Explorations of the BDI Multi-Agent support for the Knowledge Discovery in Databases Process

Alejandro Guerra-Hernández, Rosibelda Mondragón-Becerra, and Nicandro Cruz-Ramírez

Departamento de Inteligencia Artificial
Universidad Veracruzana
Facultad de Física e Inteligencia Artificial
Sebastián Camacho No. 5, Xalapa, Ver., México, 91000
{aguerra, rmondragon, ncruz}@uv.mx

Abstract. Knowledge Discovery in Databases (KDD) is the process of finding valid, novel, useful and understandable patterns in data, to verify hypothesis of the user or to describe/predict the future behavior of some event. The KDD process involves diverse techniques provided by tools like the Waikato Environment for Knowledge Analysis (WEKA), but usually without guidance. This work is an exploration of the use of Multi-Agent Systems (MAS) methodologies and tools to provide support in the KDD process while using such tools. The Belief-Desire-Intention (BDI) model of agency provides the right level of abstraction to approach this problem. First, the Prometheus methodology is used to analyse the KDD process in terms of MAS of BDI agents. Then, a MAS of decision trees inducers and Bayesian networks builders, that compete to generate the "best" hypothesis for a given KDD problem, is implemented. The main result of this exploration is a framework where it is possible to implement AgentSpeak(L) agents that use primitive actions of WEKA to form intentions for solving problems in the KDD process. Extensions in terms of the number of agents and their capabilities are easy to implement in this framework.

1 Introduction

The Knowledge Discovery in Databases (KDD) process [13] consists in selecting, preprocessing and transforming a data set obtained from several heterogeneous sources such as databases, plain files, data warehouses, etc., in order to facilitate the application of data mining algorithms that obtain the hidden patterns in this dataset. Subsequently, these patterns are interpreted and evaluated to select those that represent useful and novel knowledge [9].

There are several algorithms to carry out each one of the tasks of this process, implemented in tools as Clementine [34], DBMiner [20], Waikato Environment for Knowledge Analysis (WEKA) [36], among others. The difficulty arises when a neophyte has to choose the suitable algorithms for a given problem, since this decision depends on the nature of data, their representation, the mining task,

Advances in Computer Science and Artificial Intelligence.
Research in Computing Science 39, 2008, pp. 221-238

Received 21/06/08
Accepted 01/08/08
Final version 11/09/08

among other factors. Therefore, KDD is a complex process, that requires well trained users in a variety of disciplines including machine learning, statistics and domain knowledge. But even in this case, some parts of the KDD process can be simply tedious.

Multi-Agent Systems (MAS) have been proposed as a solution to the problems mentioned above. MAS are composed by agents specialized in data mining, which sometimes are distributed in a network [32, 23, 22, 3]. However, we observe that these approaches, often does not provide the level of abstraction required to reason about the KDD process and their participants. This is due to a weak notion of agency, which focuses on basic properties of agents as autonomy, reactivity, pro-activity and social ability [37]. The strong notion of agency conceptualizes and implements the agents as intentional systems. The Belief-Desire-Intention (BDI) model of agency [15, 33] is the best known of these approaches. In this model the agents perform practical reasoning using plans to form intentions to satisfy their desires. The folk-psychology language used in the model, is argued to offer a more effective communication among the participants in a KDD process in order to analyse it to implement support using BDI agents.

Our explorations for such a support are as follows: In order to design a MAS of rational agents that help the human experts in the KDD process, this one is analysed as performed by a MAS of human experts, following the *Prometheus* methodology [30]. The *Prometheus Design Tool* was used to get assistance for the design process and to generate a specification of the MAS easy to be implemented. As a result of the analysis we decided to implement a BDI MAS of six agents: a coordinator that receives requests to mine a given data set; a preprocessing specialist; and agents capable to execute ID3, C4.5, Naive Bayes, and TAN algorithms. These last four agents compete to produce the "best hypothesis" for a given problem, and cooperate with the preprocessing agent if necessary. The MAS was implemented in Jason [4], an interpreter of the BDI agent oriented programming language AgentSpeak(L) [33]. The idea was that these agents were capable of executing actions in WEKA [36], as part of their plans. We chose Jason because it is implemented in Java, as well as WEKA, so that the implementation of the intended agents seemed easier in this way.

Finally, the MAS was tested with different data sets, in order to know if such agent competency has sense. The results show that there is not such a thing as the "best method" for all the data sets, and that the exploration automatised by the MAS could be useful. But the main result is that we obtained a framework where it is possible to easily define new agents in the system and to improve the capabilities of the existent ones, i.e., extending their plan library or their beliefs.

The rest of this paper is organized as follows: Section two presents the antecedents, mainly the KDD process and the BDI agents as defined in AgentSpeak(L); section three introduces the tools and methods that were used to implement the MAS; section four comments some aspects of the design and implementation of the KDD process like a BDI MAS; section five reports the results of our experiments; and finally, section six presents the conclusions and discusses future work.

2 Antecedents

In this section, we describe the KDD process, as well as AgentSpeak(L), an agent oriented programming language under the BDI model of agency.

