Multi-Agent Oriented Reorganisation within the JaCaMo infrastructure

Alexandru Sorici[1,2], Gauthier Picard[2], Olivier Boissier[2], Andrea Santi[3], and Jomi F. Hübner[4]

[1] *"Politehnica" University of Bucharest , Dpt of Computer Science and Engineering , Bucharest, Romania,* `alex.sorici@cti.pub.ro`

[2]*Ecole Nationale Supérieure des Mines , FAYOL-EMSE, LSTI , F-42023 Saint-Etienne,* {`picard,boissier`}`@emse.fr`

[3]*DEIS, Alma Mater Studiorum , Universita di Bologna , 47521 Cesena (FC), Italy,* `a.santi@unibo.it`

[4]*DAS-UFSC , Federal University of Santa Catarina, CP 476, 88040-900 Florianópolis SC, Brazil,* `jomi@das.ufsc.br`

Abstract

Current IT applications are often meant to be used in complex and highly dynamic environments. Multi-Agent Oriented technologies and related programming languages offer high level abstractions that can ease the realisation of such applications. However, providing the means to handle adaptation and endogenous reorganisation, is still a challenging issue even if multiple state-of-the art works propose reorganisation models at the agent level. Adopting a multi-agent oriented approach, in this paper we define a dedicated artifact (tool) that agents use to design and implement transitions to new organisations. The agents' compliance with a specific organisation specification that we formulate helps regulate and coordinate this reorganisation process. We show how this proposal has been realised within the JaCaMo multi-agent oriented programming platform and how this constitutes a step toward really adaptive systems.

Keywords: multi-agent oriented programming, reorganisation, organisation centred multi-agent systems

1 Introduction

Current applications are more and more meant to be used in complex, distributed and highly dynamic environments. Among others, their features stress heterogeneity, context-awareness, anticipation of users' desires and absence of a central control. So, given this context, adaptation becomes the key to face the dynamics and the evolution of situations in which these applications operate. Multi-agent technologies and abstractions, and in particular organisation oriented ones, can provide concepts and tools that give possible answers to this issue. However, even if multiple works in the state-of-the art propose several reorganisation models at the agent level for dealing with system adaptation, it is still an open issue how to put these models in practice: performing the changes at the right time, in the right order, without disrupting the functioning of the entire organisation.

In this paper we introduce a proper infrastructure for managing reorganisation using a multi-agent oriented programming approach based on the JaCaMo [3] framework. The JaCaMo framework proposes a novel multi-agent oriented programming approach and a related development platform for engineering and executing distributed and open software systems by integrating three multi-agent programming dimensions, namely the agent, environment, and organisation levels in a synergistic way. In JaCaMo the use of organisation-centric abstractions (e.g. roles, groups, norms, missions) offers the possibility to express and impose cooperation schemes that govern the functioning of the autonomous agents in the environment. Currently agents have the means to inspect and reason about the organisations in which they partake. However, for being able to support endogenous adaptation [1] of systems by using systematic reorganisational processes, agents, besides abilities to reason on organisational constructs, must also have the proper means to coordinate and cooperatively modify the organisation as soon as they deem it to be improper for the current objectives. In this paper, we propose to realise such a support by using the different first class abstractions related to environment and organisation that are part of a multi-agent programming approach. Our contribution consists in a tool, the reorganisation artifact, and the definition of an organisational specification that regulates and guides agent activity during reorganisation (an *organisation for reorganisation*). We use and extend the JaCaMo platform with these different elements.

The remainder of the paper is organised as follows: In Section 2 we provide the reader with the required background for the JaCaMo multi-agent oriented programming approach upon which is realized our reorganisation proposal. Section 3 presents the rationale and general overview of the envisioned reorganisation process and Section 4 and Section 5 introduce the JaCaMo based infrastructure used to support it. In Section 6 we compare our approach with existing related works in literature. Finally, Section 7 concludes the paper.

2 Background

This section provides the required background needed for properly introducing our reorganisation infrastructure. First, we briefly present the JaCaMo framework and its programming model (Section 2.1), and then we specifically focus on organisation programming in JaCaMo (Section 2.2) for introducing the founding concepts used in the proposed multi-agent oriented programming of reorganisation.

2.1 The JaCaMo Multi-Agent Programming Framework

JaCaMo [3] is a newborn multi-agent oriented conceptual framework and platform, which provides high-level first-class support for engineering Multi-Agent Systems (MAS) taking into account agent, environment, and organisation dimensions in synergy. The framework integrates three existing agent-based platforms – i.e. *Jason* [4], CArtAgO [10], and Moise [8] – by defining in particular a semantic link among concepts of the different programming dimensions at the meta-model and programming levels, in order to obtain a uniform and consistent programming model aimed at simplifying the combination of

[1] Endogenous adaptation refers to an adaptation process realized by the agents participating to the organisation themselves.

those dimensions when programming a MAS [3]. A JaCaMo multi-agent system (i.e., a software system programmed in JaCaMo) is given by an agent organisation programmed in $\mathcal{M}$OISE, organising autonomous agents programmed in *Jason*, working in shared and distributed artifact-based environments programmed in CArtAgO.

