An investigation into the issues of Multi-Agent Data Mining

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy
by
Kamal Ali Albashiri

February 2010

Dedication

To the joy of my life, Amin.

Preface

This thesis is submitted to the University of Liverpool in support of my application for admission to the degree of Doctor of Philosophy. No part of it has been submitted in support of an application for another degree or qualification of this or any other institution of learning. This thesis is predominantly my own work and the sources from which material is drawn are identified within. Some parts of this thesis have appeared in the following publications:

Refereed Conferences:

[1] K.A. Albashiri, F. Coenen, R. Sanderson, and P. Leng. *Frequent Set Meta Mining: Towards Multi-Agent Data Mining.* Research and Development in Intelligent Systems XXIII, Proceedings AI'2007, Springer, London, pages (139-151), 2007.

[2] K. A. Albashiri, F. Coenen, and P. Leng. *Agent Based Frequent Set Meta Mining: Introducing EMADS.* Artificial Intelligence in Theory and Practice II, Proceedings IFIP'2008, Springer, pages (23-32), 2008.

[3] K. A. Albashiri, F. Coenen, and P. Leng. *EMADS: An Extendible Multi-Agent Data Miner.* Research and Development in Intelligent Systems XXIII, Springer, London, Proceedings AI'2008, Springer, London, pages (263-276), 2008. **Winner of The 2008 British Computer Society Specialist Group on AI (BCS-SGAI) Prize for Best Paper with Student as the First Author.**

[4] K. A. Albashiri, and F. Coenen. *Agent-Enriched Data Mining Using an Extendible Framework.* International Workshop on Agents and Data Mining Interaction, Proceedings ADMI'2009, collocated with AMAS'09, pages (89-106), 2009.

[5] K. A. Albashiri, F. Coenen. *A Generic and Extendible Multi-Agent Data-Mining Framework.* 4th International Conference on Hybrid Artificial Intelligence

Systems, Proceedings HAIS'2009, Springer LNAI 5572, pages (203-210), 2009.

Journal papers:

[6] K. A. Albashiri, F. Coenen, and P. Leng. *EMADS: An Extendible Multi-Agent Data Miner.* Volume 22, Issue 7, pages (523-528). Knowledge-Based Systems (KBS) journal, 2009.

Book chapter contributions:

[7] K. A. Albashiri, and F. Coenen. *The EMADS Extendible Multi-Agent Data Mining Framework.* In Cao, Longbing (Ed.), Data Mining and Multi-agent Integration, Springer, ISBN: 978-1-4419-0521-5, pages (189-200), 2009.

[8] K. A. Albashiri, F. Coenen, and P. Leng. *An investigation into the issues of Multi-Agent Data Mining.* In Bouca, D. and Gafagnao, A. (Eds), Agent Based Computing, Nova Science Publishers, ISBN: 978-1-60876-684-0, 2010.

Abstract

Very often data relevant to one search is not located at a single site, it may be widely-distributed and in many different forms. Similarly there may be a number of algorithms that may be applied to a single Knowledge Discovery in Databases (KDD) task with no obvious "best" algorithm. There is a clear advantage to be gained from a software organisation that can locate, evaluate, consolidate and mine data from diverse sources and/or apply a diverse number of algorithms.

Multi-agent systems (MAS) often deal with complex applications that require distributed problem solving. Since MAS are often distributed and agents have proactive and reactive features, combining Data Mining (DM) with MAS for Data Mining (DM) intensive applications is therefore appealing.

This thesis discusses a number of research issues concerned with the viability of Multi-Agent systems for Data Mining (MADM). The problem addressed by this thesis is that of investigating the usefulness of MAS in the context of DM. This thesis also examines the issues affecting the design and implementation of a generic and extendible agent-based data mining framework.

The principal research issues associated with MADM are those of experience and resource sharing, flexibility and extendibility, and protection of privacy and intellectual property rights. To investigate and evaluate proposed solutions to MADM issues, an Extendible Multi-Agent Data mining System (EMADS) was developed. This framework promotes the ideas of high-availability and high performance without compromising data or DM algorithm integrity. The proposed framework provides a highly flexible and extendible data-mining platform. The resulting system allows users to build collaborative DM approaches. The proposed framework has been applied to a number of DM scenarios. Experimental tests on real data have confirmed its effectiveness.

Contents

List of Figures

List of Tables

Acknowledgement

Thanks to God, my creator, who helped me in the creation of this dissertation.

I would like to especially thank my main supervisor, Dr. Frans Coenen, for his continued support and guidance throughout the past four years. His enthusiasm, constructive criticism, research ideas and always making time for discussions have made the completion of my Ph.D. possible and highly enjoyable. I also would like to express my gratitude to my second supervisor, Prof. Paul Leng, for his assistance, suggestions and valuable comments.

The Department of Computer Science in general at the University of Liverpool has been an excellent place in which to conduct research, and all members of staff and colleagues have been encouraging and helpful whenever needed.

I wish to kindly thank the Libyan's Secretariat of Higher Education for sponsoring my postgraduate studies and living expenses. I also wish to thank my sister and brothers for their encouragement and support.

Last, but by no means least, I am eternally indebted to my wife for her invariable support and encouragement, without whom the completion of my Ph.D. would not have been possible to accomplish.

Chapter 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 harmonise more effectively 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 analysis.

KDD is concerned with the extraction of hidden knowledge from data. Very often data relevant to a particular application of KDD is not located at a single site, it may be widely-distributed and in many different forms. Similarly the solution to a specific single KDD task may be achieved using a variety of algorithms with no obvious advanced indication of which is the most appropriate ("best"). There is therefore a clear advantage to be gained from a software organisation that can (in an automated manner) locate, evaluate, consolidate and mine data from diverse sources and/or apply a diverse number of algorithms.

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

[1] Weka Tool Kit `http://www.cs.waikato.ac.nz/ml/weka/`, and the LUCS-KDD Software Library `http://www.csc.liv.ac.uk/~frans/KDD/Software/`

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.

Multi-Agent Systems (MASs) 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 the challenge of analysing distributed data and offers many algorithmic solutions to perform different data analysis and mining operations in a fundamentally distributed manner.

This thesis addresses a number of research issues concerned with the use of Multi-Agent systems for Data Mining (MADM). The thesis also examines the issues affecting the design and implementation of an extendible framework for achieving the desired MADM. To evaluate the ideas promoted in this thesis a generic MADM framework was established, the Extendible Multi-Agent Data mining System (EMADS) framework. EMADS was developed primarily as a vehicle for promoting the ideas espoused in this thesis, but has also proved to be a useful MADM tool in its own right.

The rest of this chapter is organised as follows. The motivation for the work is presented in Section 1.1. The research objectives of the work are presented in Section 1.2, and the programme of work in Section 1.3. The evaluation strategy is described in Section 1.4. To act as a focus for the research described in this thesis a number of MADM scenarios were considered; these are listed in Section 1.5. The thesis organisation is described in Section 1.6.

1.1 Motivation

There are a number of issues that Data Mining (DM) researchers are currently addressing, including: accuracy, efficiency and effectiveness, privacy and security, and scalability. Accuracy is especially significant in the context of classification. Issues of efficiency and effectiveness pervade the discipline of DM. Issues of privacy and security centre around legal issues and the desire of many owners of data to maintain the copyright they hold on that data. The scalability issue is particularly 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.

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

1. Autonomy.
2. Decentralised control.
3. Robustness.
4. Simple extendibility.
5. Sharing of expertise.
6. Sharing of resources.

Autonomy and decentralised control are, arguably, the most significant features of MAS that serve to distinguish such systems from distributed or parallel approaches to computation. Autonomy and decentralised control imply that individual agents, within MAS, operate in an autonomous manner and are (in some sense) self deterministic [126]. Robustness, in turn, is a feature of the decentralised control, where the overall system continues to operate even though a number of individual agents have disconnected ("crashed"). Decentralised 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 KDD where a considerable collection of tools and techniques are current. There are many specific areas where MASs can be seen to offer benefits with respect to DM, and by extension KDD. Some examples, of specific relevance to this thesis, are as follows:

- **Location of Data**. Very often the data relevant to a particular DM application is not located as a single site, it may be widely-distributed and in many different forms. MASs can be seen as offering a potential solution to mining such data.

- **Sensitivity of Data**. By its nature DM is often applied to sensitive data. Usage of MAS Technology may allow the remote mining of such data without compromising data privacy or security.
- **Protection of Property rights**. MASs can provide a means whereby DM software can be made available to practitioners while at the same time maintaining the intellectual property rights in the software.
- **Diversity of DM Algorithms**. There are a great many DM algorithms available. From the reported research it is clear that there is no single "best" algorithm for any specific DM application. An approach to DM, such as that facilitated by MASs, that can make use of numbers of such algorithms and compare and extract the "best" result, seems desirable.
- **Distributed and Parallel Processing**. Much work on distributed and parallel DM has been conducted. It is suggested that the use of MAS can provide an alternative mechanism for facilitating distributed and parallel DM in a manner that avoids some of the associated overheads of the latter.

The broad motivation for the work described in this thesis is, thus the potential advantages that MAS can offer in the context of DM. 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 decentralised control.

The MADM 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, and DM algorithms 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 thesis is to evaluate the MADM vision, as identified 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?

There are many research challenges to be met if we are to realise the full implementation of the MADM vision. This is, of course, a significant task and the work presented in this thesis can not find the answers to all of the stated issues for all possible cases. Instead, the aim is to identify and study cases where the answer to such issues would benefit answering the research question the most.

The above research question encompasses a number of specific research 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. The usage of such a system and the mechanisms for adding/removing agents.

5. The mechanisms for achieving extendibility to provide the means to easily accept and incorporate new data sources and new DM techniques.

6. The techniques required to support the sharing of resources and expertise.

7. The mechanisms required to ensure the protection of privacy and intellectual property rights.

8. How best to deal with scalability and efficiency concerns.

In summary, the thesis presents an investigation of the MADM vision and the associated conceptualisation. The research was, in part, focused on the construction of EMADS; which was developed to provide a test bed to experiment with approaches and methodologies to address the identified MADM research issues.

Each of the identified research issues was investigated using a number of DM scenarios, and applications commonly found in the domain of DM, and implemented in EMADS.

1.3 Programme of Work

To address the above research objective and associated issues a four phase programme of work was implemented.

1. Investigation into the nature of the agent framework/platform to be used and the establishment of a multi-agent framework to test and evaluate ideas concerning the operation of the envisaged MADM.

2. Research, analysis and evaluation of MADM approaches to address a number of traditional DM activities (scenarios) starting with an agent based approach to Association Rule Mining (ARM), followed by a number of further investigations of DM activities where MADM may have specific benefits. The considered approaches may be itemised as follows:

   (a) Investigation of the operation and benefits of meta Data Mining (meta DM), a process for combining local DM results into a global result.

   (b) Investigations of the usage of MAS to achieve parallel/distributed DM.

   (c) Investigation of a mechanism whereby results from a number of DM algorithms can be compared (and the “best” selected).

   The aim of these studies was to evaluate the effectiveness of the MADM approach in various DM contexts, and to identify and assess issues arising in this realisation in a MADM context.

3. Investigation of the issues associated with the extendibility of MADM with respect to the addition of further functionality, and the adding of additional agents.

4. Final analysis and evaluation of the overall MADM vision.

1.4 Evaluation

This section itemises the criteria used to critically evaluate the work presented in later chapters. The principal aim of the evaluation was to establish whether the MADM proposed vision answered the research question described in Section 1.2. For this to be so, any generic MADM framework will be evaluated according to the following requirements:

- **Generality and Re-usability.** In order to be generic, the framework tasks need to be coordinated. The number of tasks is not known a priori, and may evolve over time. The framework should also be reactive since it must accommodate new tasks as they are created in the environment. Thus the framework should provide for generality. Furthermore, 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 whereby agents can find other agents supporting certain capabilities. It is difficult to define a measure of generality or re-usability. Therefore, in this thesis, generality will be measured by considering the applicability of the proposed MADM solution to a diverse collection of DM scenarios. If the MADM can effectively be applied to these selected scenarios, then it can be argued that the requirement of "generality" has been achieved. In the case of "re-usability" a similar approach will be adopted. A number of scenarios will be considered and observations made of how existing agents from previous scenarios can be used. If the functionality of a reasonable number of existing agents can be re-used to resolve a new DM task that the requirement of "re-usability" can be considered to be fulfilled.

- **Extendibility and scalability.** Extendibility requires the provision of a framework that could easily accept new data sources and new DM techniques. Scalability can be evaluated by considering the following questions:

    1. Can the framework be scaled up ("grow") to accommodate larger, and possibly physically distributed, data sets?
    2. Can DM agents interact with each other at run-time with ease? note

that in other non-agent based systems, these interactions must be determined at design-time (this is discussed further in Chapter 3).

Extendibility, on the other hand, can be evaluated by considering the following questions:

- Is the framework compliant with the standard agent communication language i.e. FIPA ACL [37]?
- Can new DM techniques be added to the system and out-of-date techniques be deleted from the system dynamically? Ease of use and extendibility is evaluated by considering the simplicity where by new agents can be included (and removed) in MADM (this is discussed further in Chapter 4).

Extendibility is one of the principal issues of MADM. Extendibility requires the provision of a framework that can easily accept new data sources and new DM 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; for example, adding a new data source or DM techniques by a simple process. It is suggested in this thesis that the desired extendibility can be achieved using a system of wrappers. An agent wrapper "agentifies" an existing application (i.e. an implementation of a DM algorithm) by encapsulating its implementation to become a MADM agent. In other words, MADM wrappers are used to "wrap" up data mining artefacts so that they become agents that can be incorporated into a MADM framework and consequently can communicate with other agents held in the framework [7].

Three broad categories of wrapper are identified in this thesis: (i) data wrappers, (ii) task wrappers, and (iii) DM wrappers. These are discussed in more detail in Chapter 5.

Overall the aim of the evaluation is to show that the MADM research issues addressed in this thesis (as realised by EMADS) can be addressed by considering a sequence of data mining scenarios designed to determine whether the above listed requirements are achieved. The primary vehicle for conducting the evaluation was the EMADS framework. EMADS is essentially a realisation of the MADM vision promoted in this thesis, and is referred to throughout the rest of this work. More details on the scenarios that were considered are given in Section 1.5. Chapter 4 provides a detailed description of the desired MADM extendibility feature in terms of its requirements, design and implementation.

1.5 Overview of Selected MADM Scenarios

As indicated above, the aim of the demonstration scenarios was to evaluate the effectiveness of the proposed MADM approach in various DM contexts, while at the same time acting as a focus for the research. These scenarios, referred to throughout this thesis, are introduced in this section.

1.5.1 Meta Association Rule Mining (ARM)

The standard, centralised, approach to data mining is to collate data into a single location. In this central location, a model is then computed from the data. Although this process is easy to understand, and the data mining software design is straightforward, there are a number of drawbacks to this centralised approach as descried in Section 1.1. The main objective, in the context of this scenario, is to take advantage of the inherent parallelism and distributed nature of MADM to design and implement a powerful and practical distributed DM system. The scenario assumes several data sites interconnected through an intranet or internet; the goal is then to provide the means for data owners to utilise their own local data and, at the same time, benefit from the data that is available at other data sites without transferring or directly accessing that data (thus maintaining privacy and security). This is realised in the context of MADM, by agents that execute at remote data sites and generate DM models that can subsequently be transferred and merged into one global model.

Meta ARM was chosen as a first scenario because it represented an interesting data mining application which would benefit from a MAS implementation but also lent itself to (relatively straight forward) resolution using a MADM approach that could be used as a "proof of concept" scenario with respect to the MADM mechanisms proposed in this thesis. The scenario also acted as a vehicle to investigate a number of the issues identified in Section 1.2, namely: minimisation of communication, efficiency and scalability, and data privacy protection.

The scenario 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 centralised 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 considered. This scenario is discussed in more detail in Chapter 5.

1.5.2 Data Partitioning and Parallel/Distributed ARM

Data sources measured in gigabytes or terabytes are quite common in DM. This has called for fast DM algorithms that can mine very large databases in a reasonable amount of time. However, despite the many algorithmic improvements proposed in many serial algorithms, the large size and dimensionality of many databases makes the DM of such databases too slow and too big to be processed using a single process. There is therefore a growing need to develop efficient parallel DM algorithms that can run on distributed systems.

The second demonstration scenario was selected to demonstrate that the MADM vision is capable of exploiting the benefits of parallel computing; particularly parallel query processing, parallel data accessing, namely parallel ARM and horizontal/vertical data partitioning. In addition the scenario provides a vehicle for demonstrating how re-usability can be achieved. This was seen as significant in the context of the scalability and efficiency issues.

The scenario comprises an approach to parallel/distributed ARM, which makes use of a vertical/horizontal data partitioning approach to distributing the input data as described in [29]. Using this approach each partition can be mined in isolation by individual agents while at the same time taking into account the possibility of the existence of large itemsets dispersed across two or more partitions (agents). This scenario is discussed in more detail in Chapter 6.

1.5.3 Generation of Classifiers

Multi-Agent Systems (MAS) have some particular potential advantages to offer with respect to KDD, in the context of sharing resources and expertise. Namely, that the MADM approach provides the possibility of greater end user access to DM techniques. MADM can make use of algorithms without necessitating their transfer to users, thus contributing to the preservation of any intellectual property rights over the algorithms.

The third demonstration scenario was chosen to investigate the advantage of MAS with respect to KDD in the context of sharing resources and expertise. The scenario demonstrates that the MAS approach provides greater end user access to DM techniques and can select between such techniques to identify a "best" technique for the considered task. The scenario also demonstrates that MAS can make use of algorithms without necessitating their transfer to users, thus contributing to the preservation of intellectual property rights.

The scenario illustrates the operation of MADM in the context of a classifier generation task where a number of classification algorithms are available. The scenario is thus 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. This scenario is discussed in more detail in Chapter 7.

1.6 Thesis Organisation

The organisation of this thesis is as follows. Chapter 2 provides an overview of the field of MADM. The concepts of association rules are presented in detail as this is central to the three scenarios considered. Other techniques, such as decision tree induction and classification are also covered briefly. Chapter 3 gives an overview of the requirements, design and implementation of EMADS. Chapter 4 provides a detailed description of the desired MADM extendibility feature in terms of its requirements, design and implementation. The three evaluation scenarios are considered in Chapters 5, 6, and 7 respectively. The last chapter presents some conclusions and directions for future work.

1.7 Summary

This chapter has provided the necessary context and background to the research described in this thesis. The motivations for the research, the specific research objectives, and the methodology for realising these objectives, have been described. A literature review of the relevant previous work, with respect to this thesis, is presented in the following chapter.

Chapter 2

Background and literature review

This chapter 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 display significant shortcomings.

The organisation of this chapter is as follows. An overview of Data Mining (DM) and Distributed Data Mining (DDM) is presented in Section 2.1 along with three specific DM techniques: Association Rule Mining (ARM), Classification Rule Mining (CRM), and Decision Tree (DT) generation; which are used for demonstration purposes later in this thesis. In Section 2.2 a general description of agent and multi-agent systems is provided together with a review of current multi-agent system platforms. Finally, the use of agent and multi-agent systems for DM, as an emerging field of DM, is reviewed in Section 2.3, where current research in this field is summarised and some 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 analyse, summarise and extract "knowledge" from this data. The phrase Knowledge Discovery in Databases (KDD) denotes the complex process of identifying valid, novel, potentially useful and ultimately understandable patterns in data [36]. DM 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.

In other words DM [48, 52, 53, 125] deals with the problem of analysing data in a scalable manner.

A considerable number of algorithms have been developed to perform DM tasks, from many fields of science [127]. Typical DM tasks are classification (the generation of classifiers which can be used to assign each record of a database to one of a predefined set of classes), clustering (finding groups of database records that are similar according to some user defined metrics) or ARM (determining implication rules for a subset of database record attributes).

The next two subsections review two data mining techniques, ARM and Classification, which have been used to drive and focus the research reported in this thesis, and to evaluate the solutions to the research issues identified in Chapter 1.

2.1.1 Association Rule Mining

The most popular task of DM is to find patterns 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), and was first introduced in Agrawal et al. [1]. Association Rules (ARs) 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 most commonly used mechanism for determining the relevance of identified ARs is the support and confidence framework. 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 support and confidence of a rule are defined as follows:

- $\mathrm{supp}(X \Rightarrow Y) = \mathrm{supp}(X \bigcup Y)$
- $\mathrm{conf}(X \Rightarrow Y) = \mathrm{supp}(X \bigcup Y) \;/\; \mathrm{supp}(X)$

Using the support and confidence framework, 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.

The most computationally demanding aspect of ARM is identifying the *frequent* sets of attribute-values, or items, whose support (occurrence in the data) exceeds some threshold. The desired ARs are then generated from the identified frequent itemsets. The issue here is that 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. Some of these algorithms are reviewed in the following subsection.

2.1.1.1 Basic ARM Algorithms

There have been many algorithms developed for mining frequent patterns, which can be classified into two categories:

1. Candidate-generation-and-test, and

2. Pattern-growth methods.

The first category, the candidate-generation-and-test approach, such as the Apriori algorithm [5], is directly based on an important property of frequent itemsets: if a pattern (set) with k items is not frequent, none of its super-patterns (supersets) with $(k + 1)$ or more items can be frequent. This is known as the “downward closure property”.

Since its introduction in 1994, the Apriori algorithm, developed by Agrawal and Srikant [5], has been the basis of many subsequent ARM and/or ARM-related algorithms. In [5], 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 "downward closure property" of itemsets in the generation of 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 Apriori algorithm is shown in Table 2.1.

```
Algorithm: Apriori

function Apriori($Dt$: a transactional database, $s$: a support threshold),
   returns a set of frequent itemsets $S$;
begin:
$k \leftarrow 1$;
$S \leftarrow$ an empty set for holding the identified frequent itemsets;
generate all candidate $k$-itemsets from $Dt$;
while (candidate $k$-itemsets exist) do
   determine support for candidate $k$-itemsets from$Dt$;
   add frequent $k$-itemsets into $S$;
   remove all candidate $k$-itemsets that are not sufficiently supported
   to give frequent $k$-itemsets;
   generate candidate $(k + 1)$-itemsets from frequent $k$-itemsets using
   "downward closure property";
   $k \leftarrow k + 1$;
end while
return ($S$);
end Algorithm
Note: A $k$-itemset represents a set of $k$ items.
```

Table 2.1: Apriori Algorithm

The Apriori algorithm 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. A candidate-generation-and-test approach iteratively generates the set of candidate patterns of length $(k + 1)$ from the set of frequent patterns of length k and checks their corresponding occurrence frequencies in the database.

The Apriori algorithm achieves good reduction on the size of candidate sets. However, when there exist a large number of frequent patterns and/or long patterns, candidate generation-and-test methods tend to produce very large numbers of candidates and require many scans of the database for frequency checking. Since, the number of database passes of the Apriori algorithm equals the size of the maximal frequent itemset, it scans the database k times even when only one k-frequent itemset exists. If the dataset is very large, the required multiple database scans can be one of the limiting factors of the Apriori algorithm.

Many algorithms have been proposed, directed at improving the performance of the Apriori algorithm, using different types of approaches. An analysis of the best known algorithms can be found in [54].

A second, more recent, category of methods, pattern-growth methods, such as FP-growth [49] and Apriori-TFP [30], have been proposed. These algorithms typically operate by recursively processing a tree structure into which the input data has been encoded.

FP-growth uses two data structures, the FP-tree and a header table, in which to store the input data. The FP-tree is a set enumeration tree structure that has links going back up, as well as down, the tree; and links between nodes representing the same item. The header table contains links to a "first" item in the tree for each item. The multiple links allow for fast processing. The input data is initially translated into a start FP-tree and header table. FP-growth then recursively processes this start FP-tree structure and header table to generate frequent itemsets staring with 1-itemsets and continuing until no more frequent itemsets can be discovered. At each recursion further FP-trees and header tables are generated. FP-growth is a very fast algorithm, certainly with respect to Apriori, but has some disadvantages. Firstly, because many FP-trees are repeatedly generated, FP-growth can have significant storage requirements. Secondly, the large number of links makes it difficult to distribute the tree. This latter disadvantage means that FP-growth does not lend itself to MADM as envisioned in this thesis. These disadvantages are particularly significant with respect to dense data sets.

Apriori-TFP (Total From Partial) also uses two data structures: a P-tree (Partial support tree) and a T-tree (Total support tree). The input data is first translated into the P-tree; this is a set enumeration tree structure that holds partial support counts for itemsets (unlike the FP-tree). Apriori-TFP then processes the P-tree to produce a T-tree. The T-tree is a "trie" data structure that combines the processing advantages of tree structures with the fast "look-up" (indexing) offered by array structures. The computational efficiency of Apriori-TFP is similar to FP-growth for sparse data sets, and outperforms FP-growth with respect to dense data sets. Apriori-TFP also facilitates simple distribution which offers advantages with respect to MADM as will be demonstrated in later chapters of this thesis.

Overall, no single ARM algorithm has been identified to fit all types of data. Real data sets can be sparse and/or dense according to their applications. For example, for telecommunication data analysis, calling patterns for home users vs. business users can be very different: some are frequent and dense (e.g. to family members and close friends), while others are large and sparse. Similar situations arise for market basket analysis, census data analysis, etc. It is hard to select an appropriate ARM method (on the fly) when no algorithm fits all.

Finally, large applications need more scalability. Many existing methods are efficient when the data set is not very large. Otherwise, their core data structures (such as FP-tree) or the intermediate results (e.g., the set of candidates in Apriori or the recursively generated sub-trees in FP-growth) may not fit in main memory and may cause "thrashing".

2.1.1.2 ARM Example

A typical example of ARM is market basket analysis. This process analyses customer-buying habits by finding associations among the different items that customers place in their shopping baskets. The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, market basket analysis may help managers optimise different store layouts. If customers who purchase milk also tend to buy bread at the same time, then placing the milk close or opposite to bread may help to increase the sales of both of these items.

TID	**Items**
1	cereal coke milk
2	cereal coke mustard sausage
3	cereal bread butter
4	cereal bread butter cigarettes milk
5	cereal bread nappies milk sausage
6	cereal bread milk
7	cereal butter candy cigarettes nappies
8	cereal candy nappies mustard
9	coke bread butter mustard sausage
10	coke bread candy
11	coke bread milk mustard sausage
12	coke butter sausage
13	candy cigarettes

Table 2.2: Market Basket Transaction Data

Table 2.2 illustrates a small database of market basket transactions, labelled with their Transaction ID (TID). The following describes the actual steps of applying the Apriori algorithm to the data shown in Table 2.2 to generate Association Rules (ARs). The support threshold has been set to 0.2 (i.e. for an item to be considered frequent it must appear in at least 20% of the records). Note that each identified frequent itemset is delimited by round brackets, and that the support for each itemset is given.

- **Limit:** support threshold ≥ 0.2
- **1st iteration:** frequent 1-itemsets
  - (cereal):0.62, (coke):0.46, (mustard):0.31, (bread):0.54, (butter):0.38, (candy):0.31, (cigarettes):0.23, (nappies):0.23, (milk):0.38, (sausage):0.38
- **2nd iteration:** frequent 2-itemsets
  - **Candidates:** all pairs of the above

  - **Frequent:** (cereal, bread):0.31, (cereal, nappies):0.23, (cereal, milk):0.31, (coke, bread):0.23, (coke, mustard):0.23, (coke, sausage):0.31, (bread, butter):0.23, (bread, milk):0.31, (bread, sausage):0.23, (mustard, sausage):0.23

- **3rd iteration:** frequent 3-itemsets
  - **Candidates:** (cereal, bread, milk), (coke, bread, sausage), (coke, mustard, sausage)
  - **Frequent:** (cereal, bread, milk):0.23, (coke, mustard, sausage):0.23

- **4th iteration:** frequent 4-itemsets
  - No more candidates

After the Apriori algorithm has discovered all the frequent itemsets, ARs can be derived from those itemsets; for example:

- **Sets** (coke, mustard, sausage):0.23 and (coke, sausage):0.31
- **Rule** (coke, sausage) → mustard
  - Support: 0.23
  - Confidence: 0.23/0.31 = 70%
- **Sets** (cereal, nappies):0.23 and (nappies):0.23
- **Rule** nappies → cereal
  - Support: 0.23
  - Confidence: 0.23/0.23 = 100%

The above gives a sample of some of the ARs that exist in the input input data. The numbers show the support (the number of occurrences of the item set in the data divided by the total number of records) and the confidence (accuracy) of the rule.

An example of the interpretation of this type of rule is the statement that “in 70% of transactions in which coke and sausage were purchased, mustard was also purchased, and 23% of all transactions contain all three items”. In this case the antecedent of this rule (X) consists of coke and sausage and the consequent (Y) is mustard. The 70% represents the confidence factor of this rule and the 23% is the support for the rule. The rule can then be specified as coke $\cup$ sausage $\rightarrow$ mustard. Both the antecedent and consequent can comprise sets of items, or can be a single item.

2.1.2 Classification Rule Mining

Classification is a well established data mining task, with roots in machine learning. 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 value (class) "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.

The aim of classification algorithms is to generate a classifier (a set of rules in a given database). The classifier may be expressed in a number of different ways; one method is as a set of Classification Rules (CRs). Classification Rule Mining (CRM) [98] is a well-known classification technique for the extraction of hidden CRs. Classification rules can be considered as 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 typically divided into two mutually exclusive data sets, the training set and the test set. The DM 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 [51].

In the following two subsections two specific classifier generation techniques, which are of particular relevance to this thesis, are described: classification association rule mining and decision tree based methods.

2.1.2.1 Classification Association Rule Mining

An overlap between ARM and CRM is CARM (Classification Association Rule Mining), which strategically solves the traditional CRM problem by applying ARM techniques. The idea of CARM, first introduced in [8], is to extract a set of Classification Association Rules (CARs) from a class-transactional database, where all database attributes and the class attribute are valued in a binary manner. A CAR is a special form of AR that describes an implicative co-occurring relationship between a set of binary-valued data attributes and a pre-defined class. As such the consequent part of the ARs is restricted (typically) to a single value of a given class attribute.

As suggested in [32], CARM offers a number of advantages over other CRM approaches with respect to performance efficiency. Coenen et al. [32] indicate:

- "Training of the classifier is generally much faster using CARM techniques than other classification generation techniques such as decision tree (induction) and Support Vector Machine (SVM) [58] approaches (particularly when handling multi-class problems as opposed to binary problems)".

- "Training sets with high dimensionality can be handled very effectively".

- "The resulting classifier is expressed as a set of rules which are easily understandable and simple to apply to unseen data (an advantage also shared by some other techniques, e.g. decision tree classifiers)".

2.1.2.2 Classification Rule Generation Using 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 [98], already include a class ruleset generator.

Let us consider a very well known example, taken from [98]. 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 2.3 shows the original training set, given as a relational table over the attributes Outlook, Temperature, Humidity, and Wind. The last column (class or target attribute) of the table represents the classification of each row.

Outlook	**Temperature**	**Humidity**	**Wind**	**Class**
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Low	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 2.3: Training set of examples on weather attributes

Several algorithms have been developed to mine a decision tree from datasets such as the one in Table 2.3. Almost all of them rely on the basic recursive schema used in the ID3 algorithm [97]. The ID3 algorithm is shown in Table 2.4. The ID3 algorithm is used to build a decision tree, given a set of non-categorical attributes R1, R2, .., Rn, the categorical attribute C, and a training set of records

(S).

