J-MADeM v.1.1: A full-fledge AgentSpeak(L) multimodal social decision library in Jason

Francisco Grimaldo[1], Miguel Lozano[1], Fernando Barber[1], Alejandro Guerra-Hernández[2]

[1] Departament d'Informàtica
Universitat de València
Av. Vicent Andrés Estellés, s/n, Burjassot, Spain, 46100
francisco.grimaldo@uv.es, miguel.lozano@uv.es, fernando.barber@uv.es

[2] Departamento de Inteligencia Artificial
Universidad Veracruzana
Facultad de Física e Inteligencia Artificial
Sebastián Camacho No. 5, Xalapa, Ver., México, 91000
aguerra@uv.mx

Abstract. In spite of the success of the Belief-Desire-Intention (BDI) model of agency, the gap between the actual implementation of BDI agents and the subjacent theories is a known problem of the approach. AgentSpeak(L), the abstract agent oriented programming language, was proposed as a solution for this problem; and Jason, its Java based implementation, provides a full featured development environment for it. However, a subtle question remains to be seen: the equilibrium between Java based efficient coding solutions, and the declarative, fully intentional, AgentSpeak(L) ones. In this paper we show how the new version of the J-MADeM library has been endowed with an AgentSpeak(L) layer for the implementation of multimodal social decisions in Jason. J-MADeM v.1.1 thus accelerates the development process and opens new lines of research to explore, e.g., intentional learning. To illustrate the benefits of this new release, we present a new example in which the J-MADeM library has been applied to solve the problem of meeting scheduling through a decision making approach that considers different points of view.

Keywords: AgentSpeak(L), Jason, MultiAgent Resource Allocation, Multimodal Social Decision Making.

1 Introduction and Related work

In spite of the success of the Belief-Desire-Intention (BDI) model of agency, the gap between the actual implementation of BDI agents and the subjacent theories is a known problem of the approach. AgentSpeak(L) [16], the abstract agent oriented programming language, was proposed as a solution for this problem; and Jason [1], its Java based implementation, provides a full featured development environment for it. However, a subtle question remains to be seen: the equilibrium between Java based efficient coding solutions, and the declarative, fully intentional, AgentSpeak(L) ones. This equilibrium is particularly important when dealing with social and organizational models, due to the inherent complexity of the problems they face.

Social and organizational models are being studied under the scope of multi-agent systems (MAS) in order to regulate the autonomy of self-interested agents. Nowadays, the performance of a MAS is determined not only by the degree of deliberativeness but also by the degree of sociability. In this sense, sociability points to the ability to communicate, cooperate, collaborate, form alliances, coalitions and teams. The assignment of individuals to an organization generally occurs in Human Societies [15], where the organization can be considered as a set of behavioural constraints that agents adopt, e.g., by the role they play [5].

The definition of a proper MAS organization is not an easy task, since it involves dealing with three dimensions: functioning, structure, and norms [14]. From the functioning perspective, systems focus on achieving the best plans and cover aspects such as: the specification of global plans, the policies to allocate tasks to agents, the coordination of plans, etc. [19, 4]. From the structural perspective, systems focus on defining the organizational structures (roles, relations among roles, groups of roles, etc.) that establishes the obligations/permissions of their agents [8, 7]. Very few models deal with both previous dimensions to support agent decision making about organizations, e.g., MOISE+ [14]. For the sake of simplicity, the third dimension is not discussed here.

Social reasoning has been extensively studied in MAS in order to incorporate social actions to cognitive agents [3]. As a result of these works, agent interaction models have evolved to social networks that try to imitate the social structures found in real life. Social dependence networks allow agents to cooperate or to perform social exchanges attending to their dependence relations [18]. Trust networks can define different delegation strategies by means of representating the attitude towards the others through the use of some kind of trust model, e.g., reputation [6]. Agents in preference networks express their preferences normally using utility functions so that personal attitudes can be represented by the differential utilitarian importance they place on the others' utilities. Following this preferential approach, the MADeM (Multi-modal Agent Decision Making) model [9] is a market-based mechanism for social decision making, capable of simulating different kinds of social welfares (e.g. elitist, utilitarian), as well as social attitudes of their members (e.g. egoism, altruism).

Few works dealing with complex social or organizational models have been delivered as a library for an agent-oriented programming language, e.g., J-MOISE+ [14]. However, such libraries are of great value to the community of MAS developers since they reduce the gap between theory and practice of MAS. In this paper we show how the new version of the J-MADeM library [10] has been endowed with an AgentSpeak(L) layer for the implementation of multimodal social decisions in Jason. J-MADeM v.1.1 thus accelerates the development process and opens new lines of research to explore, (e.g., intentional learning [12, 13]). To illustrate the benefits of this new release, we present a new example in which the

J-MADeM library has been applied to solve the well-known problem of meeting scheduling through a decision making approach that considers different points of view.

