Using MAS Technologies for Intelligent Organizations: A Report of Bottom-Up Results

Armando Robles[1,2], Pablo Noriega[1], Michael Luck[2], and Francisco J. Cantú[3]

[1] IIIA - Artificial Intelligence Research Institute
Bellaterra, Barcelona, Spain
[2] University of Southampton, Electronics and Computer Science
Southampton, United Kingdom
[3] ITESM Campus Monterrey
Research and Graduate Studies Office, Monterrey, N.L. México
{arobles, pablo}@iiia.csic.es, mml@ecs.soton.ac.uk,
fcantu@itesm.mx

Abstract. This paper is a proof of concept report for a bottom-up approach to a conceptual and engineering framework to enable Intelligent Organizations using MAS Technology. We discuss our experience of implementing different types of server agents and a rudimentary *organization engine* for two industrial-scale information systems now in operation. These server agents govern knowledge repositories and user interactions according to workflow scripts that are interpreted by the organization engine. These results show how we have implemented the bottom layer of the proposed framework architecture. They also allow us to discuss how we intend to extend the current organization engine to deal with institutional aspects of an organization other than workflows.

1 Introduction

This paper reports results on two particular aspects of our progress towards a framework to support knowledge intensive organizations: the design of server domain agents and the implementation of an organization engine.

We are proposing a framework for the design of systems enabled by electronic institutions that *drive* the operation of actual corporate information systems. This is an innovative approach to information systems design since we propose ways of stating how an organization is supposed to operate: *its institutional prescription*, and having that prescription control the information system that handles the day to day operation of the organization: *the enactment of the organization*. We are not restricting our proposal to any particular domain of application but we do have in mind organizations that are self-contained (i.e. with a boundary that separates the organization from its environment) and have a stable character (i. e., whose mode of operation does not change very rapidly). We should also make clear that our proposal is not intended for organizational design, what we are proposing is a framework for the design and deployment of agent-based systems that support already designed organizations. Finally, we should point out that we are designing a framework to be applied to new information systems but as this paper demonstrates we find it is also applicable, with some reservations, to the conversion of traditional legacy information systems.

In our framework we propose a conceptual architecture and the tools to build corporate information systems. The framework we propose is built around the notion of electronic institution (EI) [2] and uses agent-based technologies intensively. Instead of using the notion of electronic institutions to represent and harness only static procedural features —as is currently the case— we propose to extend the notion of electronic institution to capture conveniently more flexible procedural features. In order to capture other non-procedural institutional features of an organization as well, we use the extended notion of electronic institution and develop different sorts of agents and agent architectures —server agents, organization agents and user agents. In addition to those accretions we are also involved in the consequent extension of available tools in order to handle the added expressiveness and functionality.

In previous papers we have outlined the framework [10] and discussed its components from a top-down perspective [11] and reported the first implementation experiences [13]. In this paper we recount our experience with the *agentification* of two existing corporate information systems of the type we want to be able to capture with our framework and discuss how we plan to extend that experience for the intended framework. The experience of agentifying these industrial scale systems had two main outcomes: a family of actual server agents that deal with the knowledge repositories and user interfaces of the two application domains and a rough version of the organization engine that we propose for the full-blown framework.

The paper is organized as follows: after a quick review of relevant background, we present our basic proposal, in Section 3, and in Section 4 what we have accomplished in the bottom-up agentification process. We then discuss why and how we intend to evolve a workflow engine into an organizational engine in Section 5. Finally, in Section 6 we present ongoing work and conclusions.

2 Background

2.1 Organizations

We think of an organization, a firm, as a self-contained entity where a group of individuals pursue their collective or shared goals by interacting in accordance with some shared conventions and using their available resources as best they can [9,7,1]. This characterization focuses on the social processes and purpose that give substance and personality to a real entity and naturally allows to consider people, processes, information and other resources as part of the organization. We choose to use this particular notion in our discourse because at least for the moment we do not want to commit to other organization-defining criteria like sustainability, fitness in its environment or status and substitutability of personnel. We want to focus further in what have been called *k*nowledge-intensive or *intelligent* organizations whose distinguishing feature is the explicit recognition of their corporate-knowledge and know-how as an asset [6].