Algorithm: ID3

```
function ID3 ($R$: a set of non-categorical attributes,
   $C$: the categorical attribute, $S$: a training set), returns a decision tree;
begin
If S is empty, return a single node with value Failure;
If S consists of records all with the same value for the categorical attribute,
   Return a single node with that value;
If R is empty, then
   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)
Let D be the attribute that best classifies S;
Let {dj| j=1,2, .., m} be the values of attribute D;
Let {Sj| j=1,2, .., m} be the subsets of S consisting respectively of records
 with value dj for attribute D;
Return a tree with root labelled D and arcs labelled d1, d2, .., dm going
   respectively to the trees
ID3(R-D, C, S1), ID3(R-D, C, S2), .., ID3(R-D, C, Sm);
end ID3;
```

Table 2.4: ID3 Algorithm

Differences between decision tree algorithms are usually in the splitting criteria used to identify the (local) best attribute for a node. Using a standard decision tree inductive algorithm, we may obtain the decision tree in Figure. 2.1 from the training set in Table 2.3. Recall that, 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 labelled by the leaf node. For instance, given the above tree, the example Outlook = Sunny, Humidity = Low is classified as Yes, whereas the example Outlook = Sunny, Humidity = High is classified as No.

Note 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 Outlook = Sunny can be classified both as Yes or No, following the two left-most branches of the tree: in this case many decision tree

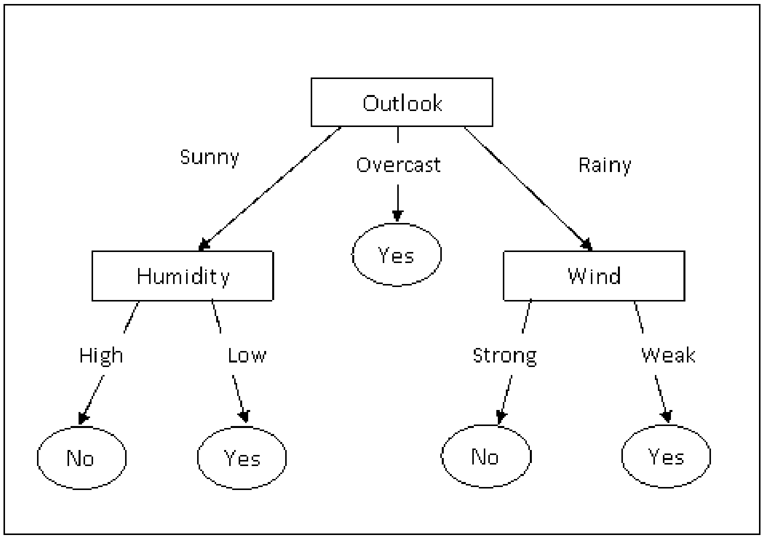


Figure 2.1: Decision Tree Example

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.

2.1.2.3 Classification Rule Generation Examples

Assume that the data shown in Table 2.3 is a record of the weather conditions during a two-week period, along with the decisions (class) of a tennis player whether or not to play tennis on each particular day. Thus tuples have been generated (or examples, instances) consisting of values of four independent variables (outlook, temperature, humidity, windy) and one dependent variable (class) play.

By applying various DM techniques, ARs and CARs can be found to extract knowledge in the forms of rules, decision trees etc., or just predict the value of the dependent variable (play) in new situations (tuples). Some examples (all produced by Weka [124]) are presented in this section demonstrating the classification generation process using: (i) CARM (Example 1) and (ii) decision tree mining (Example 2). Both examples use the dataset given in Table 2.3.

Example 1: Mining Classification Association Rules (CARs)

To find ARs in data, first the numeric attributes (a part of the data pre-processing stage in DM) have to be discretised. Thus for the data in Table 2.3, the temperature values are grouped in three intervals (hot, mild, cool) and humidity values in two (high, low) and substitute the values in data with the corresponding names. Then an ARM algorithm can be applied to generate the following ARs:

1. humidity=low, windy=weak → play=yes (0.29, 100%)
2. outlook=overcast → play=yes (0.29, 100%)
3. outlook=rainy, windy=weak → play=yes (0.21, 100%)
4. temperature=cool, windy=weak, humidity=low → play=yes (0.14, 100%)
5. temperature=cool, humidity=low, windy=weak → play=yes (0.14, 100%)

The rules show relationships within attribute value sets (the so called itemsets) that appear frequently in the data. The numbers after each rule show the support (the number of occurrences of the item set in the data divided by the total number of records) and the confidence of the rule.

Example 2: Classification by Decision Trees and Rules (CARs)
Using the ID3 algorithm, the following decision tree can be produced (shown in Figure. 2.1):

- outlook = sunny
    - humidity = high: no
    - humidity = low: yes
- outlook = overcast: yes
- outlook = rainy
    - windy = strong: no
    - windy = weak: yes

The decision tree consists of decision nodes that test the values of their corresponding attribute. Each value of this attribute leads to a sub-tree and so on, until the leaves of the tree are reached. The leaves determine the value of the dependent variable (class). Using the decision tree we can classify new tuples (not used to generate the tree). For example, according to the above tree the tuple {sunny, mild, low, weak} will be classified under play=yes.

A decision tree can be represented as a set of rules, where each rule represents a path through the tree from the root to a leaf. Given the decision tree presented in Figure 2.1 the following classification rules can be generated. Other DM techniques can produce rules directly.

1. outlook = overcast → yes
2. outlook = rainy, windy = weak → yes
3. outlook = rainy, windy = strong → no
4. outlook = sunny, humidity = low → yes
5. outlook = sunny, humidity = high → no

2.1.3 Distributed Data Mining

A basic approach for DM is to move all of the data to a central data repository (data warehouse) and then to analyse this with a single DM system. Even though this approach allows the production of a global model using all the data, it might be infeasible in many cases [96]. The cost of transferring large blocks of data may be prohibitive and result in very inefficient implementations. An alternative approach is the in-place strategy, as exemplified by DDM systems in which all the data can be locally analysed (local 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.

Distributed Data Mining (DDM) is a branch of DM that offers a framework to mine distributed data paying careful attention to the nature of the distribution and computing resources. The control and communication mechanisms used in DDM systems represent a significant computational overhead that can significantly affect the operation of these systems. Since communication is assumed to be carried out exclusively by message passing, a primary goal of many DDM methods (reported in the literature) is to minimise 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.

Surveys [64] and [90] provide a broad, up-to-date, overview of DDM, touching on DM tasks such as: clustering, ARM, basic statistics computation, Bayesian network learning, classification, and the historical roots of DDM. The collection of papers in Kargupta and Chan [61] describes a variety of DDM algorithms (ARM, clustering, classification, pre-processing, etc.), systems issues in DDM (security, architecture, etc.), and some topics in parallel DM. Survey [129] discusses parallel and distributed ARM in DDM. Survey [130] discusses a broad spectrum of issues in DDM and parallel DM and provides a survey of distributed and parallel ARM and clustering. Other DDM applications [103, 60] deal with continuous data streams. An overview of the data stream mining literature can be found in [11].

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 [101]. In the latter case, individual nodes might communicate with a resource rich centralised node, but they perform most of the tasks by communicating with neighbouring 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.

The main problems that challenge any approach to DDM are concerned not only with scalability, but also with issues of autonomy and privacy. For example, when data can be viewed from many different perspectives and at different levels of abstraction, it may threaten the goal of protecting individual data and guarding against the invasion of privacy.

The ever growing amount of data that is stored in a distributed form, over networks of heterogeneous and autonomous sources, poses several problems for knowledge discovery and DM, such as: communication minimisation, autonomy preservation, scalability, and privacy protection.

To address these issues, much research effort has been directed at the identification of more advanced approaches of combining local models built at different sites. Most of these approaches are agent-based high level learning strategies.

2.2 Agents and Multi-Agent Systems

Agents and multi-agent systems are an emergent technology that is expected to have a significant impact in realising the vision of a global and information rich services network to support dynamic discovery and interaction of digital enterprises. Significant work on multi-agent systems has already been done for more than a decade since agents were first claimed to be the next breakthrough in software development, resulting in powerful multi-agent platforms and innovative e-business applications. The rest of this section gives a brief overview of agent and multi-agent systems and then describes some of the most common multi-agent system development platforms in the literature.

2.2.1 Agents

Agents are defined by Wooldridge [126] 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 [100, 126] 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, modelled to perform specific tasks and capable of autonomous action and decision making. The agent modelling paradigm can be looked upon as a stronger encapsulation of localised 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).

2.2.2 Multi-Agent Systems

By combining multiple agents, in one system, to solve a problem, the resultant system is a Multi-Agent System (MAS) [126]. These systems are comprised of agents that individually solve problems that are simpler than the overall 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 [79] can now be solved, by localising the problem solving [119]. For example, MAS have been used in stock market prediction [77], industrial automation [123] and e-learning [40].

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

2.2.3 FIPA

The Foundation for Intelligent Physical Agents, FIPA [37] is a non-profit making international organisation established in 1996 and registered in Geneva, Switzerland. It is 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-compliant platform.

The model proposed by FIPA agent standards [88] consists of concrete specifications enabling interoperability, based on a high-level abstract architecture. The benefits of this will include increased re-use, together with ease of upgrade. Early adopters of new technology tend to be wary when there is no commonly agreed standard, which lacks the support of a large consortium of companies, FIPA provides such a standard in the context of agents. The existence of an agent standard provides added confidence to potential adopters of this technology. Finally, the standardisation process shifts the emphasis from longer-term research issues to the practicalities of realising commercial agent systems.

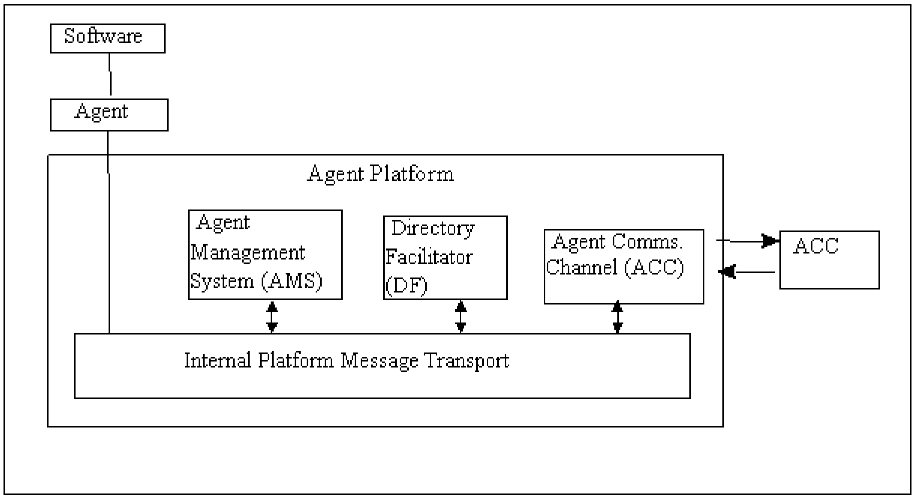

Software
Agent
Agent Platform
Agent Management System (AMS)
Directory Facilitator (DF)
Agent Comms. Channel (ACC)
ACC
Internal Platform Message Transport

Figure 2.2: FIPA Agent Management Reference Model [37]

The FIPA agent reference model (Figure 2.2) currently provides the normative framework within which FIPA Agents exist and operate. Combined with the Agent Life cycle, it establishes the logical and temporal contexts for the creation, operation and dispatching of Agents. The model comprises a number of components. The Directory Facilitator (DF), Agent Management System (AMS), Agent Communication Channel (ACC) and Internal Platform Message Transport (IPMT) form what is termed the Agent Platform (AP). The DF provides a "yellow pages" services to other agents. The AMS provides a "white-pages" services and life-cycle management services for agents, and the ACC supports inter-agent communication. The ACC supports interoperability both within and across different platforms. The Internal Platform Message Transport (IPMT) provides a message forwarding service for agents on a particular platform. These are mandatory, normative components of the model. The ACC, AMS and DF are capability sets, they may be performed by one agent or by three different agents this is left to the agent platform developer.

To be minimally FIPA-compliant requires compliance to the Agent Management FIPA specification [37], the Agent Communication Language FIPA specification and the (non-agent) software-agent integration specification.

2.2.4 MAS Development Platforms

To develop a MAS effectively, the developers have to address issues such as: agent characteristics, agent functionalities, protocols, communication, coordination and co-operation. Furthermore agent-based systems need to be robust, scalable and secure. To achieve this, the development of open, stable, scalable, and reliable architectures that allow agents to locate one another, communicate and offer services to one another, are required.

To date several agent development platforms and toolkits have been produced [19]. Examples include: AgentBuilder, AgentTool, ASL, Bee-gent, Grasshopper-2, MOLE, Open Agent Architecture, RETSINA, Zeus, FIPA-OS, and JADE. Each is briefly reviewed below.

AgentBuilder [114] 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, namely an interpreter, used as the execution environment for agent software. AgentBuilder agents are based on a model derived by the Agent-0 [106] and PLACA [116] agent models.

AgentTool [34] is a graphical environment to build heterogeneous multi-agent systems. It is a kind of CASE (Computer Assisted Software Engineering) tool, specifically oriented towards agent-oriented software engineering, whose major advantages are: (i) the complete support for the MaSE methodology [33] (developed by the same authors together with the tool), and (ii) 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 [68] is an agent platform that supports development in C/C++, Java, JESS, CLIPS and Prolog. ASL is built upon the OMG's CORBA 2.0 specifications. The use of CORBA technology facilitates seamless agent distribution and allows usage of the language bindings supported by the used CORBA implementations. Initially, ASL agents communicated through KQML messages. The platform is FIPA-compliant supporting FIPA ACL.

Bee-gent [65] is a software framework to develop FIPA-compliant agent systems that has been realised by Toshiba. The framework provides two types of agents: (i) wrapper agents used to agentify existing applications, and (ii) 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 [15] is a pure Java based Mobile Agent platform, conformant to existing agent standards, as defined by the OMG - MASIF (Mobile Agent System Interoperability Facility) [84] and FIPA specifications. 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 agents 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 [14] is an agent development system developed in Java. MOLE agents do not have a sufficient set of features to be considered to be truly agents [41, 108]. 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 [78] is a truly open architecture intended 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. The facilitator can receive tasks from agents, decompose them and award them to other agents.

RETSINA [112] offers reusable agents to realise specific 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 [87] allows the rapid development of Java agent systems by providing a library of agent components. Zeus supports: (i) a visual environment for capturing user specifications, (ii) an agent building environment that includes an automatic agent code generator, and (iii) a collection of classes that form the building blocks of individual agents. Agents are composed of five layers: API layer, definition layer, organisational layer, coordination layer and communication layer. The API layer facilitates the interaction with the non-agentised world.

JATLite (Java Agent Template, Lite) [56] is a MAS development toolkit written in the Java programming language. JATLite is a package of Java classes and programs that allow users to create new systems of software agents that communicate over the Internet in order to perform distributed computation. The agents may be newly created software or legacy software "wrapped" with software that generates and receives agent messages as an integration mechanism. JATLite allows agent communications through its message routing scheme which operates over a networked environment, however it is not a FIPA-compliant platform.

FIPA-OS (FIPA Open Source) [94] is an open source implementation of the mandatory elements contained within the FIPA specification for agent interoperability. In addition to supporting the FIPA interoperability concepts, FIPA-OS provides a component based architecture to enable the development of domain specific agents which can utilise the services of the FIPA Platform agents. The primary aim of FIPA-OS is to reduce the current barriers in the adoption of FIPA technology by supplementing the technical specification documents with managed open source code.

JADE (Java Agent Development Environment) [17], is a middle-ware that enables development of applications based on the agent paradigm. JADE is fully implemented in the Java programming language. JADE simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases. A JADE agent platform can be distributed across machines (which do not even need to share the same OS) and the configuration can be controlled via a remote GUI.

Shakshuki [102] presents a methodology for evaluating agent toolkits based on criteria such as availability, environment, development, characteristic properties and performance. His findings recommended JADE as one of the top-ranking toolkits according to their criteria. Chmiel et al. [26] 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.

Consequently the research described in this thesis has made use of the JADE platform to engineer a real world multi-agent data mining system (EMADS) for evaluation purposes. JADE was also chosen because it is open source, popular, easy to use and FIPA-compliant. JADE is described in further detail in Chapter 3.

2.3 Multi-Agent Data Mining (MADM)

There are two themes of agent and DM interaction and integration in the literature [23]: DM for agents, referred to as mining-driven agents [113]; and agents for DM, referred to as agent-driven DM, commonly known as Multi-Agent Data Mining (MADM). The former concerns issues of transforming the discovered knowledge, extracted by DM, into the inference mechanisms or simply the behaviours 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 [113]. 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 DM techniques.

MADM, rather than mining-driven agent systems, is the focus of this thesis. It is concerned with the use of agent and MAS to perform DM 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 individual agent-based DM architectures can provide. This section is not concerned with particular DM techniques. It is however concerned with the collaborative work of distributed software in the design of multi-agent systems directed at DM.

Several systems have been developed for MADM. These systems can be categorised, according to their strategy of learning, into three types:

1. Central-learning

2. Meta-learning
3. Hybrid-learning

Each is described in some further detail in the following subsections.

2.3.1 Central-learning Strategy

A central learning strategy is when all the data can be gathered at a central site and a single model 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 quantity of geographically distributed data is small and there are no issues of privacy and security that would prohibit the movement of the data. For large volumes of data, the strategy is generally very expensive but also more accurate in its DM results [13, 20] when compared to alternative approaches. 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 some appropriate merging strategy. However, as noted previously, in many cases this strategy is infeasible because of security and privacy of data requirements and for the reasons outlined in [96].

One of the earliest references to MADM that adopted a central-learning strategy can be found in Kargupta et al. [62] who describe a parallel DM system (PADMA) that uses software agents for local data accessing and analysis, and a web based interface for interactive data visualisation. PADMA has been used in medical applications.

Peng et al. [92] presented an interesting comparison between single-agent and multi-agent text classification using the central-learning strategy in terms of a number of criteria including: response time, quality of classification, and economic/privacy considerations. Their results indicated, not unexpectedly, in favour of a multi-agent approach.

2.3.2 Meta-learning Strategy

Meta-learning is the process of automatic induction of correlations between tasks and solving strategies, based on a domain characterisation. Meta-learning methods have been widely used within DM [122, 21], particularly in the area of classification and regression.

With respect to classification 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. 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 [95]).

One of the most popular MADM approaches that adopt the meta-learning strategy is the METAL project [82] whose emphasis is on helping the user to obtain a ranking of suitable DM algorithms through an on-line advisory system. Gorodetsky et al. [44] correctly consider that the core problem in MADM is not the DM 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 DDM and, specifically, classification. A more recent system, proposed in [76], uses the MAGE middle-ware [104] to build an execution engine that uses a directed acyclic graph to formalise the representation of the KDD process. In [93] a multi-agent system for KDD (AgentDiscover) has been proposed. It uses task-based reasoning [24] for problem solving on a multi-agent platform. Perhaps the most mature agent-based meta-learning systems are: JAM [111], and BODHI [63]. JAM and BODHI are both intended for data classification.

2.3.3 Hybrid-learning Strategy

A hybrid learning strategy is a technique that combines local and remote learning for model building [46]. An example of a hybrid learning framework is Papyrus [12]. 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. Papyrus is a specialised system which is designed for clustering. These systems are reviewed in detail in [70].

2.4 Generic MADM

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

- Classification Task: includes [44, 111, 63, 62, 39, 43, 67].
- Association Rule Mining Task: includes [85, 89, 27, 57, 66].
- Clustering Task: includes [12, 69, 70, 107, 42].

Given the above literature review, and to the best knowledge of the author, there have been only a few MADM systems that define a generic MADM framework. An early attempt was IDM [20]. IDM is an agent-based data mining system designed to support predefined and ad-hoc data access, data analysis, data presentation, and data mining requests from non-technical users over a data warehouse. The IDM distributed environment is realised using JATLite. The developers of IDM claim that IDM can rapidly analyse data because it delegates data mining tasks to multiple agents. However, since JATLite is not FIPA-compliant, IDM can not be integrated further with FIPA-compliant applications. IDM also has limited user friendliness, meaning users are expected to correctly set the goals when interacting with IDM and to have a good understanding of the technology used in its development. Another example of JATLite based framework with very little information on the testing part of the system is the work of Zhang et al. [131].

In [10] a generic task framework was introduced, however designed to work only with spatial data. The most recent system was introduced in [35] where the authors proposed a MAS to provide a general framework for DDM applications. The effort to embed the logic of a specific domain has been minimised and is limited to the customisation of the user. However, although its customisable feature is of a considerable benefit, it still requires users to have very good DM knowledge.

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

The major criticism of the above systems is that it is not always possible to obtain an exact final DM result (knowledge model), 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 alternative approaches to MADM hardware resource usage is not optimised. If the "heavy" computation 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 knowledge models are induced from databases that have different schemas and hence are incompatible.

2.5 Summary

The literature review, presented in this chapter, has shown that: (i) KDD (in particular, data mining) has a number of outstanding problems in aspects such as scalability, security, etc. which remain the subject of research; (ii) Distributed Data Mining may solve some of these problems, but not all, and introduces other problems; (iii) MAS offers some promising possibilities. However, MADM systems, to the best knowledge of the author, are often unique to the application under consideration or very specific to one data mining task (e.g. classification).

One of the objective of the research described in this thesis is to consider whether MAS can provide a generic MADM solution.

Chapter 3

MADM: Requirements Analysis, Design, and Implementation

As described in Chapter 1, the MADM vision proposed in this thesis is that of an "anarchic" collection of persistent, autonomous (but cooperating) KDD agents operating across the Internet. To investigate the nature of the envisaged MADM, its operation, its architecture, and the underlying research issues, identified in Chapter 1, the investigation was supported by establishing a working system. This system was termed EMADS (Extendible Multi-Agent Data mining System). This chapter describes the under-pinning philosophy of MADM frameworks and, by extension, the view of the EMADS framework proposed in this thesis. This chapter concentrates on the generic features of MADM and describes the EMADS framework to support the MADM vision.

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

To realise the EMADS framework a general software development methodology was adopted that moved from problem definition and analysis to detailed design and implementation.

This chapter provides a detailed description of the framework in terms of its requirements, design and architecture. The chapter commences by reviewing the MADM (EMADS) requirements (Section 3.1). To fully understand and analyse the requirements, the section commence by considering a number of MADM issues before moving on to the structural and operational requirements. A review of the EMADS agents and users is then presented in Section 3.2 in the context of the identified requirements. In Section 3.3 the EMADS agent interaction protocols are defined; followed, in Section 3.4, with a discussion of the proposed generic data mining process using EMADS agents. In Section 3.5 details concerning the Java Agent Development Environment (JADE) [17], the agent development toolkit that was used to implement EMADS, are presented. The JADE implementation of EMADS is then detailed in Section 3.6. A discussion of how EMADS fulfils the identified MADM requirements is presented in Section 3.7. Finally, Section 3.8 presents a summary of this chapter.

3.1 MADM Requirements Analysis

Developing a data mining system that uses specialised agents with the ability to communicate with multiple information sources, as well as with other agents, requires a great deal of flexibility. For instance, adding a new information source should merely imply adding a new agent and advertising its capabilities; a process that should be facilitated in such a way that it is as simple as possible. As noted above the motivation for researching and implementing a fully operational MADM framework was to facilitate the investigation of the various MADM research challenges and issues outlined in Chapter 1.

3.1.1 Design Requirements

The realisation of the desired MADM framework requires the consideration of a number of requirements. These requirements are itemised in this section.

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

2. **Agent Coordination:** Following on from issue number 1, the framework must be reactive since it must accommodate new agents as they are created in the environment. Careful consideration therefore needs to be directed at the communication mechanisms.

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

   There are potentially a large number of agents that must be coordinated within any generic MADM 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 efficiency of the system. Most of the current generation of DM learning algorithms are computationally complex and require all data to be resident in main memory, which is clearly implausible for many realistic problems and databases. At the same time the framework must be scalable: avoiding centralised components which would create bottlenecks during execution.

5. **Portability:** A distributed data mining system should be capable of operating across multiple environments with different hardware and software configurations (e.g. across the Internet), and be able to combine multiple models with (possibly) different representations. The framework should be able to operate on any major operating system. In some cases, it is possible that the data could be downloaded and stored on the same machine as the data mining software.

6. **Compatibility:** Combining multiple models of data mining results has

been receiving increasing attention in the data mining research literature. In much of the prior work on combining multiple models, it is assumed that all models originate from the same database or from databases with identical schema. This is not always the case, and differences in the type and number of attributes among different data sets are not uncommon. The resulting model computed at a single database is directly dependent on the format of the underlying data. Minor differences, in the schema, between databases derive incompatible models, i.e. a classifier cannot be applied on data of different formats. Yet, these classifiers may target the same concept.

The framework must be able to operate using several data sources located on various machines, and in any geographic location, using some method of network communication.

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

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

In the following sections the structure and operational requirements of the desired MADM framework are considered in detail. Given their significance in the context of MADM, the extendibility requirements are discussed in detail in Chapter 4.

3.1.2 Structural Requirements

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

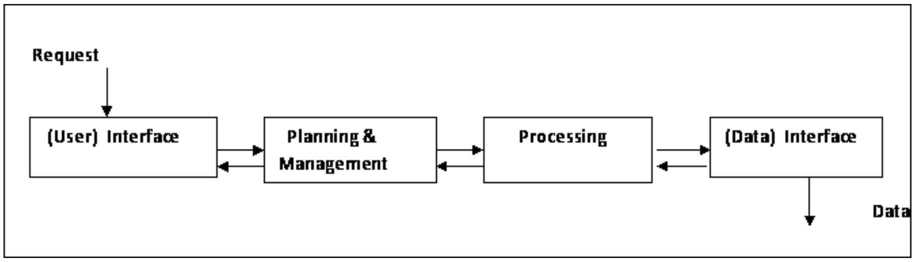


Figure 3.1: Main Domain Components

3.1.3 Operational Requirements

Most current agent-based data mining frameworks (see Chapter 2) share a similar high-level architecture, and provide common structural components, to that shown in Figure 3.1. Components of the form described above have become a template for most agent-based data mining and information retrieval systems. The structure illustrated in Figure 3.1 sets out three important elements of MADM systems: (i) agent technology, (ii) domain components, and (iii) information brokerage (middle-ware). Agent technology is a self evident element of MADM. Multi-Agent Systems (MAS) espouse the use of collaborative agents, operating across a network, and communicating by means of a high level query language such as KQML and FIPA ACL. Domain components or ontologies, give a concise, uniform description of semantic information, independent of the underlying syntactic representation of the data. Finally, information brokerage utilises specialised facilitator agents to match information needs with currently available resources, so (for example) retrieval and update requests can be properly routed to the relevant resources.

Given the above considerations the general operation of EMADS (as suggested in Figure 3.1) is as follows:

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

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

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

4. **Result:** Return the results to the user.

These steps are fairly generic and could be the foundation for any envisaged MADM.

3.2 EMADS Agents and Users

In this section the different types of EMADS agents and users are detailed. The discussion focuses on the functionality of the agents; the implementation is considered later in this chapter. Based on the requirements identified in Subsection 3.1, several types of EMADS agent were identified. However, regardless of "type", EMADS' agents adhere to the general agent definitions described in [126]. The

EMADS agent types are: User, Management, Facilitator, Data Source, Data Mining (DM), and Registration. Each falls within the domain level concepts identified in Subsection 3.1.2. The user and data source agents are all identified as interface agents because they all provide "interfaces" to either users, or data sources. User agents provide the interface between EMADS end users and the rest of the framework; whilst data source agents provide the interface between input data and the rest of the framework. Task agents and Data Mining (DM) agents are identified as processing agents because they carry out the required "processing" needed to respond to user requests, and possibly, to pre-process data within the system.

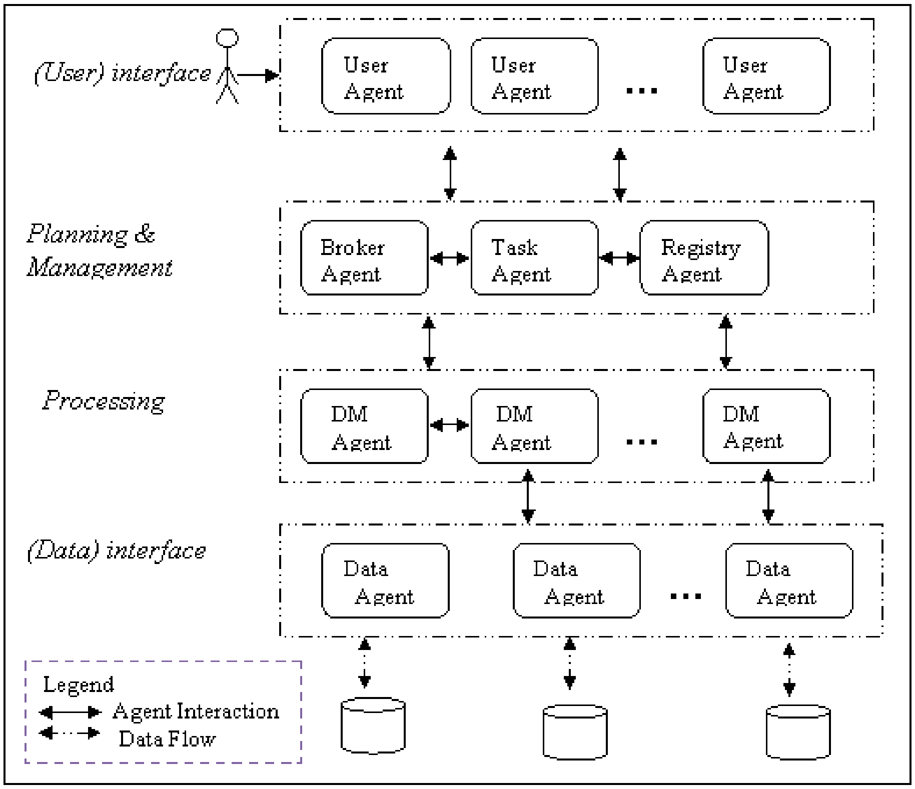


Figure 3.2: System General Architecture

Figure 3.2 shows what agents will reside in each domain component identified in the structural requirements analysis in Subsection 3.1.2. The task agent (middle of the management layer in Figure 3.2) receives a request and asks the broker agent to check all available databases (data agents) and DM agents to find: (i) which data to use, and (ii) which data mining algorithms (held by DM agents) are appropriate. The task agent then passes the task to DM agents and monitors their progress. Each database must have an interface agent (data agent) to check the database for a matching schema and then report back to a DM agent. Figure 3.2 also shows the system level inputs and outputs as they flow from agent to agent.

The agents are kept independent from any particular DM function. They were defined with respect to the high level objectives identified in the requirements presented previously in Section 3.1. In general, it is convenient to think of an EMADS agent as formed by three interacting software modules: (i) the interface module, (ii) the process module and (iii) the knowledge module. Note that modules are only included when they are needed.

The following subsections describe the structure and function of each agent from a high level perspective and in terms of the three identified software modules. 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 utilised to process an EMADS' user request in order to illustrate how each agent's function fits into the system's overall operation and contributes to supporting the other agents. A brief overview of the various categories of end users is also presented.