2.1 The KDD Process

KDD refers the non-trivial process of identifying valid, novel, potentially useful and understandable patterns in data [11, 12]. Figure 1 (adapted from Fayyad et al. [11]) shows that the KDD process has an iterative and interactive nature. Results obtained in the process are enhanced incrementally, possibly reconsidering previous decisions in the process [11, 12]. The steps of the KDD process [20, 11, 12] include:

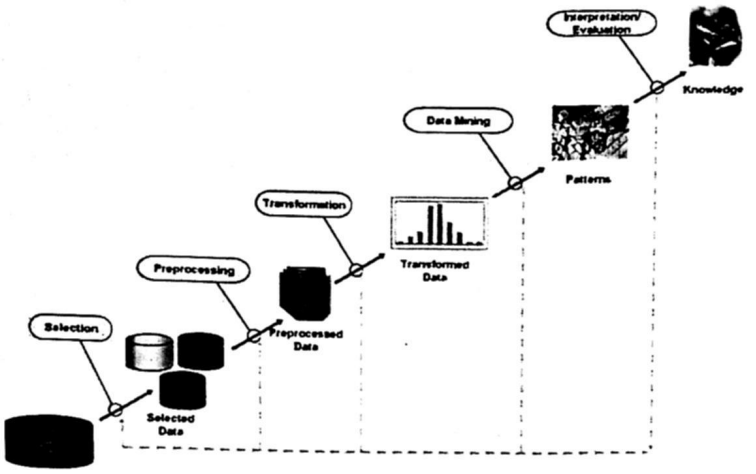

Fig. 1. Steps in the KDD process

1. Understanding the domain of the application and a background knowledge, as well as identifying the target of KDD process from the point of view of the user.
2. Selecting a subset of variables or a sample of data to create a set of target data, in which the discoveries will take place.
3. Cleaning and preprocessing the set of target data, e.g., removing noise if needed, dealing with missing data, etc.
4. Reducing and projecting data, through the selection of examples and attributes that are important for the target of the KDD process, or using dimensionality reduction or transformation methods to reduce the number

of variables under consideration or to find invariant representations for the data.
5. Selecting a data mining procedure, e.g., classification, regression, summarization, clustering.
6. Selecting some data mining algorithms and techniques to search patterns in the reduced data. The data mining expert decides which models or parameters are most appropriate.
7. Searching for patterns of interest (data mining) in a particular representational form as classification trees, rules, regression, clustering, etc.
8. Interpreting and evaluating the mined patterns, possibly reconsidering some of the steps $1 \ldots 7$.
9. Consolidating the discovered knowledge through its incorporation within another system or simply documenting and sending it to the interested participants in the process.

2.2 AgentSpeak(L) and BDI agents

BDI agents are intentional systems that continuously perceive their enviroment and take actions to modify it, based on their mental attitudes: beliefs, desires and intentions [15, 33, 8]. Beliefs represent the informational state of the agent. Desires, or goals, represent states that the agent would like to accomplish or bring about, considering its internal or external stimuli. Intentions represent the deliberative state of the agent, e.g., its commitment to some courses of action to accomplish its desires. These courses of action are built from plans in a plan library, e.g., the procedural knowledge of the agent. An event queue is usually used to process perception.

AgentSpeak(L) [33] is the language that was chosen to implement the BDI agents in this work, because it provides an abstract and elegant framework to program such agents [16]. The syntax and semantics of AgentSpeak(L) have been defined formally by means of a grammar and a operational semantics based on a transition system.

The grammar of AgentSpeak(L) as defined for its interpreter Jason [5], is shown in table 1. As usual, an agent ag is formed by a set of plans ps and beliefs bs. Each belief $b_i \in bs$ is a ground first-order term. Each plan $p \in ps$ has the form *trigger event* : *context* $\leftarrow$ *body*. A trigger event can be any update (addition or deletion) of beliefs (at) or goals (g).The context of a plan is an atom, a negation of an atom or a conjunction of them. A non empty plan body is a sequence of actions (a), goals, or belief updates. $\top$ denotes empty elements, e.g., plan bodies, contexts, intentions. Atoms (at) can be labelled with sources. Two kinds of goals are defined, achieve goals (!) and test goals (?).

The operational semantics [5] of the language, is given by a set of rules that define a transition system between configurations $\langle ag, C, M, T, s \rangle$, where:

- ag is an agent program formed by a set of beliefs bs and plans ps.
- An agent circumstance C is a tuple $\langle I, E, A \rangle$, where: I is a set of intentions $\{i, i', \ldots\}$, each $i \in I$ is a stack of partially instantiated plans $p \in ps$; E is a

Table 1. Grammar of *AgentSpeak(L)* [5]

ag	$::= bs \quad ps$	at	$::= P(t_1,\ldots,t_n) \quad (n \geq 0)$
bs	$::= b_1 \ldots b_n \quad (n \geq 0)$		$\mid P(t_1,\ldots,t_n)[s_1,\ldots,s_m] \quad (n \geq 0, m \geq 0)$
ps	$::= p_1 \ldots p_n \quad (n \geq 1)$	s	$::= \texttt{percept} \mid \texttt{self} \mid id$
p	$::= te : ct \leftarrow h$	a	$::= A(t_1,\ldots,t_n) \quad (n \geq 0)$
te	$::= +at \mid -at \mid +g \mid -g$	g	$::= !at \mid ?at$
ct	$::= ct_1 \mid \top$	u	$::= +b \mid -b$
ct_1	$::= at \mid \neg at \mid ct_1 \wedge ct_1$		
h	$::= h_1; \top \mid \top$		
h_1	$::= a \mid g \mid u \mid h_1; h_1$		

set of events $\{(te,i),(te',i'),\ldots\}$, each te is a trigger event and each i is an intention (internal events) or the empty intention $\top$ (external events); and A is a set of actions to be performed in the environment.

- M is a tuple $\langle In, Out, SI\rangle$ working as a mailbox, where: In is the mailbox of the agent; Out is a list of messages to be delivered by the agent; SI is a register of suspended intentions (intentions that wait for an answer message).
- T is a tuple $\langle R, Ap, \iota, \epsilon, \rho\rangle$ that registers temporary information as follows: R is the set of relevant plans for a given event; Ap is the set of applicable plans (the subset of applicable plans which contexts are believed true); ι, ϵ, and ρ register the current intention, event and applicable plan along one cycle of execution.
- The label s indicates the current step in the reasoning cycle of the agent.