When developing a MAS in JaCaMo environment programming is exploited to define the computational layer encapsulating functionalities and services that agents can use/explore/share at runtime [14]. Being based on the A&A meta-model [10], in CArtAgO and hence in JaCaMo, such software environments can be designed and *programmed* as a dynamic set of computational entities called *artifacts*, collected into workspaces, possibly *distributed* among various nodes of a network. In order to be used by the agents, each artifact provides a usage interface composed by a set of *operations* and *observable properties*. Operations correspond to the actions that the artifact makes it available to agents to interact with such a piece of the environment; observable properties define the observable state of the artifact, which is represented by a set of information items whose value (and value change) can be perceived by agents as percepts.
Agents obviously encapsulate the control and decision-making part of the application. Based on *Jason* an agent in the JaCaMo framework is programmed as an entity composed of a set of *beliefs*, representing agent's current state and knowledge about the environment in which it is situated, a set of *goals* (either private or bound to organisational goals), which correspond to tasks the agent has to perform/achieve, and a set of *plans* which are courses of *action*, either internal or external (mapped into artifacts' operation), triggered by *events*, and that agents can dynamically compose, instantiate and execute to achieve goals.
Organisation programming specifies and manages the overall governance strategy of the system. The following section describes how it is programmed based on $\mathcal{M}$OISE.

2.2 Organisation Programming in JaCaMo

In JaCaMo, an organisation program based on the $\mathcal{M}$OISE Organisation Modeling Language (OML) consists in an *organisation specification* (OS) stating the global structure, functioning and norms to achieve the global purpose for which the organisation is defined. The OS ($OS = \langle SS, FS, NS \rangle$) consists in two independent specifications –structural specification (SS) and functional specification (FS)– bound together by the normative specification (NS) (see [8] for further details):

- The SS ($SS = \langle R, \sqsubset, rg \rangle$) declares a set of roles R possibly connected by an inheritance relation $\sqsubset$ and a root group rg specification. A group specification can define subgroups, the set of roles contained in the group and that could be connected to each other by communication, authority and/or acquaintance links. Role-compatibility relations, agent-cardinalities for role adoption or group participation are used to constrain this setting during the life cycle of the organisation.

- The FS ($FS = \langle M, G, S \rangle$) is focused on the definition of plan-based strategies (or social schemes, S) for the achievement of collective inter-dependent goals G grouped into missions[2] M. The organisation state evolves according to the missions the agents commit to and the goal achievements.

[2] A mission represents a consistent grouping of collective or individual goals.

- The NS defines a set of norms ($norm = \langle id, c, \rho, dm, m \rangle$) that binds SS and FS together by the way of roles and missions. When the norm conditions c holds, any agent playing a role ρ has the deontic modality dm to commit to mission m.

Given the OS the set of agents A build what we call an *organisation entity* (OE) which current state can be described as $OE = \langle OS, A, GI, SI, O \rangle$. GI is the set of running group instances, i.e. groups of agents playing roles according to the specifications stated by the SS. SI is the social schemes instances SI, i.e. goals under achievement by the agents given the FS specification, following the current set of obligations O resulting from the activations of the norms stated in the NS. In order to help the agents to manage in a distributed way the frequent changes of the OE, a set of dedicated artifacts have been define (GroupBoard and SchemeBoard) to provide the agents with the proper tools to manage the OE. Such *organisational artifacts* are created for each GI or SI in the OE.

- Each group instance of GI is associated to a dedicated GroupBoard artifact that manages its current state: current set of agents in the group, playing the roles that are specified as part of that group type. This artifact provides the agents with operation for *adopting or leaving a role*, *adding or removing a social scheme* within this instance of group. Any agent that focuses on this artifact may get observable properties or different events pertaining to the *life cycle* of this group (e.g. which agent plays which role inside the group, number of roles that could be adopted, etc).

- Each social scheme instance of SI is managed by a SchemeBoard artifact. This artifact is connected to the corresponding GroupBoard managing the group instance in charge of its achievement. Thus all the agents participating to that group instance may be involved in the achievement of the goals managed by that scheme instance according to the obligations created from the activation of the norms stated in the NS. Using the operations of that artifact, agents can commit to a mission, leave it when finished, or can state that a given goal has been achieved. The SchemeBoard's observable properties provide the agents with the evolution of the coordinated goal-solving process (achieved goals, violated obligations, etc).

The Moise-based specifications of the organisation themselves are part of the information made observable by organisational artifacts to agents. This means that there is the potential for agents that understand the Moise OML to reason about the organisations in which they partake and therefore to change them at runtime. This allows for complex on-the-fly restructuring of computational systems to be done at very high level.