The rest of the paper is organized as follows: The next section introduces the MADeM model. Section 3 shows the new features of J-MADeM v.1.1 library, which allows programming MADeM decisions in Jason by only using AgentSpeak(L). In section 4 we reexamine the old J-MADeM examples, i.e, the Gold Miners and the Virtual University Bar simulation, as well as a new meeting scheduling application example. Finally, in section 5 we state the conclusions of this work and discuss about future lines of research.

2 The MADeM model

This section summarizes the MADeM (Multi-modal Agent Decision Making) model, fully explained in [9], in order to retrieve the main parameters that are necessary to use it. The MADeM model provides agents with a general mechanism to make socially acceptable decisions. In this kind of decisions, the members of an organization are required to express their preferences with regard to the different solutions for a specific decision problem. The whole model is based on the MARA (Multi-Agent Resource Allocation) theory [2], therefore, it represents each one of these solutions as a set of resource allocations. MADeM can consider both tasks and objects as plausible resources to be allocated, which it generalizes under the term *task-slots*. MADeM uses first-sealed one-round auctions as the allocation procedure [17] and a multi-criteria winner determination problem to merge the different preferences being collected according to the kind of agent or society simulated. Thus, given a MAS composed by the set of agents $\mathcal{A}$, the formal definition of a MADeM decision problem can be represented by the following tuple:

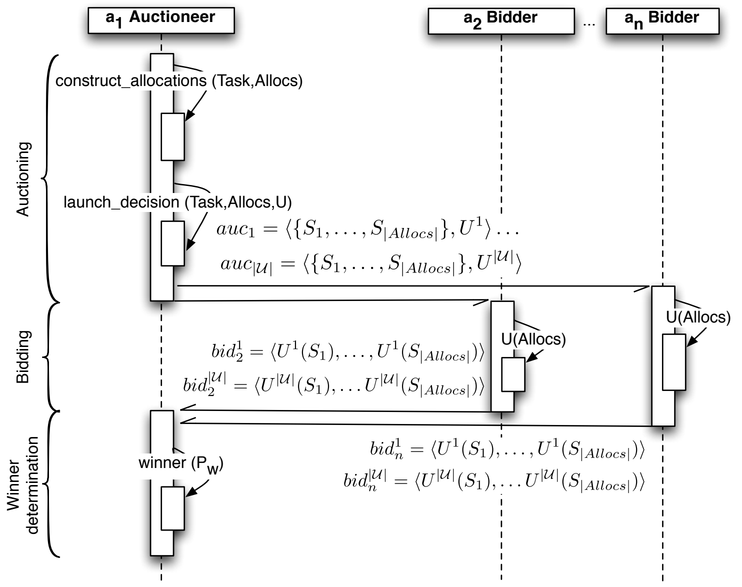
$$< a, Al, Ag, Pw, Uf, Uw, Cuf >$$

where:

- $a \in \mathcal{A}$ is the agent in charge of making a social decision or the auctioneer agent.
- Al is the set of resources or *task-slots* allocations representing all possible solutions for a specific decision problem.
- $Ag \subseteq \mathcal{A}$ is the subset of agents being consulted or bidder agents, which can be either infered from the organisational structure or maintained locally.
- $Pw : Ag \rightarrow \mathbb{R}^n$ are the personal weights (i.e. personal attitudes) that are used to balance the preferences received from each agent in Ag.
- Uf is the set of utility functions of the form $u : Al \times Ag \rightarrow \mathbb{R}$ representing the agents' preferences with regard to the resource allocations considered.
- $Uw : Uf \rightarrow \mathbb{R}$ are the utility weights that are used by the agent a to balance the importance given to each utility function in Uf when resolving the winner determination problem.

– $Cuf \in \{elitist, egalitarian, utilitarian, nash\}$ is the collective utility function representing the social welfare of the simulated society, that is, the type of society where agents are located.

Figure 1 shows an overview of the multi-modal decision making procedure followed by the agents, which mainly involves three steps:



a1 Auctioneer
a2 Bidder
...
an Bidder
Auctioning
construct_allocations (Task,Allocs)
launch_decision (Task,Allocs,U)
$auc_1 = \langle\{S_1,\ldots,S_{|Allocs|}\},U^1\rangle\ldots$
$auc_{|\mathcal{U}|} = \langle\{S_1,\ldots,S_{|Allocs|}\},U^{|\mathcal{U}|}\rangle$
Bidding
U(Allocs)
U(Allocs)
$bid_2^1 = \langle U^1(S_1),\ldots,U^1(S_{|Allocs|})\rangle$
$bid_2^{|\mathcal{U}|} = \langle U^{|\mathcal{U}|}(S_1),\ldots U^{|\mathcal{U}|}(S_{|Allocs|})\rangle$
Winner determination
winner (Pw)
$bid_n^1 = \langle U^1(S_1),\ldots,U^1(S_{|Allocs|})\rangle$
$bid_n^{|\mathcal{U}|} = \langle U^{|\mathcal{U}|}(S_1),\ldots U^{|\mathcal{U}|}(S_{|Allocs|})\rangle$

