeus [75] allows the rapid development of Java agent systems by providing a library of agent components, by supporting a visual environment for capturing user specifications, an agent building environment that includes an automatic agent code generator and a collection of classes that form the building blocks of individual agents. Agents are composed of five layers: API layer, definition layer, organizational layer, coordination layer and communication layer. The API layer allows the interaction with non-agentized world.

Shakshuki [88] presents a methodology for evaluating agent toolkits based on criteria such as availability, environment, development, characteristic properties and performance. Their findings recommended JADE [16] as one of the top-ranking toolkits according to their criteria. Chmiel et al. [24] performed experiments to test the efficiency and scalability of JADE. Their tests indicated that JADE is an efficient environment. They were able to run experiments with thousands of agents effectively migrating among eight machines and communicating by exchanging tens of thousands of ACL messages.

One of the aims of this chapter is to present our experience in using the JADE platform to engineer a real world multi-agent data mining system. JADE was chosen because it is open source, popular, easy to use and compliant with the Foundation for Physical Agents (FIPA) specifications [34]. FIPA is an international organization dedicated to promoting the industry of intelligent agents by openly developing specifications to support interoperability amongst agents and agent-based systems. FIPA defines a number of standards and specifications that include architectures to support inter-agent communication, and interaction protocols between agents, as well as communication and content languages to express the messages of these interactions. With the use of these standards, any FIPA-compliant platforms, and their agents, are able to seamlessly interact with any other FIPA complaint platform. JADE is described in detail in Section 3.

2.4 Multi-Agent Data Mining

There are two themes of agent and data mining interaction and integration in the literature [21]: data mining for agents, referred to as mining-driven agents [96], and agents for data mining, commonly known as multi-agent data mining (MADM). The former concerns issues of transforming the discovered knowledge, extracted by data mining, into the inference mechanisms or simply the behaviors of agents and multi-agent systems; as well as the arguable challenge of generating intelligence from data while transferring it to a separate, possibly autonomous, software entity. A FIPA-compliant multi-agent platform based on mining-driven

agents (Agent Academy) that offers facilities for design, implementation and deployment of multi-agent systems is proposed in [96]. The authors describe the Agent Academy as an attempt to develop a framework through which users can create an agent community having the ability to train and retrain its own agents using data mining techniques.

MADM, rather than mining-driven agent systems, is the focus of this chapter. It is concerned with the use of agent and multi-agent systems to perform data mining activities. The contribution of this section is to provide a broad review of prominent MADM approaches in the literature and a discussion of the benefits that agent-based data mining architectures provide in coping with such problems. This section is not concerned with particular data mining techniques. It is however concerned with collaborative work of distributed software in the design of multi-agent systems directed at data mining.

As mentioned in the previous section (Section 2.3.3), a basic approach for data mining is to move all of the data to a central data repository and then to analyze this with a single data mining system. Even though this guarantees accurate results of data analysis, it might be infeasible in many cases. An alternative approach is the in-place strategy in which all the data can be locally analyzed (local data model), and the local results at their local sites combined at the central site to obtain the final result (global data model). This approach is less expensive but may produce ambiguous and incorrect global results. DDM approaches require centralized control that causes a communication bottleneck that leads in turn to inefficient performance.

To make up for such a weakness, many researchers have spent great effort looking for more advanced approaches of combining local models built at different sites. Most of these approaches are agent-based high level learning strategies.

Several systems have been developed for multi-agent data mining. These systems can be categorized, according to their strategy of learning, into three types:

1. Central-learning
2. Meta-learning
3. Hybrid-learning

2.4.1 Central-learning Strategy

A central learning strategy is when all the data can be gathered at a central site and a single model can be built. The only requirement is to be able to move the data to the central location in order to merge it and then apply sequential DM algorithms.

This strategy is appropriate when the geographically distributed data is small. For large volumes of data, the strategy is generally very expansive but also more accurate in its DM results [12, 19]. The process of gathering data in general is not simply a merging step; it depends on the original distribution. For

example, different records may be placed in different sites, different attributes of the same records may be distributed across different sites, or different tables can be placed at different sites; therefore when gathering data it is necessary to adopt the proper merging strategy. However, as noted previously, this strategy in general is usually unfeasible because of security and privacy of data. Agent technology offers no great advantages in implementing this strategy, for which many KDD research groups have made software freely available for download; examples include [49, 105].

One of the earliest references to MADM adapting a central-learning strategy can be found in Kargupta et al. [55] who describe a parallel data mining system (PADMA) that uses software agents for local data accessing and analysis, and a web based interface for interactive data visualization. PADMA has been used in medical applications. Kargupta et al. describe a distributed data mining architecture and a set of protocols for a multi-agent software tool. Peng et al. [79] presented an interesting comparison between single-agent and multi-agent text classification in terms of a number of criteria including response time, quality of classification, and economic/privacy considerations. Their results indicate, not unexpectedly, in favour of a multi-agent approach.

2.4.2 Meta-learning Strategy

Meta-learning is the process of automatic induction of correlations between tasks and solving strategies, based on a domain characterization. Meta-learning methods have been widely used with data mining [20, 103], particularly in the area of classification and regression.

The meta-learning strategy offers a way to mine classifiers from homogeneously distributed data. It follows three main steps. The first is to generate base classifiers at each site using classifier learning algorithms. The second step is to collect the base classifiers at a central site, and produce meta-level data from a separate validation set and predictions generated by the base classifier on it. The third step is to generate the final classifier (meta-classifier) from meta-level data via a combiner or an arbiter. Copies of the classifier agent will exist, or be deployed, on nodes in the network being used (see for example [81]).

One of the most popular MADM approaches that adopt the meta-learning strategy is the METAL project [72] whose emphasis is on helping the user to obtain a ranking of suitable data mining algorithms through an online advisory system. Gorodetsky et al. [40] correctly consider that the core problem in MADM is not the data mining algorithms themselves (in many case these are well understood), but the most appropriate mechanisms to allow agents to collaborate. Gorodetsky et al. present a MADM system to achieve distributed data mining and, specifically, classification. A more recent system, proposed in [68], uses the MAGE middleware [90] to build an execution engine that uses a directed acyclic graph to formalize the representation of the KDD process. In [80] a multi-agent system for KDD (AgentDiscover) has been proposed. It uses task-based reasoning [22] for problem solving on a multi-agent platform. Perhaps the most mature agent-based meta-learning systems are: JAM [94],

BODHI [53], and Papyrus [11]. Papyrus is a specialized system which is designed for clusters while JAM and BODHI are designed for data classification. These are reviewed in detail in [62].

2.4.3 Hybrid-learning Strategy

A hybrid learning strategy is a technique that combines local and remote learning for model building [42]. An example of hybrid learning framework is Papyrus. Papyrus is designed to support both learning strategies. In contrast to JAM and BODHI, Papyrus can not only move models from site to site, but can also move data when that strategy is desired.

2.5 Summary

Most of the previously proposed MADM systems are used to improve the performance of one specific data mining task. Therefore, these systems can also be categorized into three groups based on the DM task they address:

- Classification Task: includes [35, 39, 40, 53, 59, 94].
- Association Rule Mining Task: includes [25, 50, 58, 74, 76].
- Clustering Task: includes [11, 30, 38, 61, 62].

Given the above literature review, and to the best knowledge of the authors, there have been only a few MADM systems that define a generic MADM framework. An early attempt was IDM [19]. It is a multiple agent architecture that attempts to do direct data mining that helps businesses gather intelligence about their internal commerce agent heuristics and architectures for knowledge discovery in databases. In [9] a generic task framework was introduced but was designed to work only with spatial data. The most recent system is introduced in [32] where the authors proposed a multi-agent system to provide a general framework for distributed data mining applications. The effort to embed the logic of a specific domain has been minimized and is limited to the customization of the user. However, although it is customizable feature is of a considerable benefit, it still requires users to have very good data mining knowledge.

Most of the MADM frameworks adapt similar architecture and provide common structural components using standard agent communication language that facilitates the interactions among agents such as KQML or FIPA-ACL.

The major criticism of such systems is that it is not always possible to obtain an exact final result, i.e. the global knowledge model obtained may be different from the one obtained by applying the one model approach (if possible) to the same data. Approximated results are not always a major concern, but it is important to be aware of their nature. Moreover, in these systems hardware resource usage is not optimized. If the "heavy" computational is always executed locally to the data, when the same data is accessed concurrently, the benefits coming from the distributed environment might vanish due to the performance

degradation. Another drawback is that occasionally, these models are induced from databases that have different schemas and hence are incompatible.

3 EMADS: REQUIREMENTS ANALYSIS, DESIGN, AND IMPLEMENTATION

As described in Section 1, the EMADS vision is that of an anarchic collection of persistent, autonomous (but cooperating) KDD agents operating across the Internet. This Section describes the under-pinning philosophy of EMADS. This Section concentrates on the generic features of EMADS and describes the agent framework to support the EMADS vision.

The motivation for researching and implementing a fully operational EMADS was to facilitate the investigation of the various MADM research challenges outlined in Section 1. Developing a data mining system that uses specialized agents with the ability to communicate with multiple information sources, as well as with other agents, allows for a great deal of flexibility. For instance, adding a new information source should merely imply adding a new agent and advertising its capabilities.

Throughout this chapter, depending on context, EMADS is referred to both as a "system", since it is a Multi-Agent System (MAS), and as "framework" because EMADS defines a framework for achieving MADM.

To realize the EMADS framework a general software development methodology was adopted that moved from problem definition and analysis to detailed design and implementation. This Section provides a detailed description of the framework in terms of its requirements, design and architecture. The Section commences by reviewing the EMADS requirements (Subsection 3.1), including the structural and operational requirements. A review of the EMADS Agents and Users is then presented in Subsection 3.4 in the context of the identified requirements. In Subsection 3.5 the EMADS agent interaction protocols are defined; followed, Subsection 3.6, with a discussion of the generic data mining process using EMADS Agents. In Subsection 3.7 details concerning Java Agent Development Environment (JADE) [16], the agent development toolkit that was used to implement EMADS, are presented. The JADE implementation of EMADS is then detailed in Subsection 3.8, together with a discussion of the implementation of the extendibility feature of EMADS. Finally, Subsection 3.9 presents a summary of this Section.

3.1 Requirements Analysis

EMADS has to fulfill the following requirements in order to satisfy the research objectives (see Section 1.2):

1. **Multiple Data Mining Tasks:** The framework must be able to provide mechanisms to allow the coordination of EMADS (data mining) tasks.

Note that the number and nature of the data mining tasks that the framework should be able to address is not known apriori, and is expected to evolve over time.

2. **Agent Coordination:** Following on from requirement 1, the framework must be reactive since it must accommodate new agents as they are created in the environment.

3. **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.

4. **Scalability:** There are potentially a large number of agents that must be coordinated by the EMADS framework. The framework must therefore be "light-weight" and scalable. In other words, it must be possible to implement efficient communication mechanisms, and the administrative overhead of the framework should not hamper the overall performance of the system. At the same time the framework must be scalable: avoiding centralized components which would create bottlenecks during execution.

5. **Extendibility**: EMADS must provide an extendible framework that can easily accept new data sources and new DM techniques.

6. **Multiple Data Sources:** 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.

7. **Multiple Data Formats:** The framework should be able to operate using data sources of both the same or heterogeneous formats.

8. **Multi-Platform:** 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 DM algorithm program.

In the following subsections the structure and operational requirements of the desired MADM framework are considered in detail. The extendibility requirements are discussed in Section 4.

3.2 Structural Requirements

The goal of the structural requirements analysis, described in this subsection, was to identify the flow of information through the desired framework and consequently clearly define the expected system input and output streams. By breaking the framework into domain level concepts, it was also possible to begin to identify the nature of the required agents. Four main domain level models were identified: (i) user interface, (ii) planning and management, (iii) processing, and (iv) data interface. The interaction (information flow) between these

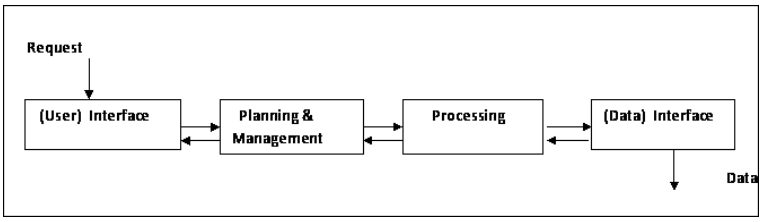


Figure 2: System Main Models

four models is shown in Figure 2. The Figure should be read from left to right. The user interface model receives DM 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 model). The identified data sources are then mined (the processing model), through access to the data interface model, and the results returned to the user via the user (interface) model.

3.3 Operational Requirements

Most current agent-based DM frameworks (see Section 2) share a similar high-level architecture, and provide common structural models, to that shown in Figure 2. Models of the form described above have become a template for most agent-based DM and information retrieval systems. The structure illustrated in Figure 2 sets out three important elements of MADM systems: (i) agent technology, (ii) domain models, and (iii) information brokerage (middleware). 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 (Foundation for Intelligent Physical Agents) [34] ACL. Domain models 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 retrieval and update requests can be properly routed to the relevant resources.

The general operation of EMADS (as suggested in Figure 2) is as follows:

1. When a request is received, select the appropriate information source or sources. One way to do this is using metadata 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.

2. Assign the appropriate mining algorithm(s).

3. 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.

4. Return the results to the user.

To simplify the issues of data heterogeneity, requirement number 6 (Section 3.1), in the case of queries that require input from multiple sources some global schema is assumed. This can be used to infer relationships between data sources, and unify heterogeneous data representations into a common object data model.

3.4 EMADS Agents and Users

In this section the different types of EMADS agent are detailed. The discussion focuses on the functionality of the agents; the implementation is considered later in this Section. Based on the requirements analysis in Section 3.1, several types of EMADS agent were identified. However, regardless of "type", EMADS agents adhere to the general agent definitions described in [107]. The EMADS agent types are: User, Management, Facilitator, Data Source, Mining, and Registration. Each falls within the domain level concepts specified in the requirement analysis phase. The User and Data Source Agents are all identified as interface agents because they all provide "interfaces" to either an outside application, other agents within the system, or data sources. User agents provide the interface between EMADS end users and the rest of the framework; whilst Data Source agents provide the interface between input data and the rest of the framework. Task Agents and DM Agents are identified as processing agents because they carry out the required "processing" needed to respond to user requests, and possibly, the pre-process data within the system.

Figure 3 shows what agents will reside on each layer (domain model) identified in the structural requirements analysis in Section 3.2. The Task Agent (middle of the management layer of Figure 3) receives a request and asks the Broker Agent to check all available databases (Data Agents) and Mining Agents to find: (i) which databases (tables) to use, and (ii) which data mining algorithms (held by Mining Agents) are appropriate. The task agent then passes the task to Mining 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. Figure 3 also shows the system level inputs and outputs as they flow from agent to agent.

Note that the agents were defined with respect to the high level objectives identified in the requirements presented previously in Section 3.1. In general, the structure of an EMADS agent may be divided into to three modules: (i) the interface module, (ii) the process module and (iii) the knowledge module.