3 Organisation Specification for Reorganisation

It is reasonable to think that adaptation of an application behaviour, in response to changes in the environment, requires a carefully selected and coordinated sequence of specific actions like monitoring, logging, reorganisation plan design, selection and implementation [7]. Being considered as an endogenous process, each of these actions will be carried out by one or several agents of the system [3]. As it is done for the coordination of the

[3]Depending of the application, agents taking part to the reorganisation may be dedicated agents or domain agents embedding particular reorganisation knowledge and skills

application, a dedicated organisation specification can be defined to coordinate and supervise the reorganisation process itself. Following the organisation programming proposed in the JaCaMo framework, we program this organisation using the $\mathcal{M}$OISE OML, installing a kind of meta-level control of the reorganisation process itself. For that purpose, we have extended the organisation for reorganisation proposed in [7]. We describe it below along the structural, functional and normative dimensions.

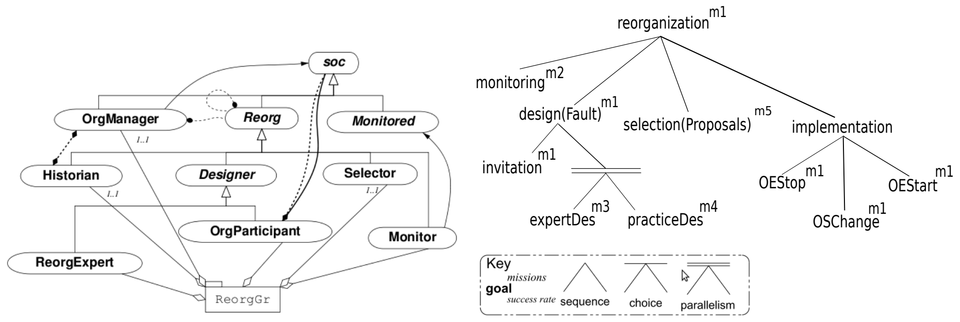

Figure 1: Graphical representations of the Reorganisation Organisation in terms of (i) Reorganisation Group and (ii) Reorganisation Social Scheme

3.1 Reorganisation Group

Figure 1-Reorganisation Group depicts the structural specification of the Reorganisation Group governing the reorganisation. We can immediately observe that, according to the names of the roles, the structure depicts the different coordination functionalities that are required to monitor, diagnose, design and install a new organisation from an existing one. All the roles are part of this group.

The OrgManager role is compatible with the Historian role. The Designer is specialized in a role that can be played by agents of the application domain (compatibility link between OrgParticipant with soc where soc is the root role of any structure of any organisation). It is also specialized into a ReorgExpert which is not compatible with any other role, forcing thus agents who adopt this role to have not adopted a role in another organisation. Let's note that the agents playing one of these roles may communicate with each other (reflexive communication link attached to the Reorg role and communication link between OrgManager and Reorg roles).

3.2 Reorganisation Social Scheme

The entire reorganisation process (see Figure 1 Reorganisation Social Scheme) consists in a monitoring phase followed by design, selection and finally implementation phases. In order to fully coordinate and implement the reorganisation process, we have extended the initial functioning presented in [7] by decomposing the *implementation* goal into *OEStop*, *OEChange* and *OEStart* subgoals, executed in sequence. The *OEStop* goal will be achieved when the *OE* will be stopped, i.e. targeted agents will stop pursuing collective goals or even leave their roles completely. The *OSChange* goal will be achieved when all required modifications to the structural, functional or normative specifications will be done. Finally, the *OEStart* goal will be achieved when the new OE will be effectively created, informing targeted agents of their new assignments (what role(s) to play, what goal(s) to pursue etc).

3.3 Normative Specification for Reorganisation

The Normative Specification as depicted in Table 3.3 assigns the roles to missions with a set of norms consisting in obligations.

Condition	**Deontic**	**Role**	**Mission**
Application Specific	*obligation*	OrgManager	*m1*
Application Specific	*obligation*	Monitor	*m2*
Application Specific	*obligation*	ReorgExpert	*m3*
Application Specific	*obligation*	OrgParticipant	*m4*
Application Specific	*obligation*	Selector	*m5*

Table 1: Normative Specification of the reorganisation process. Conditions are not specified here since they are application dependent.

As stated by the NS, the OrgManager is the role that has to instantiate the organisation and supervise the whole process (in charge of mission *m1*). The Historian is in charge of keeping a list of all the changes that the organisation has gone through – a kind of useful information for the monitoring and design phases (e.g. role adoption, mission commitments, role creation, change in the cardinalities). The Designer contains the common properties for designers. The entire reorganisation process is expected to start when the agent(s) playing the Monitor role signal(s) a fault with the current organisation. Then, during the design stage, both agents playing Reorganisation Experts and Organisation Participants can come up with solutions for the identified problem. Agents playing the Selector separate role will then have to evaluate the best proposal out of those made by the designers. At the end, the agent in charge of the OrgManager will supervise the actual transition from the old organisation to the new one (cf. goal *implementation* within *m1*).