The everyday operation of an organization consists of many activities that are somewhat structured and that involve personnel, clients and resources of different sorts. It is usual for organizations to manage and keep track of those activities through on-line information systems that are usually called corporate information systems (*CIS*). We will assume that intelligent organizations have *CIS* and we will further assume that corporate knowledge and know-how may be contained in the *CIS*.

Hotels, hospitals and other types of organizations, have conventions that structure or *institutionalize* their activity in consistent ways so that employees and clients have some certainty about what is expected of them or what to expect from each other. These conventions are usually also a convenient way of establishing procedures that save coordination and learning efforts and pinpoint issues where decision-making is regularly needed. These institutional conventions usually take the form of organizational roles, social structures, canonical documents, standard procedures, rules of conduct, guidelines, policies and records; that is, habits and objects, that participants adhere to in a more or less strict way (cf. e.g. [14]).

Our aim is to design a framework that is fit to capture such institutional aspects of an intelligent organization and make them operational as part of its *CIS*.

2.2 Electronic Institutions

We adopt the concept of electronic institution, EI, as defined in the IIIA and specified through the following components: a *dialogical* framework —that defines ontology, social structure and language conventions— and a *deontological component* that establishes the pragmatics of admissible illocutory actions and manages the obligations established within the institution [8].

EI is currently operationalized as EI_0 [2]. In particular, its deontological component is specified with two constructs that we will refer to in the rest of the paper: First, a *performative structure* that includes a network of scenes linked by transitions. Scenes are role-based interaction protocols specified as finite state machines, arcs labelled by illocutions and nodes corresponding to an institutional state. Transitions describe the role–flow policies between scenes. Second, a set of *rules of behavior* that establish role-based conventions regulating commitments. These are expressed as pre and post-conditions of the illocutions admissible by the performative structure.

There is a set of tools (EIDE)[2] that implements EI_0 electronic institutions. It includes a specification language (ISLANDER) generating an executable EI and middleware (AMELI) that activates a run-time EI to be enacted by actual agents.

We want to take advantage of these developments to capture the institutional aspects of an organization and be able to incorporate these aspects as part of a *CIS*. More precisely, we will use EI notions to represent stable institutional activities, roles, procedures and standard documents. We will also take advantage of EI as coordination artifacts to organize corporate interactions according to the (institutional) conventions of the organization. Finally, we will use an extended version of EI_0 in order to specify and implement an organization engine that enacts the institutional conventions of an organization by driving the operation of the components of its CIS.

3 A Proposal for EI-Enabled Organizations

Our aim is to design a conceptual framework to deal with the design and construction of corporate information systems. Since we intend to make such framework applicable for knowledge-intensive *CIS* and we find that the notion of electronic institution is well adapted to this purpose, we are calling it a framework for EI-enabled organizations. We are proceeding in the following manner:

Agentify the components of standard *CIS* with three types of agents owned and controlled by the organization: *server agents*, *user agents*, and *staff agents*.

Encapsulate institutional knowledge as (a) agentified knowledge repositories of different types (business rules, workflow scripts), (b) decision-making capabilities, guidelines or policies that are modelled as staff agents, and (c) the choice of functions that are delegated in staff agents.

Extend the notion of electronic institution to describe and implement properly the significant institutional aspects of an organization.

Build an operative environment where a prescriptive description of an organization governs and reacts with the *CIS* that handles the day-to-day operation of the organization.

Figure 1 (left) outlines the functional architecture of the framework. That functional architecture has been motivated and described elsewhere ([10], [11]), however, for the purpose of this paper we may say that the top layer is a prescriptive definition of the organization that the bottom layer eventually grounds on the components (users, transactions, programs, data) of the business domain. The core is an *organization engine* built around an *EI* that implements and enforces the institutional conventions through a middleware that maps them into the agentified *CIS* in the bottom layer.