The following subsections describe the structure and function of each agent from a high level perspective. The subsections also discuss the design decisions that were made and why they were made. The agents are considered in the same order as they might be utilized to process an EMADS user request in order to

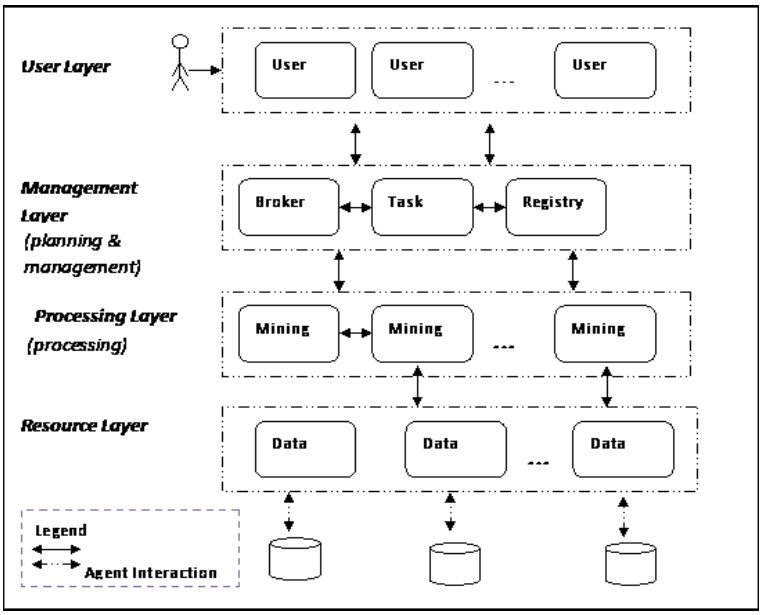


Figure 3: System General Architecture

illustrate how each agents function fits into the systems overall operation and contributes to supporting the other agents.

3.4.1 User Agent

There are three essential operations that a User Agent must be able to undertake:

1. Receive and interpret user data mining requests.
2. Present the generated results, in a suitable and easily understandable format, back to the user.
3. Expect and respond to asynchronous events, such as "a stop mining" or "end operations" instructions 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.

Note that 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 Agents 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 interaction, 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.

3.4.2 Task Agent

A Task Agent is responsible for the activation and synchronization 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 sections. 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. Note that 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.

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 that meet 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 and provides one 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.

3.4.3 Facilitator Agent (or Broker Agent)

The Facilitator 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 receives notification of a new agent who wants to advertise its services; it must add this new agents 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 a 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.

3.4.4 Data Mining (DM) Agent

A 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 tasks, 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.

3.4.5 Data Agent (or Resource Agent)

A Data 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. "Useful" in this case refers to Data Agents that are responsible for a data source that includes some part of the required data.

Note that an individual Data Agent is not specific to any particular 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 section) 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.

3.4.6 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 4. The supported participants (modes of operation) are as follows:

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

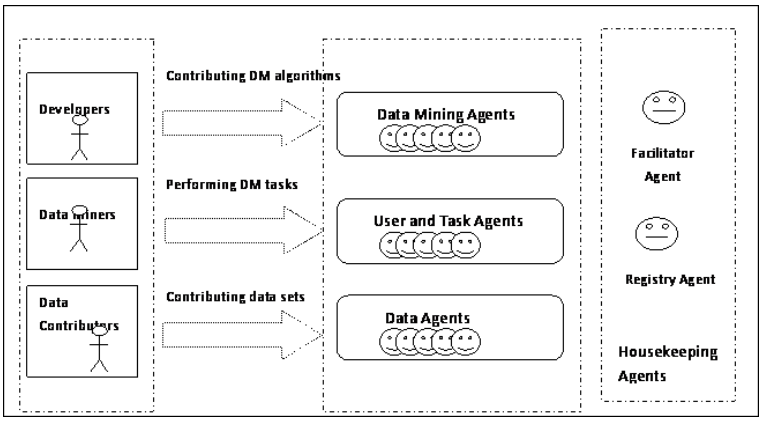


Figure 4: High level view of EMADS conceptual framework

Note that in each case, before interaction with EMADS can commence, the participant must download and launch the appropriate EMADS software. Note also 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.

3.5 Defining Interaction Protocols

In this section the various protocols required to support the identified primary functions and high-level interactions are defined for each agent. Each category of agent is considered in a separate subsection.

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:

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

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

$$< Send - Message - Tag >< Send - Content >$$
$$< Send - Action >$$

$$< Receive - Message >::=< Receive - Message - Tag > |$$
$$< Receive - Message - Tag >< Receive - Content > |$$
$$< Receive - Message - Tag >< Receive - Action > |$$
$$< Receive - Message - Tag >< Receive - Content >$$
$$< Receive - Action > |$$
$$\text{“(”} < Receive - Message > \text{“}OR\text{”} < Receive - Message > \text{“)”} |$$
$$< Receive - Message >< Receive - Message >$$

$< ProtocolName >::= \text{“}ProtocolName : \text{”}string$
$< SenderAgent >::= \text{“}Sender : \text{”}string$
$< ReceiverAgent >::= \text{“}Receiver : \text{”}string$
$< Send - Message - Tag >::= \text{“}Send : \text{”}string$
$< Send - Content >::= \text{“}Content : \text{”}string$
$< Send - Action >::= \text{“}Do : \text{”}string$
$< Receive - Message - Tag >::= \text{“}Receive : \text{”}string$
$< Receive - Content >::= \text{“}Content : \text{”}string$
$< Receive - Action >::= \text{“}Do : \text{”}string$

The first three lines of each protocol (the header) identifies the protocols 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. Note that in some cases there may be a number of alternative message 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 sequenced of replies. This format is used in the following subsections to define the seven main EMADS protocols that have been identified:

1. Find Other Agents
2. Agent Registration
3. User DM Request
4. Start Data Mining

5. Advertise Agent Capabilities
6. Perform Data Mining
7. Data Retrieval

The following subsections briefly describe each of these protocols.

3.5.1 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)

Note that 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.

3.5.2 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 agents 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.

3.5.3 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)

3.5.4 Starting Data Mining 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 DM. 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 described in Section 3.5.1 above. Thus 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 (see the User DM Request Protocol defined above). 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 protocol is given below.

Protocol Name: Start Data Mining
Sender: Task Agent
Receiver: DM Agent
send: beginMining
receive: miningComplete
content: result
do: return result
or
do: terminate

Note that 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.

3.5.5 Advertise Agent Capabilities Protocol

The Facilitator Agent primarily interacts with the Task, DM, 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 agents

information. The advertising process begins with the new agent sending an "advAgent" message that contains the new agents 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 agents 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.

3.5.6 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) and DM 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 agents data mining 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

Note that the protocol includes an option to combine results where several DM Agents or Data agents may have been used.

3.5.7 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

3.6 Data Mining with EMADS Agents

In the foregoing sections the various EMADS Agents and protocols were defined (Sections 3.4 and 3.5 respectively). In this section a generic overview of data mining with EMADS is presented. The overview is described by considering a general DM request and tracing the generation of the response through the various communication paths. The 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 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 agents. The Facilitator then returns the list of "useful" DM and Data Agents to the Task Agent. Once the Task Agent receives the list the task agent can commence the desired DM process. The nature of this 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 the following sections. 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. Once 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.

3.6.1 DM Task Planning and Flow of System Operations for One Possible Scenario

In the context of EMADS DM task planning is realized by negotiation between the EMADS end user, and EMADS agents through the message passing mechanism (described further in Section 3.8 below). The general form of the EMADS

DM process has already been partially described above in terms of the protocols used; the objective of this subsection is to bring the different elements of the process together in the form of a summary. The summary is presented by considering a generic example. The DM process commences when a user agent receives a request from an end user regarding the performance of a particular DM task. The request will include information about the task (i.e. attributes, attribute type (numeric or nominal)) associated with the request. When the User Agent receives the request from the user, it negotiates with the system facilitator to determine which processes to "talk to" for this task (i.e. which DM and Data agents are to be used). For example, if the user wants to acquire all possible association rules meeting given minimum support and confidence levels, across all available data sources, then the task agent must ask all data sources for association rules/data. It would be time consuming and wasteful to ask sources (agents) which have data unsuited to the association rule mining request. When the mining task is completed the Task Agent returns the results the User Agent from where they are passed on to the user.

3.7 The Agent Development Toolkit

In this section a discussion is presented concerning the selected agent development toolkit, JADE, in which EMADS was implemented. It is well established that 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 a more 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 differences between most development toolkits lies 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 toolkits 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, organization, 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 complaint.

Some of the reasons that 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.

Further detail concerning JADE is provided in the next subsection.

3.7.1 JADE

As noted above JADE is a software environment, fully implemented in the JAVA programming language, directed at the development of MAS. As described in Section 2 JADE is a FIPA compliant middleware 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 5. The set of one or more "active" containers is collectively referred to as a platform. 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, 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 lifecycles 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 recognized and found by other agents.

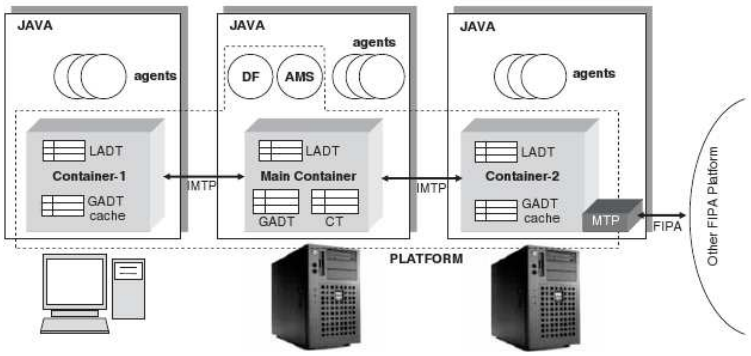


Figure 5: JADE Architecture (Bellifemine et al., 2007) [15]

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. ACL message communication between agents within the same container uses event dispatching. Message communication between agents in the same JADE platform, but in different containers, is founded on RMI. 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 recognized issue within the JADE user community. Research has been reported the 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 (discussed further in Section 3.7.2 below).

All agent communications are performed through message passing using the FIPA ACL. Each agent is equipped with an incoming message box, and message polling can be blocking or non-blocking with an optional timeout.

According to the structure of the JADE protocols, 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.

3.7.2 JADE Agent Interaction

As noted above, 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 initialized, 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 run time. Each behaviour performs its designated operation be 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).

3.8 EMADS Architecture as Implemented in Jade

This section describes the implementation of the different facets of EMADS as described in the foregoing. EMADS is implemented using the JADE MAS development framework; the rationale for this was described in Section 3.4. Some background details of JADE were presented in Section 3.7.1. Broadly JADE defines an agent platform that comprises a set of containers, which may (as in the case of EMADS) be distributed across a network. The section commences

with an overview of the EMADS JADE implementation. More specific details of the JADE implementation focusing on: Agent "Behaviours", agent interaction, mechanisms for cooperation, user request handling and extendibility; are all presented in the following subsections.

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 similar to the schema described in Section 3.5. JADE supports three types of message communication as follows:

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

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

3. Inter-Platform: Message communication between agents in different platforms uses the IIOP (Internet Inter-ORB Protocol).

Note that the last is facilitated by a special Agent Communication Channel (ACC) agent also located in the JADE platform main container. 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 (see Subsection 3.7.1) and the Agent Management System (AMS). Both the main container and the remaining containers can hold various DM agents. Note that the EMADS main container is located on the EMADS host organization site (currently the University of Liverpool in the UK), 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 four categories of agents: User Agents, Task Agents, DM Agents and Data Agents. It should be noted that EMADS containers may contain both mining and data agents simultaneously as well as user agents. DM and Data Agents are persistent, i.e. they continue to exist indefinitely and are not created for a specific DM exercise. Communication between agents is facilitated by the EMADS network.

Figure 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.

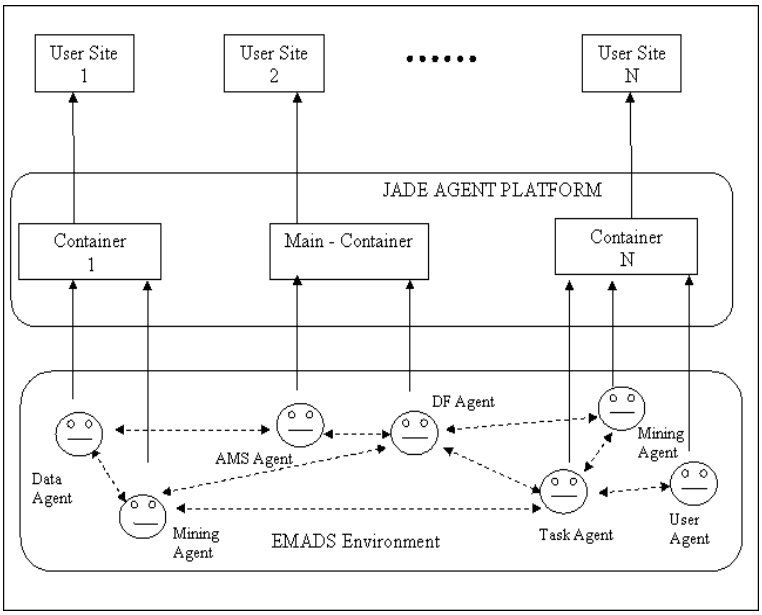


Figure 6: EMADS Architecture as Implemented in Jade

3.8.1 Mapping EMADS Protocols to JADE Behaviours

Many important design issues were considered while implementing EMADS within the JADE framework; including:

1. ACL Messages (protocols, content),
2. Data structures,
3. Algorithms and software components.

The ACL messages were defined with respect to the JADE ACL Message class fields [16] and the FIPA ACL Message Structure Specification [34].

The procedure to map the EMADS agent interaction protocols, as defined in Section 3.5, to JADE behaviours 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 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.
- WakerBehaviour: 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 provide by JADE. These composite behaviour classes include:

- SequentialBehaviour: Implementing a composite task that schedules its sub-tasks sequentially.
- FSMBehaviour: Implementing a composite task that schedules its sub-tasks according to a Finite State Machine (FSM) model.

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

3.8.2 Agent Interactions

A user agent, as shown in Figure 6, 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. In this subsection the interaction mechanism between agents is reviewed.

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; note that 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 communication 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 that should be noted are:

1. The mechanism whereby a collection of agents can be harnessed to identify a "best solution".

2. The process whereby new agents connect to the facilitator and registering their capability specifications.

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

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

3.8.3 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:

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

2. Request Posting: where User Agents (requesters of services) construct requests and relay them to a Task Agent, and

3. Processing: where the Task Agent coordinates the efforts of the appropriate service providers (Data Agents and DM Agents) to satisfy the request.

Note that Stage 1 (service registration) is not necessarily immediately followed by stage 2 and 3, it is possible that a providers 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; this is therefore described in more detail in the following subsection.

3.8.4 User Request Handling

A Task Agent is designed to handle a user request. This involves a three step process:

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

2. Optimization: Optimization of the complete task, including parallelization where appropriate; and

3. 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 facilitators knowledge of the capabilities of the available EMADS agents (and possibly of other facilitators, in a multi-facilitator system). The facilitator may also use information specified by the user (such as threshold values). 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 optimization 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.

3.9 Summary

This Section discussed the EMADS requirements, architecture, design and implementation. EMADS was envisioned as a collection of data sources scattered over a network, accessed by a group of DM agents that allow a user to data mine those data sources without needing to know the location of the supporting data, nor how the various agents interact. Additionally the expectation is that EMADS will "grow" as individual users contribute further data and DM algorithms.

In EMADS, as with most MAS, individual agents have different functionality; the system currently comprises: data agents, user agents, task agents, data mining agents and a number of "house-keeping" agents. Users of EMADS may be data providers, DM algorithm contributors or miners of data. The independence of EMADS from any particular DM function, in conjunction with the object oriented design adopted, ensures the system's capability to incorporate and use new data mining algorithms and tools.

In the following section a detail description of the design and implementation of the extendibility feature of EMADS is provided.