3.2.1 User Agent

There are four essential operations that a user agent must be able to undertake:

1. Receive and interpret user data mining requests.

2. Communicate the user request to the processing agents.

3. Return the generated results, in a suitable and easily understandable format, back to the user.

4. Expect and respond to asynchronous events, such as "a stop mining" or "end operations" instructions that may be issued by the user.

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

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

3.2.2 Registration Agent

The general excepted function of the registration is to inform all appropriate agents that are already in the system and interested of a new agent arrival. When any new agent is introduced into the system, it must first inform the registration agent that it has entered. Since the system is designed to be relatively static in terms of new data sources and data source types, the registration agent will be the least utilized agent. It should also be the first agent created and started in the system. Because the system cannot operate without a broker agent as well, the registration agent will initiate all methods, and then await a broker agent to enter the system. Once notified a broker has entered the system, it completes initialization and waits for a registration request from any new agents. When a new agent enters the system, it sends a registration request message to the registration agent. When it receives notification of a new agent, the registration agent will determine the functions the agent can perform by the information transmitted in the registration message. The broker agent will be informed of all types of agents entering the system. Once all appropriate existing system agents have been notified, the new agent will be informed it is active in the system. However, in the work described in this thesis the registration agent was eliminated and its functionality was integrated with the broker agent. The agent's interface module contains methods for inter agent communication. The knowledge module contains meta-knowledge about all agents in the system.

3.2.3 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 agent receives notification of a new agent who wants to advertise its services; it must add this new agent's identifier and its capabilities to the list of available system agents.

To fulfil a task the task agent must "talk to" the facilitator agent, asking which agents can fulfil 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.

3.2.4 Task Agent (or Management Agent)

A task agent is responsible for the activation and synchronisation of the various EMADS agents required to generate a response to a given user request. Individual categories of task agent dedicated to different, but specific, data mining operations have been identified; the various categories are considered in detail in later chapters. A task agent performs its task by first generating a work plan, and then monitoring the progress of the plan. Task agents are designed to receive data mining requests from user agents and seek the services of groups of agents to obtain and synthesise the final result to be returned to the user agents. The agent interface module is responsible for inter-agent communication; the process module contains methods for the control and coordination of the various tasks. A task agent may be required, when generating a response to a request, to: identify relevant data sources, request services from agents, generate queries, etc. The knowledge module contains meta-knowledge about data mining tasks, i.e., what steps are required for what type of task, input requirements for each of the data mining tasks, etc.

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

In the first case the task agent will ask the facilitator agent for DM agents which can fulfil the desired tasking. For instance, if the user wants all possible association rules meeting a given minimum support and confidence thresholds, across all available data sources, then the task agent will contact the appropriate DM agents with this request. The task agent accepts the result from each individual DM agent and stores it in its knowledge base. Once the DM task is completed, the task agent combines the results to provide a single result before passing it to the user agent. Note that in some cases there may be only a single result in which case there will be no requirement to combine results.

The second case may occur where the user feels (say) that the current operations are taking too long, or are no longer needed. In this case, the task agent must send "cancel" messages to all agents currently tasked and performing work.

3.2.5 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 methods, format of input data, etc. This knowledge is used by the process module in initiating and executing a particular mining algorithm for the problem at hand.

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

3.2.6 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 fulfil 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.

An individual data agent is not specific to any particular data mining task but rather is able to answer any query associated with its data. When a new data source is introduced into EMADS it must be “wrapped” (as described in the next chapter) so that a new data agent is created. During this process the presence of the new data agent will be announced to the facilitator agent so that the new agent can be recognised 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. This will be considered in further detail when extendibility is discussed in Chapter 4.

3.2.7 EMADS End User Categories

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

- **EMADS Developers:** Developers are EMADS participants, who have full access and may contribute DM algorithms and tasks.
- **EMADS Data Miners:** These are participants, with restricted access to EMADS, who may pose DM requests.
- **EMADS Data Contributors:** These are participants, again with restricted access, who are prepared to make data available to be used by EMADS data mining agents.

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

3.3 Defining Interaction Protocols

In the previous section the primary functions and interactions for generic MADM (EMADS) agents were identified. In this section the various protocols required to

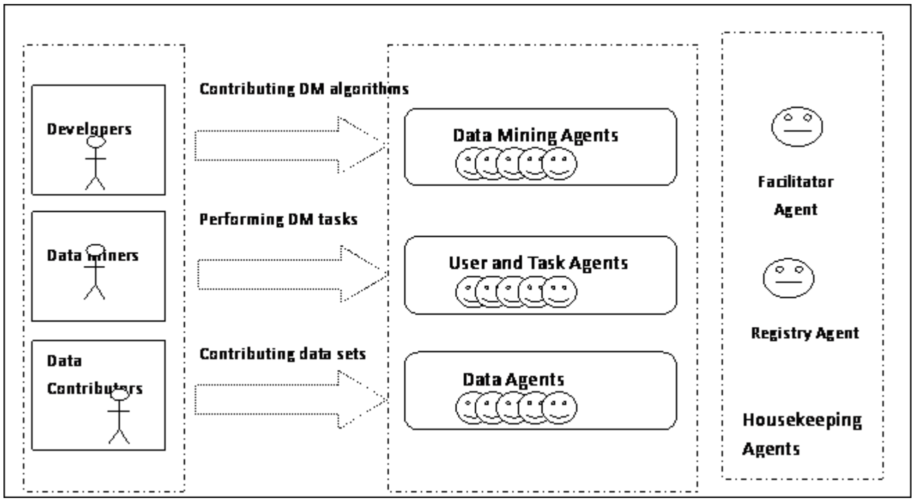


Figure 3.3: High Level View of EMADS Conceptual Framework

support the identified primary functions and high-level interactions are defined for each agent. Each protocol is considered in a separate subsection.

EMADS employs efficient distributed protocols, to avoid the consequent potential use of many messages, while maintains a generic architecture that can be extended with new DM tasks.

More generally, with respect to MAS, a protocol is some agreed message format for communication between agents that promotes some shared understanding. In MAS, protocols typically define how agents request information from one another, and how agents return responses to these requests. The agent initiating a message is called the sender, and the agent receiving it is the receiver. The response to a message usually requires the receiver to perform some action which will then generate the response to be returned to the sender. The message data is specified as content. The following syntax, expressed as a context free grammar in BackusNaur Form (BNF) [86], is used to define the protocols:

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

$$\begin{aligned} < Send-Message >::= & < Send-Message-Tag > \mid \\ & < Send-Message-Tag >< Send-Content > \mid \\ & < Send-Message-Tag >< Send-Action > \mid \end{aligned}$$

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

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

“)”|

$$< Receive - Message > \text{“}OR\text{”} < Receive - Action >$$

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

The first three lines of each protocol (the header) identifies the protocol's name (label) and the intended message “sender” and “receiver”. The sender or receiver can be one or more of the EMADS agent categories identified above. The next part of the protocol defines the nature of the message to be sent, this consists of at least a message tag, but may also define some message content and/or some action to be performed by the sender once the message has been sent. The message tag is one of a set of predefined tags (strings) that are understood by EMADS agents. The final part of the protocol defines the nature of the message to be returned to the sender (i.e. the reply) and any consequent action. This part of the protocol also consists of at least a message tag, but again may also define some message and any expected action(s) to be performed by the sender on receipt of the message. In some cases there may be a number of alternative messages that can be received, in which case this will be indicated by an "OR" (and parentheses to enhance readability). It is also possible for the sender to receive a sequence of replies of messages and actions. This format is used in the following 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 Task
5. Advertise Agent Capabilities
6. Perform Data Mining
7. Data Retrieval

3.3.1 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)

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.

3.3.2 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 agent's service and capabilities, to the facilitator agent. The sender will either receive an "accepted" tagged message, indicating that the registration was accepted and the agent is made public to other agents; or a "rejected" tagged message indicating that the registration failed.

3.3.3 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)

3.3.4 Start Data Mining Task Protocol

Task agents are central to the EMADS data mining process and interact with the user agents, the facilitator agent, and one or more DM agents. The task agent initially reacts to a request initiated by the user agent. The user agent will request that some DM operation be undertaken and will pass the appropriate attributes (for example threshold values) dependent on the nature of the desired DM task. In response to a request to be undertaken, the task agent will determine the nature of the request according to the variables sent. Once the task agent has determined the type of the request (for example an ARM request), it informs the user agent it has all the information it needs with an "accept" message and initiates the required processing.

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

The task agent will first use the Finding Other Agents protocol described in Subsection 3.3.1 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 Task protocol is given below.

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

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

3.3.5 Advertise Agent Capabilities Protocol

The facilitator agent primarily interacts with task agents, DM agents, and data agents; but has the ability to respond to any properly formatted request for agents that can fulfil 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 fulfil any given task.

To perform the first function, the facilitator must be able to communicate with any new agents entering EMADS and receive and process the new agent's information. The advertising process begins with the new agent sending an "advAgent" message that contains the new agent's full name and task list. The Advertise Agent Capabilities protocol is as follows:

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

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

3.3.6 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) to be used. Once this is received, the DM agent starts the mining algorithm associated with it by applying it to the data indicated by the specified data agent reference(s). When a DM agent's algorithm completes, the DM agent sends a "miningCompleted" reply message, with the generated results as the content, to the sending task agent. The Perform Data Mining protocol is as follows:

Protocol Name: Performing Data Mining
Sender: Task Agent
Receiver: DM Agent
send: beginMining
content: DM task type, DM and Data Agent lists

do:wait
receive: miningCompleted
content: result
do: if applicable process results and return result

The protocol includes an option to combine results where several DM agents or data agents may have been used.

3.3.7 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

3.4 Data Mining with Agents

In this section a generic overview of data mining with EMADS agents is presented. The overview is described by considering a general DM request and tracing the generation of the response through the various communication paths in the broad context of MADM. The general form of the EMADS DM process has already been partially described above in terms of the protocols used; the objective of this section is to bring the different elements of the process together in the form of a summary by considering a generic example.

In EMADS DM task, planning is realised by negotiation between EMADS agents through the message passing mechanism (described further in Section 3.6 below). The DM process begins with the user agent receiving notification from the end user describing a data mining request. The user agent picks up the user request and then starts a task agent. The task agent then asks the facilitator agent for the identifiers of all "useful" (DM and data) agents specified in the context of the request. An agent is deemed "useful" if it can potentially contribute to the resolution of the request. The facilitator agent receives the request and compiles a list of all appropriate (interested) agents. The facilitator agent then returns the list of "useful" agents to the task agent. Once the task agent receives the list, it can commence the desired DM process. The nature of this DM process will depend on the nature of the request.

A number of different categories of task agent have been identified in the context of EMADS; these are discussed in further detail in later chapters (5,6, and 7). However, in general terms, the task agent sends a request to each identified DM agent in the list (there may only be one) to begin DM together with appropriate references to the identified data agents. Each DM agent accepts the request and begins mining their applicable data source. Once completed, the DM agents send the results back to the task agent. When the task agent has all the results, it processes the results (for example it may combine them), and notifies the user agent (in some cases there may of course only be one set off results). The user agent then displays the combined results to the user.

3.5 The Agent Development Toolkit

In this section a discussion is presented concerning the selected agent development toolkit, JADE, in which EMADS was implemented (JADE was introduced in Chapter 2, Subsection 2.2.4).

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

The key difference between most development toolkits lie in the implementation and architecture of the provided communication and agent functionality. When selecting a toolkit to build some desired MAS developers should make their decision based on the MAS goals and services that are desired. Any potential toolkit should also be evaluated for potential problems related to the toolkit's strengths and weaknesses prior to any decision being made.

JADE was chosen for the proposed EMADS framework development. JADE was selected for a variety of reasons as follows:

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

A number of alternative platforms were detailed in Chapter 2, Subsection 2.2.4. Some of the reasons that these other platforms were avoided included:

- That they were still in an experimental state, abandoned, or confidentially distributed,
- Very little documentation was associated with them,
- They covered only one aspect, or a limited number of aspects, of MASs; such as single agent platforms, mobile agent platforms, interaction infrastructures toolkits,
- They were found to be too short on some construction stages, for example purely methodological models.

Further detail concerning JADE is provided in the following subsections. Subsection 3.5.1 presents JADE architecture and Subsection 3.5.2 presents agent interaction in JADE.

3.5.1 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 Chapter 2 JADE is a FIPA-compliant middle-ware that enables development of peer to peer applications based on the agent paradigm. JADE defines an agent platform that comprises a set of containers, which may be distributed across a network (as desired in the case of EMADS).

The goal of JADE is to simplify the development of MAS while at the same time ensuring FIPA compliance through a comprehensive set of system services and agents. While appearing as a single entity to the outside observer, a JADE agent platform can be distributed over several hosts each running an instance of the JADE runtime environment. A single instance of a JADE environment is called a container which can "contain" several agents as shown in Figure 3.4. The set of one or more "active" containers is collectively referred to as a platform (as indicated by the dashed perimeter line in Figure 3.4). For a platform to be active it must comprise at least one active container; further containers may be added (as they become active) through a registration process with the initial (main) container. A JADE platform includes a main container (the middle container in Figure 3.4), in which is held a number of mandatory agent services. These are the Agent Management System (AMS) and Directory Facilitator (DF) agents. The AMS agent is used to control the life-cycles of other agents on the platform, while the DF agent provides a lookup service by means of which agents can find other agents. When an agent is created, upon entry into the system, it announces itself to the DF agent after which it can be recognised and found by other agents. Further "house-keeping" agents are the Remote Monitoring Agent (RMA) and The Sniffer Agent (SA). The first keeps track of all registered agents, while the second monitors all message communications between agents.

Within JADE, agents are identified by name and communicate using the FIPA Agent Communication Language (ACL). More specifically, agents communicate

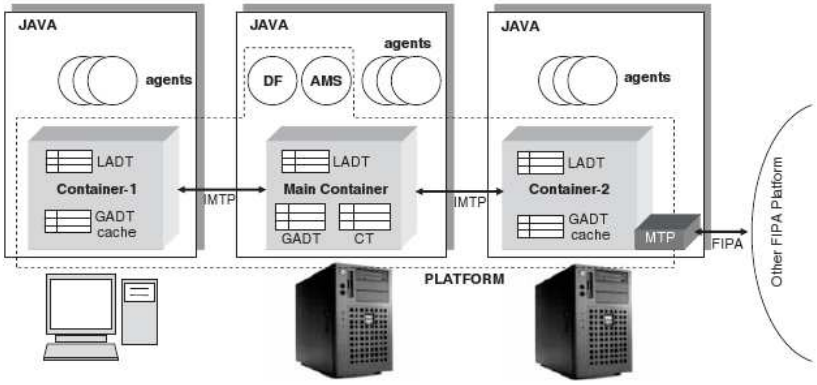


Figure 3.4: JADE Architecture (Bellifemine et al., 2007) [16]

by formulating and sending individual messages to each other and can have “conversations” using interaction protocols that range from query request protocols to negotiation protocols. JADE supports three types of message communication as follows:

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.

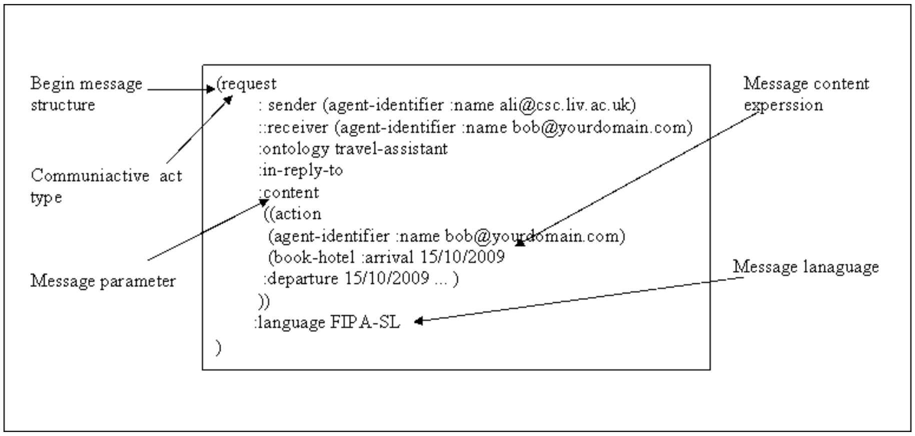
3. **Inter-platform:** Message communication between agents in different platforms uses the IIOP (Internet Inter-ORB Protocol).

. The latter is facilitated by a special Agent Communication Channel (ACC) agent also located in the JADE platform main containers.

The JADE communication architecture is intended to offer (agent transparent) flexible and efficient messaging by choosing, on an “as needed” basis, the most appropriate of the FIPA-compliant Message Transport Protocols (MTP) that are activated at platform run time. Basically, each container has a table containing details of its local agents, called the Local-Agent Descriptor Table (LADT), and also maintains a Global-Agent Descriptor Table (GADT), mapping every agent into the RMI object reference of its container. The main container has an (additional) Container Table (CT) which is the registry of the object-references and transport addresses of all container nodes. Each agent is equipped with an incoming message box and message polling can be blocking or non-blocking.

JADE uses LADTs and GADTs for its address caching technique so as to avoid querying continuously the main-container for address information and thus avoiding a potential system “bottleneck”. However, although the main-container is not a “bottleneck”, it is still a single potential point of failure within the platform. This is a recognised issue within the JADE user community. Research has been reported seeks to address this issue, for example Bellifemine et al. (2007) who described a mechanism whereby a JADE main-container replication service was used to deploy a fault-tolerant JADE platform. However, most users simply “live with the problem”; a strategy that has also been adopted with respect to the EMADS framework.

FIPA specifies a set of standard interaction protocols, such as FIPA-requests and FIPA queries. These protocols can be used to build agent “conversations” (sequences of agent interactions). In JADE, agent tasks or agent intentions are implemented through the use of behaviours (discussed further in Subsection 3.5.2 below).



```
(request
      : sender (agent-identifier :name ali@csc.liv.ac.uk)
      ::receiver (agent-identifier :name bob@yourdomain.com)
      :ontology travel-assistant
      :in-reply-to
      :content
       ((action
        (agent-identifier :name bob@yourdomain.com)
        (book-hotel :arrival 15/10/2009
       :departure 15/10/2009 ... )
      ))
     :language FIPA-SL
)
```

Figure 3.5: ACL message Example

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

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

3.5.2 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 initialised, suspended and spawned at any given time. Behaviours are implemented in terms of fields and methods contained in one or more sub-classes of a parent Behaviour class provided with JADE. Any given JADE agent keeps a task list that contains the active behaviours. JADE uses one thread per agent, instead of one thread per behaviour, to limit the number of threads running on the agent platform. A behaviour can release the execution control with the use of blocking mechanisms, or it can permanently remove itself from the queue during 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).

3.6 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 3.2. Some background details of JADE were presented in Subsection 3.5.1. Broadly speaking JADE defines an agent platform that comprises a set of containers, which may (as in the case of EMADS) be distributed across a network. This section commences with an overview of the EMADS JADE implementation. More specific details of the JADE implementation focusing on: Agent "Behaviours", agent in-

Message Type:	**Meaning:**
Accept Proposal	The action of accepting a previously submitted proposal to perform an action.
Agree	The action of agreeing to perform some action, possibly in the future.
Cancel	The action of one agent informing another agent that the first agent no longer has the intention that the second agent performs some action.
Call for Proposal	The action of calling for proposals to perform a given action.
Confirm	The sender informs the receiver that a given proposition is true, where the receiver is known to be uncertain about the proposition.
Dis-confirm	The sender informs the receiver that a given proposition is false, where the receiver is known to believe, or believe it likely that, the proposition is true.
Failure	The action of telling another agent that an action was attempted but the attempt failed.
Inform	The sender informs the receiver that a given proposition is true.
Inform If	A macro action for the agent of the action to inform the recipient whether or not a proposition is true.
Inform Ref	A macro action for sender to inform the receiver the object which corresponds to a descriptor, for example, a name.
Not Understood	The sender of the act informs the receiver that it did not understand the action the receiver just performed.
Request	The sender requests the receiver to perform some action.
Request When	The sender wants the receiver to perform some action when some given proposition becomes true.
Propose	The action of submitting a proposal to perform a certain action, given certain preconditions.
Refuse	The action of refusing to perform a given action, and explaining the reason for the refusal.

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

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

Table 3.2: FIPA pre-defined message parameters

teraction, mechanisms for cooperation, user request handling, and extendibility; are all presented in the following subsections.

Within JADE, agents can have "conversations" using interaction protocols similar to the schema described in Section 3.3.

In the case of the EMADS implementation, agents may be created and contributed by any EMADS user/contributor. One of the containers, the main container, holds the house-keeping agents and the Agent Management System (AMS) (see Subsection 3.5.1). Both the main container and the remaining containers can hold various DM agents. The EMADS main container should be located on the EMADS host organisation site, while the other containers may be held at any other sites worldwide.

Other than the house-keeping agents, held in the main container, EMADS currently supports the four categories of agents identified in Section 3.2: user agents, task agents, DM agents and data agents. Registration and facilitator functionalities are provided by JADE house-keeping agents (AMS and DF agents respectively). It should be noted that EMADS containers may contain both DM agents and data agents simultaneously as well as user agents. DM agents and data agents are persistent, i.e. they continue to exist indefinitely and are not created for a specific DM exercise as in the case of task agents. Communication between agents is facilitated by the EMADS network.

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

The principal advantage of this JADE architecture is that it does not overload a single host machine, but distributes the processing load among multiple machines. The results obtained can be correlated with one another in order to achieve computationally efficient analysis at a distributed global level.

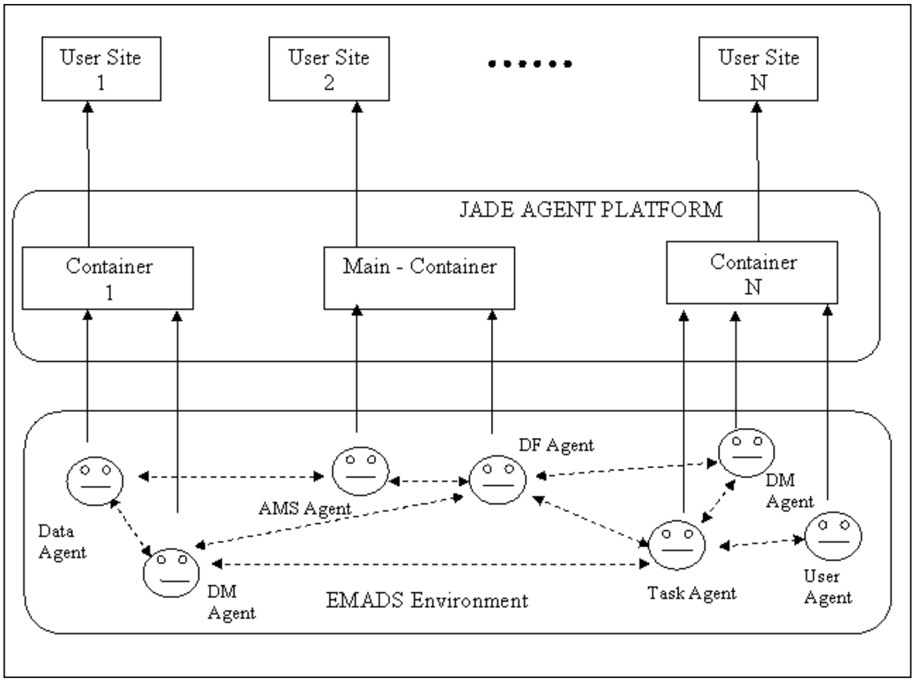


Figure 3.6: EMADS Architecture as Implemented in JADE

3.6.1 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 [17] and the FIPA ACL Message Structure Specification [37].

The procedure to map the EMADS agent interaction protocols, as defined in Section 3.3, 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 either action methods or to simple methods of behaviours. Predefined JADE behaviours that were found to be useful to EMADS were:

- **OneShotBehaviour:** Implements a task that runs once and terminates immediately.

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

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

- **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 provided 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 subtasks according to a Finite State Machine (FSM) model.

Composite behaviour can be nested and therefore there can be, for instance, a subtask of a SequentialBehaviour that is in turn a FSMBehaviour and so on. In particular, all complex responsibilities that can be modelled 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.

3.6.2 Agent Interactions

A user agent, as shown in Figure 3.6, 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; 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 in the context of agent interaction 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 register their capability specifications.

3. 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.

3.6.3 Mechanisms of Cooperation

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

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.

3. **Processing:** Where the task agent coordinates the efforts of the appropriate service providers (data agents and DM agents) to satisfy the request.

Stage 1 (service registration) is not necessarily immediately followed by stage 2 and 3; it is possible that a provider's services may never be used. Note also that the facilitator (the DF and AMS agents) maintains a knowledge base that records the capabilities of the various EMADS agents, and uses this knowledge to assist requesters and providers of services in making contact. When a service provider (i.e. data agent or DM agent) is created, it makes a connection to the facilitator. Upon connection, the new agent informs its parent facilitator of the services it can provide. When the agent is needed, the facilitator sends its address to the requester agent. An important element of the desired EMADS agent cooperation model is the function of the task agent; this is therefore described in more detail in the following subsection.

3.6.4 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. **Optimisation:** Optimisation of the complete task, including parallelisation where appropriate.

3. **Interpretation:** Interpretation of the optimised task.

Thus determination (step 1) involves the selection of one or more agents to handle each sub-task given a particular request. In doing this, the task agent uses the facilitator's knowledge of the capabilities of the available EMADS agents (and possibly of other facilitators, in a multi-facilitator system). In processing a request, an agent can also make use of a variety of capabilities provided by other agents. For example, an agent can request data from data agents that maintain data. The optimisation step results in a request whose interpretation will require as few communication exchanges as possible, between the task agent and the satisfying agents (typically DM agents and data agents), and can exploit the parallel processing capabilities of the satisfying agents. Thus, in summary, the interpretation of a task by a task agent involves: (i) the coordination of requests directed at the satisfying agents, and (ii) assembling the responses into a coherent whole, for return to the user agent.

3.7 Discussion

EMADS is a distributed, scalable, portable, extendible and adaptive agent-based system that supports the launching of agents to perform DM activates. EMADS is a realisation of the MADM ideas espoused in this thesis.

EMADS uses a facilitator approach for agent coordination. The role of the facilitator is to help agents to locate each other and to communicate for their mutual benefit based on a set of indices such as name, location, function, or interest. The usefulness of this service allows the construction of a system that is more flexible and adaptable than distributed frameworks. Individual agents can be dynamically added to the community, extending the functionality that the agent community can provide as a whole.

To better tackle the complexity of the scalability and efficiency issue, EMADS addresses it at two levels, the system architecture level and the components level. At the system architecture level, the focus is on the components of the system and the overall architecture. Assuming that the data mining system comprises several data sites, each with its own resources, databases, and agents, EMADS supports a number of protocols that allow the data sites to collaborate efficiently without hindering their progress. Employing efficient distributed protocols, however, addresses the scalability problem only partially. The scalability of the system depends greatly on the efficiency of its components (agents). The analysis of the dependencies among the agents, and their efficiency with respect to DM tasks, constitutes the other half of the scalability problem. EMADS addresses this issue in some principled basis, specific to the considered DM task. For instance, in the meta ARM problem (used to evaluate this issue and considered in detail in Chapter 5), EMADS uses a pruning technique (discarding certain infrequent itemsets early in the process) that is more efficient and scalable and at the same time achieves comparable or better predictive performance results than fully grown (un-pruned) techniques.

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

To ensure adaptivity and extendibility, EMADS was designed using object-oriented methods and was implemented independently of any particular machine learning program or any specific DM technique. Adaptivity and extendibility in EMADS were achieved using a system of wrappers. EMADS's extendibility capabilities provided the means to easily accept and incorporate new data sources and new DM techniques (considered in detail in the following chapter).

As EMADS is implemented in Java, portability is inherent within EMADS.

3.8 Summary

This chapter discussed MADM, and consequently the EMADS framework, requirements, architecture, design and implementation. EMADS was envisioned as a collection of data sources scattered over a network, accessed by a group of 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, DM 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 chapter a detailed description of the design and implementation of the extendibility feature of EMADS is provided.

Chapter 4

MADM Extendibility

As described in Chapter 1, one of the principal objectives of the research described in this thesis is the provision of an extendible MADM framework, and consequently the EMADS framework, that can easily accept new data sources and new Data Mining (DM) techniques. This was also identified as a design requirement for any generic MADM as identified in Chapter 3 (Subsection 3.1.1). The issue of extendibility may be argued as follows:

1. The aim of any MADM framework is to facilitate DM using the MAS paradigm.

2. The desired DM capability is not provided by the framework itself, this is provided by the specific agents contributed by users of the framework.

3. However, the framework should provided mechanisms to support this contribution of agents.

4. The challenge is that the extendibility mechanism, built into any MADM framework, must anticipate the nature of any likely (and unlikely) DM related agents that any end user may wish to contribute.

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 MADM extendibility is defined as the ease with which new data sources and new DM techniques can be introduced to the framework. Extendibility also implies flexibility, and MADM should support the inclusion of existing as well as future DM technology (otherwise the MADM will rapidly be rendered inadequate and therefore obsolete). Essentially adding a new data source or DM technique to a MADM should equate to the introduction of new agents.

This chapter provides a detailed discussion of extendibility in the context of MADM and how this is achieved in EMADS. The chapter commences by presenting an overview of mechanisms whereby extendibility can be achieved, in the context of MADM, in Section 4.1. The extendibility mechanisms adopted for EMADS are described in Section 4.2. The principal means for achieving this is through the use of a system of wrappers. The wrapper concept is described in Section 4.3. Section 4.4 discusses how, conceptually, the functionality of EMADS can be extended by adding new task agents, DM algorithms and data sources using the wrappers concept. The chapter also gives an overview, in Subsection 4.4.3, of the LUCS-KDD-DN software tool which was incorporated into the Data Wrapper agent for possible data normalisation requirement as to avoid the issue of data heterogeneity. The chapter ends with a summary in Section 4.5.

4.1 Extendibility

There are several reasons to ask for extendibility. For example, a piece of existing software may do nearly the right thing, but needs some addition to meet a new requirement. Or it may do the right thing, but needs to be wrapped up to provide easier handling and integration into other software components. Or, as in the EMADS case, the need to add new functionalities which make use of the services already provided.

There are several methods in the literature available to provide extendibility, including:

- **Software Wrappers.** Software wrapping refers to a re-engineering technique that surrounds a software component or tool with a new software layer to hide the internal code and the logic of the component or tool and to supply modern interfaces. One example of software wrapping is the reuse of legacy software in modern applications, where the unwanted complexity of the old software is hidden to the applications. Software wrapping removes the mismatch between the interfaces exported by a legacy software "artifact" and the interfaces required by the current software program [121].

The advantage of wrapping is that the added component becomes part of the system without changing the content of the component. Using wrappers can greatly reduce the amount of work required to build new functionalities.

- **Libraries.** Libraries are an alternative mechanism for providing a reusable abstraction of code. However, unlike wrappers, a library leaves control in the hands of the programmer. Since they do not provide any flow control by themselves, complex interactions must be built from scratch [55].

- **Plug-ins.** Plug-ins are a widely-used approach to provide application extendibility [80]. For example, makers of popular software such as Adobe and Mozilla provide plug-in architectures for their applications [1]. Plug-ins are required to interact with the host application through an API which provides access to a subset of the hosts functionality. Unfortunately, this usually means plug-ins can only provide restricted functionality to enhance the host application.