Fig. 1. MADeM Procedure

1. *Auctioning phase*: The auctioneer agent (a_1) wants to socially solve a decision problem, e.g., where to sit. A set of allocations representing all the possible solutions for the problem ($\{S_1, S_2, \ldots, S_{|Allocs|}\}$) is computed by him. These allocations have the form of task slots assignations such as $SitAt(P) \leftarrow table_1$. Then, he auctions them to the bidder agents. As complex decisions could require to consider more than one point of view, the auctioneer can start several auctions ($au_k(\{S_1, \ldots, S_{|Allocs|}\}, U^k)$) to evaluate the same set of allocations but using different utility funcions (au_1 through $au_{|U|}$).
2. *Bidding phase*: Since the auctioneer informs about both the task slot allocations and the utility functions being considered, bidders simply have to compute the requested utility functions and return the values corresponding to each auction back to the auctioneer ($bid_i^k =< U_i^k(S_1), \ldots, U_i^k(S_{|Allocs|}) >$).
3. *Winner determination phase*: In this phase, the auctioneer selects a winner allocation for each launched auction. To do this, he uses a market-based

winner determination problem to merge the different preferences being collected according to the kind of agent or society simulated. Thus, the final winner allocation will represent an acceptable decision for the society being simulated. As already mentioned, the details of these calculations are out of the scope of this paper. They are fully described in [9].

3 J-MADeM v.1.1 Implementation

The J-MADeM v.1.1 offers to the AgentSpeak programmer: (i) an agent architecture that Jason agents can use to carry out their own MADeM decisions, (ii) an ontology to express MADeM data as beliefs and rules, and (iii) a plan library to execute MADeM processes.

The agent architecture, `jmadem.MADeMAgArch`, implements a set of actions (Table 1) which perform the basic operations of the multimodal social decision model. As usual in *Jason*, actions are prefixed by the name of the library, e.g., to set the walfare of the society as a nash equilibrium, the action `jmadem.set_welfare(nash)` is executed in a plan. The action for constructing allocations is very general, basically it computes (at Java level) the cartesian product of the slots domains. Usually some kind of filtering is required to obtain "legal" allocations. The rest of the actions are devoted to set the MADeM parameters, as defined in section 2, e.g., `set_personal_weight`; and to launch the MADeM decision process, e.g., `launch_decision`.

Table 1. Actions defined in the J-MADeM library.

Action	**Description**
`add_utility_function("P.U")`	P is a Java package name and U the utility function name.
`add_utility_function(U,N)`	U is a utility name and N is fully qualified name of the function Java class.
`construct_allocations(T,S,E,Al)`	$T = t(S_1, \dots, S_n)$ is a function denoting a task t of n slots, $S \subseteq \{S_1, \dots, S_n\}$ is a set of task slots to be allocated, $E = [[e_1, \dots, e_j], \dots]$ elements in the domain of each slot, Al is the computed list of allocations
`launch_decision(A,AL,U,DId)`	A is a set of agents, AL is a set of allocations, U is a list of utilify functions, and DId is the output parameter.
`launch_decision1(A,AL,U,DId)`	As above, but it returns only 1 solution.
`remove_utility_function(U,N)`	U and N are as above.
`reset_personal_weights(PW)`	$PW = [jmadem\_personal\_weight(A, \_), \dots]$.
`reset_utility_weights(UW)`	$UW = [jmadem\_utility\_weight(U, \_), \dots]$.
`set_list_of_personal_weights(PW)`	$PW = [jmadem\_personal\_weight(A, W), \dots]$, where A is an agent and $W \in \Re$ his personal weight.
`set_list_of_utility_weights(UW)`	$UW = [jmadem\_utility\_weight(U, W), \dots]]$, where U is an utility name and $W \in \Re$ its weight.
`set_personal_weight(A,W)`	A is an agent and $W \in \Re$ is his weight.
`set_remove_MADeM_data(V)`	If V is *true* MADeM data is deleted at the Java level, once the decision is done.
`set_timeout(T)`	T is a numerical value in milliseconds (1000 by default).
`set_utility_weight(U,W)`	U is a utility name and $W \in \Re$ is its weight.
`set_welfare(W)`	$W \in \{utilitarian, egalitarian, elitist, nash\}$ is the welfare.