4 EMADS EXTENDIBILITY

As described in Section 1, one of the principal objectives of the research described in this chapter, and consequently the EMADS framework, is to provide an extendible framework that can easily accept new data sources and new DM techniques. This was also identified in the list of requirements presented in Section 2. In general, extendibility can be defined as the ease with which software can be modified to adapt to new requirements, or changes in existing requirements. In the context of EMADS the intention is to provide an extendible framework that can easily accept new data sources and new DM techniques. Adding a new data source or DM techniques to EMADS meditates the in-

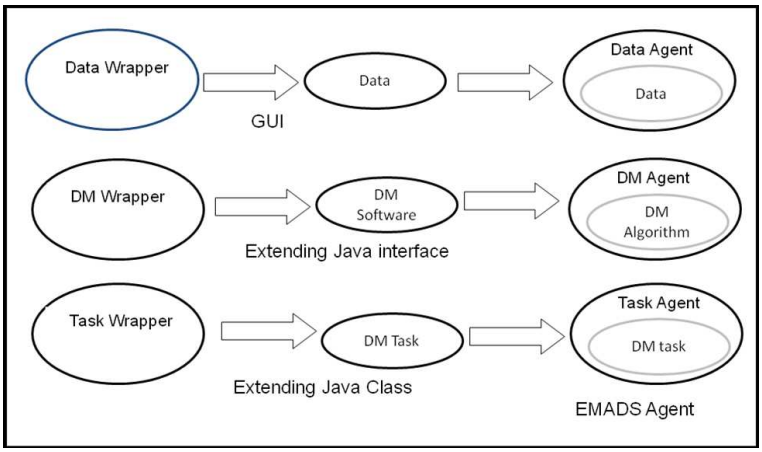


Figure 7: EMADS Wrappers

troduction of new agents. The EMADS framework was therefore designed to provide mechanism for the simple addition of agents by the EMADS body of end users.

This Section provides a detailed description of the EMADS extendibility feature in terms of its requirements, design and implementation. The principal means for achieving this is through the use of a system of wrappers. The Section commences (Subsection 4.1) by describing the wrapper concept. The section also gives an overview of the LUCS-KDD DN software which has been incorporated into the Data Wrapper agent in Subsection 4.1.4. The Section also discusses (Subsection 4.2) how, conceptually, the functionality of EMADS can be extended by adding new Task Agents and by adding Data Mining (DM) algorithms (agents) and new data sources using the wrappers concept, then the Section ends with a summary (Subsection 4.3).

4.1 Wrappers

The incorporation of data and data mining software is facilitated by a system of wrappers which allows for easy extendibility of the system. The term wrapper in the context of computer science has a number of connotations (for example "driver wrappers", "TCP wrappers" and the Java "wrapper" classes. In the context of MAS the term is used to describe a mechanism for allowing existing software systems to be incorporated into a MAS. The wrapper "agentifies" an existing application by encapsulating its implementation. The wrapper manages the states of the application, invoking the application when necessary [37].

As illustrated in Figure 7, EMADS wrappers are used to "wrap" up data mining artifacts so that they become EMADS agents and can communicate with other agents within EMADS. As such EMADS wrappers can be viewed as agents in their own right that are subsumed once they have been integrated with data

or tools to become data mining agents. The wrappers essentially provide an application interface to EMADS that has to be implemented by the end user, although this has been designed to be a fairly trivial operation. As shown in Figure 7, EMADS supports three broad categories of wrappers:

1. Data wrapper,
2. DM wrapper,
3. Task wrapper.

The first is used to create data agents, the second to create DM agents, and the third to create DM task agents. Each is described in further detail in the following three subsections.

4.1.1 Data Wrappers

In the context of EMADS, and with respect to standard data mining in general, a data source is a single text file containing data records (one line per record); where each record comprises a set of, typically comma or space separated, alpha-numeric values that subscribe to some attribute schema. This is a fairly standard tabular data format used throughout the KDD community. Data wrappers are therefore used to "wrap" a data source and consequently create a Data Agent. Conceptually the data wrapper provides the interface between the data source and the rest of the framework. Broadly a data wrapper holds: (i) the location (file path) of a data source, so that it can be accessed by other agents; and (ii) meta information about the data. To assist end users, in the application of a data wrapper to their data, a data wrapper GUI was developed. As described previously, once created, the data agent announces itself to the EMADS Facilitator (Broker) Agent as a consequence of which it becomes available to all EMADS users.

4.1.2 DM Wrappers

DM wrappers are used to "wrap" up DM software systems and to create DM agents. Generally the software systems to be wrapped will be DM tools of various kinds (classifiers, clusters, association rule miners, etc.) although they could also be (say) data normalization/discretization or visualization tools. Unlike data wrappers, DM wrappers are not supported by a GUI facility to aid their usage; instead EMADS users are expected to encode the wrapper themselves. However, the design of the wrapper is such that this would be a straight forward process.

4.1.3 Task Wrappers

It is intended that the framework will incorporate a substantial number of different tool wrappers each defined by the nature of the desired I/O which in turn will be informed by the nature of the generic DM tasks that it is desirable for

EMADS to be able to perform. Thus, EMADS users are expected to encode the wrapper themselves.

4.1.4 Discretization/Normalization

So as to avoid issue of data heterogeneity (and the consequent potential use of ontologies) all data was assumed to correspond to a common global schema.

An example of demonstrating of EMADS extendibility feature is the integration of the LUCS-KDD in Data DN with EMADS data agent. The LUCS-KDD (Liverpool University Computer Science - Knowledge Discovery in Data) DN (Discretization/ Normalization) software had been developed as a standalone application to convert data files available in the UCI data repository [17] into a binary format suitable for use with Association Rule Mining (ARM) applications. The software can, of course, equally well be used to convert data files obtained from other sources.

Discretization and normalization can be defined as follows: Discretization is the process of converting the range of possible values associated with a continuous data item (e.g. a double precision number) into a number of sub-ranges each identified by a unique integer label; and converting all the values associated with instances of this data item to the corresponding integer labels. Normalization is process of converting values associated with nominal data items so that they correspond to unique integer labels.

4.2 Implementation

As noted in the introduction to this Section, one of the most important features of EMADS, and the central theme of this chapter, is the simple extendibility requirement of the system, thus the ability for new agents to be added to the system in an effective and efficient manner. The rational is that the operation of EMADS should be independence of any particular DM method. The inclusion of a new data source or DM techniques necessitates the addition of new agents to the system. The process of adding new agents should therefore be as simple as possible. This subsection considers the implementation of the desired extendibility feature by considering each of the three categories of agents (Task, Data and DM) that may be contributed by EMADS users. It should be noted, before considering each of these categories in detail, that extendibility is achieved with respect to Task agents using an object oriented software design and implementation mechanism; while in the context of Data and DM Agents extendibility is achieved using the wrapper concept introduced earlier. Remember that in isolation wrappers can be viewed as agents in their own right which are used to build Data and DM agents, once incorporated with DM software or data the wrapper agents cease to exist in their own right as their functionality merges into the newly created Data or DM agent.

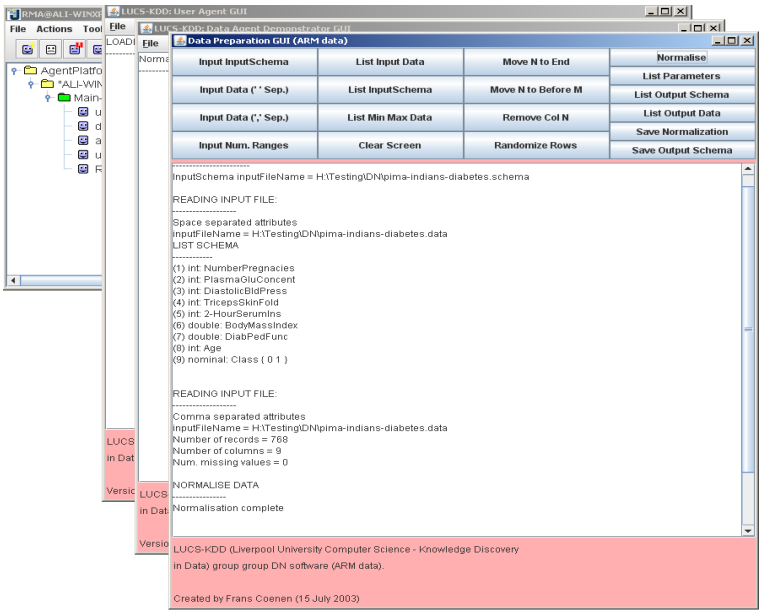


Figure 8: Data Normalization GUI

4.2.1 Task Agents

Task agents, that perform generic DM tasks, are introduced into EMADS using a predefined abstract task agent class. This is facilitated by the Object Orientation features provided by the Java programming language in which EMADS agents are expected to be implemented. Individual task agents are created by implementing a specific sub-class of this abstract class. The abstract task class defines a number of fields and method headers that are inherited by the user created sub-class used to define a specific task agent.

In effect these variables and methods define a simple and minimal interface that all task agent sub-classes are expected to comply with. As long as an agent conforms to this interface, it can be introduced and used immediately as part of the EMADS system.

4.2.2 Data Agents

New Data Agents are introduced using the predefined data wrapper introduced above. An EMADS Data Agent encapsulates a resource interface for data source specific retrieval that can be accessed by EMADS DM Agents. The resource interface holds all the information needed to interface with the specific format data source.

In the current version of the architecture, EMADS supports only flat-file access. A single Data Agent controls each separate data source. Thus, in order to bring a new data source into the system, a new Data Agent must be

instantiated with the required components. This is achieved using a purpose built software wrapper, implemented in Java and interfaced using a GUI, to facilitate the addition of new data agents by EMADS users. The user must use the Data Agent wrapper GUI (Figure 9) to refer to the data source location (file path) and provide meta information describing the data (i.e. data type, number of classes etc...). The user can also use the Data Agent GUI to launch the LUCS-KDD DN tool (Figure 8), if the data requires normalization before introducing it to EMADS. The LUCS-KDD DN tool is described further at the end of this Section.

4.2.3 DM Agents

The addition of DM Agents to the EMADS system is facilitated by a purpose built wrapper (agent). The DM wrapper agent is said to "wrap" a DM algorithm. The wrapper includes a "beginMining" method which calls a "doMine" method which must be incorporated into the mining algorithm. In effect the DM wrapper Agent can be considered to be used to hold a mining algorithm instance.

The DM wrapper Agent is implemented in Java and is designed to be an extendible component in its own right, since it can be changed or modified. As such the DM wrapper Agent is currently only compatible with DM software systems written in Java (there is no facility to incorporate foreign code). The wrapper defines an abstract interface (the mining interface) that a DM algorithm class must implement. First, the interface in which the algorithm is encapsulated must have a method that can be called to start the algorithm. This is implemented with the abstract "doMine" method; such a method must therefore be included in any DM algorithm to be wrapped. When called by the DM Agent, this method should start the algorithm operating over the data source. After the DM algorithm has finished, the DM Agent passes the results back to the Task Agent.

Because the DM algorithm must have access directly to data, it must be able to have direct visibility to it. It maintains this visibility by storing a pointer to the data as a local variable that is set through a "setResource" method. The "setResource" method is also an abstract method, and is called by the DM Agent to let the mining algorithm know the dataset it will be operating on. This component will be one of the components required when the system is extended and a new DM algorithm is added. Because of this, special consideration is being given to making it as "extendible" as possible. The DM algorithm must be able to return the result of the data mining for which it is responsible. In order to ensure this, the abstract method "getResult" is implemented that gets the results from the DM algorithm. Because each algorithm may process its data differently or not at all, the "getResult" method knows how to retrieve this.

The fact that the mining interface is an abstract interface allows any methods required in the implementation of a specific mining algorithm, to be added. It also ensures that the DM Agent can start the algorithm running, no matter

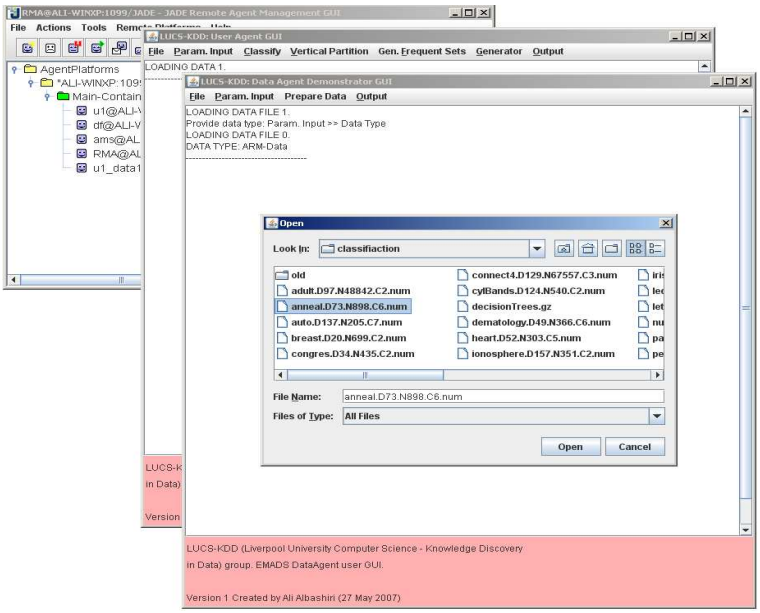


Figure 9: Data Agent GUI

what specific implementation is used. It also provides a great deal of flexibility.

4.3 Summary

This Section discussed the EMADS Extendibility requirements, architecture, design and implementation. The expectation is that EMADS will "grow" as individual users contribute data and DM algorithms. The incorporation of data and data mining software is facilitated by a system of wrappers which allows for easy extendibility of the system.

The independence of EMADS from any particular DM function, in conjunction with the object oriented design adopted, ensures the system's capability to incorporate and use new data mining algorithms and tools. As discussed above, introducing a new technique requires the sub-classing of the appropriate abstract class or the implementation of an abstract interface and the encapsulation of the tool within an object that adheres to the minimal interface. In fact, most of the existing implemented algorithms have similar interfaces already. This "plug-and-play" characteristic makes EMADS a powerful and extensible DM facility, and allows developers to employ their pre-implemented programs within EMADS agents.

5 FREQUENT SET META MINING: Meta ARM

In this Section the support that EMADS provides for the dynamic creation of communities of data mining agents is demonstrated by considering a Meta ARM (Association Rule Mining) scenario. The motivation behind the scenario is that data relevant to a particular ARM application is often owned and maintained by different, geographically dispersed, organizations. Information gathering and knowledge discovery from such distributed data sources typically entails a significant computational overheads; computational efficiency and scalability are both well established critical issue in data mining [51]. One approach to addressing problems such as the meta ARM problem is to adopt a distributed approach. However this requires expensive computation and communication costs.

The term *meta mining* describes the process of combining the individually obtained results of N applications of a data mining activity. This is typically undertaken in the context of Multi-Agent Data Mining (MADM) where the individual owners of agents wish to preserve the privacy and security of their raw data but are prepared to share the results of data mining activities. The mining activities in question could be, for example, clustering, classification or Association Rule Mining (ARM); the scenario described here concentrates on the latter — frequent set meta mining (which in this Section will be referred to as Meta ARM).

The Meta ARM problem is defined as follows: a given ARM algorithm (does not have to be the same algorithm) is applied to N raw data sets producing N collections of frequent item sets. Note that it is assumed each raw data set conforms to some globally agreed attribute schema, although each local schema will typically comprise some subset of this global schema. The objective is then to *merge* the different sets of results into a single *meta* set of frequent itemsets with the aim of generating a set of ARs or alternative a set of Classification Association Rules (CARS).

The most significant issue when combining groups of previously identified frequent sets is that wherever an itemset is frequent in a data source A but not in a data source B a check for any contribution from data source B is required (so as to obtain a global support count). The challenge is thus to combine the results from N different data sources in the most computationally efficient manner. This in turn is influenced predominantly by the magnitude (in terms of data size) of returns to the source data that are required.

There are a number of alternative mechanisms whereby ARM results can be combined to satisfy the requirements of Meta ARM. In this Section a study is presented comparing five different approaches (including a bench mark approach). The study is conducted using variations of the TFP set enumeration tree based ARM algorithm ([29,41]), however the results are equally applicable to other algorithms (such as FP growth [45]) that use set enumeration tree style structures, the support-confidence framework and an Apriori methodology of

processing/building the trees.

The Section is organised as follows. In Subsection 5.1 some background and related work is presented and discussed. A brief note on the data structures used by the Meta ARM algorithms is then presented in Subsection 5.2. The five different approaches that are to be compared are described in Subsection 5.3. This is followed, in Subsection 5.5.1, by an analysis of a sequence of experimental results used to evaluate the approaches introduced in Subsection 5.3. Finally some conclusions are presented in Subsection 5.6.