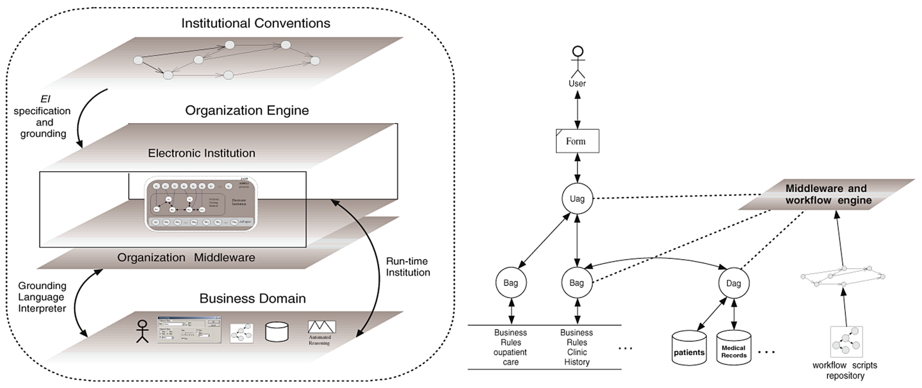
Institutional Conventions
EI specification and grounding
Organization Engine
Electronic Institution
Organization Middleware
Run-time Institution
Grounding Language Interpreter
Business Domain
Automated Reasoning
User
Form
Uag
Middleware and workflow engine
Bag
Bag
Dag
Business Rules oupatient care
Business Rules Clinic History
patients
Medical Records
workflow scripts repository

Fig. 1. Architectural diagram of the proposed framework (left) and the implemented workflow engine for the outpatient care system involving *U*ser, *B*usiness rule and *D*atabase server agents(right)

– *The electronic institution layer* implements the normative specification using as input the *performative scripts* produced by the *EI* specification language. The run-time functionalities of this layer are similar to those of AMELI [3,2] since it runs in close interaction with the organization middleware and it guarantees that all interactions comply with the institutional conventions.
– *The organization middleware layer* contains a *grounding language interpreter* and uses it to pass *grounding language commands* from the run-time EI to the CIS components and map actions in the CIS with state transitions in the run-time EI.

Thus, the grounding language is used to specify the sequencing of instantiation of *performative scripts* as well as agent behaviour in order to manage interactions with the actual *CIS* elements: users, interfaces and knowledge repositories. The basic functions of this middleware layer are:

- to log users into the organization, controlling user roles, agent resources and security issues.
- to monitor user interaction,
- to execute the *grounding language* interpreter,
- to implement interaction devices[1], and
- to control the actual mappings between the *grounding language* interpreter and domain entities.

4 A Bottom-Up Approach

4.1 The Agentified *CIS*

We have approached the design of our framework from both ends. The top-down approach is centered in the theoretical and computational extensions of the EI_0 developments (c.f. [11]). The bottom-up approach that we explore in this paper has consisted in the agentification of two *CIS* in actual operation, and the design and implementation of a rudimentary organization engine that is currently a workflow engine proficient enough to drive the MAS-ified operation of those two *CIS*.

The systems that we have been working with are integral vertical industry systems. One system implements the full operation of large hotels: reservations, room assignment, restaurant services, accounting and billing, personnel, etc. The other implements the full operation of a hospital: outpatient care, nurse protocols, pharmacy, inventory control, electronic medical records and so on. They have been developed by Grupo TCA and have been in an evolving operation for almost 20 years.[2]

Over the last 5 years, TCA has been modifying its hotel information system to facilitate its agentification. It is now a consolidated set of business rules available to a middleware workflow engine that reads workflow scripts and delegates concrete tasks and procedures to participating user and server agents. This modestly *MAS-ified CIS* whose architecture is reported in [13] is already operational in 15 hotels. In the health care domain, TCA has *MAS-ified* the outpatient care subsystem [12] as a first step for the agentification of their integral hospital information system. These two *MAS-ified CIS*, show how we intend to put our proposal to work and, as we report below, the experience brought to light many issues that should be taken into account for the development of our framework.

[1] For those domain entities that need to be in contact with external agents we have developed a special type of server agent that we call *interaction device*. These devices implement interfacing capabilities between external users and other domain elements, e.g. form handling, data base calls, business rule triggering.