MADeM agents use an ontology (Table 2) to define the input data of a decision process declaratively as beliefs. In this way, data is accessible to Test Goals and Speech Acts with *Ask*-like performatives. Utilities and filters can now be defined as beliefs or rules. For instance:

```
jmadem_utility(dummyUF,_,use(coffeeMachine,Myself),0) :- .my_name(Myself).
```

expresses that an agent is not interested in using the coffee machine following the utility function `dummyUF`. Similarly, a belief:

```
jmadem_filter(dummyFilter,use(coffeeMachine,fran)).
```

can be used to filter `fran` from a set of allocations. In addition, the agent class `jmadem.MADeMAgent` overrides the default belief revision function of Jason to delete temporal beliefs added while computing utility values.

Table 2. The ontology used by J-MADeM agents.

Belief formula	Description
`jmadem_list_of_personal_weights(PW)`	PW is a list of personal weight, as defined below.
`jmadem_list_of_utility_weights(UW)`	UW is a list of utility weights, as defined below.
`jmadem_filter(F,Al)`	F is the name of the filter Al is an allocation to be filtered.
`jmadem_personal_weight(A,W)`	A is an agent and $W \in \Re$ his weight.
`jmadem_timeout(T)`	T is the timeout in millisecond (1000 by default).
`jmadem_utility(U,N)`	U is the utility function name and N is the name of the java class.
`jmadem_utility(U,A,Al,V)`	U is the utility function name, A is the auctioneer agent, Al is an allocation, and V is the utility value assigned to Al according to U.
`jmadem_utility_weight(U,W)`	U is an utility name and $W \in \Re$ is its weight.
`jmadem_welfare(W)`	$W \in \{utilitarian, egalitarian, elitist, nash\}$.

The library of plans `jmadem.asl` enables the calling of MADeM processes as Achieve Goals. The trigger events recognized by these plans are listed in Table 3. Utilities and filters can also be defined as plans. For instance, the previous examples would be as follows:

```
+!jmadem_utility(dummyUF,_,use(coffeeMachine,Myself),0) : .my_name(Myself).
+!jmadem_filter(dummyFilter,use(coffeeMachine,fran)) : true.
```

Then, Speech Acts with *AskHow*-like performatives can be used to exchange utilities and filters defined as plans. Furthermore, there is a plan for constructing allocations after the beliefs of an agent (`!jmadem_construct_allocations` in Table 3). Instead of listing the domains for each slot, as it was the case for the corresponding action, the user defines a logical query E that computes them. Thus "legal" allocations are computed directly. Alternatively, allocations can be further filtered by means of the achive goal `!jmadem_filter_allocations`.

Table 3. Trigger Events used by J-MADeM agents.

Trigger Event	Description
`+!jmadem_get_utility_function_names(U)`	U is a list of utility names.
`+!jmadem_construct_allocations(T,E,Al)`	T is a set of task slots, E is a logic formula to compute the elements of the allocation, and Al is the resulting set of allocations.
`+!jmadem_filter_allocations(F,Al,FAls)`	F is a filter, Al is a set of allocations, $FAls$ is a set of filtered allocations.
`+!jmadem_launch_decision(A,Al,U,DId)`	A is a set of agents, Al is a set of allocations, U is a list of utility function names, DId is a decision identifier.
`+!jmadem_launch_decision1(A,Al,U,DId)`	As above, but for 1 solution.

Let us introduce a very simple example to illustrate the use of the library. Emphasis is on the different ways of defining utilities. The `test` MAS (Table 4) defines a society where four agents (lines 4–7) try to decide who prepares the coffee in an office [3]. The entry `classpath` (line 9) must include the current path to the J-MADeM library.

Table 4. A MAS for deciding who prepares the coffee.

```
 1  MAS test {
 2    infrastructure: Centralised
 3    agents:
 4      fran       agentArchClass jmadem.MADeMAgArch agentClass jmadem.MADeMAgent;
 5      miguel     agentArchClass jmadem.MADeMAgArch agentClass jmadem.MADeMAgent;
 6      fernando   agentArchClass jmadem.MADeMAgArch agentClass jmadem.MADeMAgent;
 7      alejandro  agentArchClass jmadem.MADeMAgArch agentClass jmadem.MADeMAgent;
 8
 9    classpath: "lib/jmadem.jar";
10  }
```

Agent `fran` (Table 5) launches the multimodal social decision process, believing that three other agents and himself are going to participate in it (line 4). He believes that the welfare of his society is utilitarian (line 6) and also expresses his points of view as beliefs about utility functions (lines 8–11). Beliefs are a natural and useful way to declare preferences for different agents (lines 13–15) and points of view (line 17–18). Agent `fran` wants to decide who uses the coffee machine in the office, excluding himself (lines 25–27). This exemplifies the generation of allocations filtered under some criteria, e.g., not including $fran$.

[3] A simplified version of the `test-api` MAS included in the J-MADeM v.1.1 distribution, which exemplifies much more uses of the redudant facilities provided by the library.