- **Dynamic scripting.** Dynamic scripting to influence functionality is another way to extend applications. For example, the Emacs framework [110] allows users to define new functionality on the fly or even redefine existing functionality. This ability to redefine existing behaviour makes the Emacs extendibility approach much like that of plug-ins. Furthermore the core implementation retains control over program flow. This approach is beneficial because it can be interpreted at run-time instead of compile-time. However, this carries the performance penalties associated with interpreted code.

4.2 Extendibility in EMADS

To ensure generality and extendibility, EMADS is designed using object-oriented methods and is implemented independently of any particular machine learning program or DM task. EMADS's extendible capabilities provide the means to easily incorporate any new DM algorithm/task or data sources.

The independence of EMADS from any particular DM method, in conjunction with the object oriented design, ensure the framework's capability to incorporate

[1] Johnny Stenback. Mozilla plug-ins. `http://www.mozilla.org/projects/plugins/`, and Adobe plug-in component architecture (PICA). `http://www.adobe.com/devnet`

and use new DM algorithms/tasks or data sources. DM techniques and data sources can be embedded within appropriate wrappers to be introduced into the EMADS system and subsequently become an EMADS agent. Introducing a new technique requires the use of the appropriate wrapper to encapsulate the algorithm/technique within an object that adheres to the minimal wrapper interface. In fact, most of the existing implemented DM algorithms have similar interfaces already.

EMADS wrappers define the abstract methods of the parent classes and are responsible for invoking the executables of these algorithms. This extendibility characteristic makes EMADS an extendible DM facility. Evaluation of EMADS has shown that, given an appropriate wrapper, existing data mining software can be very easily packaged to become an EMADS agent. This feature of EMADS was investigated using a number of different scenarios. These scenarios are discussed in later chapters.

4.3 Wrappers

As EMADS agents are not intended as replacements for DM algorithms or techniques, the agents must be able to interact with the algorithms or techniques. Generally speaking there are at least three possible approaches to incorporating DM software. The software could be rewritten to an appropriate form, but this is a costly approach. Alternatively a separate piece of software could be employed to act as an interpreter between the agent communication language and the native protocol of the DM software. Or thirdly, the wrapper technique could be used to augment the legacy program with code that enables it to communicate using the inter-agent language. As pointed out earlier, EMADS uses the latter approach.

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 (introduced above) 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 [41].

EMADS wrappers are used to "wrap" up DM 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 DM 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. EMADS supports three broad categories of wrapper (as shown in Figure 4.1):

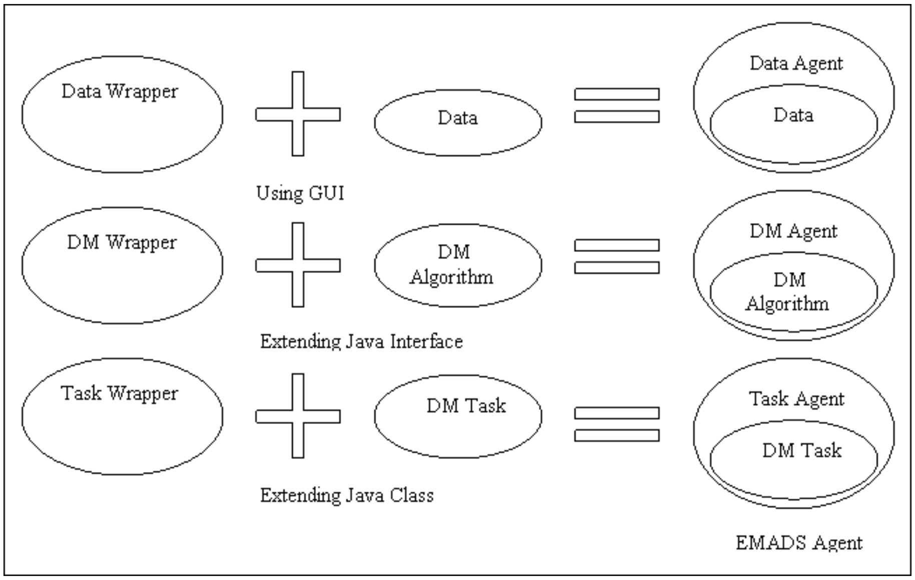


Figure 4.1: EMADS Wrappers

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 task agents. Figure 4.1 illustrates the broad "agentification" process for the three different kinds of agent. The figure should be interpreted as follows:

- Data wrapper usage is facilitated by a GUI and normally requires no programming.

- DM wrapper usage requires extending and implementing a Java interface.
- Task wrapper usage requires extending and implementing a Java class.

Each wrapper is described in further detail in the following three subsections.

4.3.1 Data Wrappers

As pointed out early, the current version of EMADS assumes well-defined schemata and datasets as to avoid issues of data heterogeneity. In the context of EMADS, 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. KDD has traditionally been concerned with tabular data sets, reflecting the strong association between DM and relational databases. DM algorithms have generally been developed with respect only to tabular data. A tabular format offers the advantage of easy integration with other tabular data sets. 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 (e.g. data type, number of classes). 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. In the case of non-standard data, then the user can use the wrapper to have the function of reformatting the data to meet the requirements of EMADS as described in Subsection 4.4.3.

4.3.2 DM Wrappers

DM wrappers are used to "wrap" up DM software algorithms and to create DM agents. Generally the algorithms to be wrapped will be DM techniques of various kinds (classifiers, clusters, association rule miners, etc.) Unlike data wrappers, DM wrappers are not supported by a GUI facility to aid their usage; instead

EMADS developers are expected to encode the wrappers themselves. However, the design of the wrapper that provides most of the required functionalities makes the encoding to be a straight forward process.

4.3.3 Task Wrappers

It is intended that the framework will incorporate a substantial number of different DM technique 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 developers are expected to encode most of the operations of the task wrappers themselves. Again, the design of the task wrapper is such that encoding of operations is not a significant process.

The realisation of these wrappers is described in further detail in the following section.

4.4 Realisation

As noted in the introduction to this chapter, one of the most important features of MADM, and consequently EMADS, and one of the central themes of this thesis, is the simple extendibility of MADM, thus the ability for new agents to be added to such systems in an effective and efficient manner. The rationale is that the operation of a MADM should be independent of any particular DM method.

The inclusion of new data sources 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 section considers the mechanisms whereby the desired extendibility feature can be realised by considering each of the three identified categories of agents (Task, Data and DM) that may be contributed by users. It should be noted, before considering each category of agent in detail, that extendibility is achieved using the wrapper concept introduced earlier. Recall that, in isolation, wrappers can be viewed as agents in their own right which are used to build data, task 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 agents. This section also describes how these mechanisms were incorporated into the EMADS.

TaskAgent abstract class to be extended by (new tasks) subclasses

```
public abstract class TaskAgent extends Agent {

// ———————————— FIELDS ————————————
/** the list of relevant data agents */
protected AID[ ] dataAgents;
/** the list of interested mining agents */
protected AID[ ] miningAgents;
/** creator Agent*/
protected String userAgent;

// ———————————— METHODS ————————————
// abstract methods
protected abstract void setup();
  .
  .

// implemented methods
/** look up data agents in the yellow pages */
protected boolean getDataAgents
               (String agentType, String dataType)
  .
  .
/** look up mining agents in the yellow pages */
protected boolean getMiningAgents
               (String agentType, String taskType)
  .
  .
}//end of class
```

Table 4.1: Prototype of the Task Agent Abstract Class

4.4.1 Task Agents

Task agents, that perform DM tasks, are introduced into EMADS using a predefined abstract task agent class (Task Wrapper). This wrapper is facilitated by the Object Orientation features provided by the Java programming language in which EMADS agents are expected to be implemented. A prototype of the task agent abstract class is shown in Table 4.1. Individual task agents are created by implementing a specific sub-class of this abstract class. The abstract task class defines a number of attributes (fields, methods and abstract method headers) that are inherited by the user created sub-class used to define a task agent specific to a DM task.

In effect the abstract class defines simple and minimal variables and methods that all task agent sub-classes are expected to comply with and use. For instance, as shown in in Table 4.1, the method *"getDataAgents(agentType, dataType)"* can be used to get a list of data agents based on the passed parameters *"agentType, dataType"*. Those parameters are used to specify the required data agents for the task under consideration (e.g. classification data). The same can be done to use the second method *"getMiningAgents(agentType, taskType)"* to get a list of matching mining agents. The method *"setup()"* is declared as an abstract and left to the user to implement according to their introduced task. As long as the sub-class task agent conforms to the abstract class attributes, it can be introduced and used immediately as part of the EMADS system.

4.4.2 DM Agents

The addition of DM agents to the EMADS system is facilitated by a second purpose built wrapper. The DM wrapper is said to "wrap" a DM algorithm. The DM wrapper 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 consists of a class and an abstract interface. The Java class defines the wrapper and provides the required code to be used to instantiate the DM algorithm class. The DM wrapping process can be achieved in two steps as follows:

1. The instantiation code of the DM algorithm class in the wrapper class must be modified to use the DM algorithm class name. A prototype of the wrapper class is shown in Table 4.2.

2. The algorithm class must extend the abstract interface and implement its abstract methods.

The wrapper abstract interface (the mining interface) that a DM algorithm class must implement is shown in Table 4.3. First, the interface in which the algorithm is encapsulated must have a method that can be called to start the

DM wrapper class to be updated by users to wrap a DM algorithm class

```
public class DMAgent extends Agent {
// ―――――――――――――――― FIELDS ――――――――――――――――――――
/** the list of relevant data agent(s) */
private AID[ ] dataAgents;
  .
// ―――――――――――――――― METHODS ――――――――――――――――――――
//** put agent initialisation here */
protected void setup() {
//** get the start-up arguments */
Object[] args = getArguments();
//** register the service in the yellow pages */
DFAgentDescription dfd = new DFAgentDescription();
.
System.out.println
   (``Mining-agent:"+getAID().getLocalName()+``␣is␣ready.");

}//end of setup
//**  agent clean-up operations  */
protected void takeDown() {
//**  de-register from the yellow pages */
.
//**  printout a dismissal message */
System.out.println
(``DM-agent "+getAID().getName()+``␣terminating.");
}//end of cleanup
/** look up Data agents in the yellow pages **/
protected void getDataAgents() {
//** specify agent info. */
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
.
}//end of method
/** begin mining method */
protected void startMining() {
//** user update is needed in here */
//set default values; e.g. support and confidence thresholds
//** create instance of the mining algorithm class */
 <Algorithm_Class_Name> instanceName =
                          new <Algorithm_Class_Name>();
//** pass arguments to the class  instance */
instanceName.setArguments(args);
//** pass the data file or the data location */
instanceName.setData(getData(dataAgent));
//** start the mining process */
instanceName.startMining();
}//end of method
}//end of class
```

Table 4.2: Prototype of the DM Agent Wrapper Class

algorithm. This is implemented with the abstract *"startMining()"* 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. In effect the DM agent can be considered to hold a data mining algorithm class instance.

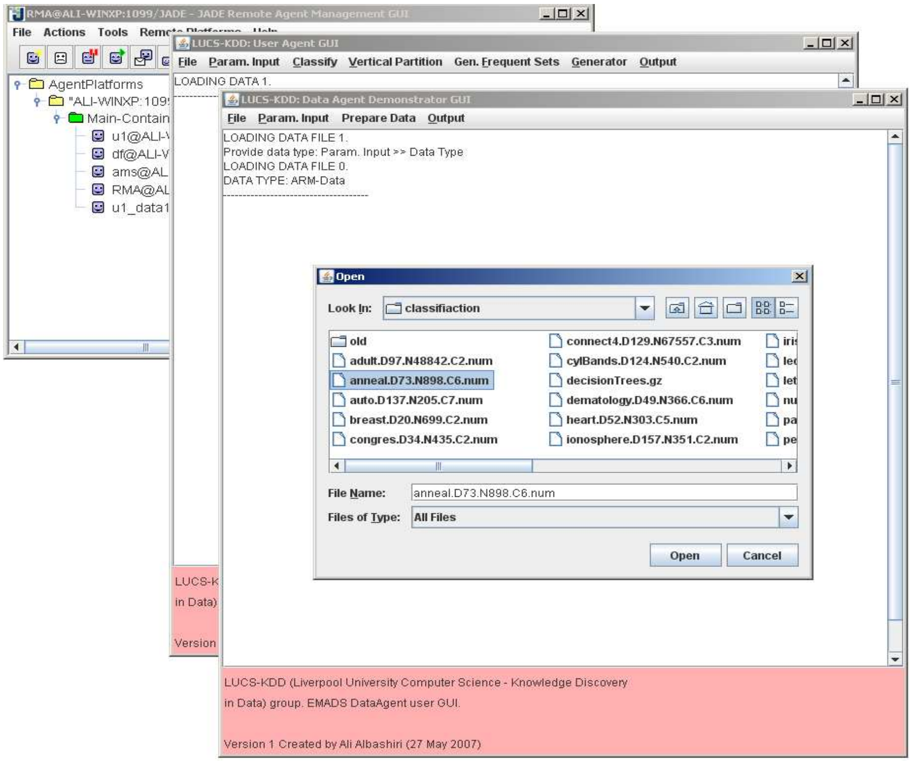
Mining abstract interface to be implemented by a DM algorithm class for the DM wrapper.

```
public interface MiningInt {

// ———————————— ABSTRACT METHODS ————————————
// abstract methods
/** can be used to pass a list of values(e.g. support
 and confidence) */
protected void setArguments(String args[]);
/** start the mining process */
boolean startMining();
/** stop the mining process */
boolean stopMining();
/** set the resource data  */
boolean setResource(short[ ][ ] data);
/** set the resource data  */
boolean setResource(String filePath);
/** return the result as a set of rules */
RuleNode getRules();
/** get the result as a double value */
double getResult();
}//end of interface
```

Table 4.3: DM Abstract Interface

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 where the dataset it will be operating on is located. This interface is one of the components required when the system is extended and new DM algorithms are added. Because of this, special consideration has been directed at making it as "extendible" as possible. The DM algorithm must be able to return the result of the DM for which it is responsible. In order to ensure this, the abstract methods *"getResult()"* and *"getRules()"* are included to obtain the desired results from the DM algorithm (as a single value (e.g. classifier accuracy), or as a list of rules). Because each algorithm may process its data differently or not at all, the *"getResult()"* method enforces the result retrieval.


RMA@ALI-WINXP:1099/JADE - JADE Remote Agent Management GUI
File Actions Tools Remote Platforms Help
AgentPlatforms
"ALI-WINXP:109
Main-Contain
u1@ALI-
df@ALI-V
ams@AL
RMA@AL
u1_data1
LUCS-KDD: User Agent GUI
File Param. Input Classify Vertical Partition Gen. Frequent Sets Generator Output
LUCS-KDD: Data Agent Demonstrator GUI
File Param. Input Prepare Data Output
Open
Look In: classifiaction
old
adult.D97.N48842.C2.num
anneal.D73.N898.C6.num
auto.D137.N205.C7.num
breast.D20.N699.C2.num
congres.D34.N435.C2.num
connect4.D129.N67557.C3.num
cylBands.D124.N540.C2.num
decisionTrees.gz
dematology.D49.N366.C6.num
heart.D52.N303.C5.num
ionosphere.D157.N351.C2.num
iris
lec
let
nu
pa
pe
Open
Cancel

```
LOADING DATA 1.
```

```
LOADING DATA FILE 1.
Provide data type: Param. Input >> Data Type
LOADING DATA FILE 0.
DATA TYPE: ARM-Data
------------------------------------
```

File Name: anneal.D73.N898.C6.num

Files of Type: All Files

LUCS-KDD (Liverpool University Computer Science - Knowledge Discovery in Data) group. EMADS DataAgent user GUI.

Version 1 Created by Ali Albashiri (27 May 2007)

Figure 4.2: Data Agent GUI

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 what specific implementation is used. It also provides a great deal of flexibility. An example on using the DM wrapper is presented in Chapter 7, Section 7.4.

4.4.3 Data Agents

Using the predefined GUI data wrapper described in Subsection 4.3.1, incorporating new data agents into EMADS is made simple. Its usage is facilitated by the use of a Java GUI only, and no programming is required for data that is in an appropriate format. 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 4.2) to refer to the data source location (file path) and provide meta information describing the data (e.g. data type, number of classes etc.). Then the wrapper creates a new data agent that will represent the data source as it is introduced to the system.

4.4.3.1 Data Discretisation/Normalisation

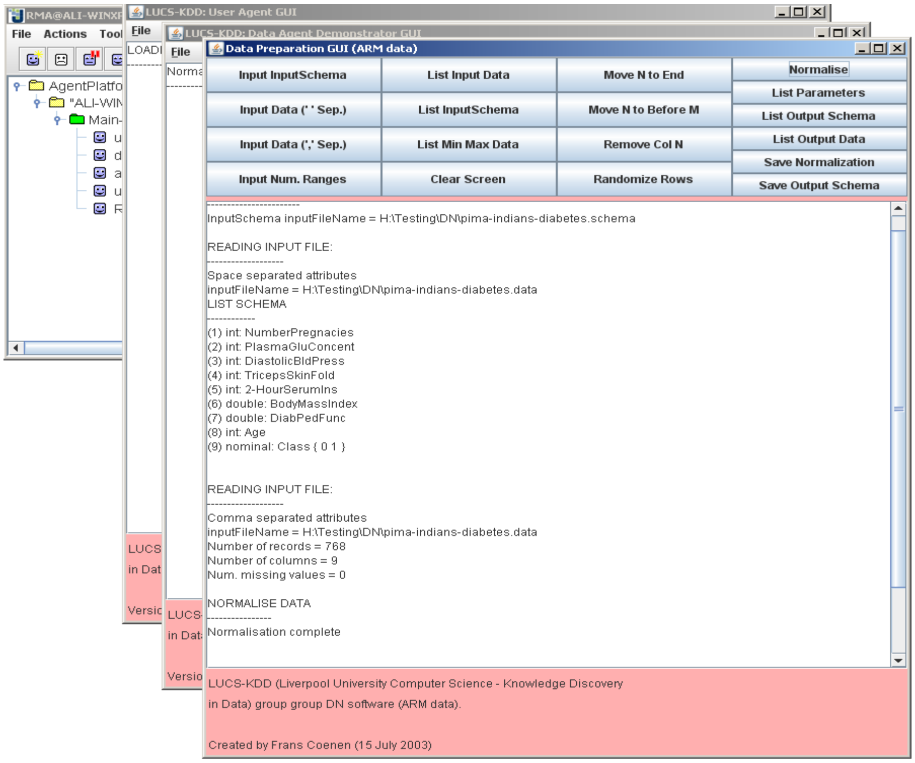
If the data requires normalisation before introducing it to EMADS, the user can use the wrapper GUI to launch the LUCS-KDD-DN (Liverpool University Computer Science - Knowledge Discovery in Data - Discretisation/Normalisation) tool (Figure 4.3) that has been integrated into the data wrapper. The LUCS-KDD-DN software tool, or DN tool for short, had been developed as a standalone Java application to convert data files available in the UCI data repository [18] into a binary format suitable for use with Association Rule Mining (ARM) applications. The DN tool can equally well be used to convert data files obtained from other sources.

Discretisation and normalisation can be defined as follows: Discretisation 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. Normalisation is the process of converting values associated with nominal data items so that they correspond to unique integer labels.

The DN tool is integrated within the data agent for data discretisation/ normalisation purposes to assure EMADS data compatibility. It is used as to avoid issues of data heterogeneity all data was assumed to correspond to a common global schema.

4.5 Summary

This chapter discussed the EMADS extendibility requirements, architecture, design and implementation. The expectation is that EMADS will “grow” as in-



```
----------------------
InputSchema inputFileName = H:\Testing\DN\pima-indians-diabetes.schema

READING INPUT FILE:
------------------
Space separated attributes
inputFileName = H:\Testing\DN\pima-indians-diabetes.data
LIST SCHEMA
-----------
(1) int: NumberPregnacies
(2) int: PlasmaGluConcent
(3) int: DiastolicBldPress
(4) int: TricepsSkinFold
(5) int: 2-HourSerumIns
(6) double: BodyMassIndex
(7) double: DiabPedFunc
(8) int: Age
(9) nominal: Class { 0 1 }


READING INPUT FILE:
------------------
Comma separated attributes
inputFileName = H:\Testing\DN\pima-indians-diabetes.data
Number of records = 768
Number of columns = 9
Num. missing values = 0

NORMALISE DATA
---------------
Normalisation complete
```

LUCS-KDD (Liverpool University Computer Science - Knowledge Discovery
in Data) group group DN software (ARM data).

Created by Frans Coenen (15 July 2003)

Figure 4.3: Data Normalisation GUI

dividual users contribute data and DM algorithms. The incorporation of data and DM techniques 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 DM algorithms and tasks. 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 technique within an object that adheres to the minimal interface. This extendibility characteristic makes EMADS an extendible DM facility, and allows developers to employ their pre-implemented programs within EMADS agents.

The following three chapters demonstrate the application of EMADS and EMADS wrappers with respect to various DM scenarios.

Chapter 5

Frequent Set Meta Mining: Meta ARM

In this, and the following two chapters, a sequence of MADM scenarios is considered. Each is selected so as to support the investigation of a sub-set of the MADM issues identified in Chapter 1.

In this chapter a *meta ARM* scenario is considered. The scenario was selected, as a first scenario, because it represented an interesting data mining application which would benefit from a MAS implementation but also leant itself to (relatively straight forward) resolution using a MADM approach. At the same time the scenario presented an opportunity to investigate some potential solutions to the issues identified in Chapter 1. More specifically the scenario facilitated investigation of MADM issues related to: scalability, efficiency, portability, extendibility, privacy protection and agent communication and interaction. The scenario also provided a first opportunity to incorporate suggested mechanisms to address these issues into EMADS, and to evaluate their effectiveness.

The remainder of this chapter is organised as follows. The motivation for the scenario is explored further in Section 5.1. In Section 5.2 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 Section 5.3. Five different agent based approaches to meta ARM are then described in Section 5.4. A meta ARM model in the context of EMADS is presented in Section 5.5. A note on the datasets used is given on Section 5.6. Extendibility consideration is described in Section 5.7. This is followed, in Section 5.8, by an analysis of a sequence of experimental results used to evaluate the approaches introduced in Section 5.4.

Discussion of how the identified research issues are addressed by this scenario is presented in Section 5.9. Finally a summary is presented in Section 5.10.

5.1 Motivation

The high level MADM problem addressed in this chapter is that of *Meta Data Mining* (meta DM), a process for combining local DM results into a global result. The main objective, in the context of MADM (and by extension EMADS), is to take advantage of the inherent parallelism and distributed nature of MADM to design and implement a powerful and practical distributed DM system. The scenario assumes several data sites interconnected through an intranet or internet; the goal is then to provide the means for data owners to utilise their own local data and, at the same time, benefit from the data that is available at other data sites without transferring or directly accessing that data (thus maintaining privacy and security). This is realised in the context of EMADS, by DM agents that execute at remote data sites and generate DM models that can subsequently be transferred and merged into one global model.

The standard, centralised, approach to data mining is to collate data into a single location. In this central location, a model is then computed from the data. Although this process is easy to understand, and the data mining software design is straightforward, there are a number of drawbacks to this centralised approach:

- The movement of data from the point(s) where it is collected to a central location requires bandwidth and takes time.

- There are substantial performance requirements at the central site: most data mining algorithms must access all of the data at least once, and often in multiple passes, and there is little exploitable locality in the access pattern. The performance of most systems is limited by the memory hierarchy, so the need to fetch records repeatedly from the bottom of the memory hierarchy is a substantial performance issue.

- All of the data is visible at the central location, so there is no way for some or all of the data-collection locations to limit access to their local data, and consequently preserve the privacy of the data. In some situations, it may not be possible to gather the data centrally because of legal restrictions arising from privacy concerns, so some forms of analysis cannot be done at all in a centralised manner.

- In some situations, data may be inherently distributed and cannot be merged into a single database for a variety of reasons including security, fault tolerance, legal constraints, and the maintenance of a “competitive edge”.

With respect to the last point there are many “real life” application domains where data is naturally collected via multiple channels, in different physical and geographically separated locations. Examples include:

- Businesses that collect information about their customers via physical stores, websites, and call centres, which are typically located in different places, perhaps even in different countries.

- Organisations that are built around networks and collect data in a distributed way such as mobile-phone companies that get data about calls made via the tower or cell in which they were initiated.

- Sensor networks that have devices that may be of different kinds and that are placed in many different locations. Each sensor only sees a small part of the total data that is collected; and thus their outputs have to be integrated to give a coherent, global picture. Akyildiz et al. [6] provide a survey on the requirements and characteristics of such sensor networks.

In such cases, it will not be possible to examine all of the data at a central processing site to compute a single global model. As dataset sizes grow and models become more complex, it is natural to consider replacing centralised data mining by parallel and distributed techniques, as a way of reducing costs/overheads.

The specific meta DM technique considered in this chapter is that of meta Association Rule Mining (meta ARM). The term *meta ARM* is used to describe the process of combining the obtained results of M applications of an ARM activity. The problem is defined as follows: an ARM algorithm (or a number of compatible algorithms) is applied to M raw data sets producing M collections of Frequent Item Sets (FISs). Note that it is assumed that 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 Item Sets (FISs) with the aim of generating a set of Association Rules (ARs).

The most significant issue when combining groups of previously identified FISs 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). This will usually demand a revisit to B to obtain counts not initially included in the FISs counted at B. The challenge is thus to combine the results from M different data sources in the most computationally efficient manner. This in turn is influenced predominantly by the number and magnitude (in terms of data size) of returns to the source data that may be required.

There are a number of alternative mechanisms whereby ARM results can be combined to satisfy the requirements of meta ARM (in the context of MADM). To fully investigate some of these alternatives, five different meta ARM approaches (including a bench mark approach) were considered. The approaches were realised using variations of the Apriori-TFP, set enumeration tree based, ARM algorithm; which, using EMADS, were wrapped into a DM agent. However, the proposed approaches may be implemented using alternative algorithms such as FP-growth; and other algorithms that use set enumeration tree style data structures, the support-confidence framework and an Apriori methodology for the processing/building of trees. The Apriori-TFP and FP-growth algorithms were described in detail in Chapter 2, Subsection 2.1.1.1. Whatever the case the meta ARM results reported in this chapter are generally applicable.

Broadly the meta DM scenario is an exemplar of more generic distributed/ parallel data mining. Issues relating to MADM (and EMADS) with respect to parallel and distributed data mining are considered in further detail in the following chapter.

In the context of MADM the meta ARM scenario allowed the investigation of a number of the MADM issues identified in Chapter 1, namely:

1. **Scalability**: Scalability and computational efficiency (see below) are both well established critical issues in DM [59]. In the context of meta ARM the MADM framework is expected to effectively support the addition of

further data agents (and DM agents) without adversely affecting the performance. Thus the proposed MADM meta ARM solution should continue to operate effectively, and without a substantial or discernible reduction in performance, as the number of agents increases. Scalability might be taken to demand that performance (execution time) scales linearly as the volume of data increases. DM algorithms, however, tend to scale polynomially or even exponentially (which is the main performance issue for them). So, in a meta ARM algorithm, scalability is defined to mean a performance not much worse than a single algorithm applied to the merged data.

2. **Efficiency**: The main issue here is the effective use of available agents to avoid use of irrelevant data, and the mechanism for moving results between agents. The latter may entail a significant computational overhead.

3. **Portability**: The MADM is expected to operate across multiple environments with different hardware and software configuration. In the context of meta ARM, this also includes the ability to combine disparate models.

4. **Extendibility**: Extendibility is essentially concerned with the capability of MADM to incorporate and use new algorithms and data sources. In other words the ease with which additional agents can be added/incorporated into the EMADS solution to the meta ARM problem.

5. **Privacy protection**: The ability of MADM to protect and preserve the privacy of data is a significant issue for meta ARM (and meta DM in general). In the context of meta ARM the solution should provide for the effective mining of the data without compromising the source of the data.

6. **Agent communication and interaction**: The ability of the agents to: (i) communicate with each other using an expressive communication language; (ii) work together cooperatively to accomplish complex goals; (iii) act on their own initiative; and (iv) use local information and knowledge to manage local resources and handle requests from peer agents; are all significant with respect to any MADM scenario.

5.2 Background and Previous Work

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, there are typically only two input data sets: (i) a large set of previously identified FISs D and (ii) a much smaller set of itemsets d to be processed in order to update D. In the case of meta ARM there can be any number of sets of input data which can be of any size. Furthermore, in this case each contributing set has already been processed to identify local FISs. I-ARM algorithms typically operate using a relative support threshold [71, 74, 120] as opposed to an absolute threshold, (i.e. a threshold expressed as a percentage rather than an absolute value); 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. 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 5.1.

	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 5.1: I-ARM itemset comparison options (relative support)

The key issue in I-ARM is that when a set that is not frequent in D is found to be frequent in d, a (costly) re-examination of D is required to obtain the necessary support-count. From the literature, three fundamental approaches to this problem can be identified. These may be categorised as follows:

1. Maintain the itemsets on the *negative border* and hope that this includes all those itemsets that may become frequent. See for example ULI [117]. The negative border itemsets are the immediate super-sets of the maximal itemsets (where maximal itemsets are frequent item sets all of whose super-sets are infrequent).

2. Make use of a second (lower) support threshold above which items are retained (similar idea to negative border). Examples include AFPIM [71] and EFPIM [74].

3. Acknowledge that, even if either or both of the above strategies are employed; it is likely, at some time or other, that counts for some itemsets will have to recomputed and consequently a data structure has to be maintained that: (i) stores all support counts, (ii) requires less space than the original structure and (iii) facilitates fast look-up, to enable updating. See for example [72].

Using a reduced support threshold and/or retaining the negative border results in a significant additional storage overhead. For example if there is a given data set with 100 attributes ($n = 100$) 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 same issues associated with I-ARM will arise, and similar strategies will need to be employed, when meta ARM is considered.

The distinction between distributed data mining (DDM) and MADM is one of control. DDM 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. 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.

Other research on meta mining includes work on meta classification. Meta classification, also sometimes referred to as meta learning, is a technique for generating a *global* classifier from M distributed data sources by first computing M *base* classifiers which are then collated to build a single *meta* classifier [95] in much the same way that ARM results are collated with respect to the meta ARM scenario.

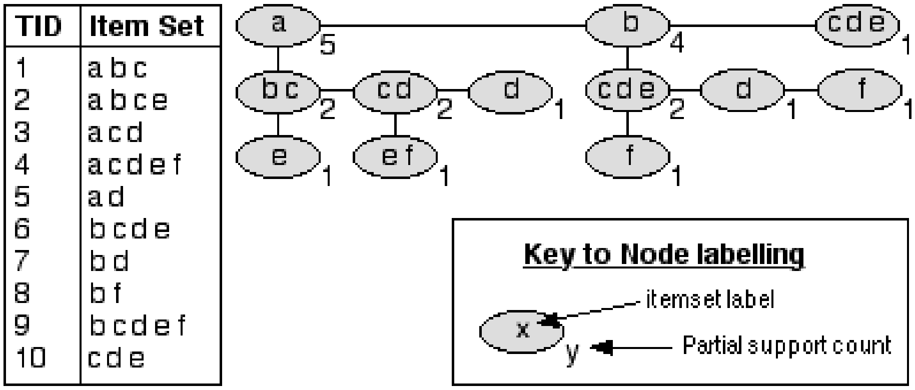
Bagging [22], boosting [38] and dagging [118] are also well known re-sampling ensemble methods that generate and combine a diversity of classifiers using the same learning algorithm for the base-classifiers. Boosting algorithms are considered stronger than bagging and dagging on noise-free data [38].