5.1 Background and Previous Work

As discussed in the previous sections, MADM research encompasses many issues. In this Section the issue of collating data mining results produced by individual agents is addressed, which is referred to as *meta-mining*.

The Meta ARM problem (as outlined in the above introduction) has similarities, and in some cases overlap, with incremental ARM (I-ARM) and distributed ARM. The distinction between I-ARM, as first proposed by Agrawal and Psaila [3], and Meta ARM is that in the case of I-ARM we typically have only two sets of results: (i) a large set of frequent itemsets D and (ii) a much smaller set of itemsets d that we wish to process in order to update D. In the case of Meta ARM there can be any number of sets of results which can be of any (or the same) size. Furthermore, in this case each contributing set has already been processed to obtain results in the form of locally frequent sets. I-ARM algorithms typically operate using a relative support threshold [63, 66, 102] as opposed to an absolute threshold; the use of relative thresholds has been adopted in the work described here. I-ARM algorithms are therefore supported by the observation that for an itemset to be *globally frequent* it must be *locally frequent* in at least one set of results regardless of the relative number of records at individual sources (note that this only works with relative support thresholds). When undertaking I-ARM four comparison options can be identified according to whether a given itemset i is: (i) frequent in d, and/or (ii) frequent in D; these are itemised in Table 4.

From the literature, three fundamental approaches to I-ARM are described. These may be categorised as follows:

1. Maintain itemsets on the border with maximal frequent item sets (the *negative border* idea) and hope that this includes all those itemsets that may become frequent. See for example ULI [99].

2. Make use of a second (lower) support threshold above which items are retained (similar idea to negative border). Examples include AFPIM [63]) and EFPIM [66].

3. Acknowledge that at some time or other we will have to recompute counts for some itemsets and consequently maintain a data structure that (a) stores all support counts, (b) requires less space than the original structure and (c) facilitates fast look-up, to enable updating. See for example [64].

	Frequent in d	Not frequent in d
Frequent in D (retained itemsets)	Increment total count for i and recalculate support	i may be globally supported, increment total count for i and recalculate support
NotFrequent in D	May be globally supported, need to obtain total support count and recalculate support (Emerging itemset)	Do nothing

Table 4: I-ARM itemset comparison options (relative support)

Using a reduced support threshold results in a significant additional storage overhead. For example if we assume a given data set with 100 ($n = 100$) attributes where all the 1 and 2 item sets are frequent but none of the other itemsets are frequent, the negative border will comprise 161700 item sets ($\frac{n(n-1)(n-2)}{3!}$) compared to 4970 supported item sets ($n + \frac{n(n-1)}{2!}$).

The data at each source is held using a P-tree (**P**artial Support Tree) data structure, the nature of which is described in further detail in Section 5.2 below. The distinction between distributed mining and MADM is one of control. Distributed ARM assumes some central control that allows for the global partitioning of either the raw data (*data distribution*) or the ARM task (*task distribution*), amongst a fixed number of processors. MADM, and by extension the Meta ARM mining described here, does not require this centralised control, instead the different sets of results are produced in an autonomous manner without any centralised control. MADM also offers the significant advantage that the privacy and security of raw data belonging to individual agents is preserved, an advantage that is desirable for both commercial and legal reasons.

In both I-ARM and distributed ARM, as well as Meta ARM, the raw data typically conforms to some agreed global schema.

Other research on meta mining that includes work on meta classification. Meta classification, also sometimes referred to as meta learning, is a technique for generating a *global* classifier from N distributed data sources by first computing N *base* classifiers which are then collated to build a single *meta* classifier [81] in much the same way that we are collating ARM results.

The term *merge mining* is used in Aref et al. [8] to describe a generalised form of incremental association rule mining which has some conceptual similarities to the ideas behind Meta ARM described here. However Aref et al. define merge mining in the context of time series analysis where additional data is to be merged with existing data as it becomes available.

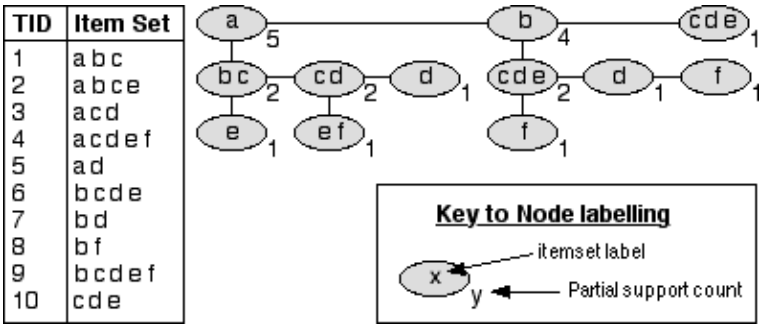


Figure 10: P-tree Example

5.2 Note on P and T Trees

The Meta ARM algorithms described here make use of two data structures, namely P-trees and T-trees. The nature of these structures is described in detail in [29, 41]; however, for completeness a brief overview is presented here.

The P-tree (**P**artial support tree) is a set enumeration tree style structure with two important differences: (i) more than one item may be stored at any individual node, and (ii) the tree includes partial support counts. The structure is sued to store a compressed version of the raw data set with partial support counts obtained during the reading of the input data. The best way of describing the P-tree is through an example such as that given in Figure 10. In the figure the data set given on the left is stored in the P-tree on the right. The advantages offered by the P-tree are of particular benefit if the raw data set contains many common leading sub-strings (prefixes). The number of such sub-strings can be increased if the data is ordered according to the frequency of the 1-itemsets contained in the raw data. The likelihood of common leading sub-strings also increases with the number of records in the raw data.

The T-tree (**T**otal support tree) is a "reverse" set enumeration tree structure that inter-leaves node records with arrays. It is used to store frequent item sets, in a compressed form, identified by processing the P-tree. An example, generated from the P-tree given in Figure 10, is presented in Figure 11.

From the figure it can be seen that the top level comprises an array of references to node structures that hold the support count and reference to the next level (providing such a level exists). Indexes equate to itemset numbers although for ease of understanding in the figure letters have been used instead of numbers. The structure can be though of as a "reverse" set enumeration tree because child nodes only contain itemsets that are lexicographically before the parent itemsets. This offers the advantage that less array storage is required (especially if the data is ordered according to the frequency of individual items.

The T-tree is generated using an algorithm called Total From Partial (TFP) which is also described in [29, 41]. The TFP algorithm is essentially an Apriori style algorithm that proceeds in a level by level manner. At each level the P-tree

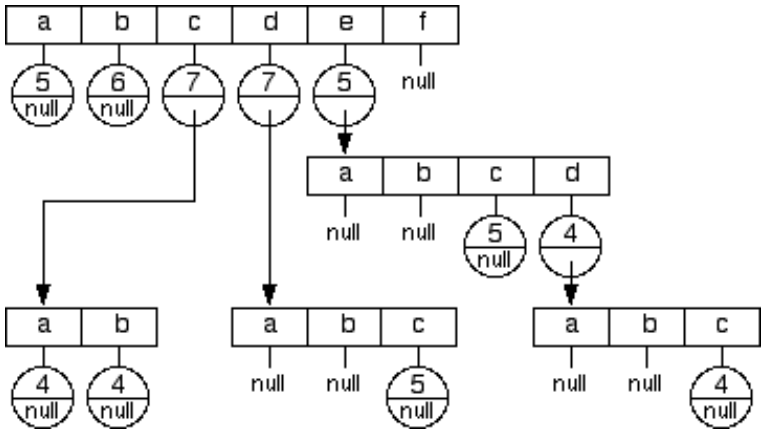


Figure 11: T-tree Example (support = 35%)

is processed to generate appropriate support counts. Note that on completion of the TFP algorithm the T-tree contains details of all the supported itemsets, in a manner that provides for fast look up during AR generation, but no information about unsupported sets (other than that they are not supported). Referring to Figure 11 unsupported sets are indicated by a *null* reference.

5.3 Proposed Meta ARM Algorithms

In this section a number of Meta ARM algorithms are described, an analysis of which is presented in section 5.5.1. It is assumed that each data source will maintain the data set in either its raw form or a compressed form. For the experiments reported here the data has been stored in a compressed form using a P-tree (Figure 10).

The first algorithm developed was a bench mark algorithm, against which the identified Meta ARM algorithms were to be compared. This is described in 5.3.1. Four Meta ARM algorithms were then constructed. For the Meta ARM algorithms it was assumed that each data source would produce a set of frequent sets using the TFP algorithm with the results stored in a T-tree. These T-trees would then be merged in some manner.

Each of the Meta ARM algorithms described below makes use of *return to data* (RTD) lists, one per data set, to contain lists of itemsets whose support was not included in the current T-tree and for which the count is to be obtained by a return to the raw data. RTD lists comprise zero, one or more tuples of the form $< I, sup >$, where I is an item set for which a count is required and *sup* is the desired count. RTD lists are constructed as a Meta ARM algorithm progresses. During RTD list construction the *sup* value will be 0, it is not until the RTD list is processed that actual values are assigned to *sup*. The processing of RTD lists may occur during, and/or at the end of, the Meta ARM process depending on the nature of the algorithm.

5.3.1 Bench Mark Algorithm

For Meta ARM to make sense the process of merging the distinct sets of discovered frequent itemsets must be faster than starting from the beginning (otherwise there is no benefit from undertaking the merging). The first algorithm developed was therefore a bench mark algorithm. This was essentially an Apriori style algorithm (see Table 5) that used a T-tree as a storage structure to support the generation process.

5.3.2 Brute Force Meta ARM Algorithm

The philosophy behind the Brute Force Meta ARM algorithm was that we simply fuse the collection of T-trees together adding items to the appropriate RTD lists as required. The algorithm comprises three phases: (i) merge, (ii) inclusion of additional counts and (iii) final prune. The merge phase commences by selecting one of the T-trees as the initial merged T-tree (from a computational perspective it is desirable that this is the largest T-tree). Each additional T-tree is then combined with the merged T-tree "sofar" in turn. The combining is undertaken by comparing each element in the top level of the merged T-tree sofar with the corresponding element in the "current" T-tree and then updating or extending the merged T-tree sofar and/or adding to the appropriate RTD lists as indicated in Table 6 (remember that we are working with relative support thresholds). Note that the algorithm only proceeds to the next branch in the merged T-tree sofar if an element represents a supported node in both the merged and current T-trees. At the end of the merge phase the RTD lists are processed and any additional counts included (the *inclusion of additional counts* phase). The final merged T-tree is then pruned in phase three to remove any unsupported frequent sets according to the user supplied support threshold (expressed as a percentage of the total number of records under consideration).

5.3.3 Apriori Meta ARM Algorithm

In the Brute Force approach the RTD lists are not processed until the end of the merge phase. This means that many itemsets may be included in the merged T-tree sofar and/or the RTD lists that are in fact not supported. The objective of the Aprori Meta ARM algorithm is to identify such unsupported itemsets much earlier on in the process. The algorithm proceeds in a similar manner to the standard Apriori algorithm (Table 5) as shown in Table 7. Note that items are added to the RTD list for data source n if a candidate itemset is not included in T-tree n.

5.3.4 Hybrid Meta ARM Algorithm 1 and 2

The Apriori Meta ARM algorithm requires less itemsets to be included in the RTD list than is the case with the Brute Force Meta ARM algorithm (as demonstrated in Section 5.5.1). However, the Apriori approach requires the RTD lists to be processed at the end of each level generation, while in the case of the

Algorithm 5.1: Bench Mark Meta ARM

```
1: begin:
2: k = 1;
3: Generate candidate k-itemsets
4: Start Loop
5:   if (k-itemsets == null break)
6:     forall N data sets get counts for k-itemsets
7:     Prune k-itemsets according to support threshold
8:     k ⇐ k+1
9:     Generate k-itemsets
10: end Loop
13: end Algorithm
```

Table 5: Bench Mark Meta ARM Algorithm

	Frequent in T-tree N	Not frequent in T-tree N
Frequent in merged T-tree sofar	Update support count for i in merged T-tree sofar and proceed to child branch in merged T-tree sofar	Add labels for all supported nodes in merge T-tree branch, starting with current node, to RTD list N
Not Frequent in merged T-tree sofar	Process current branch in T-tree N, starting with current node, adding nodes with their support to the merged T-tree sofar and recording labels for each to RTD lists 1 to $N-1$	Do nothing

Table 6: Brute Force Meta ARM itemset comparison options

Algorithm 5.2: Apriori Meta ARM

```
1: begin:
2: k = 1;
3: Generate candidate k-itemsets
4: Start Loop
5:   if (k-itemsets == null break)
6:     Add supports for level K from N T-trees or add to RTD list
7:     Prune k-itemsets according to support threshold
8:     k ⇐ k+1
9:     Generate k-itemsets
10: end Loop
13: end Algorithm
```

Table 7: Apriori Meta ARM Algorithm

Brute Force approach this is only done once. A hybrid approach, that combines the advantages offered by both the Brute Force and Apriori meta ARM algorithms therefore suggests itself. Experiments were conducted using two different version of the hybrid approach.

The Hybrid 1 algorithm commences by generating the top level of the merged T-tree in the Apriori manner described above (including processing of the RTD list); and then adds the appropriate branches, according to which top level nodes are supported, using a Brute Force approach.

The Hybrid 2 algorithm commences by generating the top two levels of the merged T-tree, instead of only the first level, as in the Hybrid 1 approach. Additional support counts are obtained by processing the RTD lists. The remaining branches are added to the supported level 2-nodes in the merged T-tree sofar (again) using the Brute Force mechanism. The philosophy behind the hybrid 2 algorithm was that we might expect all the one itemsets to be supported and included in the component T-trees therefore we might as well commence by building the top two layers of the merged T-tree.

5.4 Meta ARM EMADS Model

In EMADS, agents are responsible for accessing local data sources and for collaborative data analysis. In the context of Meta ARM each local mining agent's basic function is to generate "local" item sets (local model) from local data and provide this to the task agent in order to generate the complete global set of frequent itemsets (global model).

Figure 12 shows the Meta ARM EMADS model. The model consists of a number of data sources distributed across the network. Detailed data are stored in the DBMS (Data Base Management System) of local sites. Each local site

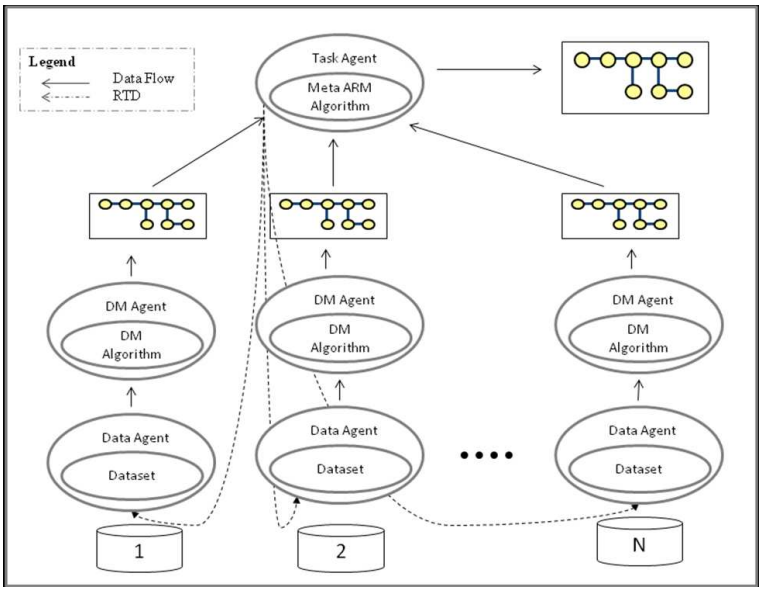


Figure 12: Meta ARM Model

has one Data Agent that is a member of EMADS. In Figure 12 the connection between a local agent and its local DBMS is not shown. A local DM Agents apply some ARM algorithm on to N raw data sets producing N collections of frequent item sets in under decentralised control and an autonomous manner. Then a Task Agent *merges* the different sets of results into a single *meta* set of frequent itemsets with the aim of generating a set of ARs or alternatively a set of Classification Association Rules (CARS).

