Design of Forces Driving Adaptation of Agent Organizations

Sergio Esparcia,[1] Olivier Boissier,[2] and Estefanía Argente[1]

[1] Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n, 46022 - Valencia (Spain)
{sesparcia,eargente}@dsic.upv.es
[2] Fayol Institute
École Nationale Supérieure des Mines de Saint-Étienne
158 Cours Fauriel, 42023 - Saint-Étienne (France)
olivier.boissier@emse.fr

Abstract. Adaptation is an important feature of human organizations. Being able to change allows them not only to survive, but to evolve to get new advantages from new situations happening in their environment or from inside the organization. The same way human organizations do, agent organizations should be able to adapt. Even if adaptation is addressed in the literature, it lacks the ability to clearly manage the reasons for change. These reasons are known in the social science bibliography as forces that drive the organizational change. These forces were introduced in a previous work in the computational domain, but only for the analysis phase of the engineering of agents organizations. In this work, a set of templates is presented to define these forces at design time. These templates have been applied in the design of components for detecting the 'obtaining resources' force, which have been implemented using Jason agents and CArtAgO artifacts within an agent organization.

Key words: multi-agent systems, agent organizations, adaptation, driving forces

1 Introduction

As stated in the studies of human organizations by Organizational Theory[3] (OT) [15], these structures are dynamic and able to adapt at runtime. The OT is one of the inspirations for Organization-Centered Multi-Agent Systems[4] (OCMAS) [10] developers. Multiple proposals have been presented to design and implement such systems, like [7, 8, 11], most of them focusing on the way the change is done, or the cost of this change in the organization. However, they do not take into account the reasons that make human organizations to change, leaving aside the forces that drive organizational change, which are an important concept when dealing with adaptation. These forces have been widely

[3] Organizational theory is the sociological study of formal social organizations, such as businesses and bureaucracies, and their interrelationship with the environment where they operate.

[4] Organization-Centered Multi-Agent Systems are Multi-Agent Systems where organizational elements (such as structure, goals, roles, norms, etc.) are explicitly defined.

studied by Social Sciences researchers such as Aldrich [2] and Lewin[5] [13]. Aldrich classifies forces into *external* and *internal* forces, depending on where the pressure for change comes from. A force is external if the reason for change comes from the organizational environment, and internal if this reason comes from inside the organization. Moreover, forces have been introduced in [9] for the analysis of OCMAS where two sets of guidelines for expressing a force are proposed: (i) *condition factors* detecting the action of the force and (ii) *solutions* for reacting to the force in the organization.

The objectives of this paper are: (i) to take the existing guidelines described in [9], which are used for the analysis of the forces, to a more formal description that works as a design step, (ii) to show how this description facilitates the implementation of forces, and finally (iii) to implement the condition factors and solutions of the forces as adaptation mechanisms distributed among agents (implemented as Jason agents [6]) and artifacts (implemented using the CArtAgO platform [16]), which can be accessed by Jason agents. Both Jason and CArtAgO are part of the JaCaMo framework [5]. Due to the lack of space, we focus on the force named **'Obtaining resources'** [2] at the design and implementation phases of an agent organization. It is an external force that states that a failure when obtaining resources can drive to an organizational change to guarantee organizational survival. Therefore, it could be necessary for organizational survival to improve the way in which resources are obtained. For example, by extending the organization to a place where resources are easily obtained, or by reaching an agreement with another organization that has a better access to the required resources.

The rest of this paper is structured as follows: Section 2 positions our contribution in the context of previous works on adaptation in agent organizations. Section 3 presents the templates to describe a force at the design time. Section 4 presents the room allocation case study. Section 5 presents the definition of a force at the design time, using the **Obtaining resources** force as an example. Section 6 describes the implementation of this force. Finally, Section 7 presents the conclusion of this paper as well as the future work on this topic.

2 Adaptation in Agent Organizations

Several works deal with adaptation in OCMAS. This section focuses on some of them to highlight the main directions that have been explored so far. Some works approach adaptation from the knowledge and skills required by the agents. For instance, DeLoach *et al.* [7] define adaptive organizations as distributed systems that can autonomously adapt to their environment thanks to organizational knowledge, based on the current goals and capabilities. In [12], the adaptation is considered from the coordination point of view by defining a *reorganization group* composed of different roles,[6] responsible of executing the *reorganization scheme*, a plan to realize the adaptation process.

