ParaMoise: Increasing Capabilities of Parallel Execution and Reorganization in an Organizational Model

Mateusz Guzek
Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg
6, rue R. Coudenhove-Kalergi
Luxembourg, Luxembourg
mateusz.guzek@uni.lu

Grégoire Danoy
Computer Science and Communications Research Unit, University of Luxembourg
6, rue R. Coudenhove-Kalergi
Luxembourg, Luxembourg
gregoire.danoy@uni.lu

Pascal Bouvry
Computer Science and Communications Research Unit, University of Luxembourg
6, rue R. Coudenhove-Kalergi
Luxembourg, Luxembourg
pascal.bouvry@uni.lu

ABSTRACT

Organization-centered top-down models can be used to effectively represent MAS organizations and their dynamics. In state-of-the-art $\mathcal{M}$oise$^+$ framework, the organization functioning is modeled as a set of goal decomposition trees, named Functional Specification (FS). However, such tree structures do not permit to model all types of goals interdependencies, which might be a limiting factor when dealing with the organization of complex, large scale systems like Cloud Computing (CC). Such systems commonly use the concept of workflow to represent dependencies between the tasks to be executed. This paper proposes to use the same workflow approach to model goals execution plans and dependencies in the FS, which permits to include properties and constraints of a strongly parallel system. The resulting organization model is ParaMoise that brings more expressiveness than $\mathcal{M}$oise$^+$: it can represent an arbitrary acyclic graph dependency structure, track the status of goals execution, and ensure mutual exclusion and progress of distributed execution. The lock mechanism enables reorganization that can affect any element of organization at system runtime. As an example, ParaMoise is used to model a realistic CC management MAS that aims to provide self-* properties to an autonomic computing system.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems; I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Theory, Design, Management, Performance, Reliability

Keywords

Agent societies and Societal issues::Environments, organisations and institutions; Agent theories, Models and Architectures::Modeling the dynamics of MAS; Agent-based system development::P2P, web services, grid computing; Systems and Organisation::Autonomic computing

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

1. INTRODUCTION

The organizational models of Multi-Agent Systems (MAS) define the roles, functioning, and rules inside MAS. Additionally, they enable conscious, explicit monitoring and management of organizations composed of agents, which can be done by external administrators of the organization, as well as by the agents themselves. The MAS paradigm encompasses autonomic, distributed, and decentralized decision making, and adaptivity to environmental changes. The demand for such properties grows together with the widespread adoption of Large Scale Distributed Systems (LSDS).

Organization-centered MAS are especially applicable to current LSDS systems that already have some pre-defined structure or organization. In such case, a top-down approach, in which the organizational model is defined a priori and modified during system runtime, is appropriate. On the other hand, reorganization behaviors are desired to ensure adaptability to changing system conditions.

State-of-the-art organizational models and frameworks suit well multiple types of MAS, e.g. robot soccer team [7], conference organization [2] or rescue team [4]. However, their current applications is limited by various drawbacks: dependency on external, often centralized entities that support or manage an organization, simplified view of agents mission planning, limited or no support of reorganization, and separation of system and execution into different, loosely connected layers. This works aim is to overcome the above mentioned points.

We propose ParaMoise, a MAS organizational model that enables parallel execution of goals structures with arbitrary precedences, alternative execution of goals, basic failure handling mechanisms, detailed mission and goal status tracking, and parallel, decentralized reorganization capabilities of any component of organization thanks to the mechanism of locks. Moreover, the reorganization is defined as a composition of goals, and can combine in this way the goals of reorganization process with the standard system goals.

The remainder of this article is organized as follows. The next section provides a state of the art on MAS organizational and reorganizational models for LSDS. Then section 3 presents the ParaMoise model, including its novel workflow-based functional specification and reorganization. An example of ParaMoise usage for modeling a CC system is given in section 4. Section 5 includes some discussion of the proposed contribution, and finally conclusions and perspectives are presented in section 6.

2. STATE OF THE ART

Various organizational models have been proposed in the literature to specify MAS functioning and dynamics. We propose here to analyze some of the most prominent top-down approaches, selected by their capacity to model LSDS.

AgentCoRe [4] is a framework designed for coordination and reorganization in MAS. The decision making modules are capable of the selection of coordination mechanisms, creation and update of a task structure, subtask assignment and reorganization (change the assignment of roles to agents, relationships between them or existing task assignment). The decision making modules are based on strategies. AgentCoRe is strongly dependent on its modules, which makes it less appropriate for general reorganization cases. Additionally, the basic components of AgentCoRe modules do not include agents as decision makers.