In the following section, we describe how we use the Artifact programming approach promoted by the JaCaMo platform to enrich the artifact-based organisation management infrastructure with a specific artifact to support the agents in the achievement of the *design* and *implementation* goals of the reorganisation social scheme.

4 Reorganisation Artifact

In the JaCaMo platform, organisation management is supported by a set of dedicated GroupBoard and SchemeBoard artifacts. Thus, the execution of the reorganisation process, as specified in the previous section, is de-facto managed by a GroupBoard for adopting/leaving roles defined in the *Reorganisation Group* and a SchemeBoard for committing to missions, achieving goals according to the specifications given in the *Reorganisation Social Scheme*. These artifacts provide the right operations for managing the execution of the generic and abstract reorganisation process. In order to complement this definition with an adapted *Design* and *Implementation* plan taking into account the real conditions in which the new organisation has to be implemented, we propose to enrich this artifact-based organisation management infrastructure with a ReorgBoard artifact. It will help the agents to build and execute organisation implementation plans. This will constitute the base infrastructure that future applications can use in order to achieve adaptive behaviours.

After a global description of this artifact, we describe in detail the dynamic structure to handle the organisation implementation plan in the system and finaly give the description of the different operations that the agents are provided with.

4.1 ReorgBoard Artifact

The management of the organisation for reorganisation as defined in the previous section is set in practice in the JaCaMo framework with the creation of (*i*) a GroupBoard to manage the created ReorgGr instance and (*ii*) a SchemeBoard artifact to manage the execution of the reorganisation social scheme. The basic operations for changing the *OE* (e.g. adopting a role, committing to a mission, etc) are de facto provided to any member of a group instance (resp. scheme instance) via the corresponding operations offered by GroupBoard (resp. SchemeBoard).

One or several of these actions changing the *OS* and/or *OE* levels of the organisation must be coordinated: agents playing a role must leave that role at a certain time, agents committed to a mission also, so on and so forth. The sequence in which it has to be done is strongly dependent on the application, on the features of the current running organisation, etc. The achievement of the *design* and *implementation* goals of the reorganisation process (see Fig. 1) require thus a specific attention. In order to help the agents in this process, we define a new kind of artifact, called ReorgBoard (cf. Fig. 2). This artifact is created by the agents playing the OrgManager role, every time the *monitoring* is achieved, launching the achievement of the goal *design* for a particular *fault* (cf. Fig. 1).

Achieving the *design* goal consists in defining a coordinated set of reorganisation changes to be done both at the *OS* and/or *OE* levels in the organisation for each of the subgoal of the *implementation* goal. This coordinated set of changes builds what we call in the sequel an implementation plan ImplPlan. The ImplPlan is stored in the ReorgBoard artifact and is made accessible to the agents via a set of observable properties of this artifact. This way, the agents in charge of implementing the new organisation will get the implementation plan as a belief and will execute it. Complementary to this observable property, the ReorgBoard usage interface proposes two distinct sets of operations to help the agents in the achievement of those two goals (see Sec. 4.3).

In the following subsections, we will first describe the implementation plan structure and then describe the different operations that are provided by this artifact.

4.2 Implementation Plan

The different reorganisation changes that have to be executed to install a new organisation may be numerous and interdependent, addressing different group instances or social scheme instances. This is why we define a hierarchical structure (ImplPlan) taking into account precedence constraints and targeted GroupBoard or SchemeBoard artifacts on which they have to operate (changes on a group instance, on a scheme instance).

Each of the three subgoals *OEStop*, *OSChange*, *OEStart* of the *implementation* goal of the Reorganisation Social Scheme (cf. Fig. 1), will be refined by the agents into a specific

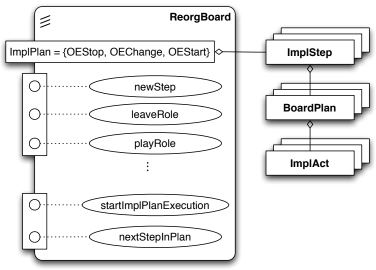

Figure 2: ReorgBoard Observable Properties (rectangles) and partial view of the operations (circles)

implementation plan ImplPlan according to the condition in which the new organisation has to be deployed. An implementation plan ImplPlan is a structured set of implementation actions ImplAct (see Fig. 2 for the implementation plan structure):

- An *OEStop* ImplPlan holds the coordinated set of actions to make the target agents stop their current work in the current organisation.
- An *OSChange* ImplPlan contains the coordinated set of actions that modify the OS which is governing the OE.
- An *OEStart* ImplPlan contains the coordinated set of actions that allow the agents playing the corresponding roles to re-instantiate a new OE according to the modified OS (e.g. telling target agents to play newly created roles, to commit to newly created missions, etc.)

