J-MADeM, an open-source library for social decision-making

Francisco Grimaldo , Miguel Lozano and Fernando Barber [1]
Departament d'informàtica, Universitat de València, Spain

Abstract. This paper presents J-MADeM, a new market-based multi-agent approach devoted to perform social simulations with BDI agents. J-MADeM is available as an open-source library integrated into Jason [2], the successful interpreter for the AgentSpeak programming language [16]. The aim of this work is to improve Jason by incorporating the main features of MADeM [10], a market-based mechanism for social decision making. Thus, J-MADeM agents can easily define the utility functions expressing their preferences and find socially acceptable decisions for specific decision problems. This paper fully explains the three main features offered by J-MADeM to AgentSpeak programmers: (i) an agent architecture that Jason agents can use to carry out their own MADeM decisions, (ii) an interface to develop utility functions that can be used along with the MADeM model and (iii) a set of internal actions to manage the parameters of these kinds of decisions.

Keywords. Group Decision Making, Multi-Agent Based Social Simulation

1. Introduction and Related Work

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 aliances, coalitions and teams. Being assigned 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) [6,13].

The definition of a proper MAS organization is not an easy task since it involves dealing whith three dimensions: functioning, structure, and norms [14]. On the one hand, systems mainly focused on functionality aim at 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. [18,5]. On the other hand, systems mainly focused on defining the organizational structure (i.e. roles, relations among roles, groups of roles, etc.) accomplish their global purpose whereas the agents follow the obligations/permissions their roles entitle them [9,8]. A few systems deal with these first two dimensions (e.g. J-MOISE+ [14]) to support agent decision making about its organization.

[1]Departament d'informàtica, Universitat de València, Dr. Moliner 50, (Burjassot) València, Spain, e-mail: {francisco.grimaldo, miguel.lozano, fernando.barber}@uv.es. Web: http://www.uv.es/grimo/jmadem

Social reasoning has been extensively studied in multi-agent systems in order to incorporate social actions to cognitive agents [4]. 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 [12]. Social dependence networks allow agents to cooperate or to perform social exchanges attending to their dependence relations (i.e. social dependence/power [17]). 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 [7]). Lastly, 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 [10] is a market-based mechanism for social decision making, capable of simulating different kinds of social welfares (e.g. elitist, utilitarian, etc.), as well as social attitudes of their members (e.g. egoism, altruism, etc.).

This paper describes how the MADeM model can be used by an agent programming language to make socially acceptable decisions available to agents eventually part of an organization. Among several languages for agent programming, we have chosen the AgentSpeak language [16] and its open source interpreter Jason [2] to program this kind of social agents. This choice was made because the language is based on the well known BDI architecture and the interpreter can be easily customised to include the MADeM support. The coupling of MADeM with Jason is inspired in other extensions of Jason, in particular J-MOISE+ [14] and hence the name J-MADeM, as it joins Jason and MADeM.

The rest of the paper is organized as follows: The next section reviews the main functionalities as well as the parameters of the MADeM model. Section 3 fully explains the J-MADeM library that provides Jason agents with the built-in feature of performing MADeM decisions. Finally, in section 4 we state some conclusions and future work.

2. The MADeM model

This section summarizes the MADeM (Multi-modal Agent Decision Making) model, fully explained in [10], 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 [3], 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 and a multi-criteria winner determination problem to merge the different preferences being collected according to the kind of agent or society simulated. Thus, the formal definition of a MADeM decision 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, where $\mathcal{A}$ is the set of all agents in the society.
- Al is the set of resource 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 target agents, which can be either infered from the organisational structure or maintained locally.
- $Pw : Ag \to \Re^n$ 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 $u : Al \times Ag \to \Re$ representing the agents' preferences with regard to the resource allocations considered.
- $Uw : Uf \to \Re$ 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.

3. J-MADeM architecture

This section describes how the MADeM model [10] has been integrated into Jason [2] as an open source library named J-MADeM. The J-MADeM built upon the *Jason Communication Infrastructure*, thus extending the communication level options available in Jason with a set of modules that provide the agents with the built-in feature of performing MADeM decisions. Figure 1a illustrates how these components are integrated into Jason. The J-MADeM basically offers to the AgentSpeak programmer: (i) an agent architecture that Jason agents can use to carry out their own MADeM decisions (Section 3.1), (ii) an interface to develop utility functions that can be used along with the MADeM model (Section 3.2) and (iii) a set of internal actions to manage the parameters of these kinds of decisions (Section 3.3).