[5] Lewin states that change is only carried out if the forces supporting the change are stronger than the forces against the change.

[6] *OrgManager* role is in charge of managing the adaptation process, *Monitor* role monitors the organizational activity, *Historian* role maintains history of the organization, *Designer* role analyzes the organization so as to identify problems and propose alternatives, and *Selector* role is in charge of selecting one of these alternatives.

Other works focus on the adaptation process itself seeing it as a state transition problem. For instance, in [8], authors propose a formal semantics framework where adaptation is treated as a design issue where changes of the organization are represented as transitions between states. Two activities are considered to realize this adaptation: (i) evaluation of the current organizational state, computing its 'distance' to the desired state; and (ii) change of organizational elements (structure, agent population, objectives) in order to achieve the desired state. The proposed strategy to decide about adaptation is based on the cost of this adaptation. A similar cost-based adaptation framework is also proposed in [1]. The costs are based on concepts like *Organization Transition Impact* and *Organization Utility*. They propose a Multi-dimensional Transition Deliberation Mechanism (MTDM) where three types of transition are considered, depending on the organizational dimension that changes: role reallocation transition, acquaintance transition, and agent population transition.

Even though these works are proposing complementary and interesting approaches for dealing with adaptation, they mainly focus on the management of the process to carry the changes out. They do not specify the reasons for change which is an important topic when dealing with adaptation. For instance, in [11] authors present a model for organizational change which states that a change in an organization is provoked by two opposing forces: *resistant* forces and *driving* forces towards a new organization. Based on these forces, authors propose a change along a three-phase process: (i) *unfreezing phase*, where the driving forces are stronger than resistant forces; (ii) *movement phase*, where all the changes are carried out; and (iii) *equilibrium*, after all changes have been deployed, where resistant forces are stronger than driving forces.

However, the authors of [11] do not specify the reasons of these forces to appear in an organization. Knowing the specific reasons of the forces facilitates the task of offering better solutions to the problems caused by such forces. Since the OT studies these forces, we are working on their definition into the OCMAS domain. After having defined a set of guidelines for the analysis phase in [9], we propose in this paper to go a step further in this direction by proposing templates to identify, describe, and implement these forces.

3 Templates to define forces

Force detection has to be carried out along the organizational life-cycle. For that purpose, guidelines may help to develop tools to identify the factors and the solutions of forces. The factors express the conditions making possible to state if the force is currently active or not. The solutions express the actions to execute in the organization in order, either to take advantage of the benefits that the force may imply, or to minimize the possible damages produced by the force in the organization. The guidelines presented in [9] focus on the analysis step, identifying and describing the factors and solutions using plain text and include one table for factors and force description, and another table depicting the solutions. In this paper, since it is addressing the design phase, we propose templates[7] for identifying and describing the factors and solutions of forces.

[7] We use the word template to differentiate the products of the design step from the guidelines produced at the analysis step, and also from the design patterns from software engineering.

Their contents are more formal, closer to an actual implementation. In this case, a force is defined by means of a template, where the factors and solutions are pointed out. Both are fully described in separate tables making possible to reuse a factor or a solution in the definition of another force.

A force (Table 1) is defined by its name, a textual description, a type stating if the force is internal or external, a set of factors participating on the detection of the action of the force, a force detection condition, a set of solutions and a selection criteria among the solutions. The force detection condition is a boolean expression bearing on the factors.

Field	**Description**
Name	Name of the force.
Description	Textual description of the force acting over the organization.
Monitor	The role of the organization responsible of monitoring the force.
Type	*Internal* or *external*.
Factors	Names of factors involved in the detection of the force.
Force detection condition	Logical combination of factors stating that the force is in action.
Solutions	Solutions that can be applied in case the force is active.
Solution choice criteria	Depicts how a solution is chosen.

Table 1. Force description template

Since we are defining templates to be filled along the organization definition, rather than guidelines, references to the organizational model such as roles, goals, etc. may appear in the definition of the factors and solutions.

3.1 Factors Stating that a Force is Acting

Table 2 defines the components of the factors for expressing the conditions testing that a force is acting over the organization. A factor is a set of monitoring mechanisms in the organization to detect if the force is acting, and it is characterized by its name, a description, the parameters referring to organizational values, and the condition which states when the factor is active.