Figure 2 shows the interpreter for *AgentSpeak(L)* as a transition system. The operational semantics rules [5] define the transitions. Because of space limitations, table 2 shows only some these rules.

3 Methods

In this section, the methodology and tools used to design and implement the MAS in this work, are introduced. First, the Prometheus [30] methodology for designing MAS, and its associated software PDT, are introduced; then the interpreter of AgentSpeak(L), Jason [27, 6, 28, 1, 35] and the KDD tool, WEKA [36], are briefly described.

3.1 The Prometheus Methodology and its PDT Tool

Prometheus [30] is a methodology for developing intelligent agents and MAS. It covers all the phases of development of a system – specification, design, implementation and testing/debugging. Prometheus consists of three main phases:

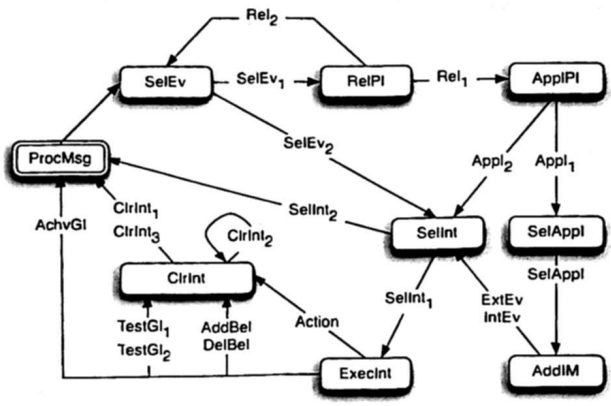

Fig. 2. The interpreter for *AgentSpeak*(L) as a transition system.

- **System Specification.** First, the goals and subgoals of the system are identified. Also, the actors (persons or roles that interact with the system) and their interactions with the system in form of perceptions and actions, are specified. Then, some scenarios are created for each actor. These scenarios show the operation of the system and consist of a series of steps including: goals, other scenarios, perceptions and actions. When the goals and scenarios are specified completely, the similar goals are grouped to form roles. Particular perceptions and actions are assigned to these roles. Finally, each step in the scenarios is assigned to their role and the data requirements for these scenarios are identified.
- **Architectural Design.** Based on the definitions generated in the previous phase, it is possible now to determine what kind of agents will be included in the system, as well as the interactions that will take place among them. To achieve this, several mechanisms are proposed, such as data coupling diagrams and agent acquaintance diagrams. In addition, in this phase is created the system overview diagram, that shows the structure of the system.
- **Detailed Design.** The internal details of each agent are designed and it is specified how the agents will carry out their jobs. Each agent is refined in terms of its capacities, internal events, plans, and data structures.

Additionally, the open software PDT [29] gives support to the Prometheus methodology. This tool provides: a graphic interface that allows to develop the definitions (diagrams) obtained through the Prometheus methodology and assists in the maintenance of a sound design because it provides verification between the diagrams; automatic spread of design elements when it is possible and appropriate; and assistance in the search for names.

Table 2. Some rules of the operational semantics of AgentSpeak(L).

(SelEv$_1$)	$\frac{S_E(C_E)=\langle te,i\rangle}{\langle ag,C,M,T,SelEv\rangle \longrightarrow \langle ag,C',M,T',RelPl\rangle}$	s.t. $C'_E = C_E \setminus \{\langle te,i\rangle\}$ $T'_\iota = \langle te,i\rangle$
(Rel$_1$)	$\frac{T_\iota=\langle te,i\rangle, RelPlans(ag_{ps},te)\neq\{\}}{\langle ag,C,M,T,RelPl\rangle \longrightarrow \langle ag,C,M,T',AppPl\rangle}$	s.t. $T'_R = RelPlans(ag_{ps},te)$
(Rel$_2$)	$\frac{RelPlans(ps,te)=\{\}}{\langle ag,C,M,T,RelPl\rangle \longrightarrow \langle ag,C,M,T,SelEv\rangle}$	
(Appl$_1$)	$\frac{ApplPlans(ag_{bs},T_R)\neq\{\}}{\langle ag,C,M,T,ApplPl\rangle \longrightarrow \langle ag,C,M,T',SelAppl\rangle}$	s.t. $T'_{Ap} = AppPlans(ag_{bs},T_R)$
(SelAppl)	$\frac{S_O(T_{Ap})=(p,\theta)}{\langle ag,C,M,T,SelAppl\rangle \longrightarrow \langle ag,C,M,T',AddIM\rangle}$	s.t. $T'_\rho = (p,\theta)$
(ExtEv)	$\frac{T_\iota=\langle te,\top\rangle, T_\rho=(p,\theta)}{\langle ag,C,M,T,AddIM\rangle \longrightarrow \langle ag,C',M,T,SelInt\rangle}$	s.t. $C'_I = C_I \cup \{[p\theta]\}$
(SelInt$_1$)	$\frac{C_I\neq\{\}, S_T(C_I)=i}{\langle ag,C,M,T,SelInt\rangle \longrightarrow \langle ag,C,M,T',ExecInt\rangle}$	s.t. $T'_\iota = i$
(SelInt$_2$)	$\frac{C_I=\{\ \}}{\langle ag,C,M,T,SelInt\rangle \longrightarrow \langle ag,C,M,T,ProcMsg\rangle}$	
(AchvGl)	$\frac{T_\iota=i[head\leftarrow !at;h]}{\langle ag,C,M,T,ExecInt\rangle \longrightarrow \langle ag,C',M,T,ProcMsg\rangle}$	s.t. $C'_E = C_E \cup \{\langle +!at,T_\iota\rangle\}$ $C'_I = C_I \setminus \{T_\iota\}$
(ClrInt$_1$)	$\frac{T_\iota=[head\leftarrow \top]}{\langle ag,C,M,T,ClrInt\rangle \longrightarrow \langle ag,C',M,T,ProcMsg\rangle}$	s.t. $C'_I = C_I \setminus \{T_\iota\}$
(ClrInt$_2$)	$\frac{T_\iota=i[head\leftarrow \top]}{\langle ag,C,M,T,ClrInt\rangle \longrightarrow \langle ag,C',M,T,ClrInt\rangle}$	s.t. $C'_I = (C_I \setminus \{T_\iota\}) \cup$ $\{k[(head' \leftarrow h)\theta]\}$ if $i = k[head' \leftarrow g;h]$ and $g\theta = TrEv(head)$
(ClrInt$_3$)	$\frac{T_\iota\neq[head\leftarrow \top] \wedge T_\iota\neq i[head\leftarrow \top]}{\langle ag,C,M,T,ClrInt\rangle \longrightarrow \langle ag,C,M,T,ProcMsg\rangle}$	