5.4.1 Dynamic Behaviour of EMADS for Meta ARM operations

EMADS initially starts up with the two central JADE agents. When a data agent wishes to make its data available for possible data mining tasks, it must publish its name and description with the DF agent. In the context of Meta ARM, each mining agent could apply a different data mining algorithm to produce its local frequent item sets T-tree. The T-trees from each local data mining agent are collected by the task agent, and used as input to Meta ARM algorithms for generating global frequent item sets (merged T-tree) making use of return to data (RTD) lists, at least one per data set, to contain lists of itemsets whose support was not included in the current T-tree and for which the count is to be obtained by a return to the raw data.

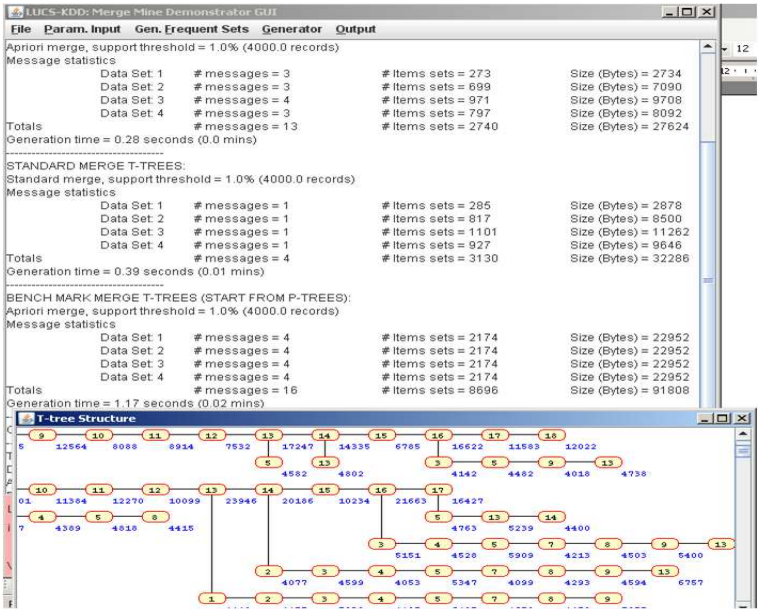


Figure 13: User Agent GUI: Meta ARM Example

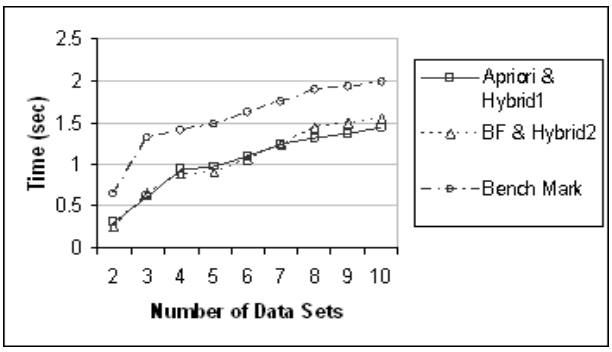
5.5 Note on datasets

The datasets used in ARM are usually presented as sequences of records comprising item (column) numbers, which in turn represent attributes of the dataset. The presence of a column number in a record indicates that the associated attribute exists for that record. This form of dataset is exemplified by shopping trolley scenarios, where records represent customer trolley-fulls of shopping purchased during a single transaction and the columns/attributes, items or groups of items available for purchase. Although ARM is directed at binary-valued attribute sets, it can be applied to non-binary valued sets and also sets where attributes are continuously valued through a pre-process of data normalization. Datasets can thus be considered to be ND tables where N is the number of columns and D is the number of records.

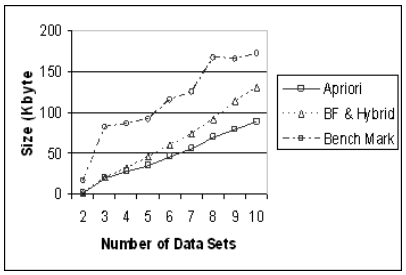
5.5.1 Experimentation and Analysis

To evaluate the five algorithms outlined above, in the context of EMADS, a number of experiments were conducted. These are described and analysed in this section. The experiments were designed to analyse the effect of the following:

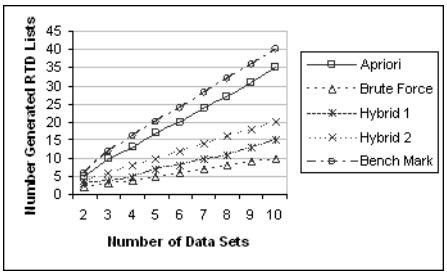
1. The number of data sources.
2. The size of the datasets in terms of number of records .



(a) Processing Time



(b) Total size of RTD lists



(c) Number of RTD lists

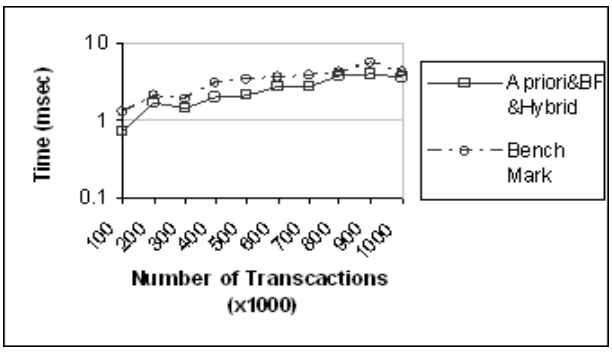
Figure 14: Effect of number of data sources

3. The size of the datasets in terms of number of attributes.

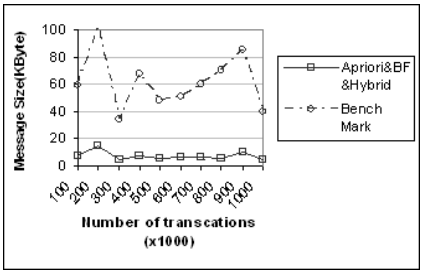
All experiments were run using a Intel Core 2 Duo E6400 CPU (2.13GHz) with 3GB of main memory (DDR2 800MHz), Fedora Core 6, Kernel version 2.6.18 running under Linux. For each of the experiments the following were measured: (i) processing time (seconds / milliseconds), (ii) the size of the RTD lists (Kbytes) and (iii) the number of RTD lists generated. The authors did not use the IBM QUEST generator [2] because many different data sets (with the same input parameters) were required and the quest generator always generated the same data given the same input parameters. Instead the authors used the LUCS KDD data generator [2]. Figure 13 shows a "screen shot" of EMADS User Agent GUI while executing the experiments.

Figure 14 shows the effect of adding additional data sources. For this experiment ten different artificial data sets were generated using $T = 4$ (average number of items per transactions), $N = 20$ (Number of attributes), $D = 100k$ (Number of transactions). Note that the selection of a relatively low value for N ensured that there were some common frequent itemsets shared across the

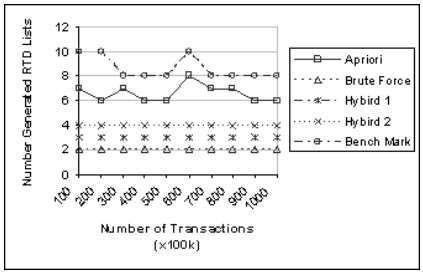
[2] $http://www.csc.liv.ac.uk/\ frans/KDD/Software//LUCS-KDD-DataGen/$



(a) Processing Time



(b) Total size of RTD lists

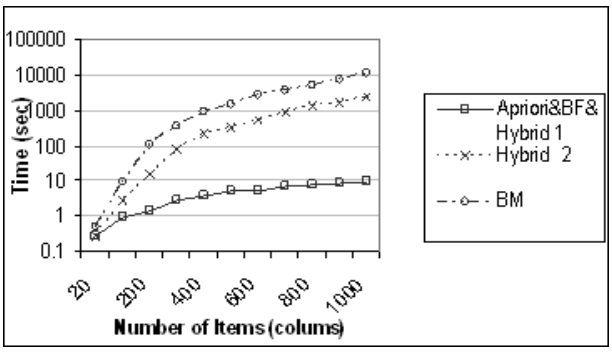


(c) Number of RTD lists

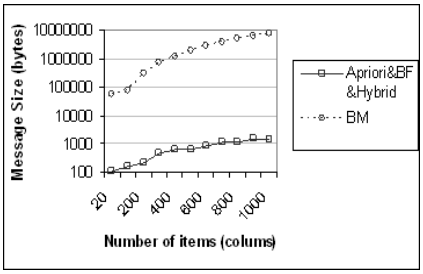
Figure 15: Effect of increasing number of records

T-trees. Experiments using $N = 100$ and above tended to produced very flat T-trees with many frequent 1-itemsets, only a few isolated frequent 2-itemsets and no frequent sets with cardinality greater than 2. For the experiments a support threshold of 1% was selected. Graph 3(a) demonstrates that all of the proposed Meta ARM algorithms worked better then the bench mark (start all over again) approach. The graph also shows that the Apriori Meta ARM algorithm, which invokes the "return to data procedure" many more times than the other algorithms, at first takes longer; however as the number of data sources increases the approach starts to produce some advantages as T-tree branches that do not include frequent sets are identified and eliminated early in the process. The amount of data passed to and from sources, shown in graph 3(b), correlates directly with the execution times in graph 3(a). Graph 3(c) shows the number of RTD lists generated in each case. The Brute Force algorithm produces one (very large) RTD list per data source. The Bench Mark algorithm produces the most RTD lists as it is constantly returning to the data sets, while the Apriori approach produces the second most (although the content is significantly less).

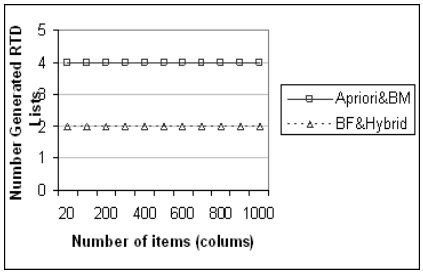
Figure 15 demonstrates the effect of increasing the number of records. The input data for this experiment was generated by producing a sequence of ten



(a) Processing Time



(b) Total size of RTD lists



(c) Number of RTD lists

Figure 16: Effect of increasing number of items (attributes)

pairs of data sets (with $T = 4$, $N = 20$) representing two sources. From graph 4(a) it can be seen that the all the meta ARM algorithms outperformed the bench mark algorithm because the size of the return to data lists were limited as no unnecessary candidate sets were generated. This is illustrated in graph 4(b). Graph 4(b) also shows that the increase in processing time in all case is due to the increase in the number of records only, the size of the RTD lists remains constant throughout as does the number of RTD lists generated (graph 4(c)).

Figure 16 shows the effect of increasing the global pool of potential attributes (remember that each data set will include some subset of this global set of attributes). For this experiment another sequence of pairs of data sets (representing two sources) was generated with $T = 4$, $D = 100K$ and N ranging from 100 to 1000. As in the case of experiment 2 the Apriori, Brute Force and Hybrid 1 algorithms work best (for similar reasons) as can be seen from graph 5(a,b). However, in this case (compared to the previous experiment), the Hybrid 2 algorithm did not work as good. The reasoning behind the Hybrid 2 algorithm slower performance is that all the 1-itemsets tended not to be all supported and because there were not eliminated and included in 2-itemsets

generation (graph 5(a)). For completeness graph 5(c) indicates the number of RTD lists sent with respect to the different algorithms.

All the Meta ARM algorithms outperformed the bench mark algorithm. The Hybrid 2 algorithm also performed in an unsatisfactory manner largely because of the size of the RTD lists sent. Of the remainder the Apriori approach coped best with a large number of data sources, while the Brute Force and Hybrid 1 approaches coped best with increases in data sizes (in terms of column/rows) again largely because of the relatively smaller RTD list sizes.

It should also be noted that the algorithms are all complete and correct, i.e. the end result produced by all the algorithms is identical to that obtained from mining the union of all the raw data sets using some established ARM algorithm. In practice, of course, the MADM scenario, which assumes that data cannot be combined in this centralised manner, would not permit this.

5.6 Summary

Traditional centralized data mining techniques may not work well in many distributed environments where data centralization may be difficult because of limited bandwidth, privacy issues and/or the demand on response time. Meta-learning data mining strategies may offer a better solution than the central approaches but are not as accurate in their results.

This Section proposed EMADS as MAS solution to address the above issues. The use of EMADS was illustrated using a meta ARM scenario where a novel extension of ARM is described where a meta set of frequent itemsets was built from a collection of component sets which had been generated in an autonomous manner without centralised control is build. This type of conglomerate was termed Meta ARM so as to distinguish it from a number of other related data mining research areas such as incremental and distributed ARM. A number of meta ARM algorithms were described and compared: (i) Bench Mark, (ii) Apriori, (iii) Brute Force, (iv) Hybrid 1 and (v) Hybrid 2.

The described experiments indicated, at least with respect to Meta ARM, that EMADS offers positive advantages in that all the Meta ARM algorithms were more computationally efficient than the bench mark algorithm. The results of the analysis also indicated that the Apriori Meta ARM approach coped best with a large number of data sources, while the Brute Force and Hybrid 1 approaches coped best with increased data sizes (in terms of column/rows). The authors are greatly encouraged by the results obtained so far and are currently undertaking further analysis of EMADS with respect to alternative data mining tasks.

6 VERTICAL PARTITIONING AND DISTRIBUTED/PARALLEL ARM

In this Section EMADS the support that EMADS provides with respect to the dynamic creation of communities of data mining agents is demonstrated. The

section also explores the capabilities of such agents and demonstrate (by experiment) their application to distributed and parallel data mining. An approach to distributed/parallel Association Rule Mining (ARM), called DATA-VP, which makes use of a vertical partitioning approach to distributing the input data is described. Using this approach each partition can be mined in isolation, while at the same time taking into account the possibility of the existence of large itemsets dispersed across two or more partitions. To facilitate the partitioning the tree data structure described in Section 5, is again used together with the Apriori-T ARM algorithm. The approach described offers significant advantages with respect to computational efficiency when compared to alternative mechanisms for (a) dividing the input data between processors and/or (b) achieving distributed/parallel ARM.

The organization of the Section is as follows: In Subsection 6.1 some previous work in the field of parallel ARM is considered. In Subsection 6.2 the Apriori-T algorithm and the the Apriori-T algorithm with vertical partitioning (DATA-VP) is presented. In Subsection 6.3 some particular considerations are given with respect to their use in distributed/parallel. The architecture and network configuration is presented in Subsection 6.4. Experimentation and Analysis is presented in Subsection 6.5 and a summary is given in Subsection 6.6.

6.1 Background and Previous Work

Notwithstanding the extensive work that has been done in the field of ARM, there still remains a need for the development of faster algorithms and alternative heuristics to increase their computational efficiency. Because of the inherent intractability of the fundamental problem, much research effort is currently directed at parallel ARM to decrease overall processing times (see [43, 78, 91, 98]), and distributed ARM to support the mining of datasets distributed over a network [23].

6.2 The T-tree and the Apriori-T algorithm

The Apriori-T algorithm combines the classic Apriori ARM algorithm, described in Section 2, with the T-tree data structure [29]. As each level is processed, candidates are added as a new level of the T-tree, their support is counted, and those that do not reach the required support threshold pruned. When the algorithm terminates, the T-tree contains only large itemsets.

At each level, new candidate itemsets of size K are generated from identified large K-1 itemsets, using the downward closure property of itemsets, which in turn may necessitate the inspection of neighbouring branches in the T-tree to determine if a particular K-1 subset is supported. This process is referred to as X-checking. Note that X-checking adds a computational overhead; offset against the additional effort required to establish whether a candidate K itemset, all of whose K-1 itemsets may not necessarily be supported, is or is not a large itemset.