The term *merge mining* is used in Aref et al. [9] to describe a generalised form of incremental ARM that has some conceptual similarities to the ideas behind meta ARM as described in this chapter. 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.

5.3 Note on P- and T-Trees

ARM algorithms require some sort of data structure to store itemsets as they are being processed. The original Apriori algorithm [5] used hash trees. The well known FP-growth [49] algorithm used a bespoke data structure called the FP-tree [50]. The meta ARM algorithms described in this chapter makes extensive use of two tree data structures, namely P-trees and T-trees. The nature of these structures is described in detail in [31, 45]; however, for completeness a brief overview is presented here.

The P-tree (**P**artial Support Tree) is a set enumeration tree style data 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 used 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 5.1. In the figure the data set given on the left is stored in the P-tree on the right. The partial count stored at each node is the count of all sets for which the node string is a leading substring. 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 P-tree can be constructed very efficiently in a single database pass, and the partial counting reduces the effort required finally to produce support-counts. This is especially useful if it becomes necessary to revisit the data to get support-counts for items not previously found to be frequent (i.e. in the I-ARM and meta ARM cases).



TID	Item Set
1	a b c
2	a b c e
3	a c d
4	a c d e f
5	a d
6	b c d e
7	b d
8	b f
9	b c d e f
10	c d e

Figure 5.1: P-tree Example

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 FISs, in a compressed form (which maybe identified by processing the P-tree). An example, generated from the P-tree given in Figure 5.1, is presented in Figure 5.2. In the figure the circles indicate node records, and the number indicates the support for the record.

From Figure 5.2 it can be seen that the top level comprises an array of references to node records that hold the support count and references to the next level (providing such a level exists). Indexes equate to item (attribute) numbers, although for ease of understanding in the figure letters have been used instead of numbers. The structure can be thought 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).

With respect to the work described here the T-tree is generated using the Apriori-TFP algorithm. The Apriori-TFP algorithm is essentially an Apriori

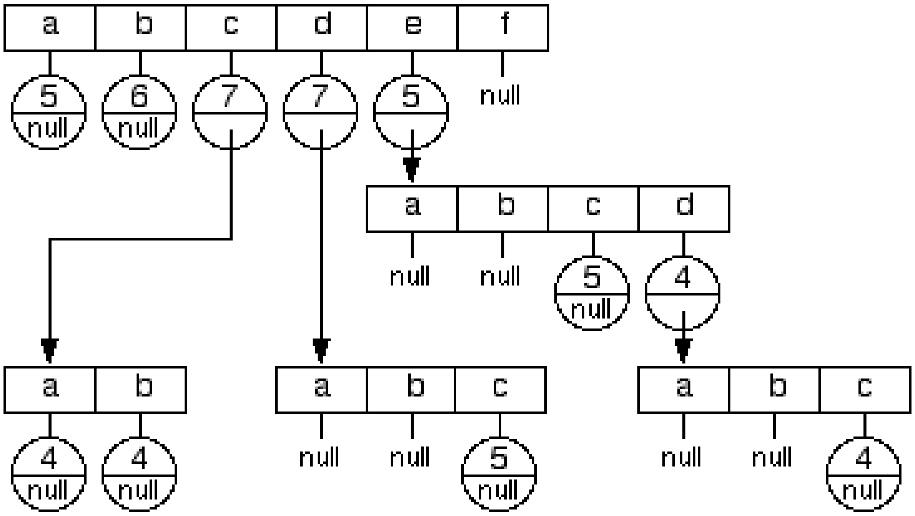


Figure 5.2: T-tree Example (support = 35%)

style algorithm that proceeds in a level by level manner. At each level the P-tree is processed to generate appropriate support counts. Note that on completion of the Apriori-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 5.2 unsupported sets are indicated by a *null* reference.

5.4 Proposed Meta ARM Algorithms

In this section a number of meta ARM MADM algorithms are described, an analysis of which is presented in Section 5.8. All algorithms are incorporated into one EMADS task agent. A data agent maintains the data set in either its raw form or a compressed form. For the experiments reported in this chapter the data has been stored in a compressed form using a P-tree as described above.

The first algorithm developed was a bench mark algorithm, against which the later developed meta ARM algorithms were to be compared. This bench mark algorithm is described in Subsection 5.4.1. Four further meta ARM MADM algorithms were then wrapped and constructed into meta ARM task agent: (i) Apriori, (ii) Brute Force, (iii) Hybrid 1 and (iv) Hybrid 2. These are described in Subsections 5.4.1, 5.4.2, 5.4.3, and 5.4.4. For the meta ARM scenario presented in this chapter, each data agent has a DM agent temporarily associated with it, which would produce a set of frequent sets using the Apriori-TFP algorithm with the results stored in a T-tree. These T-trees would then be merged in some manner using further task agents.

Each of the meta ARM MADM algorithms described below makes use of *return to data* (RTD) lists, one per data agent, that holds itemsets whose support was not included in the current T-tree and for which the count was 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 its support 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 by the appropriate task agent as detailed in Figure 5.3. 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.

5.4.1 Bench Mark Algorithm

```
Algorithm 5.1: Bench Mark Meta ARM

$k = 1$;
Generate candidate $k$-itemsets
Start Loop
  if (there is no candidate $k$-itemsets) break
  else
    for all $N$ data sets (data agents) get counts for $k$-itemsets
    Prune $k$-itemsets according to support threshold
    $k \Leftarrow k+1$
    Generate $k$-itemsets
end Loop
```

Table 5.2: Bench Mark Meta ARM Algorithm

For meta ARM to make sense the process of merging the distinct sets of discovered FISs must be faster than starting from the beginning. The first algorithm developed was therefore a bench mark algorithm. This was essentially an Apriori style algorithm (as described in Table 5.2) that used a T-tree as a storage structure to support the generation process. The algorithm involved no merging but processed the entire dataset in a single process. To be worth while any meta ARM algorithm must outperform the bench mark algorithm.

5.4.2 Brute Force Meta ARM Algorithm

The philosophy behind the Brute Force meta MADM ARM algorithm was that the collection of T-trees constructed independently at the separate data sources are simply fused 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, held at one of the DM agents, as the "start" merged T-tree (from a computational perspective it is desirable that this is the largest T-tree).

Itemset i	**Frequent in T-tree** M	**Not frequent in T-tree** M
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 merged T-tree branch, starting with current node, to RTD list M
Not Frequent in merged T-tree sofar	Process current branch in T-tree M, 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 $M-1$	Do nothing

Table 5.3: Brute Force Meta ARM itemset comparison options

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 5.3 (remember that the support threshold is relative). Note the similarity between Table 5.3 and the I-ARM itemset comparison options presented in Table 5.1. It is also worth noting 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).

5.4.3 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 Apriori 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 5.2) as shown in Table 5.4. Note that items are added to the RTD list for data source M if a candidate itemset is not included in T-tree M.

Algorithm 5.2: Apriori Meta ARM

```
$k$ = 1;
Generate candidate $k$-itemsets
Start Loop
 if (there is no candidate $k$-itemsets) break
 else
     Add supports for level $K$ from $M$ T-trees or add to RTD list
     Inclusion of additional counts
     Prune $k$-itemsets according to support threshold
     $k \Leftarrow k$+1
     Generate $k$-itemsets
end Loop
```

Table 5.4: Apriori Meta ARM Algorithm

5.4.4 Hybrid Meta ARM Algorithms 1 and 2

The Apriori meta ARM algorithm requires fewer itemsets to be included in the RTD list than in the case of the Brute Force meta ARM algorithm (as will be demonstrated in Section 5.8). However, the Apriori approach requires the RTD lists to be processed at the end of each level generation, while in the case of the 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 adding 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 all the one-item itemsets might be expected to be supported and included in the component T-trees, therefore the algorithm might as well commence by building the top two layers of the merged T-tree.

5.5 Meta ARM EMADS Model

In EMADS, agents are responsible for accessing local data sources and for collaborative data analysis. As described in Chapter 3, 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 Directory Facilitator (DF) agent.

In the context of meta ARM each local DM agent's basic function is to generate "local" item sets (a local model) from local data (provided by a local data agent) and provide this to the task agent in order to generate the complete global set of FISs (global model). 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.

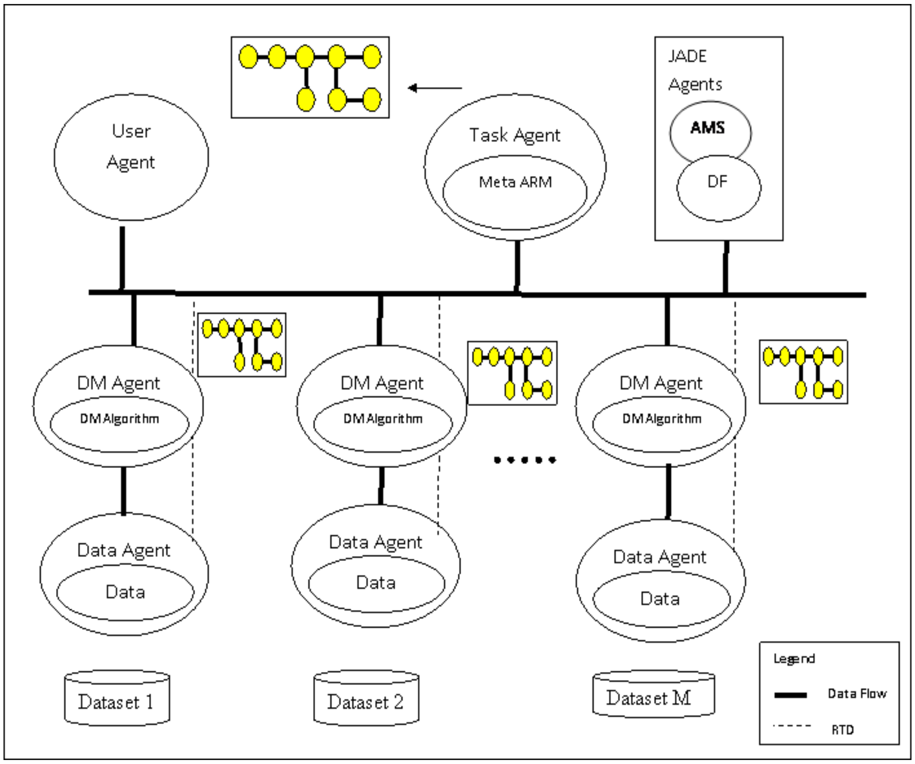


Figure 5.3: Meta ARM Model

Figure 5.3 shows the meta ARM EMADS model. The model consists of M data sources distributed across a network. Each data source has a data agent and a "local" DM agent associated with it. The merging is then done by an appropriately defined task agent. The figure also shows the house-keeping JADE agents (AMS and DF) through which agents can find each other. In Figure 5.3 data flow and RTD requests between agents are indicated. Local DM agents, operating under decentralised control and an autonomous manner, apply some ARM algorithm to their raw data sets producing M collections of frequent item sets (T-trees). Then a task agent *merges* the different sets of results (merged T-trees) into a single *meta* set of FISs 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 with the aim of generating a set of Association Rules (ARs).

5.6 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 "market basket analysis" scenarios, where records represent customer "basket-fulls" of shopping purchased during a single transaction and the columns/attributes, items or groups of items available for purchase (see Chapter 2, Subsection 2.1.1.2 for an example on market basket analysis). 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 normalisation (as described in Chapter 4, Subsection 4.4.3). Datasets can thus be considered to be NxD tables where N is the number of columns and D is the number of records.

5.7 Extendibility Consideration

The scenario presented in this chapter is a good example to illustrate that the EMADS approach offers a simple way to incorporate existing methods and data sources into a MADM framework and how new DM tasks are wrapped and incorporated. This section looks at what components (agents) are affected when new requirements are added and the extensibility of those components.

5.7.1 Incorporating New Data Source

In the current architecture of EMADS, a single data agent controls each separate data source. The data agent in-turn encapsulates and relies on an instance of an agent class. Thus, in order to bring a new data source into the system, a new data agent must be instantiated. It must also register with the system to ensure the other agents are aware of its existence and the data source can be utilised to fulfil system goals.

In the case of data agents, there is no code required to create a new data agent and introduce it to EMADS. This is achieved through the data wrapper graphical user interface (GUI) as described in Chapter 4, Subsection 4.4.3. The user must use the GUI to refer to the data source location (file path). Then the wrapper creates a new data agent that will represent the data source as it is introduced to the system.

5.7.2 Incorporating New DM algorithms

In order to bring a new DM algorithm into the EMADS system, a new DM agent must be instantiated with the two required components, namely a DM wrapper and DM algorithm. The new DM agent must also be registered with the system to ensure that other agents are aware of its existence and so that the DM algorithm can be utilised to fulfil DM requests.

The independence of the DM technique comes from the fact that the wrapper agents act only as interfaces, to the DM algorithms, to receive requests and pass back results. Because of this, the only code required to create a new DM agent and "introduce" a new DM algorithm becomes a one line instantiation as shown below:

<Algorithm_Class_Name> instanceName = new <Algorithm_Class_Name>();

The DM wrapper contains all code required to interface with the DM algorithm. In order to ensure all new DM algorithms can communicate in the system, an abstract interface was created (i.e. the mining interface, described in Chapter 4, Subsection 4.4.2). Any new DM algorithm class must extend this abstract interface and implement the abstract methods it includes.

5.7.3 Incorporating New DM Tasks

Task agents, that perform DM tasks, are introduced into EMADS using a predefined abstract task agent class (Task Wrapper), described in Chapter 4, Subsection 4.4.1. Because the task wrapper is a subclass of the abstract task agent class, any task agent subclass includes all the logic and methods that allow it to communicate within the system which gives it the flexibility to be used by other agents.

It is important to note that the parent abstract task agent class does not ensure or provide any specific methods in the data mining process. Any subclass can include any other methods or algorithms that would provide the code needed or desired for particular DM task. Because DM algorithms are task specific, these classes allow for any new methods or logic to be implemented, while retaining the ability to communicate with the system through the parent task agent. The parent abstract task agent class enforces the minimum system-specific method implementation possible by providing most of the generic methods (e.g. *"getDataAgents()", and "getMiningAgents()"*) and yet ensures the flexibility for expansion. The task agent abstract class is presented in Chapter 4, Section 4.1.

For example, when the meta ARM task agent was incorporated, the declaration code in the meta ARM agent class was as follows:

public class ARMTaskAgent extends TaskAgent

5.7.4 Registering New Agents

To complete the incorporation of a new agent into the EMADS system, the facilitator agent must be aware it exists. The facilitator agent is made aware of any new agent upon its registration (discussed in Chapter 3, Subsection 3.3.5). Once it is aware of the addition, the task agent is the only remaining agent affected. The task agent is directly affected as it is responsible for tasking all useful data agents in fulfilling a data mining request. Every time a data mining request is made, the task agent queries the facilitator agent for all useful agents to serve the request. Once a new agent is registered, it can be returned (and hence be visible) to the task agent, by the facilitator, for communications. The task agent does not maintain a "memory" of agents that are in the system from request to request. The registration of new agents has been made as easy as possible by placing all registration method implementations in the agent wrapper class. Registration then becomes one line entry *"this.register(agentName, agentType)"* in the *"setup()"* method of the agent. The facilitator agent is notified of the presence through the registration process. By reducing the otherwise complex registration process to one line, a great deal of simplicity is introduced into the extension process from the view point of the agent creator.

For example, when the Meta ARM DM agent was created, the agent's register method was called as follows:

```
this.register("ARMAgent","FrequentItemSets");
```

The registration method code is as follows:

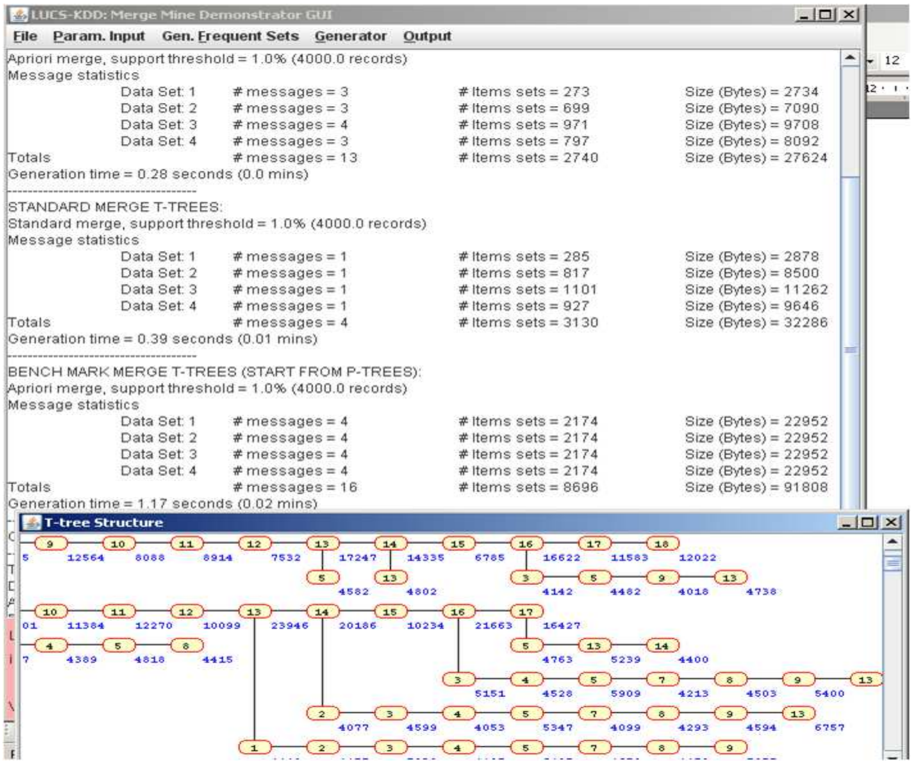
```
//use facilitator agent description
DFAgentDescription dfd = new DFAgentDescription();
//add the agent name to the facilitator agent description
dfd.setName(getAID());
//prepare the agent service description
ServiceDescription sd = new ServiceDescription();
// set the agent type and name
sd.setName(agentName);
sd.setType(agentType);
dfd.addServices(sd);
try {
 //register the agent (services) in the yellow pages
 DFService.register(this, dfd);
}
catch (FIPAException fe) {
 fe.printStackTrace();
}
```

5.8 Experimentation and Analysis

To evaluate the meta ARM scenario and 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 evaluate the issues of efficiency, scalability, and extendibility; and, to an extent, privacy of data. The experiments were particularly 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.
3. The size of the datasets in terms of number of attributes.

All experiments were run using a number of machines with an Intel Core 2 Duo E6400 CPU (2.13 GHz) with 3GB of main memory (DDR2 800MHz), Fedora Core 6, Kernel version 2.6.18 running under Linux or Windows XP. For each of the experiments the following were measured: (i) processing time (seconds



```
Apriori merge, support threshold = 1.0% (4000.0 records)
Message statistics
                    Data Set: 1    # messages = 3      # Items sets = 273      Size (Bytes) = 2734
                    Data Set: 2    # messages = 3      # Items sets = 699      Size (Bytes) = 7090
                    Data Set: 3    # messages = 4      # Items sets = 971      Size (Bytes) = 9708
                    Data Set: 4    # messages = 3      # Items sets = 797      Size (Bytes) = 8092
Totals                             # messages = 13     # Items sets = 2740     Size (Bytes) = 27624
Generation time = 0.28 seconds (0.0 mins)
-----------------------------------
STANDARD MERGE T-TREES:
Standard merge, support threshold = 1.0% (4000.0 records)
Message statistics
                    Data Set: 1    # messages = 1      # Items sets = 285      Size (Bytes) = 2878
                    Data Set: 2    # messages = 1      # Items sets = 817      Size (Bytes) = 8500
                    Data Set: 3    # messages = 1      # Items sets = 1101     Size (Bytes) = 11262
                    Data Set: 4    # messages = 1      # Items sets = 927      Size (Bytes) = 9646
Totals                             # messages = 4      # Items sets = 3130     Size (Bytes) = 32286
Generation time = 0.39 seconds (0.01 mins)
-----------------------------------
BENCH MARK MERGE T-TREES (START FROM P-TREES):
Apriori merge, support threshold = 1.0% (4000.0 records)
Message statistics
                    Data Set: 1    # messages = 4      # Items sets = 2174     Size (Bytes) = 22952
                    Data Set: 2    # messages = 4      # Items sets = 2174     Size (Bytes) = 22952
                    Data Set: 3    # messages = 4      # Items sets = 2174     Size (Bytes) = 22952
                    Data Set: 4    # messages = 4      # Items sets = 2174     Size (Bytes) = 22952
Totals                             # messages = 16     # Items sets = 8696     Size (Bytes) = 91808
Generation time = 1.17 seconds (0.02 mins)
```

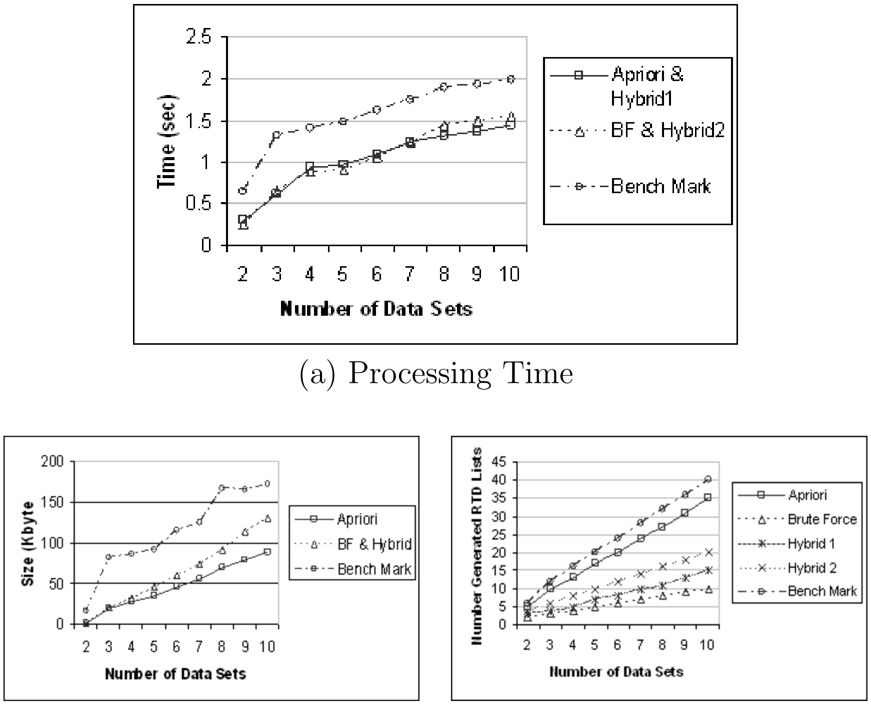
Figure 5.4: User Agent GUI: Meta ARM Example

/ milliseconds), (ii) the size of the RTD lists (Kilobytes) and (iii) the number of RTD lists generated. The IBM Quest generator [2] was not used to generate test data because many different data sets (with the same input parameters) were required and the Quest generator was found to always generated the same data given the same input parameters. Instead the LUCS-KDD data generator [1] was used.

Figure 5.4 shows a "screen shot" of the EMADS user agent GUI during the execution of one of the experiments. The GUI is showing the results of executing a number of meta ARM tasks on four datasets which were distributed on four machines. During each task execution, the GUI displays message statistics (number and size of messages, number of itemsets and their totals) and then displays the required time for the completion of each task.

Figure 5.5 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), and $D = 100k$ (Number of transactions). Note that the selection of a relatively low value for N

[1] 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 5.5: Effect of Number of Data Sources

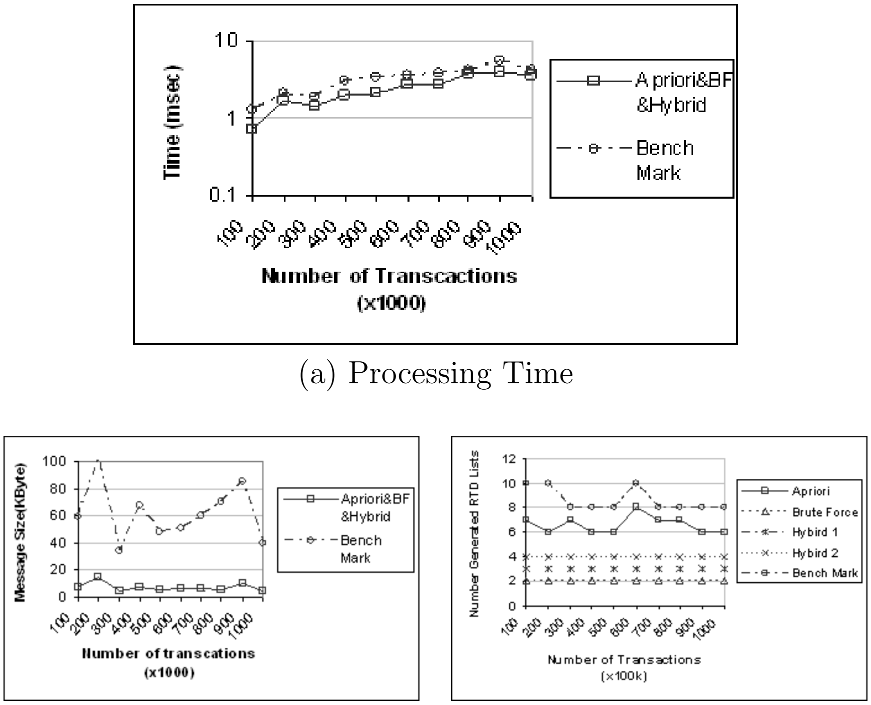
ensured that there were some common FISs shared across the T-trees.

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 5.5(a) demonstrates that all of the proposed meta ARM algorithms worked better then the bench mark (start all over again) approach, i.e. that meta ARM is worth doing. 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 5.5(b), correlates directly with the execution times in graph 5.5(a).

Graph 5.5(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).



(a) Processing Time

(b) Total size of RTD lists

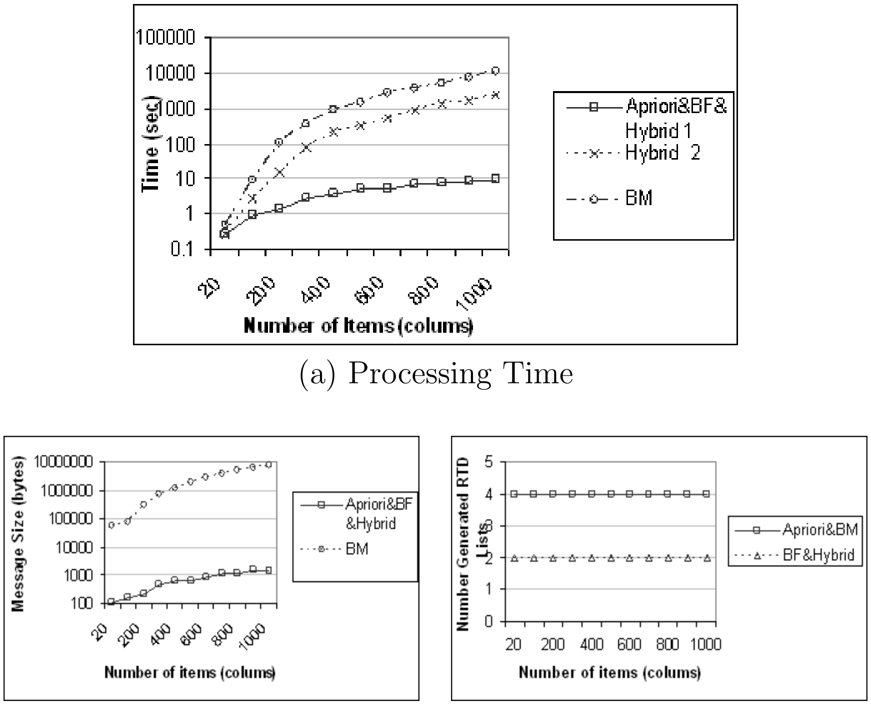
(c) Number of RTD lists

Figure 5.6: Effect of Increasing Number of Records

Figure 5.6 demonstrates the effect of increasing the number of records. The input data for this experiment was generated by producing a sequence of ten pairs of data sets (with $T = 4$, $N = 20$) representing two sources. From graph 5.6(a) it can be seen that the all the meta ARM algorithms outperformed the bench mark algorithm because the size of the RTD (return to data) lists were limited as no unnecessary candidate sets were generated. This is illustrated in graph 5.6(b). Graph 5.6(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 5.6(c)).

Figure 5.7 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 5.7(a, b). However, in this case (compared to the previous experiment), the Hybrid 2 algorithm did not work as well. The reasoning behind the Hybrid 2 algorithm's 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 5.7(a)). For completeness graph 5.7(c) indicates the number of RTD lists sent with respect to the different algorithms.



(a) Processing Time

(b) Total size of RTD lists

(c) Number of RTD lists

Figure 5.7: Effect of Increasing Number of Items (attributes)

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 generated. 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.

5.9 Discussion

This section returns to the MADM issues listed in Section 5.1 which the described meta ARM scenario was designed to contribute to, or take into consideration. The mechanisms proposed and incorporated into the resolution of the meta ARM scenario, and by extension EMADS, with respect to these issues are listed below:

1. **Scalability:** The issue here is that the proposed meta ARM mechanism must operate effectively and without a substantial or discernible reduction in performance as the number of data sites increases. This is addressed by providing specific protocols that allow the management of agents to support the collaboration of the data sites. The solution constitutes a scalable (and unifying) machine learning approach that can be applied to large amounts of data in wide area computing networks for a range of different applications. The distributed memory architecture used also contributes to scalability, in that the memory is scalable as increases in the number of data sources (agents) increases the size of memory proportionately. The proposed approach is unifying because it is algorithm and representation independent, i.e. it does not examine the internal structure and strategies of the DM algorithms themselves, but only the outputs (FISs) of the individual DM agents. As can be seen from the experimental results presented in the previous section, the meta approach scales slowly and linearly as the volume of data or the number of data sources increases.

2. **Efficiency:** Efficiency is inherently introduced into the meta ARM solution through the distributed nature of the problem, incorporated into the initial problem definition, that supports distributed/parallel processing. The proposed meta ARM strategy enhances efficiency (and scalability) firstly by executing the DM processes in parallel under decentralised control and in an autonomous manner, and secondly by applying the learning processes on smaller subsets of data that are properly partitioned and distributed to fit in main memory (a data reduction technique). In particular, doing some of the analysis on the data locally avoids performance bottlenecks.

In the context of meta DM the approach is especially attractive if there is enough processing power and sophistication at the locations where data are collected to build local models, which can then be moved to the central location and merged to produce a coherent global model. The meta ARM also enhances efficiency by taking advantage of EMADS agent advertising mechanism, to only target data agents with relevant ARM data as to avoid use of irrelevant data.

3. **Portability:** Portability is evidenced within the meta ARM solution in that agents are not tied to any particular platform. As will be seen this is also a general advantage of the EMADS framework.

4. **Extendibility:** The proposed solution may be extended using the EMADS wrapper principle. New data and DM agents can be included without adversely effecting the operation of the proposed meta ARM solution, or requiring the approach to be extended or updated in some way. As described in Section 5.7, the EMADS extendibility feature is illustrated in this scenario by showing how easy it is to incorporate new DM algorithm (i.e. Apriori-TFP), data sources and new DM task (i.e. meta ARM task) into the framework.