GPGP/STÆM [12] also proposes to decompose the organization functioning into goals and subgoals. The leaves of such a tree-structure are tasks. More complex structures can be created by adding precedence links. One of the key components is a domain-independent Design-to-Criteria (DTC) scheduler. As a result, plans bind agents with specific sets of tasks. The coordination is based on the goals allocation and the goals structure. The limitations of such an approach reside in the limited agent adaptation capabilities because of the dependency on a given schedule. Additionally, the domain-independent scheduler creates a single point of failure, and the lack of scalability of the coordination mechanism may prohibit the use of this framework for a large scale MAS.

Wang and Liang [16] propose a three layer organizational model divided into three graphs: social structure, role enactment, and agent coordination. In this approach the reorganization is a graph transformation, based on a set of predefined graph reorganization rules. Such reorganization is limited to changing the structural dimension of the organization, and thus not its functionalities.

Kota et al. [10] present an adaptive self-organizing MAS that can reorganize on task assignment (i.e. functional) and structural levels. Agents in an organization provide services and possess a set of computational resources, which correspond to a CC system. Important concepts of organization cost, benefit and efficiency are also introduced. The reorganization process is decentralized, however it includes only 4 possible actions: create/remove peer and create/remove authority.

OperA [2] is a framework for socio-technical systems. It merges roles with their objectives and presents the relations between roles as pre-order. The higher level of coordination in a system are scenes and their compositions. OperA enables adaptation of a set of agents to a pre-defined organization model, however it does not concern reorganization of the model. Moreover, the scenes composition does not support arbitrary precedences and describe system only on high level, resulting in less versatile and appropriate for effective parallel execution model than ParaMoise.

$\mathcal{M}$oise$^+$ [6] is an organization model that specifies three organizational dimensions: structural, functional and deontic. However the model is designed for small systems, as only one agent is responsible for reorganization and the reorganization implementation process is not clearly defined, and the functional specification is based on goal decomposition trees that limit the possibilities of strongly parallel executions. $\mathcal{M}$oise$^+$ includes reorganization modeling capabilities [7], an agent programming methodology [8] and it was recently transformed in an Agent & Artifact framework named ORA4MAS [5].

These latest developments are used in the JaCaMo multi-agent programming framework [13]. JaCaMo presents the reorganization implementation that inherits the single agent responsible for reorganization process from $\mathcal{M}$oise$^+$, which results in a sequentially executed implementation plan. Moreover, the organization needs to be stopped for the reorganization, which may be infeasible in a real LSDS scenario.

The objective of this work is thus to benefit from $\mathcal{M}$oise$^+$ organization and reorganization dimensions specifications, introduce a new functional specification that permits to take into account properties of dynamic LSDS and extend it to model fully parallel, reorganization at system runtime.

To this end we rely on the concept of workflow, that is well established and widely used in the Grid computing domain. The workflow is composed of tasks and their precedence relations among them, which form an acyclic graph of tasks. It provides fault tolerant distributed execution frameworks and can include alternatives among edges incoming to a node to improve failure handling or speedup the execution [9].

3. THE PARAMOISE MODEL

As previously mentioned, the ParaMoise organizational model is an extension to the $\mathcal{M}$oise$^+$ model. The $\mathcal{M}$oise$^+$ organization-centric model relies on three specifications, i.e. structural, functional and deontic, that compose the Organizational Specification (OS). An OS is instantiated by a set of agents that adopt roles and constitute the Organizational Entity (OE). The OE therefore expresses the state of specific organization following the corresponding OS. $\mathcal{M}$oise$^+$ OS can thus be defined as the tuple $\langle SS, FS, DS \rangle$ [13], where SS is the Structural Specification that defines roles, inheritance and links between roles, and groups; FS is the Functional Specification, that groups individual goals into missions and structures them in social schemes; DS is the Deontic Specification that binds the SS and FS, i.e. it assigns roles 'obligations' or 'permissions' to achieve missions with a defined time constraint. The ParaMoise model uses subsequently the above mentioned definitions of SS and DS, but changes FS as described in the next section.

For describing the OE we simplify the definition presented in [8] and add to it the set of deontic modalities (e.g. obligations [13]): an OE is a tuple $\langle OS, \mathcal{A}, \mathcal{GI}, \mathcal{SI}, \mathcal{O}, sg, ar, am \rangle$, where OS is the organizational specification; $\mathcal{A}$ is the set of agents; $\mathcal{GI}$ is the set of group instances; $\mathcal{SI}$ is the set of social schemes; $\mathcal{O}$ is the set of current deontic modalities; $sg : \mathcal{GI} \rightarrow \mathbb{P}(\mathcal{GI})$ maps each group to its subgroups; $ar : \mathcal{A} \nrightarrow \mathbb{P}(\mathcal{R} \times \mathcal{GI})$ maps agents to the roles they are playing in the groups; $am : \mathcal{A} \nrightarrow \mathbb{P}(\mathcal{M} \times \mathcal{SI})$ maps agents to the missions they are committed to in the social schemes.