Factor	
Name	The name of the factor that helps identifying the force.
Description	Textual description of the factor.
Parameters	Organizational elements concerned by the action of the force, which help in the detection of its action.
Condition	The condition stating the relations among the parameters that help in the detection of the action of the force.

Table 2. Force factor description template

3.2 Solutions to face the Force

Table 3 defines the actions that should be carried out in the organization in order to take advantage or to prevent damage from the force that has been detected. Each solution is described by a name, a textual description, a condition that points out the particular factors that need to be satisfied to execute this solution, the parameters involved in the actions of the solution, the actions to execute, and the roles of the organization that will be in charge of executing the solution.

Field	Description
Name	Name of the solution.
Description	Text describing this solution.
Condition	The condition (related to factors) that must hold in order to apply this solution.
Parameters	Describes the elements that have to be known prior to apply the solution.
Actions	The set of actions that must be carried out to apply this solution.
Cost	The cost of applying this solution to the organization.
Roles	The responsible roles for applying this solution.

Table 3. Force solution description template

4 Case study

A case study is employed to illustrate the use of the templates from the previous section. This case study focuses on how to manage the distribution of activities assigned to the different rooms of a smart building in a university.[8] The three types of activities that a room can carry out are: teaching, meeting, and brainstorming. Fig. 1 represents the organizational model issued from the analysis phase. This model is based on the graphical notation used by the GORMAS[9] methodology [4].

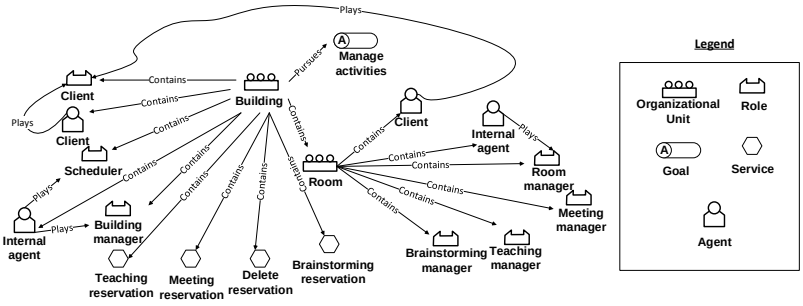

Fig. 1. Structural dimension of the building represented with the GORMAS notation.

As it can be seen in Fig. 1, the organization is composed of a **Building** organizational unit, which contains **Room** units, as many as needed. Each Room unit represents the way agents and services governing the room will be organized. It contains an Internal agent that plays the **Room Manager** role and one of the following roles: **Teaching Manager**, **Meeting Manager**, **Brainstorming Manager**. At the system initialization it only plays the Room Manager role, which is in charge of managing utilities, equipment, and other tasks related to the room management. With the objective of specifying the activity to be carried out in a specific Room at a specific time, the Internal Agent also plays one of any of the other three roles, which are exclusive between them, since

[8] This case study is inspired in [17].

[9] GORMAS is an agent-oriented software engineering methodology for the analysis, design, and implementation of OCMAS. It has been chosen because the authors have a high knowledge of it, thus facilitating the definition of the case study.

a Room can only develop one type of activity at the same time. In each Room, there can be **Client** agents that request activities. The type of activities of each Room is changed dynamically depending on the requests issued by the Client agents. All Rooms are equipped to be able to develop any of the three types of activities at any time.

Additionally, the Building unit also contains the roles **Client** (played by the Client agents), and **Scheduler** and **Building Manager**, played by Internal Agents. The Building Manager specifies the type of the activities to be carried out in each Room by assigning a specific role (Brainstorming, Meeting, or Teaching Manager) to each Internal Agent playing the Room Manager role. The agents populating the Building unit are in charge of achieving the Manage Activities goal to assure a correct organizational performance. Finally the Building Unit is composed of one service for deleting existing reservations and three types of reservation services: **Teaching Reservation**, **Meeting Reservation**, and **Brainstorming Reservation**.

The scheduling of activities that are carried out in the Rooms of the Building are controlled by the Scheduler. The Clients of the Building (external agents) send their petitions (that include the type of room, the day, the start time and the duration of the activity expressed in number of hours) to the Scheduler, who is responsible of assigning a specific activity to a room at the required time. In this application, the organization may be subject to many sources of change. For example, the number of requests received by the organization or the number of clients populating the organization are two of these sources.