[2] TCA is a medium-size privately owned information systems company that has been active in the design and development of integral information systems since 1982 for the Latin American market.

4.2 The Workflow Engine

The workflow engine (WF-engine) is currently operational and implements a restricted version of the main component of the organization middleware: the organization engine. Once initiated, WF-engine reads a workflow script from a repository and interprets the commands contained in it. The commands are executed in the sequence dictated by the workflow conditional directives, and each command triggers the inter-operation of server agents that control domain components —data bases, business rules, forms— and their interaction with human users.

Figure 1 (right) illustrates how the workflow engine supervises the agents that handle specialized domain components, such as databases or business rule repositories — a specialized *business rule* server agent (Bag) fetches, from a central repository, business rules that use data provided by another specialized *database* server agent (Dag), to provide input to a *user agent* (Uag) that displays it in a user form.

Each workflow specification is stored in a repository as a workflow script. Since each domain component is represented in the environment by a specialized *server agent*, we have implemented commands for sending requests to the corresponding *server agents* for their execution of business rules, data base accesses, reports definitions, and for end-user interactions.

Each task specified in a protocol is implemented as one of the following domain actions:

- a business rule, that could be as simple as a single computation or as complex as a complete computer program;
- a data base access to add, delete, modify or retrieve information from a data base;
- a user interaction through a specialized form; or
- a reference to another workflow script.

We have built an interpreter that takes a workflow script and produces a set of actions. This implementation involves activation of server and user agents of different types, the sequencing of their actions and the parameter loading and passing during those actions. The interpreter uses the following commands:

- read workflow specification script,
- initialize variables,
- load defaults for variables and data, and
- execute workflow commands.

Initially, the workflow interpreter reads the main workflow script and starts executing the specified commands, controlling and sequencing the interaction between the intervening agents as well as loading and executing other possible workflow scripts specified in the main workflow.

Here is a workflow script segment used in the Back Office module of the Hotel Information System to implement the task of adding a supplier.[3] The script specifies the coordination of interactions between database, business-rule and user agents (who use specialized forms).

[3] Script and business rules are taken from TCA's `hot500.wf` and `hot500.br` repositories.

```
Procedure AddSupplier
  begin
      InitializeVariables;
      Interact(UserAgent(DefineGrid(grid01)));
      Interact(UserAgent(InputFields(grid01,Supplier)));
      Interact(BRServerAgent(ConsistencyCheck));
      Interact(DBServerAgent(Suppliers,New));
  end
```

WF-Engine Functional Features. *Agent mediated interactions* The WF-engine acts as a link between the user interface and the data base and business rule repositories, but all interactions are mediated by ad-hoc server agents.

Specialized server agents for domain components. The main function of the specialized server agents is to act as a business domain components (including business rule repositories and data bases) facilitators for all user agents that may be logged-in at several client sessions.

The user interface is mediated by a user agent that is regarded as a client for the business rule and data base server agents.

Persistent communication. Once the interaction between a *user agent* and a *server agent* is established, the infrastructure makes sure that the communication between both agents is persistent until one of the agents decides to terminate it.

Business rule triggering. As shown in the previous examples, workflow scripts are currently not much more than sequences of conditional clauses that invoke, through specialized agents, the activation of specific business rules. Business rules are special-purpose programs, stored in a repository that may be accessed by a business rule agent who is able to trigger rules and use the results of such triggerings. Business rule agents (BRagents) react to workflow transitions by requesting business rule inputs either from database server agents who query a data base or from user agents that read input from a user form. With those inputs, the BRagent triggers a rule whose result is passed to a data base server agent or a user agent or, more frequently, is used by the BRagent to change the workflow state.

WF-Engine Programming Functionalities. *Context of interaction.* The system programmer is responsible for maintaining the context of all agent interactions because as agent interactions evolve, they modify the context of the world, updating data and status variables as required.

Precedence of execution. During workflow execution, event value verification takes precedence over sequential process execution; that is, in the middle of a conditional execution, it is possible to break the sequential flow and skip directly to the first command of another conditional clause.