An *implementation action* ImplAct defines an expression refering to an GroupBoard or SchemeBoard operation for changing the organisation. As mentioned, these different ImplAct actions that have to be executed to transit to a new organisation can be interdependent. To manage these dependencies, ImplPlan is decomposed in different implementation steps ImplStep which can be executed in sequence. An ImplStep can further be decomposed into board plans BoardPlans that group together implementation actions targeted at the same GroupBoard or SchemeBoard. For instance, implementation actions that are targeted to agents that play roles in the same group instance are grouped into the same BoardPlan corresponding to the GroupBoard that manages that group instance. Implementation actions that refer to changes in goals/missions are grouped in different BoardPlans according to the SchemeBoards that manage those goals/missions.

Implementation actions have different interpretations according to the organisation level they address:

- An OS level ImplAct contains an expression refering to an operation altering the OS (e.g. *addRoleObligation*, *changeRoleCardinality* etc), targeted to a GroupBoard or a SchemeBoard involved in the management of the running OE.
- An OE level ImplAct consists in an expression refering to an operation of a groupboard/schemeboard that a recipient agent has to perform (e.g. *leaveMission*, *playRole* etc). The execution of the action sends a request to the target agent to execute the change (cf. Sec. 5 for discussion on this notification mechanism).

OE level ImplAct actions are to be executed by autonomous agents. In order to control the way the action it refers to has to be executed by the agent, a "*strength*" modality is used to express the importance of the associated command and the delay with which it will have to be carried out:

- *regimentation*: immediate execution of the operation refered by the action. The agent it is addressed to is shunt: the action is directly executed in the corresponding GroupBoard or SchemeBoard.
- *obligation*: execution of the operation refered by the action as soon as the targeted agent has finished off any remaining goals from previous commitments.
- *permission*: execution of the operation refered by the action is permitted, but not required. It is mainly used when the new organisation does not depend on the successful completion of the operation.
- *interdiction*: interdiction to execute the operation refered in the action during the current reorganisation process.

4.3 Implementation & Design Operations

In order to build and manipulate the structure of the implementation plan, the ReorgBoard artifact provides the agents with a set of operations for the design and for the execution of such plans.

Design Operations. The operations for the design are used to create the structure of a ImplPlan, adding steps, boardplans. For instance, the operation `newStep(`p`, `st`)`: creates a new ImplStep *st* within the ImplPlan p, where $p \in \{OEStop, OSChange, OEStart\}$. This command binds the variable *st* with the step ID that can be used to reference the step in future commands. The table 2 shows the set of operations used for the addition of implementation actions. From the features of these actions, we can distinguish OS Level and OE Level ReorgBoard operations:

- An OS Level reorganisation design operation has the following syntax:

  $$operation(plan, step, target, type, object)$$

  It adds the implementation action "$operation(type, object)$" in the BoardPlan *target* belonging to ImplStep *step* of ImplPlan *plan*.
  For instance, the operation `addRole(`p`, `st`, `A`, `G`, `ρ`)` adds an implementation action for adding a new role ρ in the group definition G. It is included in the BoardPlan *A*, member of step st, part of the ImplPlan p.

- An OE Level reorganisation design operation has a syntax similar to the OS level design operation. The strength modality is added. Such an operation adds ImplAct in ImplBoard addressing OE Level changes.
  For instance, the operation `leaveMission(`p`, `st`, `A`, `sm`, `ag`, `mi`, `sch`)` adds to the BoardPlan *A* in step *st*, of the ImplPlan *p*, an implementation action that requests agent ag to stop working on the goals included in mission mi under execution in scheme instance of type sch managed by SchemeBoard instance A with the strength modality sm.

OS Level Operations	Description
addRole	add a role to a group specification
removeRole	remove a role from a group specification
addMission	add a mission to a scheme specification
removeMission	remove a mission from a scheme specification
addRoleObligation	add a normative assignment of a mission to a role
removeAllRoleObligations	clear a role of all its normative obligations
changeRoleCardinality	change the min..max interval of agents playing this role
changeSubgroupCardinality	change the min and max number for the instances of a subgroup
removeGroup	remove the given group specification from the organisation
removeScheme	remove the given scheme specification from the organisation
OE Level Operations	**Description**
leaveRole	target agent must leave the specified role
playRole	target agent must start playing the specified role
leaveMission	target agent must discontinue pursuing the goals in given mission
commitMission	target agent must commit to solving goals in given mission

Table 2: OE and OS Levels Reorganisation Design Operations

Implementation Operations. During the achievement of the *implementation* goal the ReorgBoard artifact acts as a coordinating entity. To this aim, it provides the following operations to the agent committed to the achievement of the implementation goals:

- `startImplPlanExecution(p)`: triggers the execution of the ImplPlan *p*, corresponding to the execution in sequence of the different *ImplStep* defining the plan.