In this paper, we focus on the aforementioned **Obtaining Resources** external force. In this case study, the resources are considered to be the rooms. Access to the rooms is managed by the reservation services. Therefore, it is considered that the access to a resource fails if the access to the service that manages the Room fails. The solution for this force implies changing the role of some Room Managers, or adding new virtual Rooms to the organization.

5 Description of Forces at the Design Step

In this paper we want to go one step further from [9] where the forces were described by means of the guidelines using plain text descriptions. Following the templates defined in Section 3, forces will be described using a formal language, which will connect the description of the forces with the organizational definition in GORMAS notation. Each element of the organizational model has associated properties or functions. They are accessed using the notation *element.property* (for accessing properties) or *element.function(parameters)* (for accessing functions). This section presents the design phase, where the templates to describe a force are filled, making references to the organizational model presented in Fig. 1. The following subsections define the *Obtaining resources* force, including the factors and solutions. Section 6 describes how an implementation of the case study has been carried out, including elements for dealing with the Obtaining resources force.

Additionally, aside from the functionalities of the organizational elements, generic functions refer to the actions for adding or removing organizational elements (e.g. *AddRole, DeleteOrgUnit, GetRole, LeaveRole, etc.*). To structure these actions in an adap-

tation process one may use sequence operator (;), choice operator (|), parallel execution (||), optional execution ([ ]), iteration (a^n) to order the different actions when adapting.

5.1 *Obtaining resources* force at the design time

In the room allocation scenario, the services that are considered to check if the Obtaining resources force is triggered or not are the services to make a reservation (one for each type of room). A service is considered to fail if the request for a service is not fulfilled because there are not enough available rooms. In this case, it is necessary to modify the type of activity the rooms have been assigned to allow the organization to satisfy the requirements of their Clients. This operation is done by modifying the role assigned to one of the Room Managers whose controlled Room has an empty schedule. The Obtaining Resources external force is described in Table 4. Its triggering factor is $FailedServiceCallsRate$ and two solutions are defined: $ChangeRoomActivity$ and $ExtendBuilding$. Their description is depicted in the next subsections.

Field	**Description**
Name	ObtainingResources
Description	A resource cannot be allocated by a service of an organization.
Monitor	Monitor element
Type	External
Factors	$FailedServiceCallsRate$
Force Detection Condition	$FailedServiceCallsRate.Condition = TRUE$
Solutions	$sol_1 = ChangeRoomActivity$, $sol_2 = ExtendBuilding$
Solution choice criteria	$max(utility_{sol1}, utility_{sol2}, utility_{noSol})$

Table 4. Definition of the Obtaining resources force, design step

FailedServiceCallsRate Factor. This factor (Table 5) takes into account the failures of the reservation services in a time duration dur and is activated if failure rate is higher than 1 - QoS. The Quality of Service (QoS) [14] defines the expected success rate when calling a service. For example, in the case of having a QoS of 90%, the maximum allowed failure rate is 10%. Each service has a different QoS, so the factor will be differently triggered depending on each service.

To detect whether the factor is active in the organization, the number of requests for activities and the number of failures of such requests by the Reservation service are considered. In our application, a request is defined as a tuple $r = \{type, time, status\}$. Let us define the set $R_{dur,res} = \{r | r.type = res \wedge res = \{teaching, meeting, brainstorming\}\}$ as the set of requests received by the (teaching, meeting, or brainstorming) reservation service in the period of time dur, and the set $R'_{dur,res} = \{r \in R_{dur,res} | r.status = fail\}$ that records the number of failures on requests to the (teaching, meeting, or brainstorming) reservation service during the same period of time. The failure rate for the service $Reservation$ for a time period dur is calculated as:

$$Monitor.Failures(Reservation, dur) = \begin{cases} \frac{|R'_{dur,Reservation}|}{|R_{dur,Reservation}|} & : |R_{dur,Reservation}| \neq 0 \\ 0 & : |R_{dur,Reservation}| = 0 \end{cases} \quad (1)$$

where *Reservation* is a parameter representing the type of reservation service to be checked (i.e., $TeachingReservation$, $MeetingReservation$, or $BrainstormingReservation$) according to our organizational model. Then, if the failure rate is higher than the expected one, it is necessary to apply one of the two possible solutions, described in the next subsection.