3.2 Jason

Jason is an interpreter for an extended version of the AgentSpeak(L) agent programming language (see section 2.2). This interpreter is written in Java and implements the operational semantics of AgentSpeak(L) [27, 6] and its extensions [28, 1, 35]. The integrated develop enviroment of Jason provides a graphical interface, that allows to edit the configuration file of a MAS and the code of the agents written in AgentSpeak(L). Through of this environment is possible to control the execution of the MAS and distribute the agents over a network in a simple way. Another tool that comes with this environment is a "mind inspector" that allows observing the internal state of the agents in run time. Jason also includes: speech acts based on KQML for communication between agents; annotations of the plans for using selection functions based on decision theory;

selection functions configurable in Java; and mechanisms of extension and use of legacy code, by means of the "internal actions" defined by the user.

3.3 WEKA

WEKA [36] is a tool implemented in Java to perform experiments in a KDD process. It provides methods for preprocessing data, e.g., replacing the missing values and performing discretization on data sets; data mining algorithms, e.g., ID3, C4.5, NB, and TAN; and pattern evaluation algorithms as the stratified cross-validation method. Other algorithms to carry out each one of the tasks of data mining, but we have focused in the methods mentioned above due to our research lines. In addition, WEKA has tools for visualizing and preprocessing data [14]. The input of all algorithms takes the form of a relational table, that can be read from a file or generated through of a database query. The file can be in *csv* format or *arff* format, which is the native format of WEKA.

4 Explorations

This section describes our explorations for the BDI MAS support for the KDD process, from the design to the implementation of the system. Later, some results on the use of the system are reported.

4.1 Design

Figure 3 shows the overview diagram for the MAS implemented for this paper. The *Coordinator* agent perceives the requests for learning from the users of the system as well as the databases associated to the requests; and their format. If a database is in *xls* or *csv* formats, she converts it into *arff* format; in any another case, she asks the user for a database in one of the mentioned formats. Then, the Coordinator asks the *Preprocessing* agent to review the database. If it is not nominal, the former tells the user that the target must be of this kind, given the nature of the learning algorithms used by the agents; otherwise, the Coordinator prints "Class is nominal" and sends a preprocessing require to the Preprocessing agent. This agent replaces the missing values (with the mode when the attributes are nominal and the mean when these ones are numerical) and discretizes the database (supervised and non-supervised). Once Preprocessing informs to Coordinator that the database has been preprocessed, the latter sends learning requests to the *ID3*, *C4.5*, *NB* and *TAN* agents. These ones learn their respective models for both kinds of discretization of data, and inform the Coordinator of their results. The Coordinator selects and reports the winning model given some criteria, e.g., the most accurate model, the fastest result obtained, etc.

To obtain the system overview diagram is necessary to identify the goals and subgoals of the system in a goal overview diagram (System Specification phase). For example, the figure 4 shows the goal and its subgoals for preprocessing the

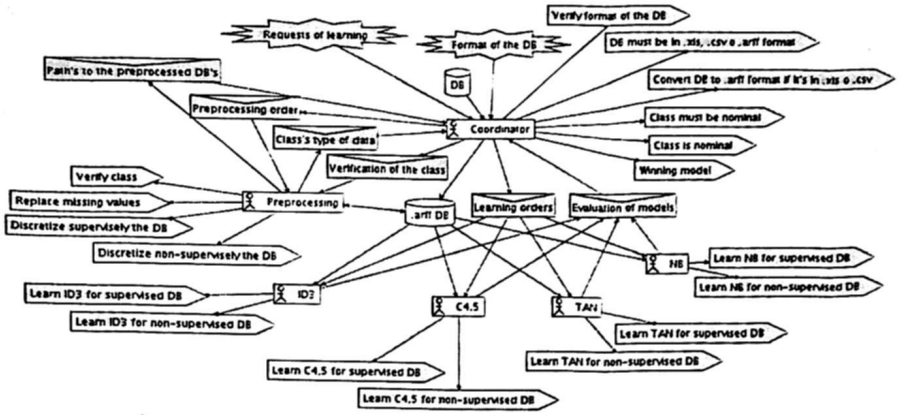

Fig. 3. Overview Diagram of our Multi-Agent System.