Workflow scope of execution. Regarding the scope of workflow execution, once a flat form or grid is addressed, all subsequent workflow commands will be made in the scope of that specific flat form or grid, until another one is addressed.

Scope of variables. Global variables are available in the scope of the workflow definition, that is, in the workflow specification the programmer can test for the value of variables defined as global by any *server agent*. It is the programmer's responsibility to define and maintain the proper scope for the required variables.

WF-Engine Limitations. The WF-engine has no control over *what is said between agents*. Because of the way workflow scripts are currently implemented, it deals only with specific conditional commands that test for contextual changes represented by changes in data and status variables. This is an important limitation whenever we want to deal with complex interactions, because we are forced to "hardwire" the control code for the execution of alternative procedures in the workflow script or in the business rules it involves.

In the WF-engine we implemented in this experiment we designed specific commands that deal with the transfer of data between the workflow engine and user or server agents. While it is natural to transfer information as data, the transfer of *control* data that may alter or modify agent behavior is undesirable but due to the limited expressiveness of workflow scripts we had to implement it in the WF-engine. We have used working memory to pass control data, but this use entails the messy problem of dealing with global variables and thus imposing severe restrictions on agent autonomy.

In the implementations described here we only use *reactive* agents. Such a primitive implementation is enough for the current needs but we may readily change their specification to involve more sophisticated behavior to take advantage of more sophisticated organization engines.

5 From WF-Engine to O-Engine

Lessons Learned. Our experience of *MAS-ifing* two *CIS*, has brought to light many pertinent issues for an organizational engine. The main lessons are:

Complexity trade-off. Considering the agentification of systems with equivalent functionalities, our experience with the MAS-ification shows that when business rules capture much of the discretional (rational) behaviour of agents, it is enough to use simple procedural rules to implement those procedures where the business rules are involved. Conversely, as business rules become simpler, the procedural requirements become more involved, the need for agent discretional behaviour is increased, and the need for handling agent commitments arises. The more "atomic" the business rules are, the more complexity is needed to handle them, both in the flow of control and in the agent behavior.

Agent commitments. These two experiments have also shown that if we do not have a structural mechanism to control the commitments generated through agent interactions, we need to hard-wire the required program logic to keep track of pending commitments inside each agent, as part of the workflow or inside some business rules. Assume that agent a performs an action x at time t and establishes a commitment to do action y at time, say $t+3$. If action x is implemented as a business rule, then we must have a mechanism to send a return value to the $BRagent$, or some way to set a variable in some kind of working memory.

Viable approach. We have described how we *MAS-ified* two *CIS*. In the process, we have outlined the construction of the required server and user agents, have developed the required business rules, and specified the workflow needed for the appropriate sequence of execution between the intervening agents. In this sense we have been able to implement two CIS that correspond roughly to the type of EI-enabled CIS we want to build with our framework.

Even though the WF-engine is an embryonic version of an organizational engine and workflow scripts are clumsy parodies of EI-performative scripts, we have shown that specialized server agents, knowledge repositories and display devices may be driven by a prescriptive specification and some intended benefits are already available even in this rudimentary examples:

- We found considerable savings in software-development time and effort avoiding duplicate code by building business rule server agents and business rule repositories, since the same agent scheme can exploit similar repositories.
- We ensured *problem separation* at system design time, allowing domain experts to define the appropriate workflow and leaving to the engineer the task of engineering server agent behaviour. By having business rules managed by server agents, the problem is reduced to implementing some control over these agents.
- Separating workflow and business rule definitions from business rule and workflow control begets a considerable simplification of the system upgrading process. This simplification allows us a glimpse at the possibility of having dynamic behavior in the CIS prescription.

Additional Functionality for the O-Engine. In our framework, we want to be able to prescribe what the valid interactions among agents are. We have decided that the only valid mechanism for agent interaction —to communicate requests, assertions, results— should be illocutions. Hence, instead of using working memory, we need a proper grounding language and a mechanism to control agent illocutions and the ensuing commitments over time. This suggests us the use of production rules and an inference mechanism that will be used to define and operate the institutional conventions of performative scripts and also to load knowledge bases of staff agents.