3.1 Workflow-based Functional Specification

The original Functional Specification in $\mathcal{M}$oise$^+$ defines how to achieve the global collective goals as a set of social schemes. A social scheme is a tree in which goals are composed of sub-goals to be executed in sequence, choice or parallel. Coherent goals to be achieved by one agent are grouped in missions. A mission is assigned a cardinality that defines the set of agents that can commit to it. Each goal belongs to a mission and may have defined its success rate.

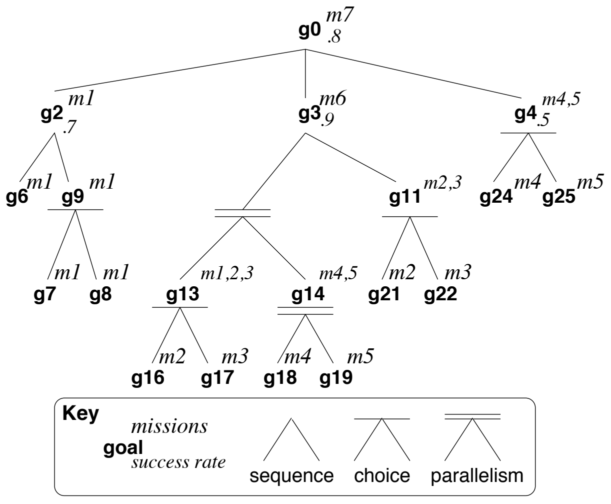

Figure 1: An example of $\mathcal{M}$oise$^+$ Social Scheme

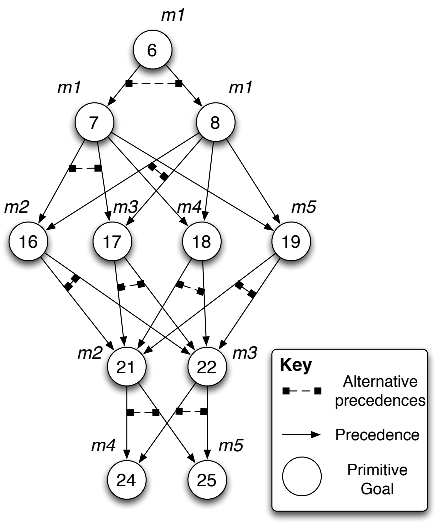

Figure 2: An example of ParaMoise WFS

An example of a social scheme from [6] that represents a social scheme to score a goal in a soccer team is presented in Figure 1.

In order to provide LSDS more verbose and parallel specification, we propose to replace the goal decomposition tree model of the FS by a workflow-based one, that we refer to as Workflow Specification (WFS).

Based on the original $\mathcal{M}$oise$^+$ social schema definition [6], the proposed WFS is defined by an extended tuple $\langle \mathcal{G}, \mathcal{E}, \mathcal{M}, mo, nm, alt, fh \rangle$, where $\mathcal{G}$ is the set of global goals; $\mathcal{E}$ is an added set of precedence relations; $\mathcal{M}$ is the set of mission labels; $mo : \mathcal{M} \to \mathbb{P}(\mathcal{G})$ is the function that specifies the mission set of goals ; $nm : \mathcal{M} \nrightarrow \mathbb{N} \times \mathbb{N}$ specifies the boundaries (min,max) of number of agents committed to the mission in well formed WFS. Two new functions (alt and fh) defining alternative execution paths and failure handling mechanisms are also added to the WFS.

The set of global goals can be defined as $\mathcal{G} = \mathcal{G}_p \cup \mathcal{G}_c$, where $\mathcal{G}_p$ is the set of *primitive* goals and $\mathcal{G}_c$ is the set of *composed* goals. A composed goal is a set of other goals, either composed or primitive. A primitive goal is either a simple goal that can be followed by an agent or a set of agents, or a nested WFS. A precedence relation is defined as a tuple: $e = \langle g_1, g_2 \rangle$, where $g_1, g_2 \in \mathcal{G}_p$, g_1 is the precedent goal and g_2 is the antecedent goal. The primitive goals and precedence relations, interpreted respectively as nodes and edges, must create a directed acyclic graph in a well formed WFS. Because of this condition and link between workflow and graph, in further text we use interchangeably primitive goals and nodes, as well as precedence relations and edges.

The $alt : \mathcal{E} \to \mathbb{P}(\mathcal{E})$ function specifies the precedence relations alternatives. Alternative edges must have the same precedent goal or antecedent goal as the mapped precedence relation. The alternatives express alternative execution paths in a workflow (in case of common precedent goal, e.g. Figure 2), or redundancy of conditions for a common antecedent (i.e. only one of the precedent goals of the alternative edges must be achieved, e.g. Figure 8).