- `nextStepInPlan(p)`: starts the execution of the next step within the *ImplPlan p*, once all the commands in the current step have completed. This last operation allows the OrgManager working with the ReorgBoard to start a new step once it has perceived the notifications that all the actions dictated by the previous commands have been performed.

Having described the internal structure of the ReorgBoard and the usage interface it exhibits, the next section will detail the interactions that exist between the ReorgBoard, the agents and the OrgArtifact instances to which it connects during the reorganisation process.

5 Multi-Agent based Reorganisation

In the previous section we have presented the ReorgBoard as an instrument that agents can use to achieve the *design* and *implementation* goals of the reorganisation process governed by the ReorgGr reorganisation group and the reorganisation social scheme described in Sec. 3. In this section, we bring together the pieces of the jigsaw and describe the use of this artifact in the global context of a reorganisation. We describe the different interactions that take place between the ReorgBoard, the agents involved in the reorganisation and GroupBoard or SchemeBoard instances that are impacted by the reorganisation (see. Fig. 3). We focus here on the design and implementation phases that are the core of the paper [4].

[4]For page limitation reasons, we cannot describe the full implementation and application. Interested readers may refer to `http://jacamo.sourceforge.net/?page_id=87` or [12]

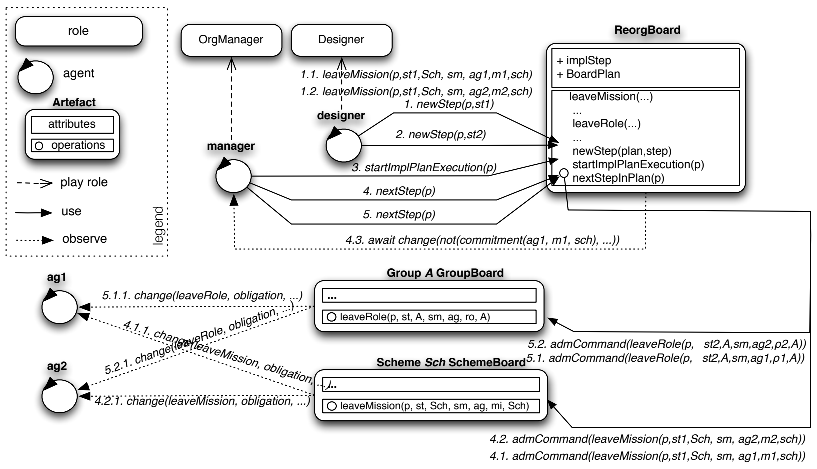
role
agent
Artefact
attributes
operations
play role
use
observe
legend
OrgManager
Designer
ReorgBoard
+ implStep
+ BoardPlan
leaveMission(...)
...
leaveRole(...)
...
newStep(plan,step)
startImplPlanExecution(p)
nextStepInPlan(p)
1.1. leaveMission(p,st1,Sch, sm, ag1,m1,sch)
1.2. leaveMission(p,st1,Sch, sm, ag2,m2,sch)
1. newStep(p,st1)
designer
2. newStep(p,st2)
manager
3. startImplPlanExecution(p)
4. nextStep(p)
5. nextStep(p)
4.3. await change(not(commitment(ag1, m1, sch), ...))
ag1
Group A GroupBoard
...
leaveRole(p, st, A, sm, ag, ro, A)
5.1.1. change(leaveRole, obligation, ...)
4.1.1. change(leaveMission, obligation, ...)
5.2.1. change(leaveRole, obligation, ...)
5.2. admCommand(leaveRole(p, st2,A,sm,ag2,ρ2,A))
5.1. admCommand(leaveRole(p, st2,A,sm,ag1,ρ1,A))
ag2
Scheme Sch SchemeBoard
...
leaveMission(p, st, Sch, sm, ag, mi, Sch)
4.2.1. change(leaveMission, obligation, ...)
4.2. admCommand(leaveMission(p,st1,Sch, sm, ag2,m2,sch))
4.1. admCommand(leaveMission(p,st1,Sch, sm, ag1,m1,sch))

Figure 3: Collaboration diagram of the reorganisation process.