3.1. J-MADeM Agent Architecture

The *J-MADeM Agent Architecture* extends the *Jason Agent Architecture* in order to incorporate all the necessary modules that allow MADeM decisions to be automatically carried out. The main components of the *J-MADeM Agent Architecture* are shown in the figure 1b, where we can identify the following elements:

- *MADeM Parameters* : This data storage contains the MADeM context currently defined for the agent. Essentially, it stores the personal weights, the utility weights, the collective utility function and the bid timeout to be used in future MADeM decisions.
- *Decision Launcher* : This module starts the MADeM process for a particular decision. Firstly, it stores the MADeM context for this decision into the *Decision Data* storage, thus allowing other decisions to be concurrently performed with different MADeM parameters. Secondly, it auctions each of the allocations being considered as solutions to the target agents.

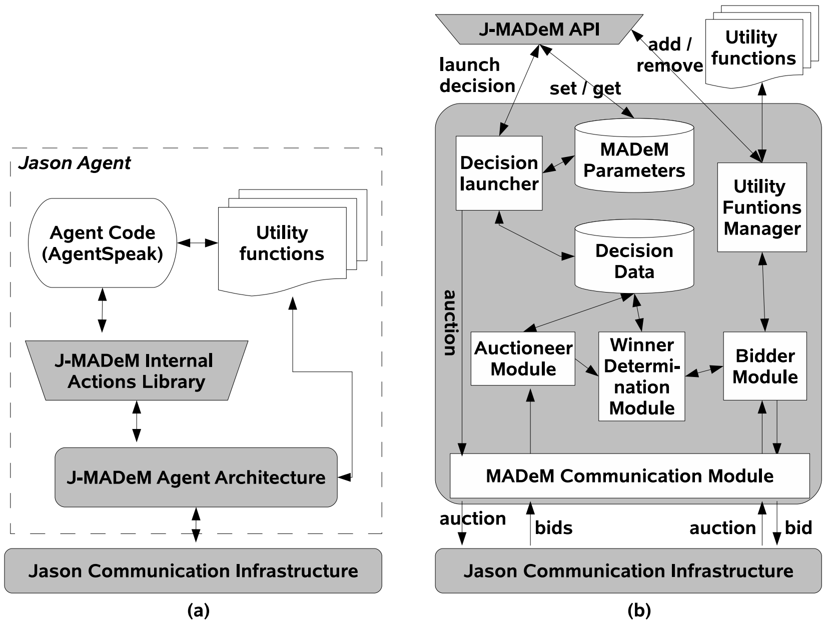

Figure 1. (a) Overview of the J-MADeM architecture and (b) detailed view of the J-MADeM Agent Architecture.

- *Decision Data* : This data storage holds all the information related to the MADeM decisions still in process. Therefore, it contains their MADeM context, their considered allocations and the preferences received for each of them.
- *MADeM Communication Module* : This module extends the Jason agent communication module in order to deal with MADeM messages. When it receives a MADeM auction, it invokes the *Bidder Module* to get the agent's preferences over the considered allocations. On the other hand, when it receives a MADeM bid, it informs the *Auctioneer Module* about the received preferences.
- *Bidder Module* : This module manages the reception of a MADeM auction. It extracts the considered allocations and bids for them according to the agent's preferences. To express these preferences it relies on the utility values provided by the *Utility Functions Manager.*
- *Utility Functions Manager* : This component acts as an interface between the built-in MADeM mechanism and the user defined *Utility Functions*. Thus, it is in charge of locating and invoking them in order to calulate the agents' utilities for the set of considered allocations.
- *Auctioneer Module* : This module manages the reception of MADeM bids. It extracts the sender's preferences and stores them into the *Decision Data*. As soon as the preferences from all the target agents have been received, it calls the *Winner Determination Module* to solve the decision.
- *Winner Determination Module* : This module solves the MADeM winner determination problem using the information stored into the *Decision Data* for the decision being resolved (i.e. considered allocations, agents' preferences, personal weights, utility weights, social welfare,...). Once resolved, it notifies the agent

about the winner solution by means of a belief base addition event of the type `+madem_result(Id, WinnerAllocation)`.