Since all MADeM data have been declared at the *AgentSpeak*(L) level, subgoals as `!jmadem_get_utility_function_names/1` and internal actions as `.findall` can be used to configure the inputs for the decision process (line 28–30). Anyway the utility functions are still defined as Java classes in the `fran` package. On the contrary, the agent `fernando` (Table 6) declares his utilities as beliefs (rules) and the agent `miguel` (Table 7) does it as plans. Agent `alejandro` (Table 8), as a newcomer, profits of the approach to ask `miguel` his preferences and adopt them.

Table 5. Agent `fran` launches the social decision process.

```
1   { include("lib/asl/jmadem.asl") }   // J-MADeM Plan Library
2
3   /* Beliefs */
4   ag(fran). ag(fernando). ag(miguel). ag(alejandro).
5
6   jmadem_welfare(utilitarian).
7
8   jmadem_utility(minimumUtilityFunction,
9                  "fran.MinimumUtilityFunction").
10  jmadem_utility(maximumUtilityFunction,
11                 "fran.MaximumUtilityFunction").
12
13  jmadem_personal_weight(fernando,0.25).
14  jmadem_personal_weight(miguel,0.5).
15  jmadem_personal_weight(alejandro,0.25).
16
17  jmadem_utility_weight(minimumUtilityFunction,0.25).
18  jmadem_utility_weight(maximumUtilityFunction,0.75).
19
20  /* Initial goal */
21  !test.
22
23  /* Test plans */
24  +!test
25    <- TaskSlot = use(coffeeMachine, Agent);
26       Elements = (ag(Agent) & Agent \== fran);
27       !jmadem_construct_allocations(TaskSlot, Elements, Allocs);
28       .findall(Ag, ag(Ag), Ags);
29       !jmadem_get_utility_function_names(Us);
30       jmadem.launch_decision(Ags, Allocs, Us, DId);
31       .println("Launched MADeM decision with identifier: ", DId).
```

4 Experiments and Results

This section summarizes the results obtained in three application examples developed to test J-MADeM agents. First, we revisit the Gold Miners problem [1], a classical simulation scenario where agents must compete for the resources (gold) located at the environment. This example allows us to evaluate the efficiency of the auction-based method proposed and to experiment with dynamic organizations. Second, in order to test the sociability features provided by J-MADeM, we

Table 6. Agent `fernando`: utilities as beliefs.

```
{ include("lib/asl/jmadem.asl") }      // J-MADeM Plan Library

/* Beliefs */

jmadem_utility(minimumUtilityFunction,_, Allocation, 0.2) :-
  .my_name(AgName) & Allocation = use(_,AgName).
jmadem_utility(minimumUtilityFunction,_, Allocation, 1)   :-
  .my_name(MyName) & Allocation = use(_,AgName) & MyName \== AgName.

jmadem_utility(maximumUtilityFunction,_, Allocation, 0.8) :-
  .my_name(AgName) & Allocation = use(_,AgName).
jmadem_utility(maximumUtilityFunction,_, Allocation, 0)   :-
  .my_name(MyName) & Allocation = use(_,AgName) & MyName \== AgName.

jmadem_utility(zeroUtilityFunction,_,_,0).

jmadem_utility(noneUtilityFunction,_,_,none).
```

have created a virtual university bar simulation. In this scenario waiter agents serve the orders placed by customer agents. According to the model parameters and the society being simulated, waiters are able to combine social behaviors (i.e. chatting) and efficiency at work. In the last example we use J-MADeM to solve a Meeting Scheduling Problem (MSP).

4.1 The Gold Miners

In this example a team of gold-mining agents has to find a set of chunks of gold, randomly scattered in a grid-like territory, in order to carry them to a depot. The original Jason team consists of a leader that assigns each miner to a quadrant of the grid. The miners then explore and pick up the pieces of gold they find in the environment. To be more efficient, when an agent finds a gold in his way and can not pick it up, he can use an auction-based model to inform the others about the new gold location and assign the gold to the winner agent.

Following the original Jason implementation, we have created an equivalent MAS with J-MADeM agents to be able to compare both. We have implemented an utility function (*goldDistance*), equivalent to the distance function used by the original miners, so that J-MADeM miners can express their preferences with regard to the gold units. Next, we find the definition of one of the cases of this utility function, defined entirely at the BDI level:

```
+!utility(goldDistance,_,gold(X,Y,MinerName), Distance)
   : .my_name(MinerName) & free & not carrying_gold
     & not allocated(gold(X,Y,_))
 <-?pos(MyX, MyY);
    // Distance between two points
    jia.dist(MyX,MyY,X,Y,Dist2Gold);
    Distance = 1000 - Dist2Gold.
```

Two versions of this example are included in the J-MADeM v.1.1 distribution: the original `gold-miners-jmadem` and the `gold-miners-structured`,