Factor	
Name	FailedServiceCallsRate
Description	If the failed service calls rate (i.e., requesting a spot for developing an activity) is higher than the allowed failure rate threshold, then the force is considered as acting.
Parameters	$Reservation \in \{TeachingReservation, MeetingReservation, BrainstorminReservation\}$, dur
Condition	$Monitor.Failures(Reservation, dur) > ((1 - Reservation.QoS) * Monitor.Requests(Reservation, dur))$

Table 5. Description of the FailedServiceCallsRate factor, design step

Applying the solution of the force After the detection of the force, it is necessary to take a decision about the adaptation action to develop into the building. In our use case, the Obtaining Resources Force defines two possible solutions: ChangeRoomActivity and ExtendBuilding (cf. Table 6 and 7). As can be seen in the solution choice criterion of the force in table 4, the solution which provides the highest utility will be deployed.

The first solution (cf. Table 6) consists, as expressed in the field 'Action', in the modification of the type of activity being developed inside one or more rooms, to get free spots in the schedule for developing activities. Once the FailedServiceCallsRate is active, the Scheduler builds the *RoomManagerList* set, containing pairs of the form $\langle rm_i, nr_i \rangle$ containing the Room Manager agent (rm_i) of Room_i whose role is required to change, and the new role that the agent will take (nr_i). This gives the Room_i the opportunity to host a new type of activities. This solution is, in most situations, the less costly. This cost is calculated as:

$$cost(ChangeRoomActivity) = \sum_{\forall \langle rm_i, nr_i \rangle \in RoomManagerList} (CostPlay(nr_i) + CostChange(rm_i.Role, nr_i) - CostPlay(rm_i.Role)) \quad (2)$$

This is, for each room that might change its type of activity, it is calculated the cost of having the new role in the organization compared to the cost of the current role the room manager is playing, and also the cost of changing from one role to another. Playing a role has a cost because depending on the role, a different subset of the room equipment is used, thus having different costs in terms of energy consumption, etc.

The second solution proposed by this force (cf. Table 7) is to extend the building with more virtual rooms to make possible more activities in it at the same time. This change will suppose adding one or more Room organizational units inside the building. The types of the rooms to be added are described inside the *NewRoles* set. Each role in this set corresponds to a type of room to be added. Therefore, for each role nr_i in the set, the BuildingManager adds a new room $Room_i$ to the building by using the function

Field	Description
Name	ChangeRoomActivity
Description	The type of activity being carried out in a room is modified by changing the role of the room manager agent.
Condition	FailedServiceCallsRate
Parameters	$RoomManagerList = \{\langle rm_i, nr_i\rangle$, where $rm_i \in RoomManager \wedge$ $nr_i \in \{MeetingManager, TeachingManager, BrainstormManager\}$
Actions	$\forall \langle rm_i, nr_i\rangle \in RoomManagerList : rm_i.LeaveRole(Role); rm_i.GetRole(nr_i)$
Cost	$\sum_{\forall \langle rm_i, nr_i\rangle \in RoomManagerList} (CostPlay(nr_i) + CostChange(rm_i.Role, nr_i) - CostPlay(rm_i.Role))$
Roles	Scheduler

Table 6. ChangeRoomActivity solution, design step

AddOU, then adds a new internal agent $RoomManager_i$ in this room (*AddAgent*), and finally assigns the role $nr_i \in NewRoles$ to this newly created agent. The cost of this solution is calculated as the cost of creating all the new rooms. Creating a room implies to add a new organizational unit (*CostAddOU*), a new internal agent that manages the room (*CostAddAgent*), and also includes the cost of this agent playing a specific role (*CostPlay*). Then, this cost is calculated as:

$$cost(ExtendBuilding) = |NewRoles| * (CostAddOU(Room) + CostAddAgent(InternalAgent)) + \sum_{nr_i \in NewRoles} CostPlay(nr_i) \quad (3)$$

Field	Description
Name	ExtendBuilding
Description	The building is extended with new rooms that allow it to fulfil all the received petitions.
Condition	FailedServiceCallsRate
Parameters	$NewRoles = 2^{\{MeetingManager, TeachingManager, BrainstormManager\}}$
Actions	$\forall nr_i \in NewRoles : Building.AddOU(Room_i); Building.AddAgent(RoomManager_i);$ $RoomManager_i.GetRole(nr_i)$
Cost	$\lvert NewRoles\rvert * (CostAddOU(Room) + CostAddAgent(InternalAgent)) + \sum_{nr_i \in NewRoles} CostPlay(nr_i)$
Roles	Building manager

Table 7. ExtendBuilding solution, design step

5.2 Selection between solutions

As stated in [1], to select between different options for change it is necessary to have a utility function that has to express the costs and benefits (both direct and indirect) of both the current and future states of the system, as well as the adaptation costs.