5.1 Reorganisation Design

In order to achieve the design goal, the agents playing the OrgManager role and the other agents playing the Designer role use the ReorgBoard's usage interface to construct the content of each implementation plan (steps, boardplans and implementation actions) by using different reorganisation design operations (cf. Table 2).
As an example, let's consider dynamic role (re)allocation in rescue teams. In such a use case, a *role shift* implies a sequence of actions ranging from the leaving of current roles (*OEStop*), to possible structural modifications (*OSChange* - e.g. change in minimum or maximum cardinalities of agents playing a given role) and adoption of the new roles (*OEStart*). Note that the knowledge and strategy (e.g. negotiation or machine learning) used to come up with the proposed role shift is particular to each Designer agent. The ReorgBoard and its reorganisation design operations act as the *mechanism* used by agents to implement the required changes.
To show the exact use of the artifact in the context of reorganisation, let's focus on the *OEStop* subgoal of a hypothetical role shift, where the agent in charge of the design decides that, in the current ImplPlan p, the agents (ag_1 and ag_2) belonging to group A have to leave their roles (ρ_1 and ρ_2).

Since, before leaving their roles, the agents must first leave their missions, the agent playing the Designer role defines two ReorgSteps (`newStep(p,` $\mathtt{st_1}$`)` and `newStep(p,` $\mathtt{st_2}$`)`), which will be executed in sequence: the first one contains an ImplAct for the given agents to stop their work on the goals contained within the missions they are responsible for, whereas the second one comprises actions that tell those two agents to leave their current role (see. 1 and 2 on Fig. 3).

The commands to leave the missions are included in the ImplStep referenced by st_1. Both actions are grouped under the BoardPlan that concerns the SchemeBoard instance (*Sch*) managing those missions. To do this, the designer agent invokes the following operations ((see. 1.1 and 1.2 on Fig. 3): `leaveMission(`p, st_1, Sch, sm, ag_1, m_1, sch`)` and `leaveMission(`p, st_1, Sch, sm, ag_2, m_2, sch`)`, which bare the same semantics as those described in section 4.3.

The commands to leave the roles are contained in the second ImplStep. The agents ag_1 and ag_2 being members of the same group, the actions are put together under the same BoardPlan that will be sent to the GroupBoard instance managing group A. The designer agent achieves this by calling `leaveRole(`p`, `st_2`, `A`, `sm`, `ag_1`, `ρ_1`, `A`)` and `leaveRole(`p`, `st_2`, `A`, `sm`, `ag_2`, `ρ_2`, `A`)` on the ReorgBoard artifact. With this, the design process for the considered *OEStop* implementation subgoal is completed [5]

5.2 Reorganisation Implementation

The achievement of the implementation goal consists in the direct interaction of the agent playing the OrgManager role with the ReorgBoard artifact. Using the different implementation operations, it ensures the proper and ordered execution of the implementation actions contained in each of the implementation plans produced during the achievement of the design goal.

Pursuing with our previous example of the *OEStop* implementation plan p, the agent OrgManager uses the `startImplPlanExecution(p)` operation to start the first step (st_1) of the ImplPlan. When handling the step, the ReorgBoard executes the list of BoardPlans, in the case of st_1, those directed to the SchemeBoard instance *Sch* managing missions m_1 and m_2. The ReorgBoard links itself to the receiving OrgArtifact instance in order to send all of the implementation actions included in the BoardPlan.[6]
When an OrgArtifact instance receives a ImplAct, it looks at the level this action is situated (OE Level or OS Level).

- In case of an OS level action, the changes contained within the action directly alter the internal specifications of the organisational artifact instance.
- In case of an OE level action, such as in our example, a first notification is done between the ReorgBoard artifact to the GroupBoard/SchemeBoard concerned by the change. A second notification is then done by this latter artifact, informing the targeted agent (e.g. ag_1, ag_2) of the pending change stated by this command. As part of the step st_1, agent ag_1 would, for instance, receive an expression like: $change(leaveMission, obligation, ag_1, m_1, sch)$ (see. 4.1, 4.1.1 and 4.2, 4.2.1 and 5.1, 5.1.1 and 5.2, 5.2.1 on Fig. 3).

Meanwhile, for implementation actions with a strength attribute lower than *regimentation* (like in the example above), the ReorgBoard uses the same observable events mechanism to inform the OrgManager of the perceived organisational statements he has to wait for before he can move on to the next step.

In our example the agent playing the OrgManager will receive, among others, a notification from the ReorgBoard of the form: $await\_change(not(commitment(ag_1, m_1, sch), ...))$, meaning the agent will have to wait until she perceives the organisational fact that agent ag_1 actually quit handling the goals included in mission m_1 from the scheme *sch*, before proceeding with other actions (i.e. calling the `nextStepInPlan(p)` operation on the ReorgBoard). The agent will be noticed of that fact thanks to the observable properties of the SchemeBoard Sch.

[5]For clarity reasons, we didn't add the call to these operations on the Fig. 3).

[6]Both GroupBoard and SchemeBoard artifacts provide an operation which allow for the handling of internal administration tasks. This operation, called `admCommand`, is exploited by the ReorgBoard to link to the required organisational artifact when sending a ImplAction.

When all steps in a plan have been executed, the agent playing the OrgManager role is again informed via an observable event sent out by the ReorgBoard that it is possible to proceed to the next plan. When the last plan has finished executing, the reorganisation process is complete and the agent playing the OrgManager role can discard the ReorgBoard artifact.