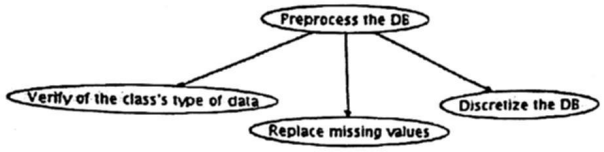

Fig. 4. Goal and its subgoals preprocess the database.

database: one of them verifies what kind the class is, another one replaces the missing values and the last one discretizes this database.

After that, the different possible scenarios for the system were defined in a scenario diagram (System Specification phase). For example, the figure 5 shows the scenario that will take place when the database has to be discretized supervisely, e.g., when the process of discretization takes into account the value of the class to create the different categories of data. It is worth mentioning that the discretization is based on the Minimum Description Length method (MDL) [10].

The roles of the system were created once the goals and scenarios were defined. These roles are formed by clustering similar goals, perceptions and actions in a system role diagram (System Specification phase). For example, the figure 6 shows the "Preprocessing of the DB" role. Observing the scenario shown in figure 5, we can see that both steps of the scenario are associated with this role.

The data coupling diagram and the agent-role grouping diagram (Architectural Design phase) were useful to identify the types of the agents in the MAS. Figure 7 shows the first of these diagrams, with the roles of the system and the data identified in the scenarios. Figure 8 shows the agent-role grouping di-

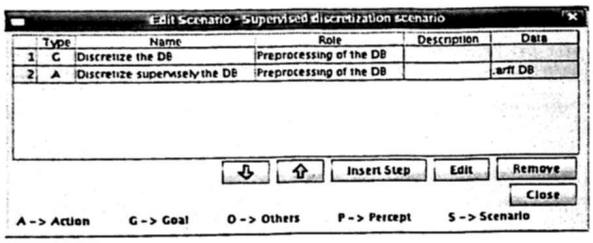

Fig. 5. Supervised discretization scenario.

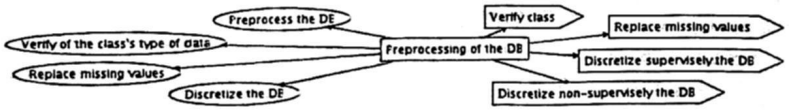

Fig. 6. Preprocessing of the DB role.

agram, which shows the roles assigned for each agent. Thus, the Coordinator is responsible for managing the input/output of the system and the database, etc.

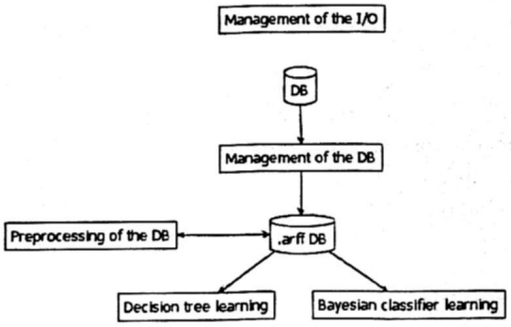

Fig. 7. Data Coupling Diagram.

In the last phase of this process (Detailed Design phase), each agent is refined in terms of its capacities, internal events, plans and structures of data. For example, the figure 9 shows the Preprocessing agent overview diagram, including two plans: "Verify the class" and "Preprocess the DB". The first plan is executed when a request to verify the target class is perceived, then it executes the action that verifies the class and finally it sends the type of data of this target class.

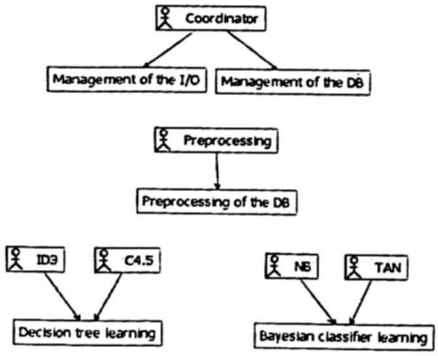

Fig. 8. Agent-Role Grouping Diagram.

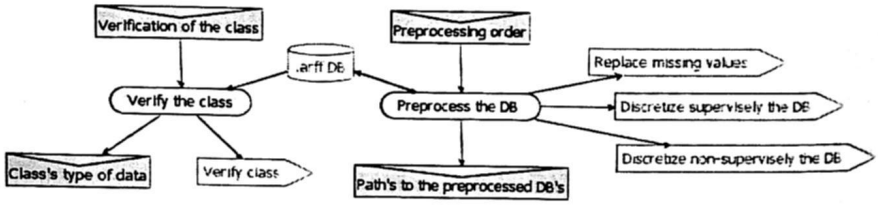

Fig. 9. Preprocessing-Agent Overview Diagram.

4.2 Implementation

As mentioned, the MAS was implemented in AgentSpeak(L)'s interpreter Jason v.1.0 (section 3.2). The development was carried out in a laptop with the following characteristics: Fedora Core 5 as operative system, Intel(R) Pentium(R) M 1.70 GHz processor, 1 Gb in RAM, 80 Gb in Hard Drive.

According to the design described in the previous section, the MAS consists of six agents: *Coordinator*, *Preprocessing*, *ID3*, *C4.5*, *NB* y *TAN*. The features of each one of these agents are shown in the table 3. The code in table 4 shows the initial beliefs and some plans of the Coordinator agent.

Table 3. Features of the agents.