The partial function $fh : \mathcal{G}_p \nrightarrow \mathbb{N}$ specifies the failure handling mechanism for a primitive goal. In case the function is undefined, there is no specified failure handling mechanism, like in $\mathcal{M}$oise$^+$ original FS. Otherwise, the primitive goal is treated as a transaction and it is either fully successful or failed at the end of its execution. The function fh specifies the number of times the transaction can be repeated in case of failure. If the number of allowed repetitions is reached, the whole workflow execution is aborted. In such case a rescue mechanism shall be defined as another WFS, triggered by the abortion.

Figure 2 presents how the original $\mathcal{M}$oise$^+$ scheme presented in Figure 1 can be transformed into a workflow. The nodes of $\mathcal{M}$oise$^+$ social schema that are not leafs are translated into composed nodes in WFS, e.g. $\mathcal{G}_c = \{g_9 = \{g_7, g_8\}, g_2 = \{g_6, g_9\}, \dots\}$, and should be listed separately.

3.2 Workflow on Organizational Entity level

The Workflow (WF) is an instance of WFS and contains the information necessary for a specific execution. It is defined as a tuple $\langle WFS, es, gs, exe, gf \rangle$, where WFS is the workflow specification, $es : \mathcal{E} \to \{active, inactive, discarded\}$ is the function that maps edges to their activity status label; $gs : \mathcal{G}_p \to \{waiting, possible, executing, suspended, achieved, discarded\}$ is the function that specifies statuses of primitive goals; $exe : \mathcal{G}_p \nrightarrow \mathbb{P}(\mathcal{A})$ is the function that specifies the set of agents executing a goal; $gf : \mathcal{G}_p \to \mathbb{N}$ specifies numbers of repetitions of primitive goals. Each primitive goal of a WFS that is itself a nested WFS must be instantiated as a nested WF, to be a primitive goal of a WF.

The execution of a WF must follow a set of rules. The corresponding transition diagram of state labelings is illustrated in Figure 3. Initially, all goals are labeled as *waiting*, except the goal(s) without incoming precedences that are marked as *possible*, and all precedences are labeled as *active*. A goal transits from *waiting* to *possible* state if there is no *active* incoming precedence and at least one incoming precedence is *inactive*. If an agent or a set of agents start execution of a goal, its label is changed to *executing* and it cannot be executed concurrently by other agents. In case other agents would like commit to that goal, they should join the set of

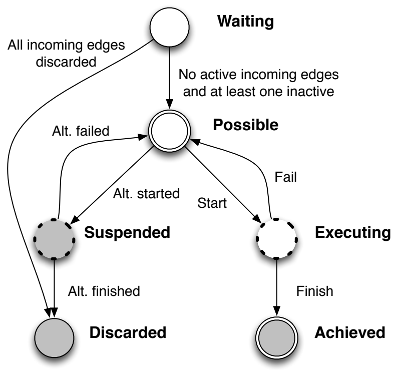

Figure 3: State transition diagram of goals status. The transitions from each state (except Achieved) to discarded in case of discarding all outgoing edges are omitted.

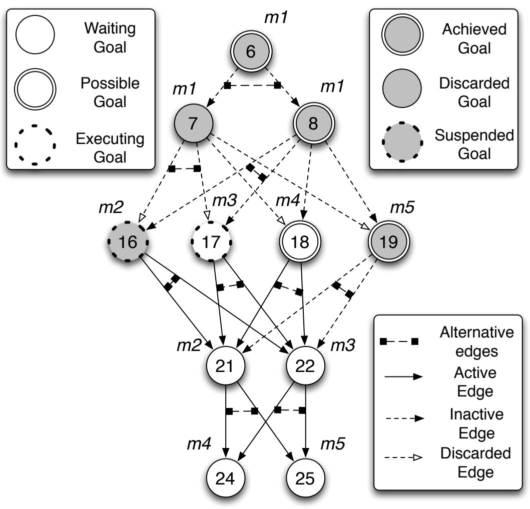

Figure 4: An example of WF during execution

agents that is already executing the goal. After successful end of execution, the goal label is changed to *achieved* and all of the precedences outgoing from this goal are marked as *inactive*.

The possibility of alternative edges extend the set of execution rules as follows. In case an edge incoming to a goal that starts its execution has alternatives with the same precedent goal, then all goals that are the antecedents of the alternatives are labeled as *suspended*. Once the executing goal is achieved, the nodes labeled as *suspended* in the previous step are labeled as *discarded*. If a goal becomes labeled as *discarded*, the labels of all its outgoing edges are set as *discarded*. If all edges incoming to a goal have status *discarded*, the goal's label is set to *discarded*. If an edge has alternatives with the same antecedent goal and it becomes labeled as *inactive*, then all other alternatives with the same antecedent goal are labeled as *discarded*. If all outgoing edges of a goal are labeled as *discarded*, the goal's label is set to *discarded*, unless the goal was previously set as *achieved*.