6 Related Works

There is a huge literature on reorganisation in multiagent systems. Some use an exogenous reorganisation process where the user or a dedicated MAS itself reorganise the whole system (e.g. [13, 6] or [5]). Others [9, 11], like our proposal or the one used as a starting point, use an endogenous approach where the agent themselves modify the organisation. However, we make clear and explicit the organisation controlling the reorganisation process itself. This is a clear difference with the other approaches promoting an endogenous approach where the reorganisation process is hard coded in the agents themselves.

An approach similar to ours can be found in the work by Alberola et al. [1]. The Reorganisation Facilitator Service they propose has a functionality and purpose which are close to those of the ReorgBoard. However, even if the approach in [1] allows for the finding of the minimal-cost transition from an initial organisation instance to a newly desired one (which can actually be viewed as a particular strategy), the work does not describe an explicit coordination and supervision mechanism for the implementation of the required transition operations.

Using the reorganisation artifact approach, we go a step further in the sense that we have proposed a set of tools that can be used to support and engineer this process. This is a strong added value with respect to the one proposed by Hubner et al. [7]. Thanks to this artifacted process, we have been able to contribute to different ways of changing the organisation as in [2] and to address the practical setting of organisation changes: when and how to stop, when and how to implement the changes, when and how to restart the organisation process.

7 Conclusions

This work has presented a clear mechanism that is used to engineer and support the process of reorganisation in multi-agent organisations. Its realization was made possible by means of the JaCaMo platform and it represents an extension of previous work on the modeling of the reorganisation process started in [7]. Specifically, the business logic of the developed ReorgBoard artifact helps extend the functional specification of the "organisation for reorganisation" with the explicit stages of the implementation phase: *stopping* the affected part of the organisation, *applying changes* to the organisational specifications themselves and *starting* the new instance of the organisation. The ReorgBoard offers an internal structuring and a set of operations to instrument both the *design* and the coordinated *implementation* of the reorganisation process.

The benefit of this extended model for reorganisation is that each involved step (*monitoring*, *design* and *implementation*) is clearly defined and controllable. By means of the artifact-based infrastructure offering the required coordination, agents can be seen to employ whichever application-specific adaptation strategies or heuristics, thus achieving a separation of concerns between the underlying reorganisation mechanism and the considered adaptation policy.

8 Acknowledgments

This work has been supported by French Rhône-Alpes Région CMIRA and ERASMUS/SOCRATES funds.

References

[1] J. M. Alberola. A cost-oriented reorganization reasoning for multiagent systems organization transitions. In L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, editors, *AAMAS*, pages 1349–1350. IFAAMAS, 2011.

[2] H. Aldewereld, F. Dignum, V. Dignum, and L. Penserini. A formal specification for organizational adaptation. In M. P. Gleizes and J. J. Gómez-Sanz, editors, *AOSE*, volume 6038 of *Lecture Notes in Computer Science*, pages 18–31. Springer, 2009.

[3] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, (0):–, 2011.

[4] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using **Jason***. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

[5] E. Bou, M. López-Sánchez, J. A. Rodríguez-Aguilar, and J. S. Sichman. Adapting autonomic electronic institutions to heterogeneous agent societies. In G. A. Vouros, A. Artikis, K. Stathis, and J. V. Pitt, editors, *AAMAS-OAMAS*, volume 5368 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2008.

[6] B. Horling, B. Benyo, and V. R. Lesser. Using self-diagnosis to adapt organizational structures. In *Agents*, pages 529–536, 2001.

[7] J. Hübner, J. Sichman, and O. Boissier. Using the moise+ for a cooperative framework of mas reorganisation. In *SBIA*, volume 2004, pages 506–515. Springer Verlag, 2004.

[8] 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. *Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.

[9] S. Kamboj. Analyzing the tradeoffs between breakup and cloning in the context of organizational self-design. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, editors, *AAMAS (2)*, pages 829–836. IFAAMAS, 2009.

[10] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in CArtAgO. In R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications, Vol. 2*. Springer, 2009.

[11] M. Sims, C. V. Goldman, and V. R. Lesser. Self-organization through bottom-up coalition formation. In *AAMAS*, pages 867–874. ACM, 2003.

[12] A. Sorici, O. Boissier, G. Picard, and A. Santi. Exploiting the jacamo framework for realising an adaptive room governance application. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, &#38; VMIL'11*, SPLASH '11 Workshops, pages 239–242, New York, NY, USA, 2011. ACM.

[13] M. Tambe, D. V. Pynadath, and N. Chauvat. Building dynamic agent organizations in cyberspace. *IEEE Internet Computing*, 4(2):65–73, 2000.

[14] D. Weyns, A. Omicini, and J. J. Odell. Environment as a first-class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, Feb. 2007.