Coordinator	**Preprocessing**	**ID3, C4.5, NB and TAN**
Manage I/O	Replace missing values	Learn data mining models
Manage the database	Discretize the database	

Table 4. A fragment of the code for the Coordinator agent

```
1  //Coordinator in project metaclasif.mas2j
2
3  //BELIEFS
4  start. //Start the execution of the MAS
5  file("./DATABASES/Iris.csv").
6
7  //PLANS
8  @pi
9  +start: true
10    <- .print("Coordinator agent, who manages the MAS...");
11       ?file(PathDB);
12       weka.verifyFormatDB(PathDB,Format);
13       .print("DB given in format ",Format);
14       !pVerifyFormatDB(PathDB,Format).
15
16 //Plans take actions depending on what type the database is
17 @pVFBD1
18 +!pVerifyFormatDB(PathDB,Format): not (Format == ".xls" | Format == ".csv" |
19                                        Format == ".arff")
20    <- .print("DB must be given in format .xls, .csv o .arff").
21
22 @pVFBD2
23 +!pVerifyFormatDB(PathDB,Format): Format == ".xls" | Format == ".csv" |
24                                   Format == ".arff"
25    <- !pConviertDB(PathDB,Format);
26       ?file(PathDBArff);
27       .send(preprocessing,achieve,verifyClass(PathDBArff));
28       .wait("+typeClass(TypeClass)");
29       !printTClass.
```

Table 5. Preprocessing Agent.

Plan	Internal Actions	WEKA Classes
@ppBD	replaceMissing	ArffLoader, ArffSaver Instances, Instance Filter, ReplaceMissingValues
	discretizeS	ArffLoader, ArffSaver Instances, Instance Filter, Discretize
	discretizeNS	ArffLoader ArffSaver, Instances Instance, Filter, Filter

The internal actions used to implement the plan library for each agent are built from WEKA classes. Table 5 illustrates this for the Preprocessing agent. For example, the *@ppBD* plan uses the *replaceMissing*, *discretizeS*, and *discretizeNS* internal actions, implemented using methods inherited from the *ArffLoader*, *ArffSaver* classes, etc., provided by WEKA.

5 Tests and Results

The MAS was tested with different databases from the repository at the University of California [2] to observe its viability offering support. Table 6 describes databases used in the tests. Performance, in terms of accuracy, was estimated using the *Stratified 10-fold Cross-Validation*, since it has been reported [24] as the best method to select one model from a set of them. Tables 7 and 8 show the percentage of correctly classified instances for each model learned by the agents, with the databases discretized supervised and non-supervised, respectively.

Table 6. Description of the Databases.

DB's	$\|Instances\|$	$\|Attributes\|$	$\|Classes\|$
Anneal	798	39	5
Balance_Scale	625	5	3
Credit_S	690	16	2
Diabetes	768	9	2
Ecoli	336	9	8
Hypothyroid	3163	26	2
Ionosphere	351	35	2
Iris	150	5	3
Lymphography	148	19	4
Segment	2310	20	7
Soybean	307	36	19
Vehicle	846	19	4
Zoo	101	18	7

The performance results show that there is no such a thing as the "best method" and the MAS automatizes the search for the best classifier using the selected algorithms. More importantly, the MAS avoids getting the "worst method" results, which in some cases (Zoo/ID3 and Ecoli/ID3) is quite relevant. The dynamics of the experiments suggests that the coordinator or another new agent, can be programmed to produce comparative reports, instead of the "the winner is..." behavior. Tables 9 and 10 provide the time taken to build the models learned by the agents, with the data discretized supervised and non-supervised, respectively.

The results of these tables indicate us that for both kinds of data discretization, NB always builds its models in the least time (because it only learns the parameters of the model, since this one is always the same) than the rest of the classifiers, whereas TAN is the one takes the most time to learn its models. In addition, we can see that when the data are discretized non-supervised, in general, the models are built in the least time.

Table 7. Percentage of correctly classified instances obtained with the data discretized supervised.

DB's	%ID3	%C4.5	%NB	%TAN
Anneal	**93.23**	92.98	92.1	92.1
Balance_Scale	70.72	71.04	71.68	**72.32**
Credit_S	77.97	86.96	86.23	**87.25**
Diabetes	75.26	77.73	77.99	**78.12**
Ecoli	0	80.95	**85.42**	85.12
Hypothyroid	98.86	**99.24**	98.58	98.83
Ionosphere	89.17	88.89	90.6	**91.17**
Iris	**95.33**	93.33	94.67	94
Lymphography	79.05	77.7	**83.78**	82.43
Segment	94.85	**95.41**	91.77	92.21
Soybean	91.21	**92.51**	85.99	88.6
Vehicle	**71.63**	70.45	62.53	69.5
Zoo	1.98	94.06	**96.04**	**96.04**
μ	**72.25**	**86.25**	**85.95**	**86.74**

Table 8. Percentage of correctly classified instances obtained with the data discretized non-supervised.

DB's	%ID3	%C4.5	%NB	%TAN
Anneal	89.6	**89.97**	88.47	87.72
Balance_Scale	39.52	65.92	**90.88**	86.72
Credit_S	74.64	**85.8**	84.35	84.64
Diabetes	59.89	72.66	75.52	**76.95**
Ecoli	0	73.21	**83.03**	78.27
Hypothyroid	95.92	**97.53**	96.9	97.03
Ionosphere	85.75	88.6	**90.6**	**90.6**
Iris	91.33	**96**	95.33	91.33
Lymphography	76.35	75	**83.11**	79.05
Segment	91.13	**93.16**	88.96	89.52
Soybean	84.69	**89.25**	80.78	83.71
Vehicle	58.51	71.63	59.93	**72.46**
Zoo	1.98	92.08	90.1	**97.03**
μ	**65.33**	**83.91**	**85.23**	**85.77**

6 Conclusions

This paper presents the design and implementation of a BDI MAS to support the KDD process. More precisely we present a framework that uses methodology and tools proposed in the MAS literature, to approach the automatizing of support for the use of common tools, as WEKA, in the KDD process. The MAS was designed according to the *Prometheus* methodology with the support of the

Table 9. Time taken to build the models learned with data discretized supervisely.