To define the status of composed goals, a total ordering of nodes status is used: $\{discarded \prec achieved \prec waiting \prec suspended \prec executing \prec possible\}$. The status of a composed goal is the status of the highest ordered primitive goal that is included into the composed goal. The proposed ordering is designed to present the most important information about the composed goals for agents in the system and ultimately to speed-up the execution of WFs. Underlining existence of *possible* and then *executing* goals facilitates choosing composed goals that agents can commit to.

An example of a WF during execution is presented in Figure 4. From the current state of the system, we can infer that after achieving g_6, goal g_8 was selected from two possible alternatives. After achieving g_8, the execution of two goals: g_{17} and g_{19} started. The goal g_{19} was achieved, while the goal g_{17} is currently executing. The agents that committed to role m_4 have possibility to start execution of goal g_{18}, but they have not yet committed to it .

3.3 Reorganization

Reorganization is the process of introducing changes in an organization. In this work, we consider reorganization on two levels: OS, that defines changes of the core elements of the organization and OE that concerns the entities of the particular instance, and thus can be seen as a reconfiguration of the MAS. However, as OE by definition includes OS, single reorganization can affect both of these levels.

The novel reorganization process definition of ParaMoise exploits the concepts of WFS and WF to provide a fully parallel mechanism that enables concurrent execution of multiple non-conflicting reorganizations.

At the structural level, the reorganization group that includes all roles responsible for the reorganization process was originally defined for $\mathcal{M}$oise$^+$ [7]. ParaMoise redefines it by relaxing the constraint of a single OrgManager (organization manager). Additionally, the roles inheriting from designer as well as historian role are neglected as they are out of the scope of this paper. The separate role of Selector (responsible for selecting one from possibly multiple reorganization designs) is added as in [13]. An additional abstract role *Org* is created, that is the root of all roles in the reorganization group. The purpose of the *Org* role is to create an anchor for compatibility links with other roles in the organization. In this way, the organizational management roles can be fully independent from other roles in the MAS. The final ParaMoise reorganization group structure is presented in Figure 5. Note that all roles cardinalities are default, i.e. $1 \dots \infty$.

The exact mechanisms of monitoring, design, and selection phases of the reorganization are out of the scope of this paper. We assume that agents in the system have capabilities to monitor the organization and send reorganization triggers, then design and select a valid reorganization WFS. The tackled problem is the parallel implementation of a reorganization WFS.

The reorganization plan implementation is represented as a regular WF. The only additional construct needed for reorganization is a *lock* mechanism that prevents contradictory changes in an organization, e.g. removing a role from an OS while this role is adopted by an agent at the same time. In this way, multiple reorganizations can be done in parallel, as long as they do not have overlapping locks. Two types of

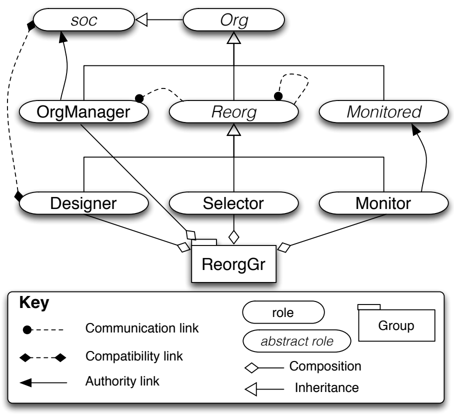

Figure 5: The ParaMoise reorganization group

locks are introduced: *read* and *write*. The *read* lock specifies which part of the organization must remain unchanged for the successful reorganization, while *write* lock specifies which part of the organization will be changed by the reorganization process. For example, when an agent is going to adopt a role, the role specification must be included in read lock. Contrary to that if the role specification is going to be removed, its specification must be included in write lock.

The lock is defined as a tuple $\langle ROE, type \rangle$, where ROE is the reduced organizational entity and $type \in \{read, write\}$ specifies the type of the lock. The core of a lock is its ROE, which is a tuple of structure identical to OE, but containing only the elements of the OE, that need to be locked (as a result some of the fields of the tuple may be empty). As OS is an element of OE, the presented locks can work on both OS and OE levels. Additionally, a lock may be limited to a function's subset of a domain, to reduce the scope of the lock. In such case, the lock on function fun on a subset $\mathcal{S}$ of its domain, is denoted as $fun(\mathcal{S})$. The example of usage of such lock is a change of role assignment of agent a_i: instead of locking the whole function ar, it is enough to create lock that includes $ar(a_i)$.