The number of candidate nodes generated during the construction of a T-tree, and consequently the computational effort required, is very much depen-

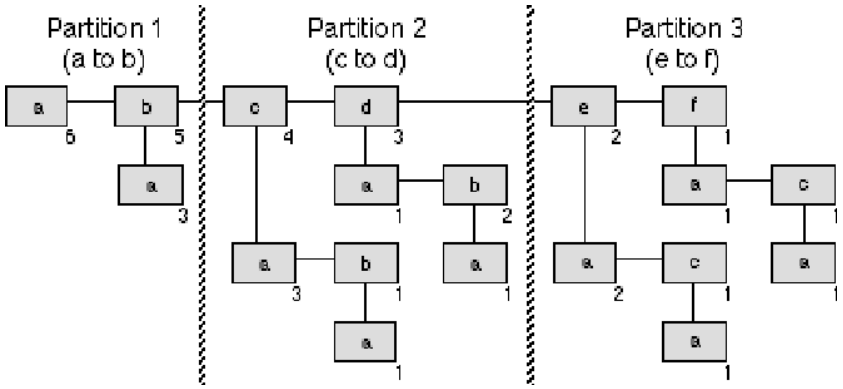


Figure 17: Vertical Partitioning of a T-tree Example

dent on the distribution of columns within the input data. Best results are produced by reordering the dataset, according to the support counts for the 1-itemsets, so that the most frequent 1-itemsets occur first ([26]).

6.3 The Distributed and Parallel Task with Vertical Partitioning (DATA-VP) Algorithm

The DATA-VP algorithm commences by distributing the input dataset over the available number of data mining (DM) agents using a vertical partitioning strategy. Initially the set of single attributes (columns) is split equally between the available DM agents so that an allocationItemSet (a sequence of single attributes) is defined for each DM Agent in terms of a startColNum and endColNum:

$$\textbf{allocationItemSet} = \{\textbf{n} | \textbf{startColNum} < \textbf{n} = \textbf{endColNum}\}$$

Each DM Agent will have its own allocationItemSet which is then used to determine the subset of the input dataset to be considered by the DM Agent. Using its allocationItemSet the Task Agent will partition the data among DM Agents as follows:

1. Remove all records in the input dataset that do not intersect with the allocationItemSet.

2. From the remaining records remove those attributes whose column number is greater than endColNum. Attributes whose identifiers are less than startColNum cannot be removed because these may represent the leading sub-string of large itemset to be included in the sub T-tree counted by the process.

3. Send the allocated data partition to the corresponding DM Agent

TID	Item Set
1	acf
2	b
3	ace
4	bd
5	ae
6	abc
7	d
8	ab
9	c
10	abd

Table 8: Dataset Example

The input dataset distribution procedure, given an *allocationItemSet*, can be summarised as follows:

$\forall$ *records* $\in$ *input data*
if (records $\in$ *allocationItemSet = true)*
record $= \{n | n \in$ *record* $n <=$ *endColNum*$\}$
else delete record

For example − given the data set in Table 8, and assuming three worker agents are participating, the above partitioning process will result in three dataset partitions:

Process 1 (a to b): {{*a*}, {*b*}, {*a*}, {*b*}, {*a*}, {*a, b*}, {}, {*a, b*}, {}, {*a, b*}}
Process 2 (c to d): {{*a, c*}, {}, {*a, c*}, {*b, d*}, {}, {*a, b, c*}, {*d*}, {}, {*c*}, {*a, b, d*}}
Process 3 (e to f): {{*a, c, f*}, {}, {*a, c, e*}, {}, {*a, e*}, {}, {}, {}, {}}

Figure 17 shows the resulting sub T-trees assuming all combinations represented by each partition are supported. Note that because the input dataset is ordered according to the frequency of 1-itemsets the size of the individual partitioned sets does not necessarily increase as the *endColNum* approaches N (the number of columns in the input dataset); in the later partitions, the lower frequency leads to more records being eliminated. Thus the computational effort required to process each partition is roughly. Once partitioning is complete each partition can be mined, using the Apriori-T algorithm, in isolation.

The DATA-VP scenario can thus be summarized as follows:

1. Task Agent starts a number of DM agents; they will be referred to as Workers.

2. Task Agent determines the division of *allocationItemSet* according to the total number of available workers (agents) and transmits this information to them.

3. Task Agent transmits only allocated partition of the data to each Worker.

4. Each worker then generates a T-tree for its allocated partition (a sub tree of the final T-tree).

5. On completion each process transmits its partition of the T-tree to the Task Agent which are then merged into a single T-tree (the final T-tree ready for the next stage in the ARM process rule generation).

The T-tree generation process begins with a top-level tree comprising only those 1-itemsets included in its allocationItemSet. The process will then generate the candidate 2-itemsets that belong in its sub T-tree. These will comprise all the possible pairings between each element in the allocationItemSet and the lexicographically preceding attribute; of those elements (see Figure 17). The support values for the candidate 2-itemsets are then determined and the sets pruned to leave only frequent 2-itemsets. Candidate sets for the third level are then generated. Again, no attributes from succeeding allocationItemSet are considered, but the possible candidates will, in general, have subsets which are contained in preceding allocationItemSet and which, therefore, are being counted by some other process. To avoid the overhead involved in X-checking, which in this case would require message-passing between the DM agents concerned, X-checking does not take place. Instead, the process will generate its candidates assuming, where necessary, that any subsets outside its local T-tree are large.

6.4 Architecture and network configuration

The architecture shown in Figure 18 assumes the availability of at least two agents (preferably more), one Task Agent and one or more DM Agents (workers). Figure 18 shows the assumed distribution of agents and shared data across the network. The figure also shows JADE Agents through which the agents find each other. All agents described here have been implemented using the EMADS framework.

6.4.1 Messaging

Distributed/parallel ARM may entail much exchange of data messaging as the task proceeds. Messaging represents a significant computational overhead in some cases outweighing any other advantage gained. Usually the number of messages sent and the size of the content of the message have significant performance factor. It is therefore expedient in the context of the techniques described here to minimize the number of messages that are required to be sent and their sizes.

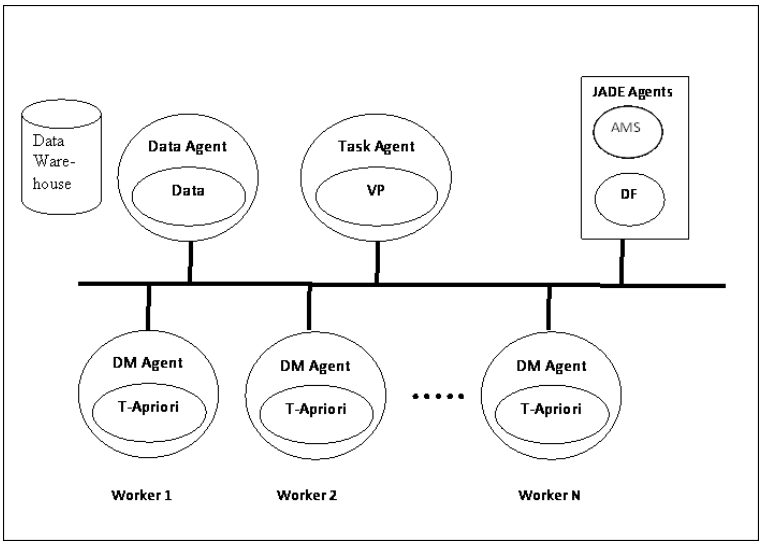


Figure 18: Parallel/Distributed ARM Model Architecture

The scenario described here is using the One-to-Many approach, where only the Task Agent can send/receive messages to/from DM Agents which involves fewer operations; the significance of this advantage increases as the number of agents used increases. Also, the exchange of data requires each Task Agent to send data to other DM Agents. So as to minimize the size of data being sent, DM agents only process their allocated data partition and not the whole datasets.

6.5 Experimentation and Analysis

To evaluate the five Meta ARM algorithms, in the context of the EMADS vision, a number of experiments were conducted. These are described and analysed in this section. The experiments presented here used up to five workers (DM Agents) and the data set T20.I10.D500K.N500 (generated using the IBM Quest generator used in Agrawal and Srikant [2]). In all case the dataset had been preprocessed so that it is ordered.

The most significant overhead of any distributed/parallel system is the number and size of messages sent and received between agents. For this DATA-VP demonstrator, the number of messages sent is independent of the number of levels in the T-tree; communication takes place only at the end of the tree construction. Therefore, DATA-VP has a clear advantage in terms of the number of messages sent. DATA-VP passes entire pruned sub T-trees (pruned T-tree branches, not entire levels). Consequently the amount of data passed between agents when using DATA-VP is significantly small.

Figure 19 shows the effect of increasing the number of workers for a range of

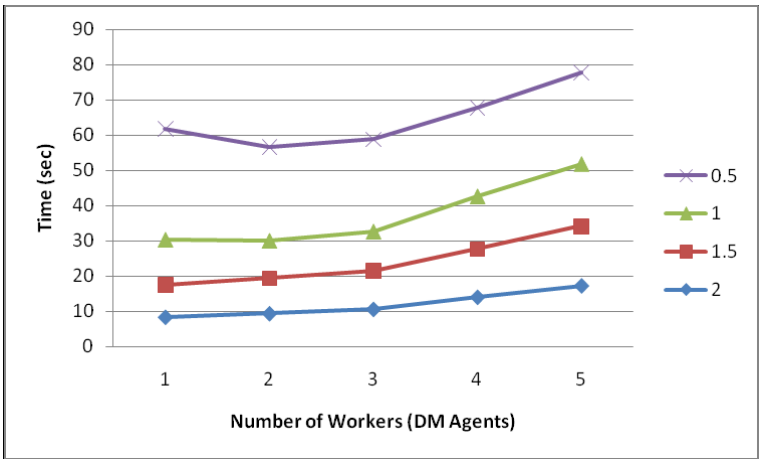


Figure 19: Effect of number of workers

support thresholds. The experimental clearly demonstrates that the approach performs very well. This is largely due to the T-tree data structure which: (a) facilitates vertical distribution of the input dataset, and (b) readily lends itself to distribution/parallelization. Note also that, for DATA-VP, as further processes are added, the increasing overhead of messaging more than outweighs any gain from using additional processes, so that distribution/parallelization becomes counter productive.

6.6 Summary

In this Section we considered a technique (DATA-VP) for distributed (and parallel) Association Rule Mining that makes use of a vertical partitioning technique to distribute the input data amongst a number of agents. The proposed vertical partitioning is facilitated by the T-tree data structure, and an associated mining algorithm (Apriori-T), that allows for computationally effective distributed/ parallel ARM when employed on EMADS.

The aim has been to identify methods that will enable efficient counting of frequent sets in cases where the data is much too large to be contained in primary memory, and also where the density of the data means that the number of candidates to be considered becomes very large.

An MADM method for distributed/ ARM has been described. The advantage of this approach is that it allows both the original data to be partitioned into more manageable subsets, and also partitions the candidate sets to be counted. The latter results in both lower memory requirements and also faster counting.

The reported experimental results show that the Tree Partitioning method described is extremely effective in limiting the maximal memory requirements of the algorithm, while its execution time scales only slowly and linearly with

increasing data dimensions. Its overall performance, both in execution time and especially in memory requirements, is significantly better than that obtained from either simple data segmentation or a method that aims to complete counting in only two passes of the data.

7 CLASSIFIER GENERATION

To further illustrate the features of EMADS a classifier generation scenario is considered in this Section. The point of this section is to show how MADM can be used to generate a number of classifiers from which the "best" can be selected.

MAS has some particular advantages to offer with respect to KDD, in the context of sharing resources and expertise. Namely that the MAS approach provides greater end user access to data mining techniques. MAS can make use of algorithms without necessitating their transfer to users, thus contributing to the preservation of intellectual property rights.

This Section considers a collection of classifier data mining agents applied to a number of standard "benchmark" data sets held by data agents.

The Section is organised as follows. A brief review of some background on data classification is presented in Subsection 7.1. The operation of EMADS with respect to classification is illustrated in Subsection 7.2. Experimentation and analysis are described in Subsection 7.3. A summary is given in Subsection 7.4.

7.1 Background

This scenario is a classification scenario where the objective is to generate a classifier (predictor) fitted to a specified dataset. It has been well established within the data mining research community, that there is no single best classification algorithm. The aim of this third scenario is therefore to identify a "best classifier given a particular dataset. Best in this context is measured in terms of classification accuracy. The experiment thus not only serves to illustrate the advantageous of EMADS, but also provides an interesting comparison of a variety of classification techniques and algorithms.

7.2 EMADS Operation: Classifier Generation

Conceptually the nature of EMADS data mining requests, that may be posted by EMADS users, is extensive. In the context of classification, the following types of generic request are supported:

- Find the "best" classifier (to be used by the requester at some later date in off line mode) for a data set provided by the user.
- Find the "best" classifier for the indicated data set (i.e. provided by some other EMADS participant).

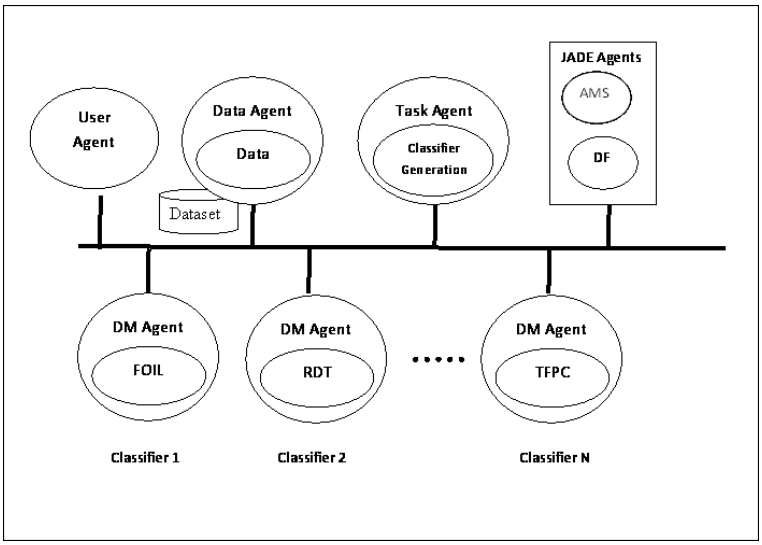


Figure 20: Classifier Generation EMADS Model Architecture

To obtain the "best" classifier EMADS will attempt to access and communicate with as many classifier generator data mining agents as possible and select the best result.

Figure 20 gives an overview of the the assumed distribution of agents and shared data across the network. The figure also shows the JADE house keeping agents (AMS & DF). All agents described here have been implemented using the EMADS framework. The principal advantage of this 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.

The scenario is that of an end user who wishes to obtain a "best" classifier founded on a given, pre-labelled, data set; which can then be applied to further unlabelled data. The assumption is that the given data set is binary valued and that the user requires a single-label, as opposed to a multi-labelled, classifier. The request is made using the individual's user agent which in turn will spawn an appropriate task agent.

For this scenario the task agent identifies mining agents that hold single labelled classifier generators that take binary valued data as input. Each of these mining agents is then accessed and a classifier, together with an accuracy estimate, requested. The task agent then selects the classifier with the best accuracy and returns this to the user agent.