In some situations two forces may apply their solutions to the same organizational elements, thus being necessary to take a decision about which option to take. For this reason, applying the solution for one of these forces may make the organization to solve the effects of both forces. Therefore, in order to exactly choose one of those solutions, the one which maximizes the utility is selected.

For calculating the utility, not only the cost of applying the solution is taken into account, but also the cost of having failures, and the benefits obtained by the organization. In the case of the *Obtaining resources* force, its benefits are represented as the

requests that have been correctly placed in a room. Therefore, the utility of a solution to this force is calculated as:

$$utility_{sol} = benefit_{sol} - (cost(sol) + cost_{sol}(Failures)) \quad (4)$$

where $benefit_{sol}$ is the benefit obtained for placing the activities into the schedule of the rooms, calculated as $benefit_{sol} = AccReq * UnitBen$ where $AccReq$ is the number of requests accepted and placed in the schedule of a room and $UnitBen$ is the unitary benefit of having a request correctly scheduled. $cost(sol)$ is the cost associated to the solution sol. Finally, $cost_{sol}(Failures)$ is the cost of the remaining failures among the requests, calculated as $cost_{sol}(Failures) = RemFails * UnitFail$, where $RemFails$ is the number of remaining failures after applying the solution, and $UnitFail$ is the unitary cost of having a failure in the organization.

Then, the action to apply in the organization is decided following this equation:

$$max(utility_{sol1}, utility_{sol2}, utility_{noSol}) \quad (5)$$

where $utility_{sol1}$ is the utility of the first solution, $utility_{sol2}$ is the utility of the second solution, and $utility_{noSol}$ is the utility without applying any solution. As it can be noticed, in some situations where the cost of applying the solution is too high it is recommended to not take any action because it is the option with the maximum utility.

Applying the solution of a force could provoke the triggering of a factor of another force. Then, an action to solve the newly appeared force would have to be taken.

6 Implementation

In order to exemplify the implementation of the forces described in the previous section in the context of the use case described in Section 4, let us set a scenario with a building of three rooms (*room*1, *room*2, and *room*3) each being able to host one of the three following activities: teaching, meeting, and brainstorming. Each room is managed by a room manager (*RoomManager*1, *RoomManager*2, and *RoomManager*3, respectively) which has to check that the room is properly running. The building manager sets the following QoS: 90% for teaching, 60% for meeting, and 50% for brainstorming reservations.[10] At the start of a week, the rooms can be randomly distributed, or they can follow the distribution of the week before. In this example, they are randomly distributed.

This scenario, as described in Section 4, is based on the GORMAS organizational model. Agents and forces are implemented using Jason [6], for programming autonomous agents, and CArtAgO [16] for programming environment artifacts. Both components are available in the Multi-Agent Programming framework JaCaMo [5] that provides the infrastructure and abstractions for running distributed multi-agent systems combining agents, artifacts and organizations.

– **Agents** are: *Building manager* (responsible of selecting the solution and to apply the ExtendBuilding solution), *client* (generates a request for activity), *room manager* (manages the room and the role it plays is the activity carried out inside the

[10] Such QoS mean that the building management will only accept a maximum of 10%, 40%, and 50% of failures for teaching, meeting, and brainstorming requests for activities, respectively.

room), and *scheduler* (distributes the petitions around the different rooms, calculates the failures of the petitions, and is responsible of the ChangeRoomActivity solution). Each type of agent has a different set of skills and capabilities, related to the roles defined at the design time.

– **Artifacts**, implemented as CArtAgO classes, provide functionalities to the agents. The *monitor* and *room* artifacts store the number of requests and occupancies. The *room* artifact is controlled by the *room manager* agent. The *monitor* artifact (mentioned in Table 4) controls that the behavior and performance of the system is correct by checking the factor of the *Obtaining resources* force (by taking into account the QoS of each service). To carry out the experimentations, a *date generator* artifact generates random requests of activities in the system.