A reorganization WF, w_r, can be created if no overlap exists between the existing locks and w_r locks or if the overlap is limited to read locks. Only one w_r can be created at a time, and during this procedure the feasibility of new locks must be checked. This constraint is important only for the instantiation of a new w_r, since multiple WFs that are instances of the same WFS may exist and be executed concurrently. The w_r has only one mission to be committed by OrgManagers, i.e. all reorganization goals can be interchangeably executed by OrgManagers. After successful reorganization, the corresponding reorganization locks are destroyed.

The w_r is composed of goals that can modify the OS and/or the OE, possibly combined with auxiliary standard goals. ParaMoise extends the set of state-of-the-art reorganization goals from [13]. On the OS level, reorganization goals include adding, modifying, or removing roles, missions, deontic modalities, groups, WFS, and goals; and changing cardinalities of roles and groups. On the OE level, reorganization goals permit to agents to join or leave the OE, to adopt or leave a role, to swap a role, to commit to or leave a mission, to create or destroy group instances and WFs, and

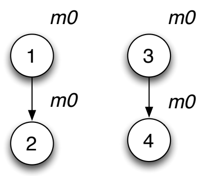

Figure 6: The reorganization WF example

modify the set of the current deontic modalities. A read lock is automatically created for each element created by the w_r and it holds until the end of w_r.

The w_r is capable to represent the reorganization as a parallel process, as presented on the example originally executed sequentially by JaCaMo [13]. Moreover, the JaCaMO considers the reorganization process as global sequence, in which the whole OE is stopped, the OS changed and the new OE started. On the contrary, ParaMoise reorganization requires only to lock the concerned part of organization, and as such does not impact the functioning of whole OE, which might not be acceptable in real LSDS.

The example of reorganization WFS ws_r is presented in Figure 6. The reorganization has only one mission m_0, which is obligatory for all OrgManagers. The goals are: g_1: agent a_1 leaves mission m_1 of WF s_1, g_2: agent a_1 leaves role r_1 of WF s_1, g_3: agent a_2 leaves mission m_2 of WF s_1, g_4: agent a_2 leaves role r_2 of WF s_1. The lock for ws_r is:
$ROE = \langle\langle\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, ar(\{a_1, a_2\}), am(\{a_1, a_2\})\rangle, write\rangle$, as ws_r changes only the mapping of agents to roles and missions.

4. CLOUD COMPUTING MANAGEMENT SYSTEM

In this section, the ParaMoise model is used to specify the organization and reorganization of an exemplary LSDS, a Cloud Computing (CC) system. The next subsection provides a description of the considered system while the second subsection presents its organization modeling using ParaMoise specifications. Finally, the third subsection uses examples to demonstrate ParaMoise ability to model the reorganization in a CC system.

4.1 System Description

Cloud Computing (CC) [14] is one of the major contemporary incarnations of distributed systems, and has its roots in outsourcing IT services to large and efficient data centers. CC services are accessible for any user with an access to the Internet. CC users do not need to invest in infrastructure and maintenance of an IT system and can dynamically increase or decrease the utilization of their services. As a result, CC systems are large scale, complex and dynamic.

The main aim of a CC system is to provide services to its customers. The price and the performance guarantees of a service are described in terms of Service Level Agreement (SLA), which is a part of the contract between a client and a CC system operator. In the proposed model, a service is a group responsible for a set of tasks. A task is an atomic unit of workload processed by the CC system and belongs to only one service. Services have defined in SLA the requested quality of service, the price of the service and the potential penalties in case of violation of SLA conditions. As a result, the goal of a CC system is to fulfill the terms of SLAs, rather than to maximize the amount of completed tasks, as in classical Grid systems.

SLA conditions could have multiple objectives: CC systems should provide services ordered by customers and ensure high performance and availability even in a case of failures of system components (hardware and software). On the other hand, such systems shall minimize their energy consumption for environmental and economical reasons [1].

Typical approach for resource sharing in CC is usage of Virtual Machines (VMs). VM is a container that enables isolation of running operating system as well as additional features like VM runtime migration.

Contemporary CC management systems (e.g. OpenNebula, Nimbus, OpenStack) include only basic autonomic computing capabilities. They use simple mechanisms, e.g. a threshold-based rules for VMs consolidation[1]. A notable state-of-the-art solution is Snooze [3], which is based on self-organizing hierarchical MAS. As a result, the system can self-optimize and self-heal. One of the aims of this work is to complement such approach by providing a model which can generalize reorganization and functioning of a system to increase its performance and capabilities.

