J-MADeM v1.0: 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] Computer Science Department
University of Valencia
Dr. Moliner 50, (Burjassot) Valencia, Spain 46000
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 v1.0 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 an example in which the J-MADeM library has been applied to solve the problem of meeting scheduling through an agent-based approach.

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

1 Introduction

2 J-MADeM v.1.0 Implementation

The library provides the `jmadem.MADeMAgArch` agent architecture class, implementing a set of actions (Table 1) which perform the basic operations of the multimodal social decision model. As usual in *Jason*, actions are suffixed by the

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 its Java class.
`launch_decision(A,AL,U,DId)`	A is a set of agents, AL as set of allocations, U 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)`	$PW = [jmadem\_personal\_weight(A, \_) \ldots]$.
`reset_utility_weights(UW)`	$UW = [jmadem\_utility\_weight(U,) \ldots]$.
`set_list_of_personal_weights(PW)`	$PW = [jmadem\_personal\_weight(A, W) \ldots]$, where A and W are as above.
`set_list_of_utility_weights(UW)`	$UW = [jmadem\_utility\_weight(U, W)]$, where U and W are as above.
`set_personal_weight(A,W)`	A is an agent and $W \in \Re$ is his weight in the process.
`set_remove_MADeM_data(V)`	If V is *true* MADeM data is deleted when finished.
`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 is its weight.
`set_welfare(W)`	$W \in \{utilitarian, egalitarian, elitist, nash\}$ is the welfare of the society.

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 library also implements the `jmadem.MADeMAgent` agent class, overriding the default belief revision funcion of Jason, to parse the beliefs about utility functions. The ontology associated to J-MADeM agents is listed in Table 2. This means that the data associated to MADeM can now be declarative defined as beliefs of the agents, so usable in test goals and speech acts of performative *Ask*.

Furthermore, there is also the `jmadem.asl` library of plans, that enable the definition of MADeM data as achieve goals. The trigger events recognized by these plans are listed in Table 3. Speech acts of performative *AskHow* can be used to exchange such plans.

For instance, the `test` (Table 4) MAS defines a society where four agents trying to decide who prepares the coffee in an office [3]. The entry `classpath` points to the *J-MADeM* library.

[3] A simplified version of the `test-api` MAS included in the *J-MADeM* distribution, which exemplifies much more uses of the new constructs provides by *J-MADeM v.1.0.*

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

Belief formula	Description
`jmadem_list_of_personal_weights(PW)`	PW is a list of personal weights.
`jmadem_list_of_utility_weights(UW)`	UW is a list of utility weights.
`jmadem_personal_weight(A,W)`	Agent A has personal weight W.
`jmadem_timeout(T)`	T is the timeout in millisecond.
`jmadem_utility_function(U,N)`	U is the utility function name N is the name of the java class,
`jmadem_utility_weight(U,W)`	U is an utility name and $W \in \Re$ is its weight.
`jmadem_welfare(W)`	$W \in \{utilitarian, egalitarian, elitist, nash\}$ is the welfare of the society.

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 arguments 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.

Agent `fran` (Table 4) launches the multimodal social decision process, believing that other three agents and himself are going to participate in it (line 4). Agent `fran` 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 also natural and useful 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). Since all data about the MADeM process has been declared at the BDI level, internal actions as `.findall` and subgoals as `!jmadem_utility_function_names/1` can be used to configure the call to launch the decision process (line 30); although the real code for the utility functions are still defined as a methods in the Java `fran` package. On the contrary, the agent `fernando` 6 declares his utilities as believes (rules) and the agent `miguel` 8 as plans. Agent `alejandro` 6, as a newcomer, profits of the approach to ask `miguel` his preferences and adopt them..

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  }
```

3 Experiment: Meeting scheduling

Traditionaly, a 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:

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

4 Discussion and future work

Acknowledgments.

A. Guerra-Hernández is supported by Conacyt CB-2007-78910 fundings.

References

1. F. Grimaldo, M. Lozano, and F. Barber. Madem: a multi-modal decision making for social mas. In L. Padgham, D. C. Parkes, J. P. Müller, and S. Parsons, editors, *AAMAS (1)*, pages 183–190. IFAAMAS, 2008.

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

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

/* Beliefs */
ag(fran). ag(fernando). ag(miguel). ag(alejandro).

jmadem_welfare(utilitarian).

jmadem_utility_function(minimumUtilityFunction,
                        "fran.MinimumUtilityFunction").
jmadem_utility_function(maximumUtilityFunction,
                        "fran.MaximumUtilityFunction").

jmadem_personal_weight(fernando,0.25).
jmadem_personal_weight(miguel,0.5).
jmadem_personal_weight(alejandro,0.25).

jmadem_utility_weight(minimumUtilityFunction,0.25).
jmadem_utility_weight(maximumUtilityFunction,0.75).

/* Initial goal */
!test.

/* Test plans */
+!test
  <- TaskSlot = use(coffeeMachine, Agent);
     Elements = (ag(Agent) & Agent \== fran);
     !jmadem_construct_allocations(TaskSlot, Elements, Allocs);
     .findall(Ag, ag(Ag), Ags);
     !jmadem_get_utility_function_names(Us);
     jmadem.launch_decision(Ags, Allocs, Us, DId);
     .println("Launched MADeM decision with identifier: ", DId).
```

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

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

/* Beliefs */

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

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

utility(zeroUtilityFunction,_,_,0).

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

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

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

/* Plans */

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

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

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

+!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, {+!utility(_,_,_,_)}).
```