Table 7. Agent `miguel`: utilities as plans.

```
{ include("lib/asl/jmadem.asl") }      // J-MADeM Plan Library

/* Plans */

+!jmadem_utility(minimumUtilityFunction, Auctioneer, Allocation, 0.3)
    : .my_name(AgName) & Allocation = use(_,AgName).
+!jmadem_utility(minimumUtilityFunction, Auctioneer, Allocation, 1)
    : .my_name(MyName) & Allocation = use(_,AgName) & MyName \== AgName.

+!jmadem_utility(maximumUtilityFunction, Auctioneer, Allocation, 0.9)
    : .my_name(AgName) & Allocation = use(_,AgName).
+!jmadem_utility(maximumUtilityFunction, Auctioneer, Allocation, 0)
    : .my_name(MyName) & Allocation = use(_,AgName) & MyName \== AgName.

+!jmadem_utility(zeroUtilityFunction,_,_,0).

+!jmadem_utility(noneUtilityFunction,_,_,none).
```

Table 8. Agent `alejandro`: uses Speech Acts to form his utilities.

```
{ include("lib/asl/jmadem.asl") }      // J-MADeM Plan Library

/* Initial goal */

!askUF.

/* Plans */

// Alejandro gets the utility functions from Miguel
+!askUF
  <- .send(miguel, askHow, {+!jmadem_utility(_,_,_,_)}).
```

where a new multi-agent organization is proposed to adapt better to different gold distributions. For a complete description of both experiments, see [11].

4.2 Virtual university bar

This social scenario represents a virtual university bar where waiters take orders placed by customers. Both waiters and customers carry out tasks in the virtual bar and they use J-MADeM to decide among different task slot assignments. For a full description of the simulated scenario see [9].

In this scenario, we test the ability of J-MADeM to model different social behaviors by means of utility weights (UW) and personal weights (PW). We have modelled the waiters with three different utility functions: one utility function for performance, another utility function for sociability with other waiters, and another one for representing the tiredness of waiters.

Preferences about these points of view are expressed as beliefs, e.g., for a *coordinated waiter* focusing on performance:

```
jmadem_utility_weight(performanceUF, 0.75).
jmadem_utility_weight(socialUF, 0.125).
jmadem_utility_weight(tirednessUF, 0.125).
```

Furthermore, using personal weights, waiters can model their attitude towards others. For instance, an altruist waiter `fernando` defines the following beliefs:

```
jmadem_personal_weight(fernando, 0.25).
jmadem_personal_weight(miguel, 0.75).
jmadem_personal_weight(alejandro, 0.75).
jmadem_personal_weight(fran, 0.75).
```

This two dimensions of weights may be combined for obtaining more complex and richer behaviors for the agents. Detailed statistics of the different behaviors may be seen in [9].

4.3 The Meeting Scheduling Problem (MSP)

This section presents a J-MADeM implementation of the MSP problem, a well known scheduling problem to find the best allocation for a meeting within an organization. Traditionaly, a meeting scheduling problem is defined as a pair $\langle A, M\rangle$, where A is a set of agents and M a set of meetings. A time slot is denoted by $\langle D, H\rangle$ where D is a day and H is an hour. A meeting $m_i \in M$ is usually defined as a tuple $\langle A_i, h_i, l_i, w_i, S_i, \mathcal{T}_i\rangle$, where A_i is the set of agents assisting to the meeting; $h_i \in A$ is the host of the meeting; l_i is the duration of the meeting; w_i is the priority assigned to the meeting; and S_i is the starting time; and $\mathcal{T}_i = \langle D_i, H_i\rangle$ is the time slot assigned to the meeting or a list of time slots representing diffent solutions for the scheduling.

Global performance has been suggested to be measured as the weighted success ratio in scheduling n meetings:

$$\eta = \frac{\sum_{i=1}^{n} w_i \times \rho_i}{\sum_{i=1}^{n} w_i}$$

where:

$$\rho_i = \begin{cases} 1 & \text{if } m_i \text{ has been scheduled} \\ 0 & \text{otherwise} \end{cases}$$

The example proposed is about school meetings in a MAS where four agents $A = \{director, teacher, father, father\_worker\}$ try to schedule some meetings proposed by the host $h_i = director$ to $A_i = A - \{director\}$ agents. All meetings last $l_i = 2$ hours and their starting time can be $S_i \in H = \{9, 11, 15\}$ on days $D = \{monday, tuesday\}$. Two kinds of meetings are considered: Usual monomodal meetings, based on laboral preferences weighted with $w_i = 1$; and MADeM Multimodal meetings considering two weights: $w_l = math.random(1)$ for labor preferences; and $w_p = 1 - w_l$ for personal preferences. Performace η varies slightly since ρ_i is the number of attendants for the meeting m_i. This variation takes into account that autonomous agents can accept to assist to the scheduled meeting or not, following their preferences.