DB's	%ID3	%C4.5	%NB	%TAN
Anneal	0.96	0.91	**0.22**	2.48
Balance_Scale	0.24	0.41	**0.11**	0.16
Credit_S	0.6	0.78	**0.15**	0.53
Diabetes	0.66	0.81	**0.2**	0.42
Ecoli	1.62	1.28	**0.3**	1.65
Hypothyroid	1.57	1.15	**0.33**	3.25
Ionosphere	0.63	0.65	**0.21**	1.47
Iris	0.31	0.17	**0.1**	0.26
Lymphography	0.48	0.46	**0.21**	0.37
Segment	1.04	0.57	**0.16**	2.72
Soybean	0.48	0.2	**0.05**	3.48
Vehicle	0.51	0.44	**0.04**	1.3
Zoo	0.12	0.05	**0.01**	0.53
μ	**0.71**	**0.61**	**0.16**	**1.43**

Table 10. Time taken to build the models learned with data discretized non-supervised.

DB's	%ID3	%C4.5	%NB	%TAN
Anneal	0.34	0.14	**0.02**	0.33
Balance_Scale	0.17	0.11	**0**	0.02
Credit_S	0.15	0.08	**0.01**	0.07
Diabetes	0.2	0.13	**0**	0.06
Ecoli	0.31	0.08	**0**	0.28
Hypothyroid	0.51	0.31	**0.06**	0.56
Ionosphere	0.21	0.06	**0.01**	0.14
Iris	0.06	0.13	**0**	0.04
Lymphography	0.12	0.06	**0**	0.16
Segment	0.4	0.18	**0.02**	0.37
Soybean	0.14	0.13	**0.01**	2.73
Vehicle	0.34	0.11	**0.02**	0.28
Zoo	0.13	0.06	**0.02**	0.29
μ	**0.24**	**0.12**	**0.01**	**0.41**

PDT tool. Some advantages that were found when using Prometheus to model the KDD process like a BDI MAS are:

- The KDD process was clearly understood, because of the goal oriented analysis to define the MAS. The use of diagrams in the design was also helpful.
- The transition between the design and implementation of the BDI MAS was really easy. The diagrams obtained with Prometheus are expressed in terms of beliefs, goals, plans, events, etc., which are the natural constructors for the BDI agents and its AgentSpeak(L) programming language.

– Prometheus can be used to model any process within an organization, since its diagrams offer a high level of abstraction, in the sense that they can be referred in a similar language to ours (for example, through beliefs, goals, scenarios, roles, plans, events, actions, etc.).

The MAS was implemented in the *AgentSpeak(L)* agent programming language, through the *Jason* interpreter. The main point in favor of Jason, regarding other development platforms of MAS (as JACK [7], JAM [21], JADEX [31], among others), is its theoretical basis, which gets to implement the operational semantics of AgentSpeak(L).

Java helps since both WEKA and Jason are written in this language, so that inherit methods in both senses was natural. We have opted for taking methods in WEKA to built internal actions in Jason agents. Future work will explore another possibility: considering WEKA as the environment for the MAS implemented in Jason. This should enable a richer interaction between the agents and the users of WEKA, e.g., an agent can execute directly commands from WEKA, in the same way the user does. Another contribution due to Java is portability. The MAS has been executed successfully on Windows, Linux, Solaris, Mac OS X, all for free.

Our exploration provides the basis to built more elaborated MAS in this context, e.g, extending the number of agents (more learning algorithms adopted) or extending their competences (wiser agents using better the algorithms). A much more ambitious goal is to approach meta-learning by this way, e.g., the agents learn intentionally [17, 18] to become wiser. This kind of learning enables the agents to learn when a given plan is really useful, given the desires and beliefs of an agent, and its past experience, i.e., agents that learn to learn. Given that, such a system would be much more elaborated, we are currently developing formal tools for AgentSpeak(L) program verification [19].

Acknowledgments. The second author is supported by the CONACYT scholarship 197800 and DIP-UJAT (DAIS-02 UJAT-EGRESADA/2005) agreement.

References

1. Ancona, D., Mascardi, V., Hübner, J., Bordini, R.: Coo-AgentSpeak: Cooperation in AgentSpeak through plan exchange. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2004), pages 698-705. New York, NY (2004)
2. Asuncion, A., Newman, D.: UCI Machine Learning Repository. Irvine, CA: University of California, Department of Information and Computer Science. http://www.ics.uci.edu/ mlearn/MLRepository.html [Consulted: friday, June 27, 2007].
3. Bailey, S., Grossman, R., Sivakumar, H., Turinsky, A.: Papyrus: A System for Data Mining Over Local and Wide Area Clusters and Super-clusters. In Proceedings of the 1999 ACM/IEEE conference on Supercomputing, page 63. ACM Press, Portland, OR (1999)