4.2 Organizational Specification

The following sections describe the CC organization using ParaMoise Structural, Functional and Deontic Specifications.

4.2.1 Structural Specification

The default structural specification of a CC system (Figure 7) is based on the four main roles we identified in the functioning of an Infrastructure as a Service CC system: Hypervisor, Virtual Machine, Service Auditor, and Resource Allocator. As these components are software entities from the CC system, we make an assumption that System roles are mutually exclusive: agent already playing one System role cannot play another one, instead a new agent shall be dynamically instantiated to play this new role. Despite that, all System roles are compatible with Organization roles. In this way the number of agents automatically scales with the size of the CC system, to assure correct functioning of the CC management system. Moreover, all system roles can communicate, if they belong to the same group.

Agents playing the *Hypervisor* role are installed on the servers and are responsible for VM and hardware management. Examples of hypervisors are Xen [15] or KVM [11]. The cardinality of the Hypervisors role is determined by the system configuration, e.g. the number of servers. However, hypervisor role cardinality can vary, as the system changes over time. A hypervisor can directly monitor its server, e.g. check its utilization, power consumption and temperature.

Agents playing the *VM* role are hosted by hypervisors that have authority on them. They are responsible for processing the tasks in the cloud system. The number of VMs is not bounded, but it is constrained by the hosting system capabilities, as VMs reserve resources of hypervisors.

Agents playing the *Service Auditor* role are specialized in monitoring the performance of services. The auditors can

[1]Aggregation of VMs on a smaller group of hypervisors to turn off the rest of hypervisors and reduce CC system energy consumption in this way.

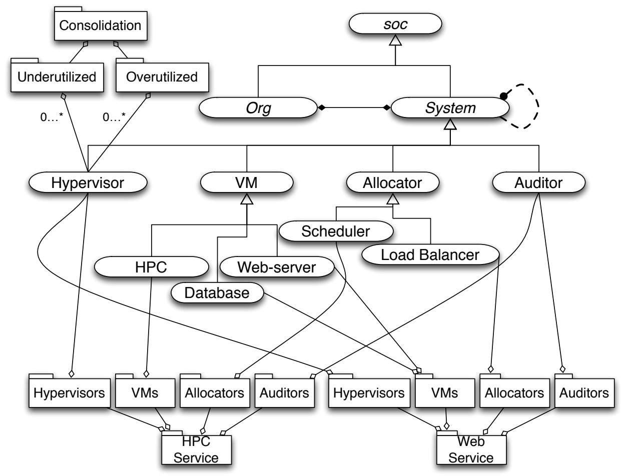

Figure 7: Structural specification of a simple CC system

be a specialized and separated group of VM or they can be hosted externally. In any case, they must provide high availability of the audit, which is the main reason for their separation.

Finally, each agent who plays the *Resource Allocators* role is specialized in allocating tasks to VMs. The resource allocators should have dedicated resources, as their high availability is crucial for the whole system.

There are additional roles inheriting from VM, that correspond to VM specialization: High Performance Computing (HPC) server, Database server, and Web-server. Similarly, the roles inherited from Resource Allocator are Load Balancer (specialized in balancing the large amount of small requests for web servers or databases) and Scheduler (specialized in optimizing the execution of HPC tasks). Two service groups are defined, one for HPC and one for Web Service. These service groups are composed of subgroups which include the roles specific for each service . The default composition rule in this example is 1...∗. Additionally, two sub-groups of hypervisors are defined: Overutilized and Underutilized. Hypervisors in these groups can communicate via the Consolidation group, to aggregate VMs.

This example is a simplification of a real CC scenario in which additional service types with specific requirements of group members, complex inheritance trees for all roles, and additional groups may be needed. Nevertheless, this example is sufficient to present the expressiveness of the ParaMoise organizational model for a CC system.

4.2.2 Functional Specification

The Functional Specification of a CC management system is complex even for a simple system. Due to space limitations, one representative example of WFS is presented.

Small HPC Job is shown in Figure 8. It uses other WFSs as some of its primitive goals and includes multiple missions and precedence relations. g_1 is the job acceptance goal, including selection of the sufficient number of VM for a given deadline. g_2 is the preparation of VMs and g_3 is the preparation of the audit. Goals $g_4 - g_9$ are task execution goals, which are themselves WFSs that concern the specific HPC application. In this case there are 3 data preprocessing goals

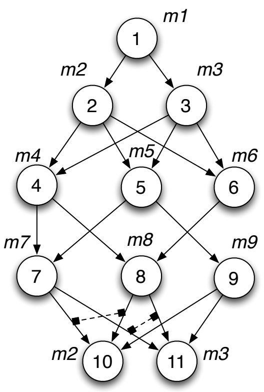

Figure 8: Small HPC Job WFS

$g_4 - g_6$ and three Monte Carlo simulation goals $g_7 - g_9$. Finally, the HPC job is finished by two goals: g_{10} that decommission used VMs and g_{11} that finishes the audit and creates its result. The goals g_7 and g_8 are redundant: completion of any of them is sufficient to complete the WFS.