Bidder agents in A_i express their constraints as beliefs. Laboral and personal constraints values are in the range [0..1], where 1 indicates a full availability of the agent to attend the meeting and 0 indicates quite the opposite. A constraint of value 0.6 means: I could attend, but I'd prefer to schedule the meeting in another way. For instance, the `father_worker` agent has indifferent personal constraints for both days (anonymous variables) at 9 and 15 hours (lines 1–2); and a labor hard constraint both days at 11 hours (line 3).

```
1   constraint(personal,_,9,0.5).
2   constraint(personal,_,15,0.5).
3   constraint(labor,_,11,0).
```

Tables 9 and 10 show the utility values defined for this example. Following the minimun constraint values, utilities are computed using the following rules:

```
jmadem_utility(laborUF,_,meeting(_,Day,Hour), Min)
   :- .findall(V, constraint(labor,Day,Hour,V), LValues) &
      .min(LValues, Min).

jmadem_utility(personalUF,_,meeting(_,Day,Hour), Min)
   :- .findall(V, constraint(_,Day,Hour,V), LValues) &
      .min(LValues, Min).

jmadem_utility(_,_,_,1) :- true.
```

Table 9. Labor Utility Function: [0 | 1]

	Monday			Tuesday		
time	teacher	father	father_worker	teacher	father	father_worker
9	0	1	1	0	1	1
11	1	1	0	1	1	0
15	1	1	1	0	1	1

Table 10. Personal Utility Function: [0..1]

	Monday			Tuesday		
time	teacher	father	father_worker	teacher	father	father_worker
9	0	0.2	0.5	0	0.2	0.5
11	1	1	0	1	1	0
15	0.3	0.2	0.5	0	0.2	0.5

The host agent believes `jmadem_welfare(utilitarian)` and the weights of each utility function, w_l and w_p, are defined as specified above.

The best choice, considering the monomodal case is $\mathcal{T}_i = \{\langle monday, 15\rangle\}$, with a value of 3.0; while $\mathcal{T}_i = \{\langle monday, 11\rangle, \langle tuesday, 11\rangle\}$ with a value of 2.0, is the best solution for the multimodal case. The winner determination will depend on the weights applied to each utility function. For example, if the host agent sets the personal weight $w_p = 0.5$ and the labor one $w_l = 0.5$, then the winner allocation will be $\max(0.5 \times 3.0, 0.5 \times 2.0)$, and the labor modality will govern the result (Monday at 15). However it is easy to realize that if personal constraints are considered more important (for instance, $w_p > 0.6$), the winner allocation will change to the other one (Monday or Tuesday at 11).

Figure 2 shows the weighted success ratio η computed using the number of attendants ρ_i. For this running, the set of agents A contains: the director, 20 `teacher` agents, 20 `father` agents and 20 `father_worker` agents. It can be seen how the average number of attendants is higher when applying multimodality, as it offers intermediate solutions between both laboral and personal points of view.

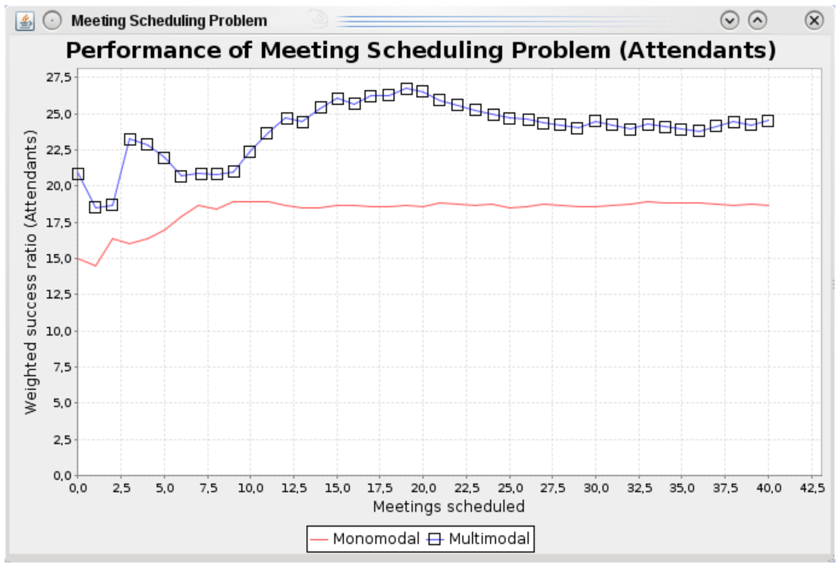


Fig. 2. Performance of the Meeting Scheduling Problem

5 Conclusions and Future work

In this paper we have shown the new version of the J-MADeM library, which is endowed with an AgentSpeak(L) layer for the implementation of multimodal social decisions in Jason. We have tested it with various examples: the gold-miners, a virtual university bar, and a Meeting Scheduling Problem. J-MADeM v.1.1 has demonstrated through these examples to accelerate the development process. It also offers the developer with the possibility to choose the more adequate way for implementing utility functions and the auctioning process, ranging from Java based efficient solutions to AgentSpeak(L) based declarative ones, as shown in the examples.