We need to design a proper *grounding language* to map the sequencing and instantiation of *performative scripts* and server agents illocutions in order to manage interactions with the domain components.

How to Define and Implement the O-Engine. In order to address the issues mentioned in this section, we need to change the definition of a workflow engine into a more sophisticated organization engine that handles performative scripts —that capture more information than workflows— illocutory interactions and dynamic agent commitments.

We will implement this required functionality by extending the concept of Electronic Institution. In fact each *performative script* is built as an electronic institution and an extension of the current machinery for transitions is used to intertwine the scripts. We will also need to extend the expressiveness of ISLANDER by having sets of modal formulae (norms) as a way of specifying performative scripts [4,5]. The grounding language will be a gradual extension —as we increase the functionality and autonomy of server agents— of the primitive commands that we use to load WF scripts and sequence the interaction of intervening agents and their calls to business rule and databases that we now hide in the WF interpreter. Once the *performative script* is modelled and specified (using an extension of IIIA's ISLANDER tool), it is saved in a *performative scripts* repository. The organizational engine reads and instantiates each *performative script* as needed.

The current EI_0 operationalization of EI [2] will be taken as the starting point for these purposes but we are extending it to have a better representation of organizational activities, the extended functionality and a leaner execution.

6 Final Remarks

Recapitulation. In [10] we took a top–down approach for the definition of a framework for enacting intelligent organizations. We proposed having a prescriptive specification that drives the organization's information system day to day operation with an organizational engine based on electronic institutions. In this paper we report our experiences with a bottom-up approach where we tested and proved adequate a rudimentary version of the proposed framework. In this paper we also discussed how we expect to attain the convergence of the top–down and bottom–up approaches by, on one hand, transforming the WF-engine that is now functional in two industrial-scale *CIS*s into an *organization engine* that may deal with more elaborate organizational issues and, on the other hand, implementing the extensions of EI_0 that the organizational engine entails.

Programme. Our intention is to be able to build and support large information systems that are effective and flexible. What we are doing is to device ways of stating how an organization should work and, in fact, making sure that the prescribed interactions are isomorphic with the actions that happen in the information system. We realize that there is a tension between the detailed specification of procedures and the need to a continuous updating of the system and since we know that the ideal functioning will be changing, we want that the actual operation changes as well. In order to achieve this flexibility we are following three paths:

- Making the information system *agent-pervasive*. This way we make sure that all interactions in the CIS become, in fact, illocutions in the organizational engine, and then we may profit from all the advantages that electronic institutions bring about to express complex interaction protocols and enforce them.
- Simultaneously we are going for *plug-able components* —performative scripts, business rule and knowledge repositories, server agents, user agents— that are easy to specify, assemble, tune and update so that we can use them to deploy interaction protocols that are stable, quickly, and thus allowing us to update these protocols parsimoniously.
- We count on *staff agents* that are reliable and disciplined (since they are part of the organization) and, because they may have better decision-making capabilities and because we can localize their knowledge, we can build into them the flexibility needed to accommodate less frequent interactions or atypical situations (and thus simplify interaction protocols) and also to accommodate more volatile conventions (and thus save us from more frequent updates).

We entertain the expectation that we will be able to incorporate autonomic features into our systems.

Next steps. An outline. In the top-down strategy we are (a) looking into the formal and conceptual extensions of the EI_0 so that we may handle complex performative structures and assemble them from simpler performative scripts. (b) Devising ways of expressing deontological conventions declaratively, so that we may specify performative scripts declaratively and logically enforce them. (c) Defining the guidelines for a grounding language that translates EI manageable illocutions into CIS components actions.

In the bottom-up approach we will (a) start enriching server agents so they can interact with EI_0 performative structures, with "more atomic" business rules and with the other application domain entities. (b) We will also develop user agents and interaction devices further, so that we have better access and control for external users of the system. (c) We will also start implementing actual performative scripts, staff agents and appropriate business rules, on one side, and a grounding language to handle their interactions on the other. (d) We will extend the current WF-engine to handle (c).