5. **Privacy protection:** The suggested meta ARM solution contributes to the important issue of data privacy by giving access to data at a local level only. Each data source has its own local memory; which, in the case of the meta ARM scenario, can only be accessed by a dedicated DM agent. These agents are executed locally and only allowed to share their data mining findings (results) and not the data itself.

6. **Agent communication and interaction:** The suggested meta ARM solution demonstrated the efficient use of EMADS protocols to manage the agents and transfer data to support the collaboration of the data sites.

The reported investigation of the meta ARM MADM scenario also provided an opportunity to investigate, define and compare the operation of a number of different agent based meta ARM techniques. The work described in this chapter therefore also illustrates the use of EMADS as an affective research tool to support distributed/parallel data mining related investigations.

In the context of meta DM the advantages of the approach also serve to highlight the weaknesses of a centralised data mining approach. More specifically:

- There is no longer a need to move raw data from the points of collection to the central site; instead only models or pieces of models need to be moved, and these are of much smaller size than the data.

- Processing takes place at each local site, so the computational load is spread over many processors. Perhaps more importantly, the data are distributed over multiple shallower memory hierarchies, reducing the overheads of data access which dominates most algorithms.

- Because only models leave local sites, details of the raw data can, at least in principle, remain hidden. Hence a distributed approach permits privacy-preserving data mining in a natural way. For example, insurance companies that are competitors could still share their fraud models, and so reduce the costs of fraud to all, without revealing sensitive information.

5.10 Summary

In this chapter the use of EMADS was illustrated in the context of a meta ARM scenario; a novel extension of ARM where a meta set of FISs was built from a collection of component sets which had been generated in an autonomous manner without centralised control. 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. More specifically the work described in this chapter investigated some of the MADM issues identified in Chapter 1, namely: scalability, efficiency, portability, extendibility and privacy protection.

The described experiments indicated, at least with respect to meta ARM, that MADM and by extension 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 main MADM advantages of the described approach may be summarised as follows: (i) its flexibility to allow sharing of DM models (results) without disclosing data proprietary and (ii) its ability to scale as the number and size of databases grow.

In the following chapter, a second demonstration scenario is presented to investigate the usage of MADM to achieve a parallel/distributed data mining scenario namely parallel ARM. This was seen as significant in the context of the scalability and efficiency issues and also to demonstrate MADM re-usability.

Chapter 6

Data Partitioning and Parallel/Distributed ARM

This chapter explores and demonstrates (by experiment) the capabilities of MADM (EMADS) agents in the context of parallel and distributed Data Mining (DM). The exploration, like the previous chapter, is conducted by considering a specific parallel/distributed DM scenario, namely data partitioning to achieve parallel/distributed ARM. The aim of the scenario is to demonstrate that the MADM vision is capable of exploiting the benefits of parallel computing; particularly parallel query processing and parallel data accessing. In addition the scenario provides a vehicle for demonstrating how re-usability can be achieved.

The organisation of the chapter is as follows: In Section 6.1 some background on data distribution and the motivation for the scenario is described. Data partitioning is introduced in Section 6.2. Data partitioning may be achieved in either a horizontal or vertical manner. In Section 6.3 a parallel/distributed task with Data Horizontal Segmentation (DATA-HS) algorithm is described. Before describing the vertical approach the Apriori-T algorithm is briefly described in Section 6.4. The parallel/distributed task with Data Vertical Partitioning (DATA-VP) algorithm (which is founded on Apriori-T) is then described in Section 6.5. The DATA-VP MADM task architecture and network configuration is presented in Section 6.6. Experimentation and Analysis, comparing the operation of DATA-HS and DATA-VP, is then presented in Section 6.7. Discussion of how this scenario addresses the research goals of this thesis is presented in Section 6.8. Finally a summary is given in Section 6.9.

6.1 Motivation

A common feature of most DM tasks is that they are resource intensive and operate on large sets of data. Data sources measured in gigabytes or terabytes are quite common in DM. This has called for fast DM algorithms that can mine very large databases in a reasonable amount of time. However, despite the many algorithmic improvements proposed in many serial algorithms, the large size and dimensionality of many databases makes the DM of such databases too slow and too big to be processed using a single process. There is therefore a growing need to develop efficient parallel DM algorithms that can run on distributed systems.

There are several ways in which data distribution can occur, and these require different approaches to model construction, including:

- **Horizontal Data Distribution.** The most straight forward form of distribution is horizontal partitioning, in which different records are collected at different sites, but each record contains all of the attributes for the object it describes. This is the most common and natural way in which data may be distributed. For example, a multinational company deals with customers in several countries, collecting data about different customers in each country. It may want to understand its customers worldwide in order to construct a global advertising campaign.

- **Vertical Data Distribution.** The second form of distribution is vertical partitioning, in which different attributes of the same set of records are collected at different sites. Each site collects the values of one or more attributes for each record and so, in a sense, each site has a different view of the data. For example, a credit-card company may collect data about transactions by the same customer in different countries and may want to treat the transactions in different countries as different aspects of the customers total card usage. Vertically partitioned data is still rare, but it is becoming more common and important [81].

This chapter addresses a second generic MADM scenario, that of distributed/parallel DM. This scenario assumes an end user who owns a large data set and wishes to obtain DM results but lacks the required resources (i.e. processors and memory). The data set is partitioned into horizontal or vertical partitions that can be distributed among a number of processors (agents) and independently processed, to identify local itemsets, on each process. In addition to addressing the MADM efficiency and scalability issues, the scenario is also used to demonstrate agent re-usability in that use is made of the meta ARM agents identified in the meta ARM scenario described in the previous chapter.

In the context of MADM the parallel ARM scenario allowed the investigation of a number of the research issues identified in Chapter 1, namely:

1. **Scalability and efficiency**: Scalability and computational efficiency. Scalability might be taken to demand that performance (execution time) scales linearly as the volume of data increases. DM algorithms, however, tend to scale polynomially or even exponentially (which is the main performance issue for them). Distributed and parallel DM can improve both efficiency and scalability first by executing the DM processes in parallel improving the run-time efficiency and second, by applying the DM processes on smaller subsets of data that are properly partitioned and distributed to fit in main memory.

2. **Extendibility:** Extendibility is essentially concerned with the capability of MADM to incorporate and use new algorithms and data sources. In other words the ease with which additional agents can be added/incorporated into the EMADS solution to the parallel ARM problem.

3. **Re-usability:** 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 whereby agents can find other agents supporting certain capabilities.

4. **Agent communication and interaction:** The ability of the agents to communicate with each other and work together cooperatively and in parallel to accomplish complex goals is significant with respect to any this scenario.

In the exploration of the applicability of MADM to parallel/distributed ARM, the two parallel ARM approaches, based on the Apriori algorithm, are described and their performance evaluated. Recall that DATA-HS makes use of a horizontal partitioning of the data. The data is apportioned amongst the processes, typically by horizontally segmenting the dataset into sets of records. DATA-VP makes use of a vertical partitioning approach to distributing the input dataset over the available number of DM (worker) agents. To facilitate the vertical data partitioning the tree data structure, described in Chapter 5, Section 5.3, is again used together with the Apriori-T ARM algorithm [29]. Using both approaches each partition can be mined in isolation, while at the same time taking into account the possibility of the existence of frequent itemsets dispersed across two or more partitions. In the first approach, DATA-HS, the scenario complements the meta ARM scenario described in the previous chapter.

6.2 Data Segmentation and Partitioning

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 has been directed at parallel ARM to decrease overall processing times (see [47, 91, 105, 115]), and distributed ARM to support the mining of datasets distributed over a network [25]. The main challenges associated with parallel DM include:

- Minimising I/O.
- Minimising synchronisation and communication.
- Effective load balancing.
- Effective data layout (horizontal vs. vertical).
- Good data decomposition.
- Minimising/avoiding duplication of work.

To allow the data to be mined using a number of cooperating agents the most obvious approach is to allocate different subsets of the data to each agent. There are essentially two fundamental approaches to partitioning/segmenting the data:

1. Horizontal segmentation where the data is divided according to row number.

2. Vertical partitioning where the data is divided according to column number.

Note that in this chapter the term partitioning is used to indicate vertical sub-division of data, and segmentation to indicate horizontal sub-division of data.

Horizontal segmentation, is in general more straightforward. Assuming a uniform/homogeneous dataset it is sufficient to divide the number of records by the number of available agents and allocate each resulting segment accordingly.

The most natural method to vertically partition a dataset is to divide the number of columns by the number of available agents so each is allocated an equal number of columns.

Many parallel DM algorithms have been developed based on the Apriori algorithm or variations of the Apriori algorithm. The most common parallel methods are [4, 47]:

- **Count Distribution.** This method follows a data-parallel strategy and statically partitions the database into horizontal partitions that are independently scanned for the local counts of all candidate itemsets on each process. At the end of each iteration, the local counts are summed across all processes to form the global counts so that frequent itemsets can be identified.

- **Data Distribution.** The Data Distribution method attempts to utilise the aggregate main memory of parallel machines by partitioning both the database and the candidate itemsets. Since each candidate itemset is counted by only one process, all processes have to exchange database partitions during each iteration in order for each process to get the global counts of the assigned candidate itemsets.

- **Candidate Distribution.** The Candidate Distribution method also partitions candidate itemsets but selectively replicates, instead of "partition-and-exchanging" the database transactions, so that each process can proceed independently.

Experiments show that the Count Distribution method exhibits better performance and scalability than the other two methods [4]. The steps for the Count Distribution method may be generalised as follows (for distributed-memory multiprocessors):

1. Divide the database evenly into horizontal partitions among all processes.

2. Each process scans its local database partition to collect the local count of each item.

3. All processes exchange and sum up the local counts to get the global counts of all items and find frequent 1-itemsets.

4. Set level $k = 2$.

5. All processes generate candidate k-itemsets from the mined frequent (k-1)-itemsets.

6. Each process scans its local database partition to collect the local count of each candidate k-itemset.

7. All processes exchange and sum up the local counts into the global counts of all candidate k-itemsets and find frequent k-itemsets among them.

8. Repeat (5) - (8) with $k = k + 1$ until no more frequent itemsets are found.

In the following sections two MADM tasks, using both vertical partitioning and horizontal segmentation, are introduced. These tasks were implemented using the task wrapper, as described in Chapter 4, so that they could be incorporated into EMADS as task agents.

6.3 The Parallel/Distributed Task with Horizontal Segmentation (DATA-HS) Algorithm

The Data Horizontal Segmentation (DATA-HS) algorithm uses horizontal segmentation, dividing the dataset into segments each containing an equal number of records. ARM in this case involves the generation of a number of T-trees, holding frequent itemsets, one for each segment; and then merging these T-trees to create one global T-tree. As a demonstration of MADM re-usability, this is carried out using the meta ARM task agent which is defined from the previous scenario.

Assuming that a data agent representing the large dataset has been launched by a user, the DATA-HS MADM algorithm comprises the following steps:

1. User agent requests the task agent to horizontally segment the dataset according to the total number of segments required.

2. The task agent assigns and sends each data segment to an interested data agent; if none exist then the task agent launches new data agents.

3. Then a meta ARM task is called (in the experiments described in this chapter the Apriori meta ARM task was used) to obtain the Association Rules (ARs) as described in Chapter 5.

6.4 The Apriori-T Algorithm

The Apriori-T (Apriori Total) algorithm is an Association Rule Mining (ARM) algorithm [31] that combines the classic Apriori ARM algorithm with the T-tree data structure. 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 frequent itemsets. The Apriori-T algorithm was developed as part of the more sophisticated ARM algorithm The Apriori-TFP. The Apriori and Apriori-TFP algorithms were described in Chapter 2, Subsection 2.1.1.1.

At each level, new candidate itemsets of size k are generated from identified frequent 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 frequent itemset.

The number of candidate nodes generated during the construction of a T-tree, and consequently the computational effort required, is very much dependent on the distribution of columns within the input data. Best results are produced by ordering the dataset, according to the support counts for the 1-itemsets, so that the most frequent 1-itemsets occur first [28].

6.5 The Parallel/Distributed Task with Vertical Partitioning (DATA-VP) Algorithm

The second algorithm considered in the exploration of the applicability of MADM to parallel/distributed ARM is the Data Vertical Partitioning (DATA-VP). The DATA-VP algorithm commences by distributing the input dataset over the available number of workers (DM agents) using a vertical partitioning strategy. Initially the set of single attributes (columns) is split equally between the available workers so that an *allocationItemSet* (a sequence of single attributes) is defined for each DM agent in terms of a $startColNum$ and $endColNum$:

$$\mathbf{allocationItemSet} = \{\mathbf{n}|\mathbf{startColNum} < \mathbf{n} \leq \mathbf{endColNum}\}$$

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 their *allocationItemSet* the task agent will partition the data among the workers (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 still need to be included in the subtree counted by the DM agent.

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

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

```
$\forall$ records $\in$ input data
if (record $\cap$ allocationItemSet $\equiv$ true)
record = $\{n|n \in$ record, $n \leq$ endColNum$\}$
else delete record
```

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

Table 6.1: Dataset Example

As an example, the ordered data set in Table 6.1 has items with 6 attributes, a, b, c, d, e and f. Assuming three worker agents are participating, the above partitioning process will result in three dataset partitions, with *allocationItemSets* {a, b}, {c, d} and {e, f}. Application of the above algorithm will create partitions as follows (but note that the empty sets, here shown for clarity, will in fact not be included in the partitions):

Partition 1 (a to b): $\{\{a\}, \{b\}, \{a\}, \{b\}, \{a\}, \{a, b\}, \{\}, \{a, b\}, \{\}, \{a, b\}\}$

Partition 2 (c to d): $\{\{a, c\}, \{\}, \{a, c\}, \{b, d\}, \{\}, \{a, b, c\}, \{d\}, \{\}, \{c\}, \{a, b, d\}\}$

Partition 3 (e to f): $\{\{a, c, f\}, \{\}, \{a, c, e\}, \{\}, \{a, e\}, \{\}, \{\}, \{\}, \{\}\}$

Once partitioning is complete each partition can be mined, using the Apriori-T algorithm, in isolation while at the same time taking into account the possibility of the existence of frequent itemsets dispersed across two or more partitions. Figure 6.1 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 balanced.

The DATA-VP MADM task can thus be summarised as follows:

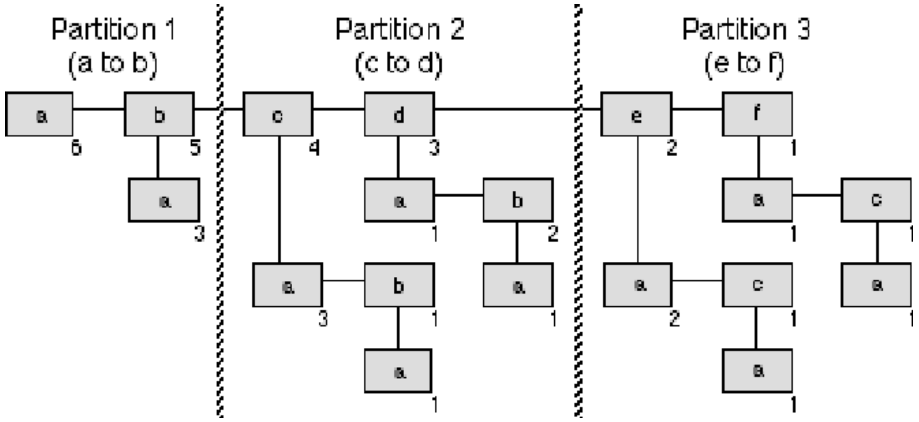


Figure 6.1: Vertical Partitioning of a T-tree Example [29]

1. A task agent starts a number of workers (DM agents); these will be referred to as *workers*.

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

3. The task agent transmits the allocated partition of the data (calculated as described above) to each worker.

4. Each worker then generates a T-tree for its allocated partition (a sub T-tree of the final T-tree) using the Apriori-T algorithm as described below.

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

The local T-tree generation process begins with a top-level "tree" comprising only those 1-itemsets included in each worker (DM agent) *allocationItemSet*. The DM agent will then generate the candidate 2-itemsets that belong in its sub (local) T-tree. These will comprise all the possible pairings between each element in the *allocationItemSet* and the lexicographically preceding attributes of those elements (see Figure 6.1). 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 DM agents. To avoid the overhead involved in the X-checking process, described in Section 6.4, which in this case would require message-passing between the DM agents concerned, X-checking does not take place. Instead, the DM agent will generate its candidates assuming, where necessary, that any subsets outside its local T-tree are frequent.

6.6 DATA-VP Task Architecture and Network Configuration

The DATA-VP task architecture shown in Figure 6.2 assumes the availability of at least one worker (DM agent), preferably more. Figure 6.2 shows the assumed distribution of agents and shared data across the network. The figure also shows the house-keeping JADE agents (AMS and DF) through which agents find each other.

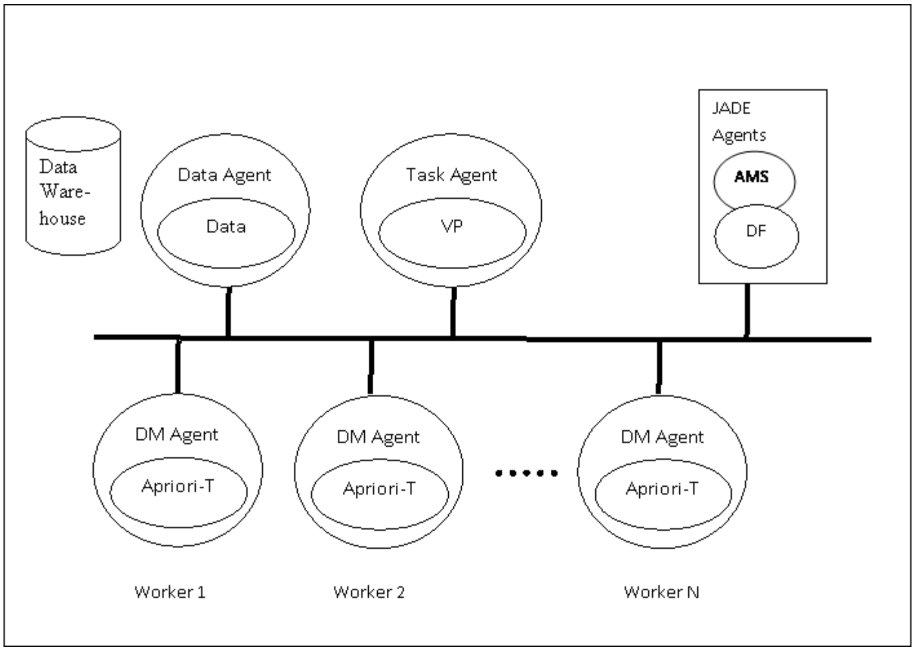


Figure 6.2: Parallel/Distributed ARM Model for DATA-VP Task Architecture

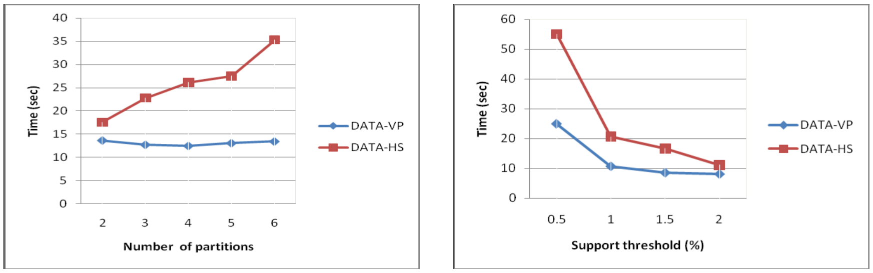
6.6.1 Messaging

Parallel/distributed ARM tends to 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 are significant factors affecting performance. It is therefore expedient, in the context of the techniques described here, to minimise the number of messages that are required to be sent as well as their size.

The technique described here is One-to-Many approach, where only the task agent can send/receive messages to/from DM agents. This involves fewer operations, although, the significance of this advantage decreases as the number of agents used increases.

6.7 Experimentation and Analysis

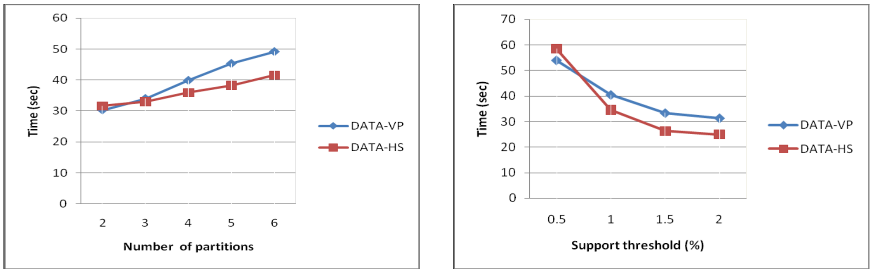
To evaluate the two approaches, in the context of the EMADS vision, a number of experiments were conducted. These are described and analysed in this section. The experiments presented here were run on one machine with an Intel Core 2 Duo E6400 CPU (2.13 GHz) with 3GB of main memory (DDR2 800MHz), Fedora Core 6, Kernel version 2.6.18 running under Linux and used up to six data partitions and two artificial datasets: (i) T20.D100K.N250.num, and (ii) T20.D500K.N500.num where $T = 20$ (average number of items per transactions), $D = 100K$ or $D = 500K$ (Number of transactions), and $N = 500$ or $N = 250$ (Number of attributes) are used. The datasets were generated using the IBM Quest generator used in Agrawal and Srikant [2].



(a) Number of Data Partitions (b) Support Threshold

Figure 6.3: Average of Execution Time for Dataset T20.D100K.N250.num

As noted above the most significant overhead of any parallel/distributed system is the number and size of messages sent and received between agents. For the



(a) Number of Data Partitions (b) Support Threshold

Figure 6.4: Average of Execution Time for Dataset T20.D500K.N500.num

DATA-VP EMADS approach, 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. DATA-VP passes entire pruned sub (local) T-tree branches. Therefore, DATA-VP has a clear advantage in terms of the number of messages sent.

Figure 6.3 and Figure 6.4 show the effect of increasing the number of data partitions with respect to a range of support thresholds. As shown in Figure 6.3 the DATA-VP algorithm shows better performance compared to the DATA-HS algorithm. This is largely due to the smaller size of the dataset and the T-tree data structure which: (i) facilitates vertical distribution of the input dataset, and (ii) readily lends itself to parallelisation/distribution. However, when the data size is increased as in the second experiment, and further DM (worker) agents are added (increasing the number of data partitions), the results shown in Figure 6.4, show that the increasing overhead of messaging size outweighs any gain from using additional agents, so that parallelisation/distribution becomes counter productive. Therefore DATA-HS showed better performance from the addition of further data agents compared to the DATA-VP approach.

6.8 Discussion

MADM can be viewed as an effective distributed and parallel environment where the constituent agents function autonomously and (occasionally) exchange information with each other. EMADS is designed with asynchronous, distributed communication protocols that enable the participating agents to operate independently and collaborate with other peer agents as necessary, thus eliminating centralised control and synchronisation barriers.

Distributed and parallel DM can improve both efficiency and scalability first by executing the DM processes in parallel improving the run-time efficiency and second, by applying the DM processes on smaller subsets of data that are properly partitioned and distributed to fit in main memory (a data reduction technique).

The scenario, described in this chapter, demonstrated that MADM provides suitable mechanisms for exploiting the benefits of parallel computing; particularly parallel data processing. The scenario also demonstrated that MADM is suitable for re-usability and illustrated how it is supported by re-employing the meta ARM task agent, described in the previous chapter, with the DATA-HS task.

6.9 Summary

In this chapter a MADM method for parallel/distributed ARM has been described so as to explore the MADM issues of scalability and re-usability. Scalability is explored by parallel processing of the data and re-usability is explored by re-employing the meta ARM task agent with the DATA-HS task.

The solution to the scenario considered in this chapter made use of a vertical data partitioning or a horizontal data segmentation technique to distribute the input data amongst a number of agents. In the horizontal data segmentation (DATA-HS) method, the dataset was simply divided into segments each comprising an equal number of records. Each segment was then assigned to a data agent that allowed for using the meta ARM task when employed on EMADS. Each DM agent then used its local data agent to generate a complete local T-tree for its allocated segment. Finally, the local T-trees were collated into a single tree which contained the overall frequent itemsets. The proposed vertical partitioning (DATA-VP) was facilitated by the T-tree data structure, and an associated mining algorithm (Apriori-T), that allowed for computationally effective parallel/distributed ARM when employed on EMADS.

The reported experimental results showed that the data partitioning methods described are extremely effective in limiting the maximal memory requirements of the algorithm, while their execution time scale only slowly and linearly with increasing data dimensions. Their overall performance, both in execution time and especially in memory requirements has brought significant improvement.

Chapter 7

Classifier Generation

To further explore the issues associated with the concept of a generic MADM, a classifier generation scenario is considered in this chapter. The point of this scenario is to investigate how MADM can be used to support the sharing of resources and expertise while at the same time preserving intellectual property rights that may be attached to Data Mining (DM) software contributed by MADM users. The scenario is also intended to further demonstrate how extendibility is supported. The scenario considered is where a user wishes to generate a classifier for a specified dataset. In this scenario, a collection of classification (DM) agents is applied to a number of standard benchmark datasets held by data agents. A number of classifiers are generated, from which the best can be selected.

The chapter is organised as follows. In Section 7.1 the motivation for the scenario is described. Some background on data classification is presented in Section 7.2. The operation of EMADS with respect to classification is then described in Section 7.3. Extendibility consideration is described in Section 7.4. The evaluation of the MADM classification process is then described in Section 7.5. A summary is given in Section 7.6.

7.1 Motivation

Multi-Agent Systems (MAS) have some particular potential advantages to offer with respect to KDD, in the context of sharing resources and expertise. Namely that the MADM approach provides the possibility of greater end user access to Data Mining (DM) techniques. MADM can make use of algorithms without necessitating their transfer to users, thus contributing to the preservation of any intellectual property rights over the algorithms. This is a significant issue with respect to the EMADS vision espoused in this thesis. Recall that the vision is that of an anarchic collection of agents that can organically grow (using the EMADS extendibility feature). For this to happen contributors must have confidence in the "security" of any software they may contribute (i.e. the intellectual property rights in the software). This feature of MADM may be illustrated in a number of ways, however the scenario considered in this chapter is in the context of classification.

Although the preservation of the intellectual property in DM algorithms is the primary motivation for the work described in this chapter there are also other secondary motivations, namely:

- To further illustrate the EMADS extendibility feature.
- To demonstrate how EMADS agents can cooperate to produce on "optimum" solutions to a DM problem.

In the scenario considered in this chapter, in addition to the preservation of the intellectual property in DM algorithms, EMADS is used to illustrate how MADM can provide a dynamic solution of the problem of selecting the best classifier for a given dataset. "Best" in this context is measured in terms of classification accuracy (although some other metric, such as precision, could equally well be used). The experiments reported in this chapter thus not only serve to illustrate the advantages of MADM, but also provides an interesting comparison of a variety of classification techniques and algorithms.

7.2 Background

As described earlier in the thesis, classification is a well established data mining task, with roots in machine learning. 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.

This section presents some background to the classification scenario that forms the focus of this chapter. It has been well established within the DM research community, that there is no single best classification algorithm that works for all datasets (a similar observation can be made for clustering algorithms). Selecting an optimal classifier for a pattern recognition application is a challenge; while different classification algorithms can be used to perform the same task, each algorithm might produce a different result (i.e. different accuracy). There is some reported work on best classifier selection. For example the STATLOG project [83] where several classification algorithms were compared based on some empirical datasets and a meta-level machine learning rule on the algorithm selection was provided. Another example is the meta analysis of classification algorithms approach, proposed by Sohn et al. [109], where a statistical meta model to predict the expected classification performance of classification algorithms, as a function of data characteristics, is used to provide a relative ranking of classification algorithms.

Notwithstanding these attempts, it appears clear that no single algorithm will always generate the best classifier in every case. The alternative is an empirical approach, to select what is likely to be a good classifier in the particular circumstances of the problem under consideration.

7.3 Classifier Generation Operation

Conceptually the nature of EMADS DM requests that may be posted by EMADS users is extensive. In the context of classifier generation, the following request is currently supported:

- Find the "best" classifier (to be used by the requester at some later date in off line mode) for a dataset provided by the user.

In other words, the problem can described as follows. Given M datasets and N classification algorithms, produce a mapping returning, for each dataset, the algorithm that returns the best classification for that dataset.

To obtain the "best" classifier EMADS will attempt to access and communicate with as many classifier generators (DM agents) as possible and select the best result.

Figure 7.1 gives an overview of the distribution of agents and shared data across the network. All agents described here have been implemented using the EMADS framework. The figure includes a user agent, a single data agent, a dedicated task agent, and a sequence of N classification (DM) agents each equipped with a classifier generator (FOIL, RDT and TFPC are all classifier generator algorithms, and among those featured in the scenario considered in this chapter). The figure also shows the JADE house-keeping agents (AMS and DF). The principal advantage of this architecture is that it does not overload a single host machine, but distributes the processing load among those multiple machines. The results obtained can be correlated with one another in order to achieve computationally efficient analysis at a distributed global level.

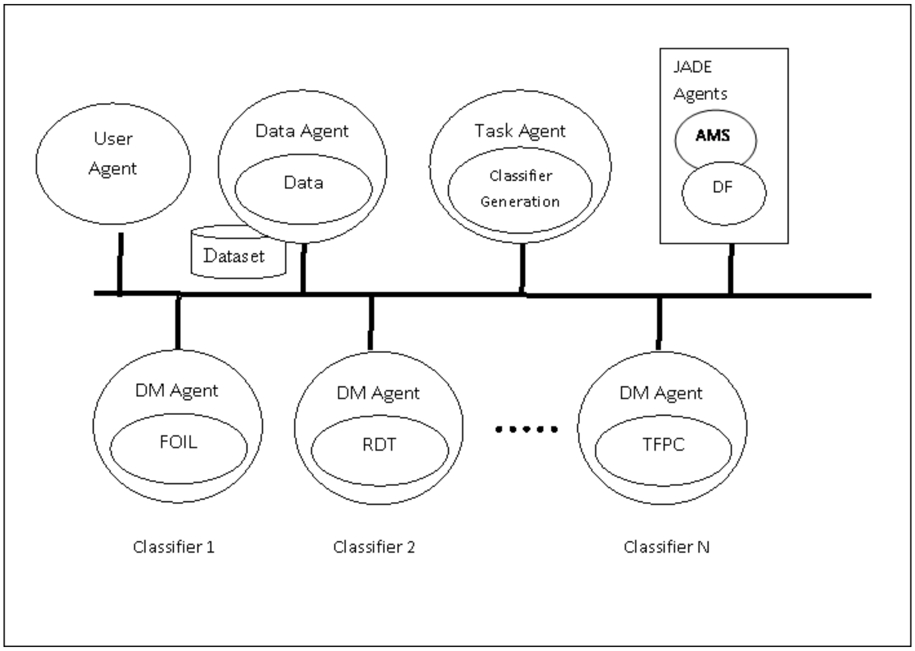


Figure 7.1: Classifier Generation EMADS Model Architecture

For the purpose of the scenario it has been assumed that datasets are binary valued and that the user requires a single-label, as opposed to a multi-labelled, classifier. A single-label classification is the task of assigning an instance to exactly one class, when there are two or more classes. Multi-label classification is the task of assigning an instance simultaneously to one or multiple classes. The request is made using the individual's user agent which in turn will spawn an appropriate task agent.