The use of J-MADeM v.1.1 suggests new directions for future research taking into account the intentional learning framework [12, 13], where $AgentSpeak(L)$ agents can learn reasons to adopt and abandon intentions for the benefit of their commitment strategies. MADeM data stated declaratively, enable the agents to build first-order trainning examples required for the framework and to communicate effectively for performing social learning. Dynamic adaptative multimodal social decision procedures are envisaged from this perspective.

Acknowledgements

This work has been jointly supported by the Spanish MEC and the European Commission FEDER funds, under grants Consolider-Ingenio 2010 CSD2006-00046, TIN2009-14475-C04-04 and JC2009-00342. The last author is supported by Conacyt CB-2007-78910 fundings.

References

1. R. H. Bordini, J. F. Hübner, and M. Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, 2007.
2. Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lemaitre, N. Maudet, J. Padget, S. Phelps, J. A. Rodriguez-Aguilar, and P. Sousa. Issues in multiagent resource allocation. *Informatica*, 30:3–31, 2006.
3. R. Conte and C. Castelfranchi. *Cognitive and Social Action*. UCL Press, London, 1995.
4. K. S. Decker. *Simulating Organizations: Computational Models of Institutions and Groups*, chapter Task environment centered simulation, pages 105–128. AAAI Press / MIT Press, Menlo Park, 1998.
5. V. Dignum and F. Dignum. Modelling agent societies: Co-ordination frameworks and institutions. In P. Brazdil and A. Jorge, editors, *Procs. of the 10th Portuguese Conference on Artficial Intelligence (EPIA'01)*, volume 2258 of *LNAI*, pages 191–204, Berlin, 2001. Springer.
6. R. Falcone, G. Pezzulo, C. Castelfranchi, and G. Calvi. Why a cognitive trustier performs better: Simulating trust-based contract nets. In *Proc. of AAMAS'04: Autonomous Agents and Multi-Agent Systems*, pages 1392–1393. ACM, 2004.
7. J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agents systems. In *Proc. of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135. IEEE Press, 1998.
8. M. S. Fox, M. Barbuceanu, M. Gruninger, and J. Lon. *Simulating Organizations: Computational Models of Institutions and Groups*, chapter An organizational ontology for enterprise modeling., pages 131–152. AAAI Press / MIT Press, Menlo Park, 1998.

9. F. Grimaldo, M. Lozano, and F. Barber. MADeM: a multi-modal decision making for social MAS. In *Proc. of AAMAS'08: Autonomous Agents and Multi-Agent Systems*, pages 183–190. ACM, 2008.
10. F. Grimaldo, M. Lozano, and F. Barber. J-MADeM, an open-source library for social decision-making. In *Proc. of CCIA'09: International Conference of the Catalan Association for Artificial Intelligence*, pages 207–214. IOS Press, 2009.
11. F. Grimaldo, M. Lozano, and F. Barber. J-MADeM, a market based model for complex decision problems. *Logic Journal of IGPL. doi:10.1093/jigpal/jzq028*, 2010.
12. A. Guerra-Hernández, A. El-Fallah-Seghrouchni, and H. Soldano. Learning in BDI Multi-agent Systems. In J. Dix and J. Leite, editors, *Computational Logic in Multi-Agent Systems: 4th International Workshop, CLIMA IV, Fort Lauderdale, FL, USA, January 6–7, 2004, Revised and Selected Papers*, volume 3259 of *Lecture Notes in Computer Science*, pages 218–233, Berlin Heidelberg, 2004. Springer-Verlag.
13. A. Guerra-Hernández and G. Ortíz-Hernández. Toward BDI sapient agents: Learning intentionally. In R. V. M. V. Mayorga and L. I. Perlovsky, editors, *Toward Artificial Sapience: Principles and Methods for Wise Systems*, pages 77–91. Springer, London, 2008.
14. J. F. Hübner, J. S. Sichman, and O. Boissier. Developing organised multi-agent systems using the Moise+ model: Programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.
15. M. Prietula, K. Carley, and L. Gasser, editors. *Simulating Organizations: Computational Models of Institutions and Groups.* AAAI Press / MIT press, 1998.
16. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In S. Verlag, editor, *Proc. of MAAMAW'96*, number 1038 in LNAI, pages 42–55, 1996.
17. T. W. Sandholm. Distributed Rational Decision Making. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 201–258. The MIT Press, Cambridge, MA, USA, 1999.
18. J. Sichman and Y. Demazeau. On social reasoning in multi-agent systems. *Revista Ibero-Americana de Inteligencia Artificial*, 13:68–84, 2001.
19. M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.