The data mining agent wrapper in this case provides the interface that allows input for: (i) the data; and (ii) the number of class attributes (a value that the mining agent cannot currently deduce for itself) while the user agent interface allows input for threshold values (such as support and confidence values). The

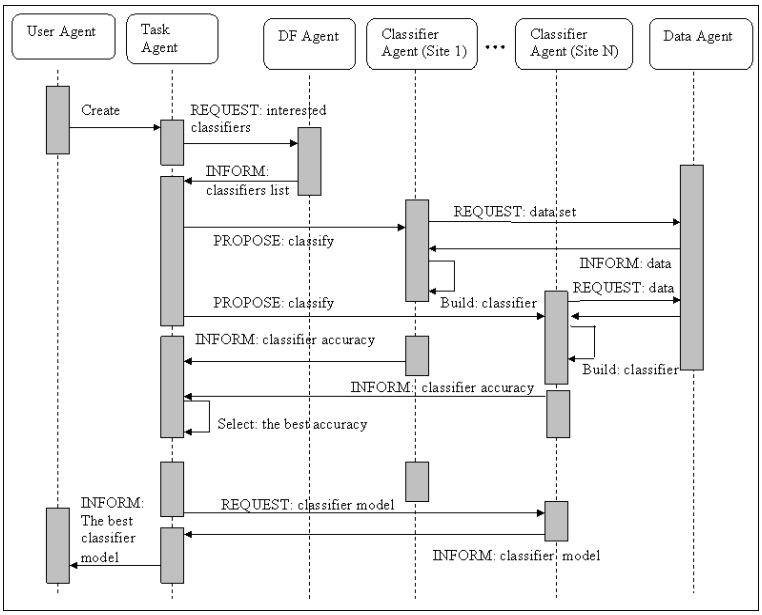


Figure 21: Classification Task Sequence Diagram.

output is a classifier together with an accuracy measure. To obtain the accuracy measures the classifier generator (data mining agent) builds the classifier using the first half of the input data as the "training" set and the second half of the data as the "test" set. An alternative approach might have been to use Ten Cross Validation (TCV) to identify the best accuracy.

From the literature there are many reported techniques available for generating classifiers. For this scenario, eight different algorithm implementations were used [3]:

1. FOIL (First Order Inductive Learner) [85] the well established inductive learning algorithm for generating Classification Association Rules (CARs).
2. TFPC (Total From Partial Classification) CAR generator [28] founded on the P- and T-tree set enumeration tree data structures.
3. PRM (Predictive Rule Mining) [109] an extension of FOIL.
4. CPAR (Classification based on Predictive Association Rules) [109] a further development from FOIL and PRM.
5. IGDT (Information Gain Decision Tree) classifier, an implementation of the well established decision tree based classifier using most information gain as the "splitting criteria".

[3] taken from the LUCS-KDD repository at *http* : *//www.csc.liv.ac.uk/f̃rans/KDD/Software/*

Data Set	Classifier	Accuracy	Gen. Time (sec)
connect4.D129.N67557.C3	RDT	79.76	502.65
adult.D97.N48842.C2	IGDT	86.05	86.17
letRecog.D106.N20000.C26	RDT	91.79	31.52
anneal.D73.N898.C6	FOIL	98.44	5.82
breast.D20.N699.C2	IGDT	93.98	1.28
congres.D34.N435.C2	RDT	100	3.69
cylBands.D124.N540.C2	RDT	97.78	41.9
dematology.D49.N366.C6	RDT	96.17	11.28
heart.D52.N303.C5	RDT	96.02	3.04
auto.D137.N205.C7	IGDT	76.47	12.17
penDigits.D89.N10992.C10	RDT	99.18	13.77
soybean-large.D118.N683.C19	RDT	98.83	13.22
waveform.D101.N5000.C3	RDT	96.81	11.97

Table 9: Classification Results

6. RDT (Random Decision Tree) classifier, a decision tree based classifier that uses most frequent current attribute as the "splitting criteria" (so not really random).
7. CMAR (Classification based on Multiple Association Rules) is a Classification Association Rule Mining (CARM) algorithm [65] .
8. CBA (Classification Based on Associations) is a CARM algorithm [67].

These were placed within an appropriately defined tool wrapper to produce eight (single label binary data classifier generator) data mining agents. This was a trivial operation indicating the versatility of the wrapper concept.

Thus each mining agent's basic function is to generate a classification model using its own classifier and provide this to the task agent. The task agent then evaluates all the classifier models and chooses the most accurate model to be returned to the user agent. The negotiation process amongst the agents is represented by the sequence diagram given in Figure 21 (the figure assumes that an appropriate data agent has ready been created). In the figure includes N classification agents. The sequence of events commences with a user agent which spawns a (classification) task agent, which in turn announces itself to the DF agent. The DF agent returns a list of classifier data mining agents that can potentially be used to generate the desired classifier. The task agent then contacts these data mining agents who each generate a classifier and return statistical information regarding the accuracy of their classifier. The task agent selects the data mining agent that has produced the best accuracy and requests the associated classifier, this is then passed back to the user agent.

Note that the users make the data that they desire to be mined (classified) available by launching their own data agents (which in turn publish their name and description using the DF agent as described above).

7.3 Experimentation and Analysis

To evaluate the classification scenario, as described above, a sequence of data sets taken from the UCI machine learning data repository [17] were used (pre-processed by data agents so that they were discretized/normalized into a binary valued format). The results are presented in Table 1. Each row in the table represents a particular request and gives the name of the data set, the selected best algorithm as identified from the interaction between the EMADS agents, the resulting best accuracy and the total EMADS execution time from creation of the initial task agent to the final "best classifier being returned to the user agent. The naming convention used in the Table is that: D equals the number of attributes (after discretization/normalization), N the number of records and C the number of classes (although EMADS has no requirement for the adoption of this convention).

The results demonstrate firstly that EMADS can usefully be adopted to produces a best classifier from a selection of classifiers. Secondly that operation of EMADS is not significantly hindered by agent communication overheads, although this has some effect. Generation time, in most cases does not seem to be an issue, so further classifier generator mining agents could easily be added. The results also reinforce the often observed phenomena that there is no single best classifier generator suited to all kinds of data set.

7.4 Summary

This Section described the illustration of EMADS in the context of a classification scenario. The scenario is that of an end user who wishes to obtain a "best" classifier founded on a given, pre-labelled, data set; which can then be applied to further unlabelled data. The scenario demonstrated that EMADS can usefully be adopted to produces a best classifier from a selection of classifiers. The scenario also illustration some of the principal advantages offered by EMADS include: experience and resource sharing, flexibility and extendibility, and intellectual property rights.

8 CONCLUSION

The research described in this chapter presents an MADM vision, and describes a multi-agent framework for generic data mining (EMADS). The principal advantages envisaged are those of experience and resource sharing, flexibility and extendibility, and (to an extent) protection of privacy and intellectual property rights. To investigate and evaluate the approach, the EMADS framework was developed. Wrappers have been used to incorporate existing software into EMADS. Experience indicates that, given an appropriate wrapper, existing data mining software can be very easily packaged to become an EMADS data mining agent. The use of EMADS was illustrated using a number different of scenarios.

The EMADS requirements, architecture, design and implementation were discussed in detail. EMADS is envisioned as a collection of data sources scat-

tered over a network and a group of DM agents that allow a user to data mine those data sources without needing to know the location of the supporting data, nor how the various agents interact. Additionally the expectation is that EMADS will "grow" as individual users contribute data and DM algorithms.

In EMADS, as with most MAS, individual agents have different functionality; the system currently comprises: data agents, user agents, task agents, data mining agents and a number of "house-keeping" agents. Users of EMADS may be data providers, DM algorithm contributors or miners of data. The independence of EMADS from any particular DM function, in conjunction with the object oriented design adopted, ensures the system's capability to incorporate and use new data mining algorithms and tools.

EMADS Extendibility is also discussed. The expectation is that EMADS will "grow" as individual users contribute data and DM algorithms. The incorporation of data and data mining software is facilitated by a system of wrappers which allows for easy extendibility of the system.

The independence of EMADS from any particular DM function, in conjunction with the object oriented design adopted, ensures the system's capability to incorporate and use new data mining algorithms and tools. As discussed above, introducing a new technique requires the sub-classing of the appropriate abstract class or the implementation of an abstract interface and the encapsulation of the tool within an object that adheres to the minimal interface. In fact, most of the existing implemented algorithms have similar interfaces already. This "plug-and-play" characteristic makes EMADS a powerful and extensible DM facility. This feature allows developers to employ their pre-implemented programs within EMADS agents.

Meta ARM scenario represented a novel extension of ARM where a meta set of frequent itemsets from a collection of component sets, which have been generated in an autonomous manner without centralised control, is built. This type of conglomerate was termed meta ARM so as to distinguish it from a number of other related data mining research areas such as incremental and distributed ARM. A number of meta ARM algorithms were described and compared: (i) Bench Mark, (ii) Apriori, (iii) Brute Force, (iv) Hybrid 1 and (v) Hybrid 2. The described experiments indicated, at least with respect to Meta ARM, that EMADS offers positive advantages in that all the Meta ARM algorithms were more computationally efficient than the bench mark algorithm.

The second scenario considering a technique (DATA-VP) for distributed (and parallel) Association Rule Mining that made use of a vertical partitioning technique to distribute the input data amongst a number of agents. The proposed vertical partitioning was facilitated by a set enumeration tree data structure (the T-tree), and an associated mining algorithm (Apriori-T), that allows for computationally effective distributed/parallel ARM when employed on EMADS. The experimental results obtained showed that the Tree Partitioning method described was extremely effective in limiting the maximal memory requirements of the algorithm, while its execution time scales only slowly and linearly with increasing data dimensions. The scenario demonstrated how EMADS can be used to achieve distributed/parallel data mining in a MADM context.

The use of EMADS was also illustrated with using a third classification scenario. The results demonstrated firstly that EMADS can usefully be adopted to produces a best classifier from a selection of classifiers. Secondly that the operation of EMADS is not significantly hindered by agent communication overheads, although this has some effect. Generation time, in most cases does not seem to be an issue, so further classifier generator mining agents could easily be added. The results also reinforce the often observed phenomena that there is no single best classifier generator suited to all kinds of data set.

In conclusion a good foundation has been established for both data mining research and genuine application based MADM. It is acknowledged that the current functionality of EMADS is limited to classification and ARM. The research team is at present working towards increasing the diversity of mining tasks that EMADS can address. There are many directions in which the work can (and is being) taken forward. One interesting direction is to build on the wealth of distributed data mining research that is currently available and progress this in a MAS context. The research team is also enhancing the systems robustness so as to make it publicly available. It is hoped that once the system is live other interested data mining practitioners will be prepared to contribute algorithms and data.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. *Mining association rules between sets of items in large databases.* In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 1993.

[2] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger. *The Quest Data Mining System.* Proceedings 2nd Int. Conf. Knowledge Discovery and Data Mining, (KDD1996), 1996.

[3] R. Agrawal and G. Psaila. *Active Data Mining.* Proceedings 1st Int. Conf. Knowledge Discovery in Data Mining, AAAI, 1995.

[4] R. Agrawal and R. Srikant. *Fast algorithm for mining association rules.* In: Bocca, J.B., Jarke, M., and Zaniolo, C. (Eds.): Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, 1994.

[5] K. A. Albashiri, F. P. Coenen, and P. Leng. *Agent Based Frequent Set Meta Mining: Introducing EMADS.* Artificial Intelligence in Theory and Practice II, IFIP'2008, Springer, London, UK, 2008.

[6] K. A. Albashiri, F. P. Coenen, and P. Leng. *EMADS: An Extendible Multi-Agent Data Miner*, volume XXIII. Research and Development in Intelligent Systems, AI'2008, Springer, London, UK, 2008.

[7] K. A. Albashiri, F. P. Coenen, P. Leng, and R. Sanderson. *Frequent Set Meta Mining: Towards Multi-Agent Data Mining*, volume XXIV. Research

and Development in Intelligent Systems, AI'2007, Springer, London, UK, 2007.

[8] W. G. Aref, M. G. lfeky, and A. K. Elmagarmid. *Incremental, Online, and Merge Mining of Partial Periodic Patterns in Time-Series Databases*, volume 16(3). IEEE Transaction in Knowledge and Data Engineering, 2004.

[9] H. Baazaoui, S. Faiz, R. Ben Hamed, and H. Ben Ghezala. *A Framework for data mining based multi-agent: an application to spatial data*. 3rd World Enformatika Conference WEC'05, Avril, 2005, Istanbu, 2005.

[10] B. Babcock, S. Babuand M. Datar, R. Motwani, and J. Widom. *Models and Issues in Data Stream Systems*. In Proceedings of the 21th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), 2002.

[11] S. Bailey, R. Grossman, H. Sivakumar, and A. Turinsky. *Papyrus: a system for data mining over local and wide area clusters and super-clusters*. In Proceedings Conference on Supercomputing, ACM Press, 1999.

[12] Julien Balter, Annick Labarre-Vila, Danielle Zibelin, and Catherine Garbay. *A Platform Integrating Knowledge and Data Management for EMG Studies*. AIME, 2001.

[13] J. Baumann, F. Hohl, K. Rothermel, and M. StraBer. *Mole - Concepts of a Mobile Agent System*. World Wide Web,1(3), 1998.

[14] C. Baumer and T. Magedanz. *GrassHopper 2, an intelligent mobile agent platform written in 100% pure Java*. In Sahin Albayrak, editor, Proceedings of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA-99), Springer 1699 of LNAI, Berlin, Germany, 1999.

[15] F. Bellifemine, G. Cairo, and D. Greenwood. *Developing Multi-Agent Systems with Jade*. Wiley Series in Agent Technology, ISBN: 9780470057476, 2007.

[16] F. Bellifemine, A. Poggi, and G. Rimassi. *JADE: A FIPA-Compliant agent framework*. Proceedings Practical Applications of Intelligent Agents and Multi-Agents, 1999. http://www.jade.tilab.com.

[17] C. L. Blake and C. J. Merz. *UCI repository of machine learning databases*. Irvine, CA: University of California, Department of Information and Computer Science, 1998. http://archive.ics.uci.edu/ml/.

[18] R. H. Bordini, L. Braubach, M. Dastani, A. E. F. Seghrouchni, J. J. G. Sanz, G. OHare J. Leite, A. Pokahr, and A. Ricci. *A survey of programming languages and platforms for multi-agent systems*. In Proceedings of the IEEE International Conference on Cognitive Informatics, 2006.

[19] R Bose and V Sugumaran. *IDM: An Intelligent Software Agent Based Data Mining Environment.* In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 1998.

[20] J. A. Bota, A. F. Gmez-Skarmeta, M. Valds, and A. Padilla. Metala. *A meta-learning architecture.* Fuzzy Days, 2001.

[21] L. Cao, C. Luo, and C. Zhang. *Agent-Mining Interaction: An Emerging Area.* AIS-ADM07, LNAI 4476, Springer - Verlag, Berlin, Germany, 2007.

[22] B. Chandrasekaran and T. R. Johnson. *Generic tasks and task structures: History, critique and new directions.* In J. M. David and J. P. Krivine and R. Simmons (Ed.), Second Generation Expert Systems, Springer - Verlag, Berlin, Germany, 1993.

[23] D. Cheung and Y. Xiao. *Effect of Data Distribution in Parallel Mining of Associations.* Proceedings in Data Mining and Knowledge Discovery 3(3), 1999.

[24] Krzysztof Chmiel, Dominik Tomiak, Maciej Gawinecki, Pawel Karczmarek, Michal Szymczak, and Marcin Paprzycki. *Testing the Efficiency of JADE Agent Platform.* In Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04), 2004.

[25] G. Christopher and B. S. Marks. *Extensible Multi-Agent System for Heterogeneous Database Association Rule Mining and Unification.* Master's thesis, Air University, 1994. http://people.cis.ksu.edu/s̃deloach/publications/Thesis/marks.pdf.

[26] F. Coenen and P. Leng. *Optimising Association Rule Algorithms Using Itemset Ordering.* Research and Development in Intelligent Systems XVIII: Proceedings ES2001 Conference, eds M Bramer, F Coenen and A Preece, Springer, 2001.

[27] F. Coenen, P. Leng, and S. Ahmed. *T-Trees, Vertical Partitioning, and Distributed Association Rule Mining.* Proceedings IEEE Int. Conf. on Data Mining (ICDM 2003), Florida, eds. X Wu, A Tuzhilin and J Shavlik: IEEE Press, 2003.

[28] F. Coenen, P. Leng, and L. Zhang. *Threshold Tuning for Improved Classification Association Rule Mining.* Proceeding PAKDD'05, LNAI3158, Springer, 2005.