The task agent identifies DM agents that hold single labelled classifier generators that take binary valued data as input by asking the directory facilitator agent. Each of these DM 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.

When the data agent is launched, it prompts the user through its GUI to provide input of: (i) the data file location; and (ii) the number of class attributes (a value that the DM agent cannot currently deduce for itself) while the user agent GUI allows input for threshold values that may be required by the algorithm (such as support and confidence values). The output is a classifier together with an accuracy measure. To obtain the accuracy measure a classifier generator (a DM 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.

7.3.1 Employed Classifier Generators

From the literature there are many reported techniques available for generating classifiers. With respect to the classifier generation scenario, eight different algorithm implementations were used [1]:

1. FOIL (First Order Inductive Learner) [99]: The well established inductive learning algorithm for generating Classification Association Rules (CARs).
2. TFPC (Total From Partial Classification): A CAR generator [32] founded on the P and T-tree set enumeration tree data structures.
3. PRM (Predictive Rule Mining) [128]: An extension of FOIL.
4. CPAR (Classification based on Predictive Association Rules) [128]: A further development from FOIL and PRM.
5. IGDT (Information Gain Decision Tree) classifier: An implementation of the well known decision tree based classifier using information gain as the "splitting criteria".
6. RDT (Random Decision Tree) classifier: A decision tree based classifier that uses most frequent current attribute as the "splitting criteria" (as not really random as the name suggests).
7. CMAR (Classification based on Multiple Association Rules): A Classification Association Rule Mining (CARM) algorithm [73] frequently cited in the literature.

[1] taken from the LUCS-KDD repository at `http://www.csc.liv.ac.uk/~frans/KDD/Software/`

8. CBA (Classification Based on Associations): Another frequently cited CARM algorithm [75].

These were placed within appropriately defined DM wrappers as described in Chapter 4, Subsection 4.3.2, to produce eight (single label binary data classifier generator) DM agents. This was found to be a trivial operation, requiring little coding expertise, thus indicating the versatility of the wrapper concept. Then the DM agents were incorporated from multiple machines (two agents per one machine) as to imitate the scenario of having these DM agents remotely incorporated by contributors who wish to preserve their software intellectual property.

7.3.2 Dynamic Behaviour of Agents

Each DM (classification) agent's basic function was to generate a classification model (together with an accuracy estimate) 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 7.2 (the figure assumes that an appropriate data agent has already been launched or exists).

The figure includes N classification (DM) 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 (DM) agents that can potentially be used to generate the desired classifier. The task agent then contacts these DM agents which each generate classifier and return statistical information regarding the accuracy of their classifier. The task agent selects the DM agent that has produced the best accuracy and requests the associated classifier, which is then passed back to the user agent.

It is worth noting that the users can either use their own data by making the data that they desire to be mined (classified) available by launching their own data agents (which in turn results in the publication of the new data agent's name and description via the DF agent) or they can use data that has already been introduced through existing data agents.

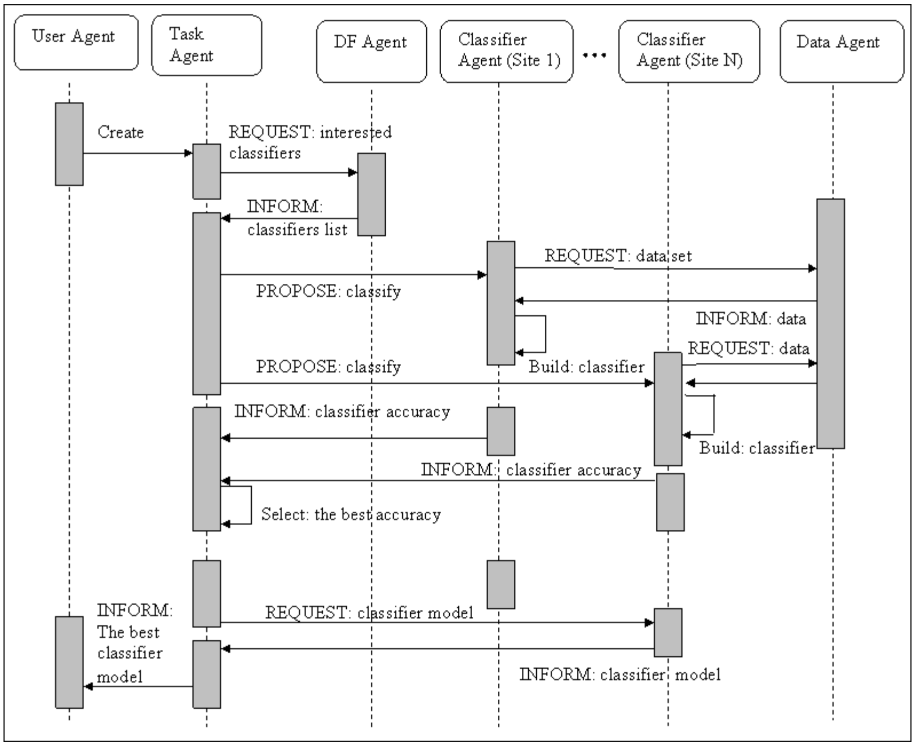

User Agent
Task Agent
DF Agent
Classifier Agent (Site 1)
...
Classifier Agent (Site N)
Data Agent
Create
REQUEST: interested classifiers
INFORM: classifiers list
REQUEST: data set
PROPOSE: classify
INFORM: data
REQUEST: data
PROPOSE: classify
Build: classifier
INFORM: classifier accuracy
Build: classifier
INFORM: classifier accuracy
Select: the best accuracy
INFORM: The best classifier model
REQUEST: classifier model
INFORM: classifier model

Figure 7.2: Classification Task Sequence Diagram.

7.4 Extendibility Consideration

As in the meta ARM scenario presented in Chapter 5, this classification scenario is a good example to illustrate that the EMADS approach offers a simple way to incorporate existing DM methods and data sources into a MADM framework. This section looks at what components (agents) are affected when new classification (DM) algorithms are added and the extendibility of those components.

In order to bring a new classification algorithm into the EMADS system, a new DM agent must be instantiated with the required components, namely a DM wrapper and the classification algorithm. As described previously in Chapter 5, Subsection 5.7.2 the new DM agent must also be registered with the system to ensure that other agents are aware of its existence, and so that the DM algorithm can be utilised to fulfil DM requests.

Recall that the independence of the DM algorithm comes from the fact that the DM wrapper acts only as interface, to the DM algorithm, to receive requests and pass back results. Because of this, the only code required to create a new DM agent and “introduce” a new DM algorithm into EMADS becomes a one line instantiation as shown below:

<Algorithm_Class_Name> instanceName = new <Algorithm_Class_Name>();

For example, when the FOIL algorithm with the Java class name "Foil" was incorporated, the instantiation code in the DM agent was as follows:

Foil foil = new Foil();

As also described in Chapter Subsection 5.7.2, the DM wrapper contains all code required to interface with the DM algorithm. In order to ensure all new DM algorithms can communicate in the system, an abstract interface was created (the mining interface). Any new DM algorithm class must extend this abstract interface and implement the abstract methods it includes.

It is important to note that the DM algorithm class does not have to be a subclass of the abstract Agent class. Because the DM wrapper, that includes all required methods for communication, encapsulates it. The DM algorithm does not need to communicate with any other agents. It will receive all information it requires from its DM agent. The mining interface has certain methods to allow the DM agent to communicate with it. It also allows for the DM algorithm and any addition methods to be included. Because the DM algorithms themselves can be coded in a variety of ways, abstract methods that would force a particular processing of the algorithm were not included. The included abstract methods are presented in Chapter 4, Subsection 4.4.2. For instance, the abstract "startMine()" method allows the DM agent to begin mining in response to a request from the task agent. It expects a list of results in the form of tree-structure to be returned so that it may return them to the user agent indicating completion.

Extendibility is illustrated in this scenario through the simple incorporation of a number of classifier generator algorithms (e.g. FOIL, TFPC, etc..) into the framework.

7.5 Experimentation and Analysis

To evaluate the classification scenario, in the context of MADM, a number of experiments were conducted on a sequence of datasets taken from the UCI machine learning data repository [18]. The datasets were pre-processed by data agents so that they were discretised/normalised into a binary valued format. The support threshold has been set to 0.2% and the confidence threshold to 80% that are required by the CARM-based algorithms.

Dataset	FOIL	TFPC	PRM	CPAR	IGDT	RDT	CMAR	CBA
connect4.D129.N67557.C3	67.64	65.94	68.55	52.32	44.13	79.76	64.29	71.64
adult.D97.N48842.C2	84.15	83.38	75.79	75.79	86.05	86.02	81.17	82.79
letRecog.D106.N20000.C26	51.04	35.75	55.53	55.10	69.39	91.79	33.61	37.94
anneal.D73.N898.C6	98.44	77.51	95.55	95.55	84.40	84.41	77.07	79.73
breast.D20.N699.C2	85.10	89.11	92.84	92.84	93.98	93.98	88.82	93.70
dematology.D49.N366.C6	75.96	63.93	74.86	76.50	74.31	96.17	74.30	75.41
flare.D39.N1389.C9	82.71	84.01	67.00	67.00	87.03	87.00	84.00	84.00
heart.D52.N303.C5	56.29	54.97	58.28	54.30	89.40	96.02	53.64	59.60
penDigits.D89.N10992.C10	85.90	87.03	82.46	82.77	88.57	99.18	86.74	90.92
pima.D38.N768.C2	72.66	54.97	73.44	73.70	82.55	80.99	15.88	73.70
soybean-large.D118.N683.C19	88.27	66.86	78.89	80.65	58.36	98.83	75.66	90.03
waveform.D101.N5000.C3	74.92	64.40	69.04	68.92	73.04	96.81	75.64	71.56

Table 7.1: Classification Results

The results of the experiments are presented in Table 7.1 and Table 7.2. Table 7.1 shows the accuracy results of all the applied algorithms for each dataset. Table 7.2 shows the “best” accuracy results for each dataset. Each row in the table represents a particular request for a best classifier and gives the name of the dataset, 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 Table 7.1 and Table 7.2 is that: D equals the number of attributes (after discretisation/normalisation), N the number of records and C the number of classes (although EMADS has no requirement for the adoption of this convention).

The results demonstrated firstly that EMADS can usefully be adapted to produces a “best” classifier from a selection of classifiers. Secondly, that the operation of EMADS is not significantly hindered by agent communication overheads because the generation time does not seem to be an issue, so further classifier generator (DM) agents could easily be added. The results also reinforce the often observed phenomenon that there is no single best classifier generator suited to

Dataset	Classifier	Accuracy	Gen. Time (sec)
connect4.D129.N67557.C3	RDT	79.76	157.65
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
dematology.D49.N366.C6	RDT	96.17	11.28
flare.D39.N1389.C9	IGDT	87.03	4.55
heart.D52.N303.C5	RDT	96.02	3.04
penDigits.D89.N10992.C10	RDT	99.18	13.77
pima.D38.N768.C2	IGDT	82.22	3.6
soybean-large.D118.N683.C19	RDT	98.83	13.22
waveform.D101.N5000.C3	RDT	96.81	11.97

Table 7.2: “Best” Accuracy Results

all kinds of dataset. Although inspection of Table 7.2 indicates that decision tree algorithms seen to work well.

7.6 Summary

This chapter considered the operation of MADM 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, dataset; which can then be applied to further unlabelled data. The scenario demonstrated that EMADS can usefully be adapted to produces a best classifier from a selection of classifiers. The scenario also illustrated how MADM may be applied to permit: experience and resource sharing, flexibility and extendibility, and the preservation of intellectual property rights.

Chapter 8

Conclusion

KDD, and by extension data mining, is an application area of increasing relevance as is evidenced by continuously growing interest in the subject. Much of this interest is related to the explosion of data available in network-accessible sites. However, these sites are characteristically heterogeneous and disparate in many ways. The research proposed in this thesis will open up many ways to make use effective of available information. Domains where the work may prove particularly useful are those where large amounts of geographically distributed data are available which continuously but irregularly collected and updated by different organisations, and the organisations have an interest in sharing the data. Some example application areas where data is available in this format include: the study of medical diagnostic/epidemiological information, market forecasting and other forms of commercial/financial data management, meteorological/oceanographic data analysis, and climate/weather forecasting.

Data mining (DM) refers to the process of extracting novel, useful, and understandable pieces of information (e.g., patterns, rules, regularities, constraints) from data in databases. One way of acquiring knowledge from databases is to apply various machine learning algorithms that search for patterns that may be exhibited in the data and compute descriptive representations.

DM techniques have been successfully applied in many problems in diverse areas with very good results. Although the field of machine learning has made substantial progress over the past few years, both empirically and theoretically, one of the continuing challenges is the development of learning techniques that effectively scale up to large and possibly physically distributed datasets. Most of the current generation of learning algorithms are computationally complex and typically require all data to be resident in main memory, which is clearly unrealistic for many genuine problems and databases. Furthermore, in certain situations, data may be inherently distributed and cannot easily be merged into a single database for a variety of reasons including security, fault tolerance, legal constraints and competitive reasons. In such cases, it may not be possible to examine all of the data at a central processing site to compute a single global result.

Similarly there may be a number of algorithms that may be applied to a single DM task with no obvious “best” algorithm. Therefore, there is a clear advantage to be gained from a software organisation that can (in an automated manner) locate, evaluate, consolidate and mine data from diverse sources and/or apply a diverse number of algorithms. The research described in this thesis has investigated the use of Multi-Agent Systems to provide this organisation.

The organisation of this concluding chapter is as follows. An overall summary of the research presented in this thesis is provided in Section 8.1. This is followed by a review of the main findings and contributions of the thesis in Section 8.2. Some further research directions are discussed in Section 8.3.

8.1 Summary

MADM, as proposed in this thesis, seeks to benefit from the possibilities offered by MASs to improve overall DM performance; or in other words, the management of intelligent agents with the capacity to perform data mining. The term “management” denotes the ability to dispatch such agents across participant sites, and also the potential to support the communications between these agents. MADM allows each participant to utilise their own local data and algorithms and, at the same time, benefit from the data and algorithms that are made available by the other participants world wide. The principal MADM advantages envisaged are those of experience and resource sharing, flexibility and extendibility, and (to an extent) protection of privacy and intellectual property rights.

This thesis presented a MADM vision, and investigated the issues that arise when using MAS for data mining activities. To assist the investigation EMADS was designed and implemented. EMADS is a generic and extendible multi-agent data mining framework that supports the creation and dispatching of DM agents across multiple sites and employs DM techniques to combine the separately and locally learned results into a higher level representation.

EMADS comprises a collection of data sources scattered 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. EMADS also provides simple mechanisms to incorporate new DM techniques and data sources. 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 (or all three). 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 DM algorithms/techniques. The expectation is that EMADS will "grow" as individual users contribute data and DM algorithms. The incorporation of data and DM techniques is facilitated by a system of wrappers which allows for easy extendibility of the system. Experience indicates that by using the wrappers provided with EMADS, existing data mining software can be very easily packaged to become an EMADS data mining agent. As discussed in Chapter 5, 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. This "simple extendibility" characteristic makes EMADS an extendible DM facility. This feature allows developers to employ their pre-implemented programs in the form of EMADS agents.

8.2 Main Findings and Contributions

In the course of the design and implementation of EMADS, several obstacles were identified. The implementation addressed many issues related to MADM, including: (i) dealing with heterogeneous platforms, (ii) dealing with multiple databases and (possibly) different schemata, (iii) the design and implementation of scalable protocols, and (iv) the efficient use of the results that are collected from remote sites. Other important problems, intrinsic to data mining that were also addressed included: (i) the ability of the system to adapt to environment changes (e.g., when data or targets change over time), and (ii) the capacity to extend it with new machine learning methods and data mining techniques. To be more specific, addressing these issues and referring back to Chapter 1, required the MADM framework to be:

- **Scalable.** To operate efficiently and without substantial or discernible reduction in performance as the number of data sites increases. The system was designed with asynchronous and distributed communication protocols that enable the participating agents to function independently and collaborate with each other as necessary, thus eliminating centralised control, congestion and synchronisation points.

- **Adaptive.** To accommodate new input patterns in anticipation of such patterns changing over time. This was achieved by facilitating the generation of new results based on the newly collected data and by extending meta mining techniques to incorporate and combine them with existing results.

- **Extendible.** To incorporate new DM techniques or datasets. This was attained via well-defined wrappers that enabled the decoupling of the framework from DM techniques and data.

- **Compatible.** To facilitate the use of databases with different schemata. This was attained through the definition and embedding of a specific Discretisation/Normalisation (DN) tool.

- **Efficient.** To make effective use of the system resources. This was achieved by adapting effective DM methods (i.e. the use of special pruning methods before and after the meta mining of results).

- **Effective.** To compute highly predictive and accurate results. This property is attained by allowing the evaluation of the generated results and the selection of the best results under various methods depending on the particular task and the imposed constraints (e.g. classifier accuracy).

- **Reusable.** To improve software development productivity and reduce implementation time.

- **Portable.** To execute on multiple platforms including Solaris, Windows and Linux. The framework was built using Java technology and algorithm-independent DM techniques.

A further important advantage of MADM is not to speed up the process, but to be able to tap larger amounts of data whose owners are not willing, or legally not allowed, to share data details. Privacy-preserving data mining allows the investigation of distributed data sources without having to merge the data. As an example, consider patient data. Generally speaking it is illegal for patients data to be "advertised", but using MADM, it may still be possible to detect commonalities in (say) cancer cases.

The research described in this thesis has demonstrated the above properties of MADM through the experimental implementation of EMADS and its application in a number of DM scenarios.

The 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 considered a data partition technique for parallel (and distributed) ARM that made use of vertical/horizontal partitioning techniques 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. In addition, the proposed horizontal segmentation part of the scenario demonstrated that MADM facilitates re-usability by reusing the previously developed meta ARM agents.

The use of EMADS was also illustrated using a classification scenario. The results demonstrated firstly that EMADS can usefully be adopted to produce 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 reinforced the often observed phenomenon that there is no single best classifier generator suited to all kinds of data.

The main findings of the research may be summarised as follows: (i) the protocols and communication mechanisms adopted are proven to be effective; (ii) the use of wrappers has achieved simple extendibility; (iii) the techniques used to support the sharing of recourse and expertises through the use of the facilitator approach are proven to be effective; (iv) the mechanism used for the protection of data privacy and intellectual property rights was proven to be effective; (v) scalability and efficiency were addressed efficiently; (vi) MADM approach can be viewed as an effective distributed and parallel environment; and (vii) flexibility that allows non-expert users to perform data mining is supported.

In conclusion, this thesis presented an overall picture of the advantages offered by MADM, an approach that advocates the integration of MAS and DM.

8.3 Future Work

The analysis and synthesis of the MADM challenges and issues clearly indicated the need for, and the promising potential of, the enhancement of DM and the creation of intelligent DM systems. More efforts are required to develop techniques, systems, and case studies from foundational, technological, and practical perspectives.

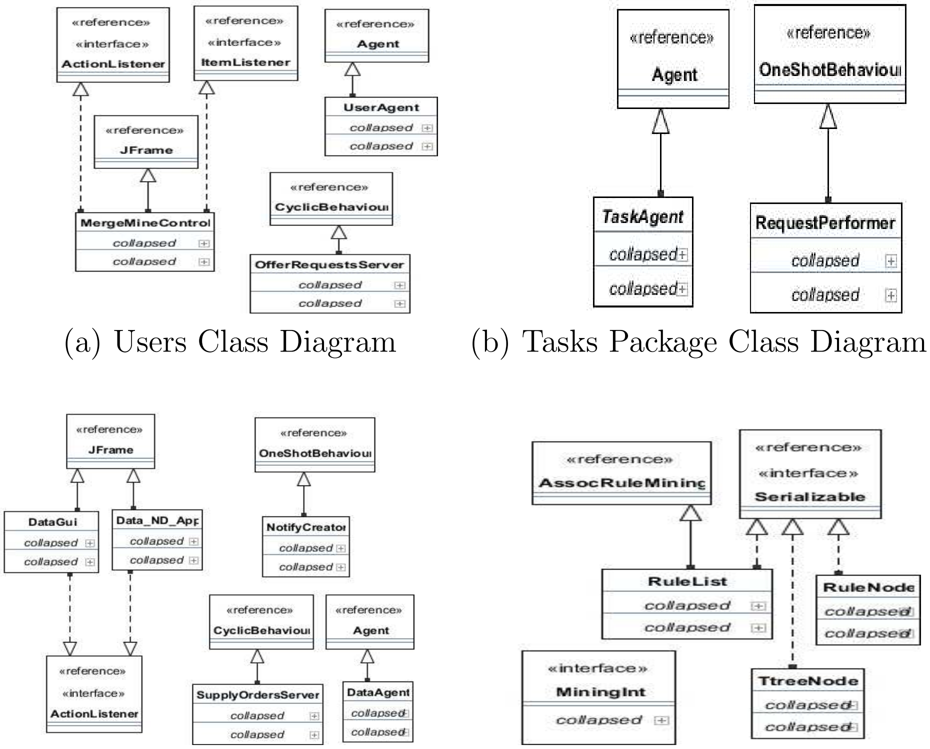
A foundation has thus been established for both data mining research and genuine application based MADM. It is acknowledged that the current functionality of EMADS is limited to classification and ARM, there is more work needed towards increasing the diversity of mining tasks that EMADS can address. Future extensions of EMADS can provide authentication capabilities, and fault tolerance. EMADS can also be extended with tools facilitating the data selection problem. The data selection problem refers to the pre-processing, transformation and projection of the available data to expressive and informative features, and is probably one of the hardest, but very important stages in the knowledge discovery process. The process depends on the particular data mining task and requires application domain knowledge. The current version of EMADS, assumes well-defined schemata and datasets. The datasets that were used in the scenarios presented in this thesis were generated and pre-processed in a separate off-line process before being used in EMADS (although a data normalisation/discretisation facility was incorporated into the data agent wrapper GUI). Introducing data pre-processing agents could solve the incompatible schema problem. Recall that working with heterogeneous databases has not been addressed in this thesis.

There are many other 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 entire EMADS system can also be made publicly available, to enhance its scope and diversity. It is hoped that once the system is live other interested data mining practitioners will be prepared to contribute algorithms and data.

Appendix A

EMADS Class Diagrams (UML representation)

A.1 Main EMADS Packages Class Diagrams



(a) Users Class Diagram (b) Tasks Package Class Diagram

(c) Data Package Class Diagram (d) Common Package Class Diagram

Figure A.1: Main Packages Class Diagrams

A.2 Meta ARM Application Class Diagram

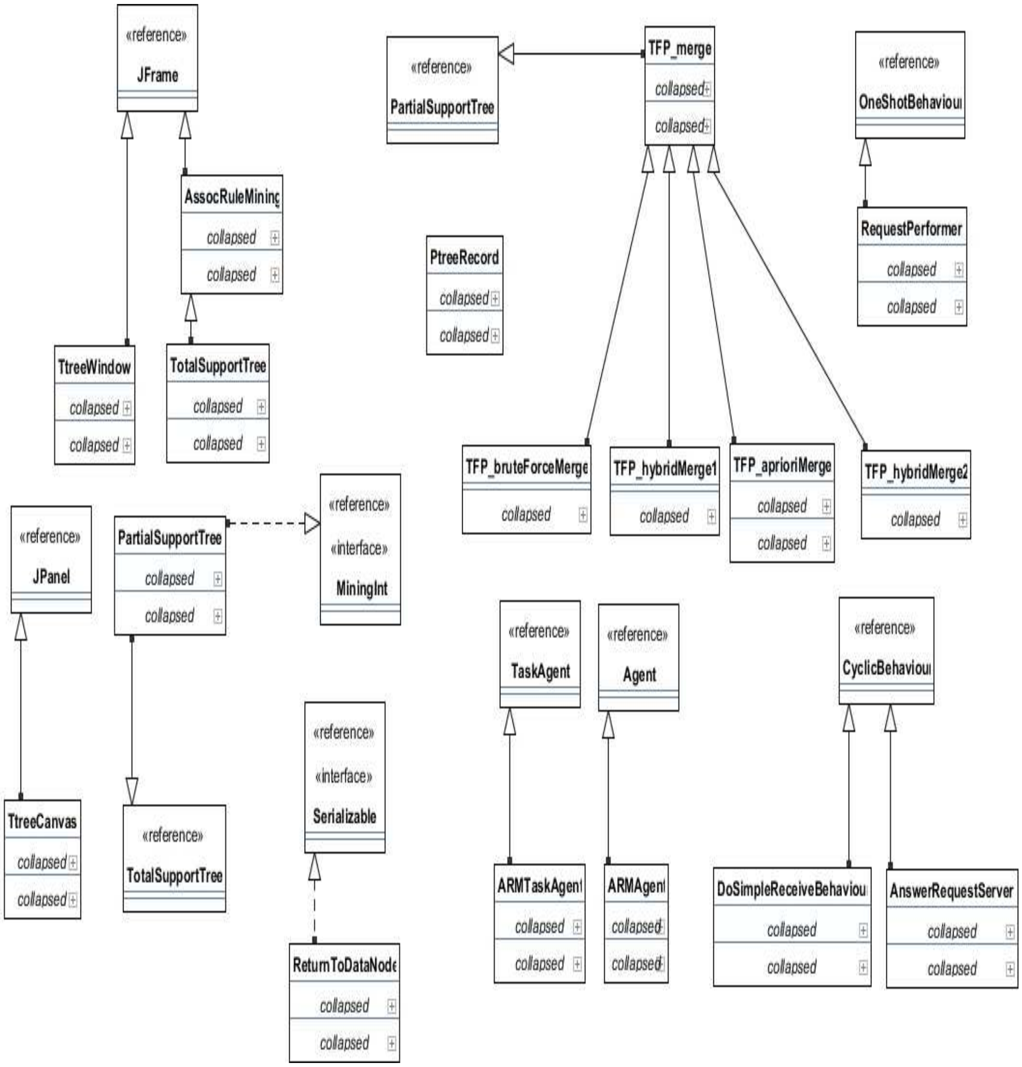


Figure A.2: Meta ARM Class Diagram

A.3 Parallel and Distributed ARM Application Class Diagram

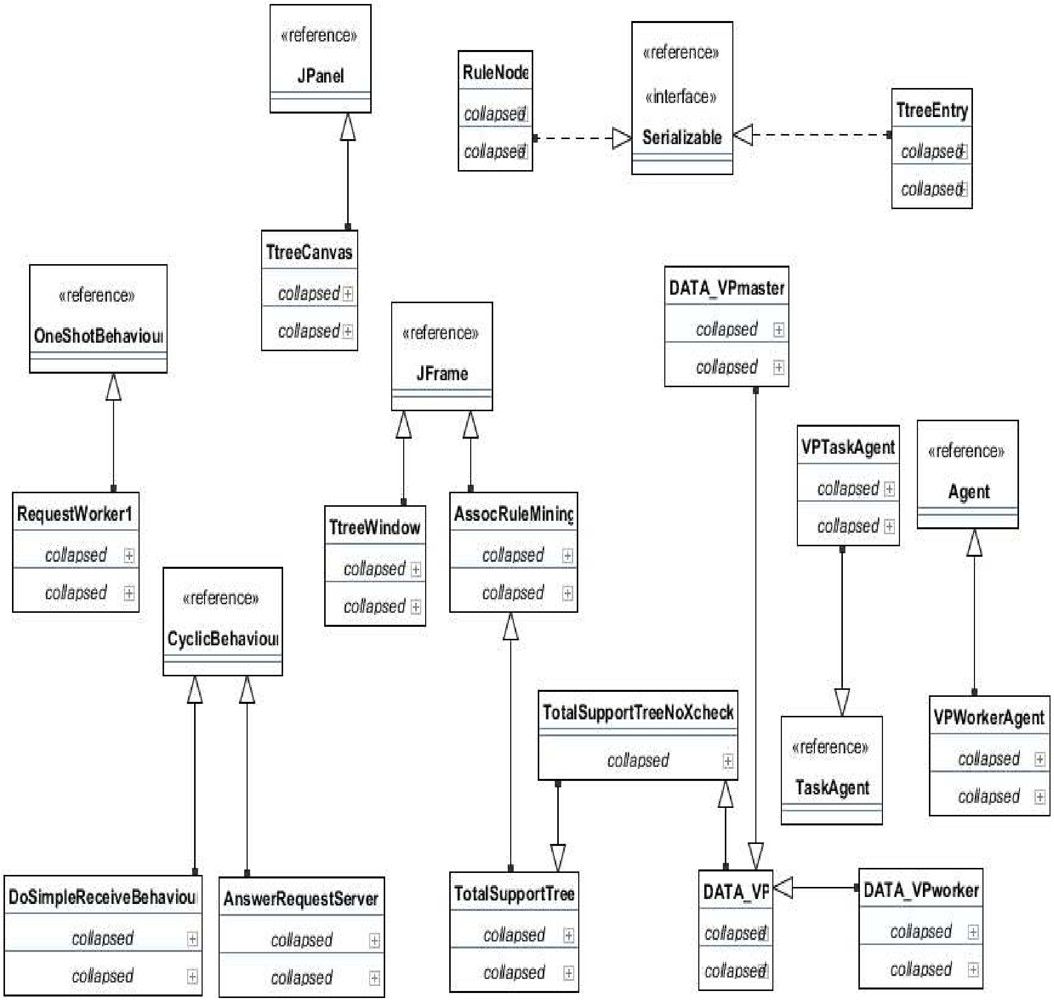


Figure A.3: Parallel/Distributed ARM Class Diagram

A.4 Classifier Generation Application Class Diagram

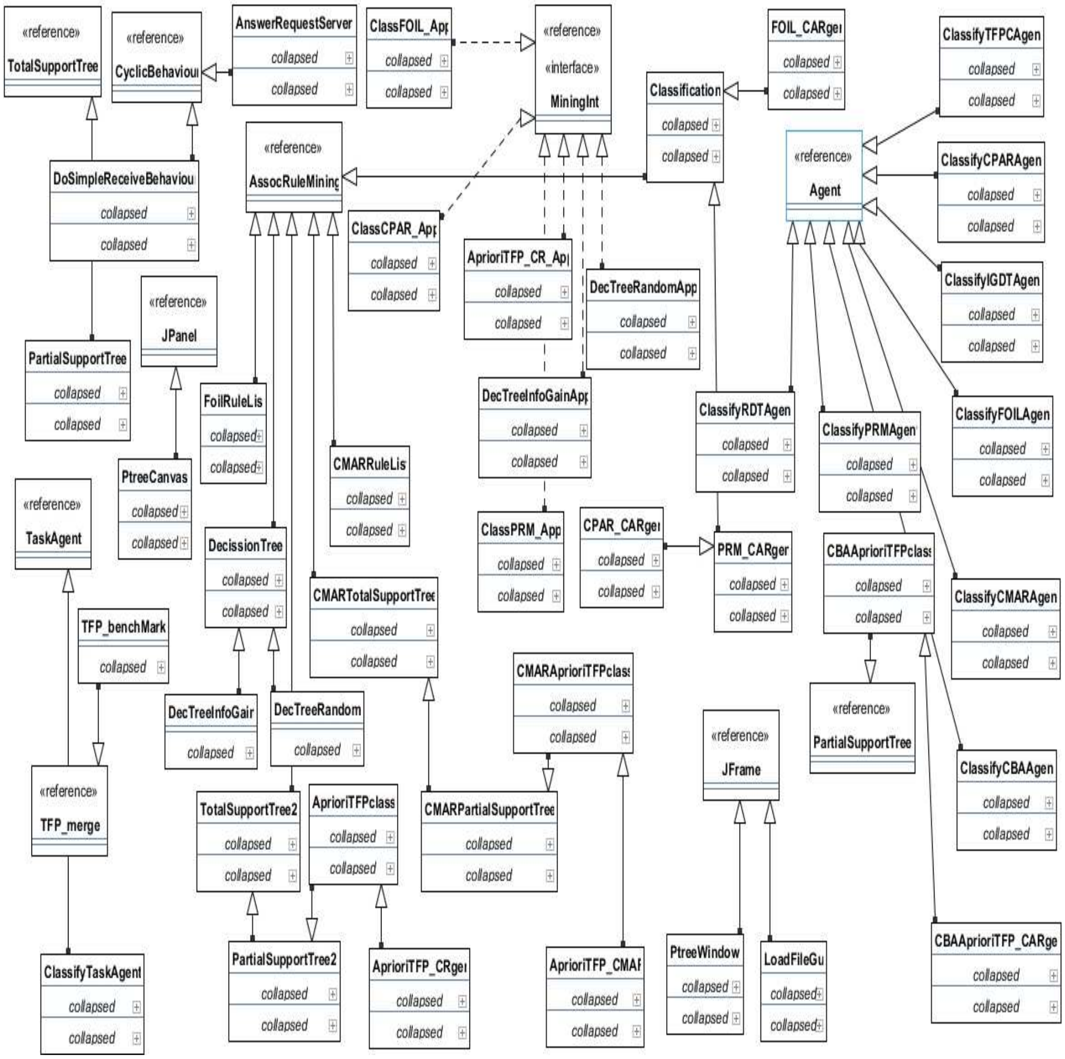


Figure A.4: Classifier Generation Class Diagram

Appendix B

EMADS Tutorial

B.1 Testing EMADS using Meta ARM

1. Go to `http://www.csc.liv.ac.uk/~ali/emads/`.

2. Download (emads.jar, jade.jar, jadeTools.jar, http.jar) files to your home directory.

3. On your home directory, start the main container:
   >> *java −classpath jade.jar jade.Boot* [*−host* “*main−container host−name*”][1] *−jade_domain_df_autocleanup true −gui.*

4. Start another container with EMADS data agent:
   >> *java −classpath jade.jar* : *emads.jar jade.Boot* [*−host* “*main − container host − name*”][1] *−container d*1 : *data.DataAgent*
   this will create a data agent and launch the data agent GUI.