In the implementational front we foresee (a) a prototype organization engine, built on top of EIDE, to handle the bottom-up developments. (b) An extension to the ISLANDER (ISLAplus) tool to handle the new expressiveness of the organizational engine. (c) A leaner version of EIDE that instantiates an ISLAplus specification into an organization engine and enacts it on a CIS.

Acknowledgments

This research is partially funded by the Spanish Ministry of Education and Science (MEC) through the Web-i-2 project (TIC-2003-08763-C02-00) and by private funds of the TCA Research Group.

References

1. Howard E. Aldrich, editor. *Organizaions and Environments*. Prentice Hall, 1979.
2. Josep Lluis Arcos, Marc Esteva, Pablo Noriega, Juan A. Rodríguez-Aguilar, and Carles Sierra. Environment engineering for multiagent systems. *Engineering Applications of Artificial Intelligence*, (submitted), October 2004.
3. Marc Esteva, Juan A. Rodriguez-Aguilar, Bruno Rosell, and Josep Lluis Arcos. AMELI: An agent-based middleware for electronic institutions. In *Third International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS'04)*, pages 236–243, New York, USA, July 19-23 2004.
4. Andres Garcia-Camino, Pablo Noriega, and Juan Antonio Rodriguez-Aguilar. Implementing norms in electronic institutions. In *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.
5. Andres Garcia-Camino, Juan Antonio Rodriguez-Aguilar, Carles Sierra, and Wamberto Vasconcelos. A Distributed Architecture for Norm-Aware Agent Societies. In *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems. Declarative Agent Languages and Technologies workshop (DALT'05)*, 2005. (forthcoming).

6. Jay Liebowitz and Tom Beckman. *Knowledge Organizations*. Saint Lucie Press, Washington, DC, 1998.
7. James G. March and Herbert A. Simon. *Organizations*. John Wiley and sons, New York, USA., 1958.
8. Pablo Noriega. *Agent Mediated Auctions: the Fishmarket Metaphor*. PhD thesis, Universitat Autònoma de Barcelona (UAB), Bellaterra, Catalonia, Spain, 1997. Published by the Institut d'Investigaci en Intelligncia Artificial. Monografies de l'IIIA Vol. 8, 1999.
9. Douglas C. North. *Institutions, Institutional change and economic performance*. Cambridge Universisy press, 40 west 20th Street, New York, NY 10011-4211, USA, 1990.
10. Armando Robles and Pablo Noriega. A Framework for building EI–enabled Intelligent Organizations using MAS technology. In M.P. Gleizes, G. Kaminka, A. Nowé, S. Ossowski, K. Tuyls, and K. Verbeeck, editors, *Proceedings of the Third European Conference in Multi Agent Systems (EUMAS05)*, pages 344–354., Brussel, Belgium, December 2005. Koninklijke Vlaamse Academie Van Belgie Voor Wetenschappen en Kunsten.
11. Armando Robles, Pablo Noriega, Francisco Cantú, and Rubén Morales. Enabling Intelligent Organizations: An Electronic Institutions Approach for Controlling and Executing Problem Solving Methods. In Alexander Gelbukh, Álvaro Albornoz, and Hugo Terashima-Marín, editors, *Advances in Artificial Intelligence: 4th Mexican International Conference on Artificial Intelligence, Proceedings ISBN: 3-540-29896-7*, pages 275 – 286, Monterrey, NL, MEX, November 2005. Springer-Verlag GmbH. ISSN: 0302-9743.
12. Armando Robles, Pablo Noriega, Michael Luck, and Francisco Cantú. Multi Agent approach for the representation and execution of Medical Protocols . In *Fourth Workshop on Agents Applied in Healthcare (ECAI 2006)*, Riva del Garda, Italy, Aug 2006.
13. Armando Robles, Pablo Noriega, Marco Robles, Hector Hernandez, Victor Soto, and Edgar Gutierrez. A Hotel Information System implementation using MAS technology. In *Industry Track – Proceedings Fifth International Joint Conference on AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS (AAMAS 2006)*, pages 1542–1548, Hakodate, Hokkaido, Japan, May 2006.
14. Pamela Tolbert and Lynn Zucker. chapter The Institutionalization of Institutional Theory, pages 175–190.