3.2. J-MADeM utility function interface

To be able to express their preferences, agents need to use utility functions. J-MADeM ease the creation of utility functions for the AgentSpeak agents by providing a Java interface, named `UtilityFunctionInt`, which just has to be implemented by the programmer. Essentially, two functions must be implemented when developing an utility function:

- `String getId(void)` : This method returns the name of the utility function so that the *Utility Functions Manager* of the *J-MADeM Agent Architecture* can locate it when necessary.
- `float computeUtility(a, Al, AgentArch)` : This method computes the utility value given by an agent to a set of allocations. In order to do this, it receives: (i) the agent asking for preference and, thus, responsible for making the decision (`a`); (ii) the allocation being evaluated (`Al`); and (iii) a reference to the architecture of the agent which is computing the utility value (`AgentArch`). Therefore, this method can access any aspect of the agent that expresses its preference (e.g. belief base, intentions, desires, etc.). Even though no restrictions are imposed by J-MADeM regarding the range of values to be covered by utility functions, for the sake of simplicity, frequently it is interesting to work with normalized utility functions. When dealing with this kind of functions, the values returned by this method must be within the interval [0, 1]; 0 meaning no preference and 1 meaning the highest preference.

3.3. J-MADeM internal actions library

The set of available actions in Jason can be extended by means of internal actions. Thus, we have created a library of internal actions termed `jmadem`that implements all the actions needed to handle MADeM decisions. The internal actions provided by J-MADeM are the following:

- `add_utility_function(Uf)` : This action registers a new utility function in the *Utility Functions Manager*. It receives one only parameter of the form `Uf = "pakage.utilityFunctionName"` that refers to the java class implementing the new utility function, which will be registered under the name provided by the `getId()` method of this java class. For more information about how to implement utility functions in J-MADeM see section 3.2.
- `remove_utility_function(Uf)` : This action removes a previously registered utility function that the agent is not interested in any more. It needs the name of the utility function as it was previously registered.
- `get_utility_function_names(ListOfNames)` : This action returns a list with the names of all the utility functions already registered in the MADeM agent.
- `set_personal_weights(Pw)` : This action sets the personal attitudes defined for a set of agents. It gets a list of pairs containing the name of the agents

and the weights to apply to their preferences. That is, it receives a paremeter of the form `Pw = [ [AgentName, Weight] | ListOfPairs ]`.

- `get_personal_weights(Pw)` : This action gets the personal attitudes already defined for the agents in the society. Again, it returns a list of pairs such as `Pw = [ [AgentName, Weight] | ListOfPairs ]`.
- `set_utility_weights(Uw)` : This action sets the level of importance for a set of utility functions. It gets a list of pairs containing the name of the already registered utility functions and the weights to apply when resolving the MADeM winner determination problem. Thus, the parameter being received must be of the form `Uw = [ [UtilityName, Weight] | ListOfPairs ]`.
- `get_utility_weights(Uw)` : This action returns the level of importance for the already registered utility functions. Thus, it returns a list of pairs such as `Uw = [ [UtilityName, Weight] | ListOfPairs ]`.
- `set_cuf(Cuf)` : This action sets the welfare of the society being simulated. Currently, J-MADeM allows selecting one of the following collective utility functions: `elitist`, `egalitarian`, `nash` and `utilitarian`.
- `get_cuf(Cuf)` : This action returns the welfare of the society being simulated (i.e. `elitist`, `egalitarian`, `nash` and `utilitarian`).
- `set_timeout(Tout)` : This action sets the bid timeout in milliseconds to be applied when performing a MADeM decision. That is, the maximum amount of time the agent in charge of deciding can wait to receive the others' preferences. If this time is reached, the agent will start the winner determination process with the preference values obtained up to that point.
- `get_timeout(Tout)` : This action returns the bid timeout being used by the agents when performing a MADeM decision.
- `construct_allocations(Task, Slots, ElementValues, Al)` : This action builds the set of possible solutions (allocations) for a decision problem in the format the MADeM model can understand. That is, it gets the task (`Task`) and the slots (`Slots`) to allocate as well as the list of element values (`ElementValues`) to be assigned to each slot and generates the list of all possible assignments (`Al`) of the elements to each *task-slot*. For instance, the following invocation will assign a waiter (either `doug` or `norman`) to the `agent executor` slot of the task `make_coffee`:

  ```
  jmadem.construct_allocations(make_coffee(ag_executor),
                   [ag_executor], [ [doug, norman] ], Al)
  ```