5. Use the data agent GUI to load a binary dataset file suitable for ARM and then close the GUI.

6. Start another container with another EMADS data agent:
   >> *java −classpath jade.jar* : *emads.jar jade.Boot* [*−host* “*main − container host − name*”][1] *−container d*2 : *data.DataAgent.*

7. To add another containers with another EMADS data agents:
   run the same command, but use different agent names (i.e. d3, d4, ...).

[1] Use -host host-name when you dispatch agents from remote machines

8. Start another container with an EMADS user agent:
   >> java − classpath jade.jar : emads.jar jade.Boot [−host "main − container host − name"][1] *− container u1 : user.UserAgent.*

9. Input support, and confidence; using the user agent GUI.

10. From the main menu choose "Gen. Frequent Sets", click on one of the listed Meta ARM tasks and wait for results.

B.2 Testing EMADS using Data Vertical Partitioning

1. Go to `http://www.csc.liv.ac.uk/~ali/emads/`.

2. Download (emads.jar, jade.jar, jadeTools.jar, http.jar) files to your home directory.

3. On your home directory, start the main container:
   >> java − classpath jade.jar jade.Boot [−host "main − container host − name"][2] *− jade_domain_df_autocleanup true − gui.*

4. Start another container with EMADS data agent:
   >> java − classpath jade.jar : emads.jar jade.Boot [−host "main − container host − name"][2] *− container d1 : data.DataAgent*
   this will create a data agent and launch the data agent GUI.

5. Use the data agent GUI to load a binary dataset file suitable for ARM and then close the GUI.

6. Start another container with an EMADS user agent:
   >> java − classpath jade.jar : emads.jar jade.Boot [−host "main − container host − name"][2] *− container u1 : user.UserAgent.*

7. Input the number of partitions, support, and confidence; using the user agent GUI.

8. From the main menu choose "Vertical Partitioning", click on one "Gen. FIS with Vertical Part." and wait for results.

[2] Use -host host-name when you dispatch agents from remote machines

B.3 Testing EMADS using Data Horizontal Segmentation

1. Go to `http://www.csc.liv.ac.uk/~ali/emads/`.

2. Download (emads.jar, jade.jar, jadeTools.jar, http.jar) files to your home directory.

3. On your home directory, start the main container:
   >> java −classpath jade.jar jade.Boot [−host “main−container host−name”][3] *−jade_domain_df_autocleanup true −gui.*

4. Start another container with EMADS data agent:
   >> java −classpath jade.jar : emads.jar jade.Boot [−host “main − container host − name”][3] *−container d1 : data.DataAgent*
   this will create a data agent and launch the data agent GUI.

5. Use the data agent GUI to load a binary dataset file suitable for ARM and then close the GUI.

6. Start another container with an EMADS user agent:
   >> java −classpath jade.jar : emads.jar jade.Boot [−host “main − container host − name”][3] *−container u1 : user.UserAgent.*

7. Input number of partitions, support, and confidence; using the user agent GUI.

8. From the main menu choose “Horizontal Partitioning”. This will create a number of data agents, according the number of partitions, assigning each data partition to a data agent.

9. From the main menu choose “Gen. Frequent Sets”, click on one of the listed Meta ARM tasks and wait for results.

B.4 Testing EMADS using Best Classifier Generation

1. Go to `http://www.csc.liv.ac.uk/~ali/emads/`.

[3] Use -host host-name when you dispatch agents from remote machines

2. Download (emads.jar, jade.jar, jadeTools.jar, http.jar) files to your home directory.

3. On your home directory, start the main container:
   $>> java\ -classpath\ jade.jar\ jade.Boot\ [-host\ “main-container\ host-name”]$[4] $-jade\_domain\_df\_autocleanup\ true\ -gui.$

4. Start another container with EMADS classifier agent:
   $>> java\ -classpath\ jade.jar : emads.jar\ jade.Boot\ [-host\ “main-container\ host-name”]$[4] $-container\ FOIL : classifier\_app.ClassifyFOILAgent.$

5. Start another container with another EMADS classifier agent:
   $>> java\ -classpath\ jade.jar : emads.jar\ jade.Boot\ [-host\ “main-container\ host-name”]$[4] $-container\ TFPC : classifier\_app.ClassifyTFPCAgent.$

6. To add another containers with another EMADS classifier agents: Run the same command, but replace TFPC with CPAR, IGDT, RDT, or PRM (the other available classifiers).

7. Start another container with an EMADS user agent:
   $>> java\ -classpath\ jade.jar : emads.jar\ jade.Boot\ [-host\ “main-container\ host-name”]$[4] $-container\ u1 : user.UserAgent.$

8. Load data using the user agent GUI will create a data agent and launch the data agent GUI.

9. Use the data agent GUI to load a binary dataset file suitable for classification task the data file and then close the GUI.

10. Input support, confidence, and the number of classes; using the user agent GUI.

11. From the main menu choose classify, click on “Generate Best Classifier” and wait for results.

[4] Use -host host-name when you dispatch agents from remote machines

Bibliography

[1] R. Agrawal, T. Imielinski, and A. Swami. *Mining association rules between sets of items in large databases.* In Proceedings of the 1993 Association for Computer Machinery (ACM) Special Interest Group on Management of Data (SIGMOD) International Conference on Management of Data, pages (207-216), 1993.

[2] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger. *The Quest Data Mining System.* In Proceedings of the 2nd International Conference Knowledge Discovery and Data Mining, (KDD), 1996.

[3] R. Agrawal and G. Psaila. *Active Data Mining.* In Proceedings of the 1st International Conference Knowledge Discovery in Data Mining, AAAI, pages (3-8), 1995.

[4] R. Agrawal and J. Shafer. *Parallel mining of association rules.* In Proceedings of the IEEE Transactions on Knowledge and Data Engineering 8(6), pages (962-969), 1996.

[5] R. Agrawal and R. Srikant. *Fast algorithm for mining association rules.* In Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, pages (487-499), 1994.

[6] I. Akyildiz, W. Su, Y. Sankara, and E. Cayirci. *A survey on sensor networks.* In IEEE Communications Magazine, pages (102-114), 2002.

[7] K. A. Albashiri, F. Coenen, and P. Leng. *EMADS: An Extendible Multi-Agent Data Miner*, volume XXIII, pages (263-276). Research and Development in Intelligent Systems, AI, Springer, London, England, 2008.

[8] K. Ali, S. Manganaris, and R. Srikant. *Partial classification using association rules.* In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD, AAAI Press), Newport Beach, CA, USA, pages (115-118), 1997.

[9] W. Aref, M. lfeky, and A. Elmagarmid. *Incremental, Online, and Merge Mining of Partial Periodic Patterns in Time-Series Databases*, volume 16(3), pages (332-342). In Proceedings of the IEEE Transaction in Knowledge and Data Engineering, 2004.

[10] H. Baazaoui, S. Faiz, R. BenHamed, and H. BenGhezala. *A Framework for data mining based multi-agent: an application to spatial data.* In Proceedings of the 3rd World Enformatika Conference (WEC), Avril, Istanbu, Turkey, 2005.

[11] B. Babcock, S. Babuand M. Datar, R. Motwani, and J. Widom. *Models and Issues in Data Stream Systems.* In Proceedings of the 21th Association for Computer Machinery (ACM) Special Interest Group on Management of Data (SIGMOD) Symposium on Principles of Database Systems (PODS), 2002.

[12] 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 of the Conference on Supercomputing, ACM Press, pages (50-63), 1999.

[13] J. Balter, A. Labarre-Vila, D. Zibelin, and C. Garbay. *A Platform Integrating Knowledge and Data Management for EMG Studies.* In Proceedings of the Artificial Intelligence in Medicine (AIME), 2101, pages (417-420), 2001.

[14] J. Baumann, F. Hohl, K. Rothermel, and M. StraBer. *MOLE - Concepts of a Mobile Agent System.* In Proceedings of the World Wide Web, 1(3), pages (123-137), 1998.

[15] C. Baumer and T. Magedanz. *GrassHopper 2, an intelligent mobile agent platform written in 100% pure Java.* In Proceedings of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA), Springer LNAI 1699, Berlin, Germany, pages (19-32), 1999.

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

[17] F. Bellifemine, A. Poggi, and G. Rimassi. *JADE: A FIPA-Compliant agent framework*. In Proceedings the Practical Applications of Intelligent Agents and Multi-Agents, pages (97-108), 1999. http://www.jade.tilab.com.

[18] C. Blake and C. 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/.

[19] R. Bordini, L. Braubach, M. Dastani, A. Seghrouchni, J. Sanz, J. Leite, G. OHare, 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, pages (33-44), 2006.

[20] 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.

[21] J. Bota, A. Gmez-Skarmeta, M. Valds, and A. Metala. *A meta-learning architecture*. In Proceedings of the Computational Intelligence. Theory and Applications, 2206/2001, pages (688-698), 2001.

[22] L. Breiman. *Bagging Predictors*, volume 24(3), pages (123-140). In Machine Learning Archive, 1996.

[23] L. Cao, C. Luo, and C. Zhang. *Agent-Mining Interaction: An Emerging Area*. In Proceedings of the AIS-ADM, LNAI 4476, Springer - Verlag, Berlin, Germany, pages (60-73), 2007.

[24] B. Chandrasekaran and T. Johnson. *Generic tasks and task structures: History, critique and new directions*. In Proceedings of the Second Generation Expert Systems, Springer - Verlag, Berlin, Germany, pages (232-272), 1993.

[25] D. Cheung and Y. Xiao. *Effect of Data Distribution in Parallel Mining of Associations*. In Proceedings of the Data Mining and Knowledge Discovery 3(3), pages (291-314), 1999.

[26] K. Chmiel, D. Tomiak, M. Gawinecki, P. Karczmarek, M. Szymczak, and M. 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, Frameworks and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar), pages (49-56), 2004.

[27] G. Christopher and B. 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.

[28] F. Coenen and P. Leng. *Optimising Association Rule Algorithms Using Itemset Ordering.* In Proceedings of the AI Conference, Research and Development in Intelligent Systems XVIII, Springer, pages (53-66), 2001.

[29] F. Coenen, P. Leng, and S. Ahmed. *T-Trees, Vertical Partitioning, and Distributed Association Rule Mining.* In Proceedings of the IEEE International Conference on Data Mining (ICDM), Florida, eds. X Wu, A Tuzhilin and J Shavlik: IEEE Press, pages (513-516), 2003.

[30] F. Coenen, P. Leng, and S. Ahmed. *Data structure for association rule mining: T-trees and p-trees.* In Proceedings of IEEE Transactions on Knowledge and Data Engineering 16(6), pages (774-778), 2004.

[31] F. Coenen, P. Leng, and G. Goulbourne. *Tree Structures for Mining Association Rules*, volume 8(1), pages (25-51). In the Journal of Data Mining and Knowledge Discovery, 2004.

[32] F. Coenen, P. Leng, and L. Zhang. *Threshold Tuning for Improved Classification Association Rule Mining.* In Proceeding of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), LNAI 3158, Springer, pages (216-225), 2005.

[33] S. DeLoach. *Analysis and Design using MaSE and agentTool.* In Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS), 2001.

[34] S. DeLoach and M. Wood. *Developing Multiagent Systems with agentTool.* In Proceedings of the Intelligent Agents VII. Agent Theories, Architectures, and Languages - 7th International Workshop, ATAL, Boston, MA, USA, 2000.

[35] G. DiFatta and G. Fortino. *A customizable multi-agent system for distributed data mining.* In Proceedings of the 2007 ACM symposium on applied computing, pages (42-47), 2007.

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

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

[38] Y. Freund and R. Schapire. *Experiments with a New Boosting Algorithm.* In Proceedings of the The International Machine Learning Society (ICML), pages (148-156), 1996.

[39] Y. Fu, W. Ke, and J. Mostafa. *Automated Text Classification Using a Multi-Agent.* In Proceedings of the 5th ACM/IEEE-CS Joint Conference on Publication, pages (157-158), 2005.

[40] A. Garro and L. Palopoli. *An XML Multi-agent System for E-learning and Skill Management.* In Proceedings of the Agent Technologies, Infrastructures, Tools, and Applications for E-Services, pages (283-294), 2002.

[41] M. Genesereth and S. Ketchpel. *Software Agents.* In Proceedings of the Communications of the Association for Computer Machinery (ACM), 37(7), pages (48-53), 1994.

[42] C. Giannella, R. Bhargava, and R. Kargupta. *Multi-agent Systems and Distributed Data Mining.* In Proceedings of the Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2004, ISSU 3191, pages (1-15), 2004.

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

[44] V. Gorodetsky, O. Karsaeyv, and V. Samoilov. *Multi-agent technology for distributed data mining and classification.* In Proceedings of the International Conference on Intelligent Agent Technology, IEEE/WIC, pages (438-441), 2003.

[45] G. Goulbourne, F. Coenen, and P. Leng. *Algorithms for Computing Association Rules Using A Partial-Support Tree.* In Proceedings of the nineteenth Annual International Conference of the British Computer Society's Specialist Group on Knowledge Based Systems (ES99), Springer, London, England, pages (132-147), 1999.

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

[47] E. Han, G. Karypis, and V. Kumar. *Scalable Parallel Data Mining for Association Rules.* In Proceedings of the ACM(Association for Computer Machinery)- Special Interest Group on Management of Data (SIGMOD), International Conference on Management of Data, ACM Press, pages (277-288), 1997.

[48] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* ISBN: 1-55860-489-8, Morgan Kaufman Publishers, San Francisco, CA, (Second Edition), 2006.

[49] J. Han, J. Pei, and Y. Yin. *Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach.* In Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD) International Conference Management of Data(SIG MOD), pages (53-87), 2004.

[50] J. Han, J. Pei, and Y. Yiwen. *Mining Frequent Patterns Without Candidate Generation.* In Proceedings of the ACM (Association for Computer Machinery)- Special Interest Group on Management of Data (SIGMOD) International Conference on Management of Data, pages (1-12), 2000.

[51] D. Hand. *Construction and Assessment of Classification Rules.* ISBN: 0472965839, John Wiley and Sons, 1997.

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

[53] T. Hastie and R. Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* ISBN:978-0-387-84857-0, Second Edition, Springer - Verlag, Berlin, Germany, 2001.

[54] R. Ivancsy, F. Kovacs, and I. Vajk. *An Analysis of Association Rule Mining Algorithms.* In Proceedings of the Fourth International ICSC Symposium on Engineering of Intelligent Systems (EIS), Island of Madeira, Portugal, pages (774-778), 2004.

[55] JDM2. *Java Data Mining (JDM).* SUN's Java Community Process listed as JSR-73 and JSR-247 for JDM 2.0, 1995. https://datamining.dev.java.net/.

[56] H. Jeon, C. Petrie, and M. Cutkosky. *JATLite: A Java Agent Infrastructure with Message Routing.* In Proceedings of the IEEE Internet Computing (4)2, pages (87-96), 2000.

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

[58] T. Joachims. *Text categorization with support vector machines: Learning with many relevant features.* In Proceedings the 10th European Conference on Machine Learning (ECML), Springer Verlag, pages (80-91), 1998.

[59] M. Kamber, L. Winstone, G. Wan, S. Shanand, and H. Jiawei. *Generalization and Decision Tree Induction: Efficient Classification in Data Mining.* In Proceedings of the Seventh International Workshop on Research Issues in Data Engineering, pages (111-120), 1997.

[60] 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 the Second Society for Industrial and Applied Mathematics (SIAM) International Conference on Data Mining, 2004.

[61] H. Kargupta and P. Chan. *Advances in Distributed and Parallel Knowledge Discovery.* In Proceedings of the Advances in Distributed and Parallel Knowledge Discovery, MIT/AAAI press, Menlo Park, CA, 2000.

[62] H. Kargupta, I. Hamzaoglu, and B. Stafford. *Scalable, Distributed Data Mining Using an Agent Based Architecture.* In Proceedings of the Knowledge Discovery and Data Mining, AAAI Press, pages (211-214), 1997.

[63] H. Kargupta, B. Hoon, D. Hershberger, and E. Johnson. *Collective Data Mining: A New Perspective Toward Distributed Data Mining.* In Proceedings of the Advances in Distributed and Parallel Knowledge Discovery, MIT/AAAI Press, 1999.

[64] H. Kargupta and K. Sivakumar. *Existential Pleasures of Distributed Data Mining. In Data Mining: Next Generation Challenges and Future Directions.* In Proceedings of the Advances in Distributed and Parallel Knowledge Discovery, MIT/AAAI Press, Cambridge, MA, New York, 2004.

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

[66] M. Kaya and R. Alhajj. *Fuzzy OLAP association rules mining-based modular reinforcement learning approach for multiagent systems*, volume 35, pages (326-338). In Proceedings of the IEEE Transactions on Systems, Man and Cybernetics, Part B, Issue 2, 2005.

[67] B. Kegl and G. Lapalme. *Performance Evaluation of an Agent Based Distributed Data Mining System.* In Proceedings of AI 2005, LNAI 3501, Springer-Verlag Berlin, Heidelberg, Germany, pages (25-32), 2005.

[68] 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.* In Proceedings of the Intelligence in Services and Networks: Technology for Ubiquitous Telecom Services, Antwerp, Belgium, 1430, pages (115-126), 1998.

[69] M. Klusch, S. Lodi, and M. Gianluca. *The role of agents in distributed data mining: issues and benefits.* In Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT), pages (211-217), 2003.

[70] M. Klusch, S. Lodi, and G. Moro. *Agent-based Distributed Data Mining: The KDEC Scheme.* In Proceedings of the Intelligent Information Agents The AgentLink Perspective. 2586, Springer-Verlag, Berlin, Germany, 2003.

[71] J. Koh and S. Shieh. *An efficient approach to maintaining association rules based on adjusting FP-tree structures.* In Proceedings of the Database Systems for Advanced Applications (DASFAA) 2004, pages (417-424), 2004.

[72] C. Leung, Q. Khan, and T. Hoque. *CanTree: A tree structure for efficient incremental mining of Frequent Patterns.* In Proceedings of the IEEE International Conference on Data Mining (ICDM), pages (274-281), 2005.

[73] W. Li, J. Han, and J. Pei. *CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules.* In Proceedings of the IEEE International Conference on Data Mining (ICDM), pages (369-376), 2001.

[74] X. Li, Z-H. Deng, and S-W Tang. *A Fast Algorithm for Maintenance of Association Rules in Incremental Databases.* In Proceedings of the Advanced Data Mining and Applications (ADMA), Springer-Verlag LNAI 4093, pages (56-63), 2006.

[75] B. Liu, W. Hsu, and Y. Ma. *Integrating Classification and Association Rule Mining.* In Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI), New York, USA, pages (80-86), 1998.

[76] 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.

[77] P. Mariano, C. Pereira, L. Correira, R. Ribeiro, V. Abramov, N. Szirbik, J. Goossenaerts, T. Marwala, and P. Wilde. *Simulation of a trading multi-agent system*, volume 5, pages (3378-3384). In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Springer, London, England, 2001.

[78] D. Martin, A. Cheyer, and D. Moran. *The Open Agent Architecture: A Framework for Building Distributed Software Systems*, volume 13, pages (91-128). In Proceedings of the Applied Artificial Intelligence, 1998.

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

[80] J. Mayer, I. Melzer, and F. Schweiggert. *Lightweight Plug-In-Based Application Development.* In Lecture Notes in Computer Science Springer Berlin / Heidelberg, volume 2591, pages (87-102), 2003.

[81] S. McConnell and D. Skillicorn. *Building predictors from vertically distributed data.* In Proceedings of the 2004 conference of the Centre for Advanced Studies conference on Collaborative research 04-07. Markham, Ontario, Canada, pages (150-162), 2004.

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

[83] D. Michie, D. Spiegel halter, and C. Taylor. *Machine Learning, Neural and Statical Classification.* In Ellis Horwood Series in Artificial Intelligence, New York, 1994.

[84] 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 Proceedings of the 2nd International Workshop Mobile Agents (MA), Lecture Notes in Computer Science, Springer 1477, Stuttgart, Germany, pages (50-67), 1998.

[85] S. Mohan, E. Park, and Y. Han. *Association Rule-Based Data Mining Agents for Personalized Web Caching*, volume 02, pages (37-43). In Proceedings of the 29th Annual international Computer Software and Applications Conference, IEEE Computer Society, Washington, DC, 2005.

[86] P. Naur. *Revised Report on the Algorithmic Language ALGOL 60*, volume 3 No.5, pages (299-314). In Communications of the Association for Computer Machinery (ACM), 1960.

[87] H. Nwana, D. Ndumu, and L. Lee. *ZEUS: An advanced Tool-Kit for Engineering Distributed Multi-Agent Systems.* In Proceedings of the Practical Aspects of Declarative Languages, London, England, pages (377-391), 1998.

[88] P. O'Brien and R. Nicol. *FIPA - towards a standard for software agents*, volume 16, no.3, pages (51-59). BT Technology Journal, 1998.

[89] R. Pairceir, S. McClean, and B. Scotney. *Discovery of multi-level rules and exceptions from a distributed database.* In Proceedings of the sixth ACM Special Interest Group on Management of Data (SIGKDD) international conference on Knowledge discovery and data mining, Boston, Massachusetts, USA, pages (523-532), 2000.

[90] 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, pages (341-358), 2003.

[91] S. Parthasarathy, M. Zaki, and W. Li. *Memory Placement Techniques for Parallel Association Mining.* In Proceedings of the 4th International Conference on Knowledge Discovery in Databases (KDD), AAAI Press, pages (304-308), 1998.

[92] 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, pages (935-944), 2001.

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

[94] S. Poslad, P. Buckle, and R. Hadingham. *The FIPA-OS agent platform: Open Source for Open Standards.* published at PAAM2000, Chichester, England, 2000. http://sourceforge.net/projects/fipa-os/files/.

[95] A. Prodromides, P. Chan, and S. Stolfo. *Meta-Learning in Distributed Data Mining Systems: Issues and Approaches.* In Proceedings of the Advances

in Distributed and Parallel Knowledge Discovery. AAAI Press/The MIT Press, pages (81-114), 2000.

[96] F. Provost. *Distributed Data Mining: Scaling Up and Beyond.* In Proceedings of the Advances in Distributed and Parallel Knowledge Discovery, MIT/AAAI Press, Cambridge, MA, New York. pages (3-27), 1999.

[97] J. Quinlan. *Induction of decision trees.* In Proceedings of the Machine Learning 1(1), pages (81-106), 1986.

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

[99] J. Quinlan and R. Cameron-Jones. *FOIL: A Midterm Report.* In Proceedings of The European Conference on Machine Learning (ECML), Vienna, Austria, pages (3-20), 1993.

[100] I. Rudowsky. *Intelligent Agents*, volume 14, pages (275-290). In Proceedings of the Communications of the Association for Information Systems, Springer, London, England, 2004.

[101] R. Schollmeier. *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications.* IN Proceedings of the First International Conference on Peer-to-Peer Computing (P2P) IEEE, 2001.

[102] E. Shakshuki. *Methodology for Evaluating Agent Toolkits.* In Proceedings of the Knowledge-Based Intelligent Information and Engineering Systems, Springer, 4694/2010, pages(941-949), 2005.

[103] S. Sharples, C. Lindemann, and O. Waldhorst. *A Multi-Agent Architecture For Intelligent Building Sensing and Control.* In Proceedings of the International Sensor Review Journal, Yesha, MIT/AAAI Press, pages (3-27), 2000.

[104] 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, pages (250-257), 2004.

[105] T. Shintani and M. Kitsuregawa. *Hash Based Parallel Algorithms for Mining Association Rules.* In Proceedings of the 4th International Conference on Parallel and Distributed Information Systems, (PIDS), IEEE Computer Society Press, pages (19-30), 1996.

[106] Y. Shoham. *Agent-oriented programming*, volume 60(1), pages (51-92). In Proceedings of the Artificial Intelligence, Elsevier Science Publishers Ltd. Essex, England, 1993.

[107] J. Silva, M. Klusch, S. Lodi, and G. Moro. *Privacy-preserving agent-based distributed data clustering.* In Proceedings of the Web Intelligence and Agent Systems 4(2), pages (221-238), 2006.

[108] Munindar P. Singh. *Write Asynchronous, Run Synchronous.* In Proceedings of the IEEE Internet Computing, 3(2), pages (4-5), 1999.

[109] S. Sohn. *Meta Analysis of Classification Algorithms for Pattern Recognition.* In Proceedings of the IEEE Transactions on PAMI 2111, pages (1137-1144), 1999.

[110] R. Stallman. *EMACS the extensible, customizable self-documenting display editor*, volume 16(6), pages (147-156). In ACM (Association for Computer Machinery) Special Interest Group on Management of Data Notices, New York, NY, USA, 1981.

[111] S. Stolfo, A. Prodromidis, S. Tselepis, and W. Lee. *JAM: Java Agents for Meta-Learning over Distributed Databases.* In Proceedings of the International Conference Knowledge Discovery and Data Mining, pages (74-81), 1997.

[112] K. Sycara, A. Pannu, M. Williamson, and D. Zeng. *Distributed Intelligent Agents*, volume 11(6), pages (36-46). IEEE Expert, 1996.

[113] A. Symeonidis and P. Mitkas. *Agent Intelligence Through Data Mining*, volume XXVI, pages (0-206). In Proceedings of the Multi-agent Systems, Artificial Societies, and Simulated Organizations, 2006.

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

[115] M. Tamura and M. Kitsuregawa. *Dynamic Load Balancing for Parallel Association Rule Mining on Heterogeneous PC Cluster Systems.* In Proceedings of the 25th Very Large Data Bases (VLDB) Conference, Morgan Kaufman, pages (162-173), 1999.

[116] S. Thomas. *The PLACA Agent Programming Language.* In Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, Germany, pages (355-370), 1994.

[117] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. *An efficient algorithm for the incremental updation of association rules.* In Proceedings of the 3rd Association for Computer Machinery (ACM) Special Interest Group on Management of Data (SIG) International Conference on Knowledge Discovery and Data Mining, pages (263-266), 1997.

[118] K. Ting and I. Witten. *Stacking Bagged and Dagged Models.* In Proceedings in the fourteenth international Conference on Machine Learning, San Francisco, CA, pages (367-375), 1997.

[119] B. van Aardt and T. Marwala. *A Study in a Hybrid Centralised-Swarm Agent Community.* In Proceedings of the IEEE 3rd International Conference on Computational Cybernetics, Mauritius, pages (169-74), 2005.

[120] A. Veloso, W. Meira, B. deCarvalho, M. Possas, S. Parthasarathy, and M. Zaki. *Mining Frequent Itemsets in Evolving Databases.* In Proceedings of the Second Society for Industrial and Applied Mathematics (SIAM) International Conference on Data Mining (SDM), 2002.

[121] M. Vigder and J. Dean. *An architectural approach to building systems from COTS software components.* In Proceedings of the Centre for Advanced Studies Conference on Collaborative research, Toronto, Ontario, Canada, pages (131-143), 1997.

[122] R. Vilalta, G. Giraud-Carrier, P. Brazdil, and C. Soares. *Using Meta-Learning to Support Data Mining.* International Journal of Computer Science and Applications, IJCSA, 1(1), pages (31-45), 2004.

[123] T. Wagner. *An Agent-Oriented Approach to Industrial Automation Systems.* In Proceedings of the Agent Technologies, Infrastructures, Tools, and Applications for E-Services, pages (314-328), 2002.

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

[125] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* ISBN: 1-55860-552-5, Morgan Kaufman Publishers, San Fransisco, 1999.

[126] M. Wooldridge. *An Introduction to Multi-Agent Systems.* ISBN-10: 0470519460, John Wiley and Sons (Chichester, England), 2003.

[127] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, Z. Zhou, M. Steinbach, J. Hand, and D. Steinberg. *Top 10 Algorithms in Data Mining, Knowledge and Information Systems*, volume 14, pages (1-37). In Proceedings of the Knowledge and Information Systems, Springer-Verlag, London Limited, 2008.

[128] X. Yin and J. Han. *CPAR: Classification based on Predictive Association Rules.* In Proceedings the Second Society for Industrial and Applied Mathematics (SIAM) International Conference on Data Mining (SDM), San Fransisco, CA, pages (331-335), 2003.

[129] M. Zaki. *Parallel and Distributed Association Mining: A Survey*, volume 7(4), pages (14-25). In the IEEE Concurrency, 1999.

[130] M. Zaki. *Parallel and Distributed Association Mining: An Introduction.* In Proceedings of the Large-Scale Parallel Data Mining (Lecture Notes in Artificial Intelligence 1759), Springer-Verlag, Berlin, Germany, pages (1-23), 2000.

[131] Z. Zhang, C. Zhang, and S. Zhang. *An Agent-Based Hybrid Framework for Database Mining.* In Proceedings of the Applied Artificial Intelligence 17, pages (383-398), 2003.