[29] F. P. Coenen, P. Leng, and G. Goulbourne. *Tree Structures for Mining Association Rules*, volume 8(1). Journal of Data Mining and Knowledge Discovery, 2004.

[30] Josenildo Costa da Silva, Matthias Klusch, Stefano Lodi, and Gianluca Moro. *Privacy-preserving agent-based distributed data clustering.* Web Intelligence and Agent Systems 4(2), 2006.

[31] S.A. DeLoach and M. Wood. *Developing Multiagent Systems with agentTool.* In C. Castelfranchi and Y. Lesprance, editors, Intelligent Agents VII. Agent Theories, Architectures, and Languages - 7th International Workshop, ATAL-2000, Boston, MA, USA, 2000.

[32] Giuseppe Di Fatta and Giancarlo Fortino. *A customizable multi-agent system for distributed data mining.* Proceedings of the 2007 ACM symposium on applied computing, 2007.

[33] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* the Association for the Advancement of Artificial Intelligence (AAAI) Press/MIT Press, 1996.

[34] Foundation for Intelligent Physical Agents. *FIPA 2002 Specification.* Geneva, Switzerland, 2002. http://www.fipa.org/specifications/index.html.

[35] Y. Fu, W. Ke, and J. Mostafa. *Automated Text Classification Using a Multi-Agent.* Proceedings of the 5th ACM/IEEE-CS Joint Conference on Publication, 2005.

[36] A. Garro and L. Palopoli. *An XML Multi-agent System for E-learning and Skill Management.* Agent Technologies, Infrastructures, Tools, and Applications for E-Services, 2002.

[37] M.R. Genesereth and S.P. Ketchpel. *Software Agents.* Communications of the ACM, 37(7), 1994.

[38] C. Giannella, R. Bhargava, and R. H. Kargupta. *Multi-agent Systems and Distributed Data Mining.* Lecture Notes in Computer Science, 2004, ISSU 3191, 2004.

[39] V. Gorodetsky, O. Karsaev, and V. Samoilov. *Infrastructural Issues for Agent-Based Distributed Learning.* Proceedings of IADM, IEEE Computer Society Press, 2006.

[40] V. Gorodetsky, O. Karsaeyv, and V. Samoilov. *Multi-agent technology for distributed data mining and classification.* In Proceedings of IAT Int. Conference on Intelligent Agent Technology, IEEE/WIC, 2003.

[41] G. Goulbourne, F. P. Coenen, and P. Leng. *Algorithms for Computing Association Rules Using A Partial-Support Tree.* Proceedings ES99, Springer, London, UK, 1999.

[42] R. Grossman and A. Turinsky. *A framework for finding distributed data mining strategies that are intermediate between centralized strategies and in-place strategies.* In KDD Workshop on Distributed Data Mining, 2000.

[43] E. H. Han, G. Karypis, and V. Kumar. *Scalable Parallel Data Mining for Association Rules*. Proceedings ACM-SIGMOD, Int. Conf. on Management of Data, ACM Press, 1997.

[44] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufman Publishers, San Francisco, CA, (Second Edition), 2006.

[45] J. Han, J. Pei, and Y. Yiwen. *Mining Frequent Patterns Without Candidate Generation*. Proceedings ACM-SIGMOD International Conference on Management of Data, 2000.

[46] D. Hand, Mannila H., and P. Smyth. *Principals of Data Mining*. MIT press, Cambridge, Mass, 2001.

[47] DJ Hand. *Construction and Assessment of Classification Rules*. John Wiley and Sons, 1997.

[48] T. Hastie and R. Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer - Verlag, Berlin, Germany, 2001.

[49] JDM2. *Java Data Mining (JDM)*. SUN's Java Community Process listed as JSR-73 and JSR-247 for JDM 2.0, 1995. http://jcp.org/en/jsr/detail?id=73.

[50] N. Jiarui. *A human-friendly MAS for mining stock data*. Proceedings of the IEEE International conference on Web Intelligence, Hong Kong, 2006.

[51] M. Kamber, L. Winstone, G. Wan, and S. Shanand H. Jiawei. *Generalization and Decision Tree Induction: Efficient Classification in Data Mining*. Proceedings of the Seventh International Workshop on Research Issues in Data Engineering, 1997.

[52] H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, and M. Klein. *VEDAS: A Mobile Distributed Data Stream Mining System for Real-Time Vehicle Monitoring*. In Proceedings of the 2004 SIAM International Conference on Data Mining, 2004.

[53] H. Kargupta, Byung-Hoon, and et al. *Collective Data Mining: A New Perspective Toward Distributed Data Mining*. Advances in Distributed and Parallel Knowledge Discovery, MIT/AAAI Press, 1999.

[54] H Kargupta and P. Chan. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI press, Menlo Park, CA, 2000.

[55] H. Kargupta, I. Hamzaoglu, and B. Stafford. *Scalable, Distributed Data Mining Using an Agent Based Architecture*. Proceedings of Knowledge Discovery and Data Mining, AAAI Press, 1997.

[56] H Kargupta and K. Sivakumar. *Existential Pleasures of Distributed Data Mining. In Data Mining: Next Generation Challenges and Future Directions.* edited by H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, MIT/AAAI Press, 2004.

[57] T. Kawamura, N. Yoshioka, T. Hasegawa, A. Ohsuga, and S. Honiden. *Bee-gent : Bonding and Encapsulation Enhancement Agent Framework for Development of Distributed Systems.* Proceedings of the 6th Asia-Pacific Software Engneering Conference, 1999.

[58] M. Kaya and R. Alhajj. *Fuzzy OLAP association rules mining-based modular reinforcement learning approach for multiagent systems*, volume 35. IEEE Transactions on Systems, Man and Cybernetics, Part B, Issue 2, 2005.

[59] B. Kegl and G. Lapalme. *Performance Evaluation of an Agent Based Distributed Data Mining System.* AI 2005, LNAI 3501, Springer - Verlag Berlin, Heidelberg, Germany, 2005.

[60] D. Kerr, D. O'Sullivan, R. Evans, R. Richardson, and F. Somers. *Experiences using Intelligent Agent Technologies as a Unifying Approach to Network and Service Management.* Proceedings of ISN 98, Antwerp, Belgium, 1998.

[61] M. Klusch, S. Lodi, and M. Gianluca. *The role of agents in distributed data mining: issues and benefits.* Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT), 2003.

[62] M. Klusch, S. Lodi, and G. Moro. *Agent-based Distributed Data Mining: The KDEC Scheme.* Intelligent Information Agents The AgentLink Perspective. Lecture Notes in Computer Science 2586, Springer - Verlag, Berlin, Germany, 2003.

[63] J. L. Koh and S. F. Shieh. *An efficient approach to maintaining association rules based on adjusting FP-tree structures.* Proceedings DASFAA 2004, 2004.

[64] C. K. S. Leung, Q. I. Khan, and T. Hoque. *CanTree: A tree structure for efficient incremental mining of Frequent Patterns.* Proceedings The IEEE International Conference on Data Mining (ICDM), 2005.

[65] W. Li, J. Han, and J. Pei. *CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules.* Proceedings The IEEE International Conference on Data Mining (ICDM), 2001.

[66] X. Li, Z-H. Deng, and S-W Tang. *A Fast Algorithm for Maintenance of Association Rules in Incremental Databases.* Proceedings ADMA 2006, Springer-Verlag LNAI 4093, 2006.

[67] B. Liu, W. Hsu, and Y. Ma. *Integrating Classification and Association Rule Mining*. Proceedings KDD-98, New York, the Association for the Advancement of Artificial Intelligence (AAAI), 1998.

[68] P. Luo, R. Huang, Q. He, F. Lin, and Z. Shi. *Execution engine of meta-learning system for kdd in multi-agent environment*. Technical report, Institute of Computing Technology, Chinese Academy of Sciences, 2005.

[69] P. Mariano, C. Pereira A, L. Correira, R. Ribeiro, V. Abramov, N. Szirbik, J. Goossenaerts, T. Marwala, and P. De Wilde. *Simulation of a trading multi-agent system*, volume 5. IEEE Int. Conf. on Systems, Man, and Cybernetics, Springer, London, UK, 2001.

[70] D.L. Martin, A.J. Cheyer, and D.B. Moran. *The Open Agent Architecture: A Framework for Building Distributed Software Systems*, volume 13. Applied Artificial Intelligence, 1998.

[71] T. Marwala and E. Hurwitz. *Multi-Agent Modeling using intelligent agents in a game of Lerpa*. eprint arXiv:0706.0280, 2007.

[72] METAL. *METAL Project*. Esprit Project METAL (no.26.357), 2002. http://www.metal-kdd.org.

[73] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. *MASIF The OMG Mobile Agent System Interoperability Facility*. In K. Rothermel and F. Hohl, Eds. Proceedings 2nd Int. Workshop Mobile Agents (MA '98), Lecture Notes in Computer Science, Springer 1477, Stuttgart, Germany, 1998.

[74] S. R. Mohan, E. K. Park, and Y. Han. *Association Rule-Based Data Mining Agents for Personalized Web Caching*, volume 02. In Proceedings of the 29th Annual international Computer Software and Applications Conference, IEEE Computer Society, Washington, DC, 2005.

[75] H.S. Nwana, D.T. Ndumu, and L.C. Lee. *ZEUS: An advanced Tool-Kit for Engineering Distributed Mulyi-Agent Systems*. In Proceedings of PAAM98, London, U.K, 1998.

[76] Ronan Pairceir, Sally McClean, and Bryan Scotney. *Discovery of multi-level rules and exceptions from a distributed database*. Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, Massachusetts, United States, 2000.

[77] B. Park and H. Kargupta. *Distributed Data Mining: Algorithms, Systems, and Applications*. In The Handbook of Data Mining, edited by N. Ye, Lawrence Erlbaum Associates, 2003.

[78] S. Parthasarathy, M. Zaki, and W. Li. *Memory Placement Techniques for Parallel Association Mining*. Proceedings 4th Int. Conf. on Knowledge Discovery in Databases (KDD'98), AAAI Press), 1998.

[79] S. Peng, S. Mukhopadhyay, R. Raje, M. Palakal, and J. Mostafa. *A Comparison Between Single-agent and Multi-agent Classification of Documents*. In Proceedings of 15th International Parallel and Distributed Processing Symposium, 2001.

[80] D. Pop, V. Negru, and C. Sru. *Multi-agent architecture for knowledge discovery*. In Proceedings of the 8th Intl. Workshop on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC, Timisoara, Romania, 2006.

[81] A. Prodromides, P. Chan, and S. Stolfo. *Meta-Learning in Distributed Data Mining Systems: Issues and Approaches*. In Kargupta, H. and Chan, P. (Eds), Advances in Distributed and Parallel Knowledge Discovery. AAAI Press/The MIT Press, 2000.

[82] F. Provost. *Distributed Data Mining: Scaling Up and Beyond*. In Advances in Distributed and Parallel Knowledge Discovery, edited by H. Kargupta, A. Joshi, and K. Sivakumar, 1999.

[83] J. R. Quinlan. *Induction of decision trees*. Machine Learning 1(1), 1986.

[84] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers, San Francisco, CA, USA. (ISBN 1-55860-238-0), 1993.

[85] J. R. Quinlan and R. M. Cameron-Jones. *FOIL: A Midterm Report*. Proceedings ECML, Vienna, Austria, 1993.

[86] I. Rudowsky. *Intelligent Agents*, volume 14. Communications of the Association for Information Systems, Springer, London, UK, 2004.

[87] R. Schollmeier. *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*. Proceedings in the First International Conference on Peer-to-Peer Computing (P2P01) IEEE, 2001.

[88] E. Shakshuki. *Methodology for Evaluating Agent Toolkits*. Coding and Computing (ITCC05), 2005.

[89] S. Sharples, C. Lindemann, and O. Waldhorst. *A Multi-Agent Architecture For Intelligent Building Sensing and Control*. In International Sensor Review Journal, Yesha, MIT/AAAI Press, 2000.

[90] Z. Shi, H. Zhang, Y. Cheng, Y. Jiang, Q. Sheng, and Z. Zhao. *Mage: An agent-oriented programming environment*. In Proceedings of the IEEE International Conference on Cognitive Informatics, 2004.

[91] T. Shintani and M. Kitsuregawa. *Hash Based Parallel Algorithms for Mining Association Rules*. Proceedings 4th Int Conf. on Parallel and Distributed Information Systems, (PIDS'96), IEEE Computer Society Press, 1996.

[92] Y. Shoham. *Agent-oriented programming*. Artificial Intelligence, 60(1), 1993.

[93] Munindar P. Singh. *Write Asynchronous, Run Synchronous*. IEEE Internet Computing, 3(2), 1999.

[94] S. Stolfo, A. L. Prodromidis, S. Tselepis, and W. Lee. *JAM: Java Agents for Meta-Learning over Distributed Databases*. In Proceedings of the International Conference on KnowledgeDiscovery and Data Mining, 1997.

[95] K. Sycara, A. Pannu, M. Williamson, and D. Zeng. *Distributed Intelligent Agents*. IEEE Expert, 11(6), 1996.

[96] Andreas L. Symeonidis and Pericles A. Mitkas. *Agent Intelligence Through Data Mining*, volume XXVI. Multi-agent Systems, Artificial Societies, and Simulated Organizations, Hardcover ISBN: 978-0-387-24352-8, 2006.

[97] Reticular Systems. *AgentBuilder - An integrated Toolkit for Constructing Intelligence Software Agents*. Acronymics, Inc., 1999. Available at http://www.agentbuilder.com.

[98] M. Tamura and M. Kitsuregawa. *Dynamic Load Balancing for Parallel Association Rule Mining on Heterogeneous PC Cluster Systems*. Proceedings 25th VLDB Conference, Morgan Kaufman, 1999.

[99] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. *An efficient algorithm for the incremental updation of association rules*. Proceedings 3rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1997.

[100] S.R. Thomas. *The PLACA Agent Programming Language*. In M.J. Wooldrige and N.R. Jennings (Eds.), Lecture Notes in Artificial Intelligence, Springer - Verlag, Berlin, Germany, 1994.

[101] B. van Aardt and T. Marwala. *A Study in a Hybrid Centralised-Swarm Agent Community*. IEEE 3rd International Conf. on Computational Cybernetics, Mauritius, 2005.

[102] A. A. Veloso, W. Meira, B. de Carvalho, M.B Possas, S. Parthasarathy, and M. J. Zaki. *Mining Frequent Itemsets in Evolving Databases*. Proceedings Second SIAM International Conference on Data Mining (SDM'2002), 2002.

[103] Ricardo Vilalta, Christophe G. Giraud-Carrier, Pavel Brazdil, and Carlos Soares. *Using Meta-Learning to Support Data Mining*. IJCSA 1(1), 2004.

[104] T. Wagner. *An Agent-Oriented Approach to Industrial Automation Systems*. Agent Technologies, Infrastructures, Tools, and Applications for E-Services, 2002.

[105] WEKA. *Data Mining Software in Java*. The University of Waikato, New Zealand, 1993. http://www.cs.waikato.ac.nz/ ml/weka/.

[106] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufman Publishers, San Fransisco, 1999.

[107] M. Wooldridge. *An Introduction to Multi-Agent Systems*. John Wiley and Sons (Chichester, England), 2003.

[108] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. *Top 10 Algorithms in Data Mining, Knowledge and Information Systems*, volume 14. Springer - Verlag, London Limited 2008, 2008.

[109] X. Yin and J. Han. *CPAR: Classification based on Predictive Association Rules*. Proceedings SIAM Int. Conf. on Data Mining (SDM'03), San Fransisco, CA, 2003.

[110] M. Zaki. *Parallel and Distributed Association Mining: A Survey*, volume 7(4). IEEE Concurrency, 1999.

[111] M. Zaki. *Parallel and Distributed Association Mining: An Introduction*. In Large-Scale Parallel Data Mining (Lecture Notes in Artificial Intelligence 1759), edited by Zaki M. and Ho C.-T., Springer -Verlag, Berlin, 2000.