4. Bordini, R., Dastani, M., Dix, J., El Fallah Seghrouchni, A.: In Multi-Agent Programming: Languages, Platforms and Applications. Springer, New York (2005)
5. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley, England (2007)
6. Bordini, R., Moreira, A.: Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L). Annuals of Mathematics and Artificial Intelligence, 42(1-3):197-226. Special Issue on Computational Logic in Multi-Agent Systems, September (2004)
7. Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents - Components for Intelligent Agents in Java. Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia (1998)
8. d'Inverno, M., Luck, M.: Engineering AgentSpeak(L): A formal computational model. Journal of Logic and Computation, 8(3). 233-260 (1998)
9. Dunham, M.: Data Mining: Introductory and Advanced Topics. Prentice Hall PTR (2002)
10. Fayyad, U., Irani, K.: Multi-interval discretization of continuous-valued attributes for classification learning. Proceedings of the 13th International Joint Conference on Artificial Intelligence, 1022-1027. Morgan Kaufmann, San Francisco, CA (1993)
11. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery in databases. AI Magazine, 17(3):37–72 (1996)
12. Fayyad, U., Piatetsky-Shapiero, G., Smyth, P.: Knowledge Discovery and Data Mining: Towards a Unifying Framework. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon (1996)
13. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From Data Mining to Knowledge Discovery: An Overview. In AKDDM, AAAI/MIT Press (1996)
14. Frank, E., Hall, M., Holmes, G., Kirkby, R., Pfahringer, B., Witten, I., Trigg, L.: WEKA - A Machine Learning Workbench for Data Mining. In O. Maimon and L. Rokach, editors, The Data Mining and Knowledge Discovery Handbook, pages 1305-1314. Springer (2005)
15. Georgeff, M. P. and Lansky, A. L.: 'Procedural knowledge. In Proceedings of the IEEE, (74)10:1383–1398 (1986)
16. Guerra-Hernández, A., Cruz-Ramírez, N., Mondragón-Becerra, R.: Exploraciones sobre el soporte Multi-Agente en Minería de Datos. Conferencia Internacional en Información, Comunicación y Diseño. Universidad Autónoma Metropolitana, Cuajimalpa, México (2006)
17. Guerra-Hernández, A., El-Fallah-Seghrouchni, A., Soldano, H.: Learning in BDI Multi-agent Systems. In: Dix, J., Leite, J. (eds.) CLIMA IV. LNCS, vol. 3259, pp. 218–233. Springer, Heidelberg (2004)
18. Guerra-Hernández, A., Ortíz-Hernández, G.: Toward BDI sapient agents: Learning intentionally. In: Mayorga, R.V., Perlovsky, L.I. (eds.) Toward Artificial Sapience: Principles and Methods for Wise Systems, pp. 77–91. Springer, London (2008)
19. Guerra-Hernández, A., Castro-Manzano, J.M., El-Fallah-Seghrouchni, A.: Toward an AgentSpeak(L) theory of commitment and intentional learning. In: LNAI 5317:848–858. Springer Verlag, Berlin Heidelberg (2008)
20. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers (2000)
21. Huber, M.: JAM: A BDI-Theoretic Mobile Agent Architecture. In Proceedings of the Third Annual Conference on Autonomous Agents, pp. 236-243, O. Etzioni, J. P. Müller, J. Bradshaw, editors, ACM Press (1999)
22. Kargupta, H., Hamzaoglu, I., Stafford, B.: Scalable, Distributed Data Mining Using

An Agent Based Architecture. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, Proceedings of Knowledge Discovery And Data Mining, pages 211-214. AAAI Press, Menlo Park, CA (1997)
23. Kargupta, H., Park, B., Hershberger, D., Johnson, E.: Collective Data Mining: A New Perspective Towards Distributed Data Mining. In Hillol Kargupta and Philip Chan, editors, Advances in Distributed and Parallel Knowledge Discovery, pages 133-184. MIT/AAAI Press (2000)
24. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann, San Mateo, CA (1995)
25. Mascardi, V., Demergasso, D., Ancona, D.: Languages for Programming BDI-style Agents: an Overview. In F. Corradini, F. De Paoli, E. Merelli, and A. Omicini, editors, WOA 2005 - Workshop From Objects to Agents, pages 9-15 (2005)
26. Mondragón-Becerra, R.: Exploraciones sobre el soporte Multi-Agente BDI en el Proceso de Descubrimiento de Conocimiento en Bases de Datos. Master's thesis submitted at the Universidad Veracruzana, December (2007)
27. Moreira, A., Bordini, R.: An operational semantics for a BDI agent-oriented programming language. In John-Jule Ch. Meyer and Michael J. Wooldridge, editors, Proceedings of the Workshop on Logics for Agent-Based Systems (LABS-02), held in conjunction with the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), pages 45-59. Toulouse, France (2002)
28. Moreira, A., Vieira, R., Bordini, R.: Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. In Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT-03), held with AAMAS-03, Melbourne, Australia (2003)
29. Padgham, L., Thangarajah, J., Winikoff, M.: Tool support for agent development using the prometheus methodology. In Fifth International Conference on Quality Software (QSIC 2005), pages 383-388. IEEE Computer Society, Melbourne, Australia (2005)
30. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. Proceedings of the Third International Workshop on Agent-Oriented Software Engineering, at AAMAS, Bologna, Italy (2002)
31. Pokahr, A., Braubach, L., Lamersdorf W.: Jadex: A BDI reasoning engine. In R. Bordini, M. Dastani, A. Seghrouchni, and J. Dix, editors, Multi-Agent Programming. Kluwer (2005)
32. Prodromidis, A., Chan, P., Stolfo, S.: Meta-learning in Distributed Data Mining Systems: Issues and Approaches. In Hillol Kargupta and Philip Chan, editors, Advances of Distributed Data Mining. MIT/AAAI Press (2000)
33. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: de Velde, W.V., Perram, J.W. (eds.) MAAMAW. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
34. SPSS. *Clementine*. http://www.spss.com/clementine/ [Consulted: thursday, May 29, 2008].
35. Vieira, R., Moreira, A., Wooldridge, M., Bordini, R.: On the formal semantics of speech-act based communication in an agent-oriented programming language. Submitted article to appear (2005)
36. Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques. 2nd Edition, Morgan Kaufmann, San Francisco (2005)
37. Wooldridge, M., Jennings, N.: Intelligent Agents: Theory and Practice. In Knowledge Engineering Review 10(2) (1995)