  where `Al` will take the following value:

  ```
  Al=[make_coffee(doug), make_coffee(norman)]
  ```

- `launch_decision(Ag, Al, Uf, Id)` : This action starts a MADeM decision that uses the current internal agent MADeM settings (i.e. personal weights, utility weights, social welfare and timeout), which can be established with the internal actions explained above. Thus, it only gets: (i) a list with the names of the agents involved in the decision (`Ag`); (ii) the allocations being considered as possible solutions (`Al`) and the names of the utility functions being questioned to express agent preferences (`Uf`). Launching MADeM is not a blocking process. That is, this action only launches the MADeM process and returns the decision identifier (`Id`) as its result. Afterwards, once MADeM has been resolved, a belief will be added to the agent's belief base of the type `+madem_result(Id, WinnerAllocation)`, so that the agent can capture the event and handle it.

4. Conclusions and Future work

This paper fully explains J-MADeM, a new open source library oriented to create different types of social simulation agents. The J-MADeM architecture and its integration into Jason have been reviewed, including the definition of the J-MADeM agents and its main parameters. Thus, J-MADeM provides agents with a general market-based social decision-making mechanism, where several utilities (personal and collective), weigths (personal and utility based) and other important functionalities allow the researcher to easily design complex social simulations.

Currently, two application examples are included along with the latest J-MADeM release, which can be downloaded from the Jason sourceforge website [1]. In the first example, we revisit the Gold Miners problem [2], 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. Also, a new multi-agent organization is proposed to better adapt to different gold distributions [11]. In the second example, we test the sociability features provided by J-MADeM, and specifically, the ability to simulate different kinds of societies (e.g. elitist, utilitarian, etc), as well as social attitudes of their members such as, egoism, altruism, indifference or reciprocity. All these features have been demonstrated in a virtual university bar simulation where waiter agents must 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 [10].

Future work will be oriented to incorporate new features to the J-MADeM library. Precisely, we aim at putting the J-MOISE+ organizational model [14] together with J-MADeM to upgrade this important aspect of the MAS (e.g. define the organizational structure, etc.). Finally, we are also interested in introducing an independent normative framework to reinforce the MAS structure.

Acknowledgements

This work has been jointly supported by the Spanish MEC and European Commission FEDER funds under grants Consolider Ingenio-2010 CSD2006-00046 and TIN2009-14475-C04-04.

References

[1] R. H. Bordini and J. F. Hübner. Jason. Available at http://jason.sourceforge.net/, March 2007.

[2] R. H. Bordini, J. F. Hübner, and M. Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, 2007.

[3] 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.

[4] R. Conte and C. Castelfranchi. *Cognitive and Social Action*. UCL Press, London, 1995.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] F. Grimaldo, M. Lozano, and F. Barber. J-MADeM, a market based model for complex decision problems. In *Proc. of CMMSE'09: Computational and Mathematical Methods in Science and Engineering*, 2009.

[12] H. Hexmoor. From inter-agents to groups. In *Proc. of ISAI'01: International Symposium on Artificial Intelligence*, 2001.

[13] J. F. Hübner, J. Sichman, and O. Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In G. Bittencourt and G. L. Ramalho, editors, *Procs. of the 16th Brazilian Symposium on Artifical Intelligence (SBIA'02)*, volume 2507 of *LNAI*, pages 118–128, Berlin, 2002. Springer-Verlag.

[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] J. Sichman and Y. Demazeau. On social reasoning in multi-agent systems. *Revista Ibero-Americana de Inteligencia Artificial*, 13:68–84, 2001.

[18] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.