At the first execution cycle, the building manager creates both the Monitor artifact and the date generator artifact. Additionally, in this phase each of the defined room manager agents randomly receives a type of role (teaching, meeting, and brainstorming), creates the room artifact it manages, and sends this information to the scheduler.

Then, on each execution cycle, each client executes an operation of the date generator artifact to generate a random petition for an activity. Each petition for a room includes a specific day of the week (from Monday to Friday), with a specific start time (from 9am to 6pm, in 1-hour intervals, thus having the opportunity to start the activity in 10 different hours each day), and a length (of 1, 2, or 3 hours). In this example, the duration *dur* taken into account when monitoring the organization refers to a week. All clients send this information to the Scheduler, which tries to allocate all the petitions around the different rooms at the required times. After all the requests have been processed, and after deciding whether they can be allocated into a room or not (failures are calculated using Equation 5.1), the Monitor artifact reacts and computes if an adaptation is required or not, following the condition of Table 5. Then, the Monitor sends a specific signal if an adaptation is required. As previously stated, an adaptation is required if any of the values of the QoS is lower than the acceptable value. In order to do this, the monitor counts the different types of the petitions separately, and then checks the QoS for all the reservation services (*TeachingReservation*, *MeetingReservation*, *BrainstormingReservation*).

In this case, the building manager requests the Monitor artifact to compute the utility of the two possible solutions (described in Tables 6 and 7) so as to decide which solution to apply, or to not take any further action.

Some experimentation has been carried out. However, due to the lack of space, this section only focuses on how a design template is implemented.

7 Conclusions and future work

In this work, templates for the design of the forces that drive organizational change, including the factors that help to detect if they are active or not, and the solutions that will take advantage of the forces or to prevent damage to the organization, have been defined. They have been used to implement the concept of forces into an OCMAS with adaptation features. These templates extend the guidelines presented in [9] that are used during the analysis phase of the development process.

As future work, more forces based on this example will be designed and implemented, and also different case studies will be studied. Finally, the implementation of the force detection and solution will be applied in other MAS-supporting frameworks such as THOMAS [3].

Acknowledgment This work is supported by the MINECO/FEDER grant TIN2012-36586-C03-01, the TIN2009-13839-C03-01 project of the Spanish government, and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

References

1. J. M. Alberola, V. Julian, and A. Garcia-Fornes. A cost-based transition approach for multi-agent systems reorganization. In *10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1221–1222. IFAAMAS, 2011.
2. H. Aldrich. *Organizations evolving*. Sage Publications Ltd, 1999.
3. E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julian, and M. Rebollo. An abstract architecture for virtual organizations: The thomas approach. *Knowledge and Information Systems*, 29(2):379–403, 2011.
4. E. Argente, V. Botti, and V. Julian. Gormas: An organizational-oriented methodological guideline for open mas. In *Agent-Oriented Software Engineering X*, pages 32–47. Springer, 2011.
5. O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 2011.
6. R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. Wiley. com, 2007.
7. S. A. Deloach, W. H. Oyenan, and E. T. Matson. A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, 16(1):13–56, 2008.
8. V. Dignum and F. Dignum. Towards formal semantics for reorganization. *Technical report UU-CS*, 2006.
9. S. Esparcia and E. Argente. Forces that drive organizational change in an adaptive virtual organization. In *2012 Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pages 46–53. IEEE, 2012.
10. J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: an organizational view of multi-agent systems. In *Agent-Oriented Software Engineering IV*, pages 214–230. Springer, 2004.
11. M. Hoogendoorn, C. Jonker, M. Schut, and J. Treur. Modeling centralized organization of organizational change. *Comput Math Organ Theory*, 13(2):147–184, 2007.
12. J. Hubner, J. Sichman, and O. Boissier. Moise+: towards a structural, functional, and deontic model for mas organization. In *Proc. of Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 501–502. ACM, 2002.
13. K. Lewin and D. Cartwright. *Field theory in social science*. Harper & Brothers, 1951.
14. M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proc. Web Information Systems Engineering. WISE 2003*, pages 3–12. IEEE, 2003.
15. D. S. Pugh and M. Weber. *Organization theory: selected readings*. Penguin, 1971.
16. A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in cartago. In *Multi-Agent Programming:*, pages 259–288. Springer, 2009.
17. A. Sorici, O. Boissier, G. Picard, and A. Santi. Exploiting the jacamo framework for realising an adaptive room governance application. In *Proc. DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11*, pages 239–242. ACM, 2011.