4.2.3 Deontic Specification

As defined in $\mathcal{M}$oise$^+$, the deontic specification is formed as a set of obligations (*obl*) and permissions (*per*) modalities [6]. In the Small HPC Job example it can be defined as: $obl(Allocator, m1, Now)$, $obl(Hypervisor, m2, Now)$, $obl(Auditor, m3, Now)$, and for $i \in [4; 9]$, $obl(HPC, mi, Now)$.

4.3 CC System Reorganization

A reorganization in a CC system may be caused by exogenous or endogenous triggers. An exogenous trigger example may be the creation of a new service group specification or service group instance. An endogenous trigger example may be the detection of thermal or utilization imbalances by hypervisors. Two sample reorganizations FS are presented in this section. In both cases, there is only one mission $m0$, which is the obligation of OrgManager role.

4.3.1 Creation of a New Service Group Instance

The Creation of a New Service Group Instance reorganization WFS (see Figure 9) is an example of an exogenously triggered reorganization on the OE level. The name of the new service is *My Service*. The following goals appear in the WFS: g_1: Create My Service group instance, g_2: Create subgroups instances of the new My Service group instance, g_3: Assign Auditors, g_4: Assign VMs and assign the Hypervisors hosting the assigned VMs to the Service Hypervisor subgroup (This primitive goal is a WFS), g_5: Start execution – create the corresponding norms in $\mathcal{O}$, g_6: Assign Allocators, g_7: Start audit – create the corresponding norm, g_8: Start allocation – create the corresponding norm. Each primitive goal can be repeated in case of failure up to 5 times, except the more complex g_4 that can be repeated up to 3 times. The repetitions limits are arbitrary and are set to assure responsiveness of the system on a level acceptable for a service user.

The locks created for this operation are read lock concerning the created My Service group specification and its subgroups specifications (OS level) and a write lock that

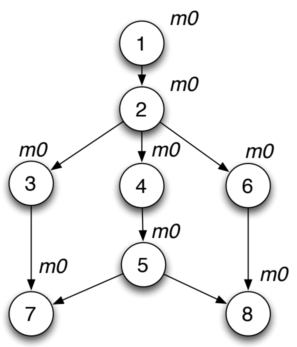

Figure 9: Creation of New Service Group Instance Reorganization WFS

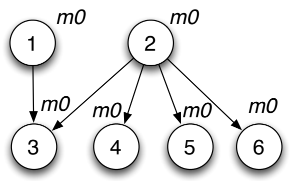

Figure 10: Thermal Emergency Reorganization WFS

includes the *ar* function for the subset of agents that will adopt the roles of the System elements: Allocator, VM, Hypervisor, Auditor (OE level).

This simple example illustrates the superior expressiveness of WFS: even for such simple case, a tree structure cannot capture the precedences in the least constraining way, presented in Figure 9. Given the time needed to perform such an operation in a real CC system, which could be estimated as at least several seconds in case there is no ready VMs, enabling parallel reorganization is critical.

4.3.2 Thermal Emergency

Thermal Emergency reorganization WFS (Figure 10) is an example of endogenously triggered reorganization on both OS and OE levels. The endogenous trigger is a detection of an abnormal thermal phenomena in a system. The WFS is composed of the following primitive goals that are WFS themselves. g_1: Identify affected hypervisors, g_2: Create the group Thermal Emergency for hypervisors and sub-roles of Hypervisor: Affected and Unaffected. g_3: Hypervisors join Thermal Emergency group and adopt corresponding roles, g_4: Notify VMs, g_5: Notify Auditors, g_6: Notify Allocators. Each of the primitive goals can be repeated only once, to assure a fast reaction of the system. The *notify goals* inform *notified roles* about the emergency. This major reorganization requires wide scope of locks: a read lock includes the roles of Hypervisor, VM, Auditor, and Allocator. The write lock include the Hypervisor role, function *ar* for agents playing the Hypervisor role, and all created entities.

5. DISCUSSION

The presented ParaMoise can be implemented using the concept of artifact, similarly to ORA4MAS [5] which is an artifact-based refinement of $\mathcal{M}$oise$^+$. The implementation details of artifact can enhance the parallel execution. For example, if all elements of OE are represented as artifacts, they can be replicated for a parallel access and reliability. A following synchronization mechanism of replica shall be lightweight, under the assumption that in the properly designed system typical reorganization concerns only OE elements excluding OS. In such cases reorganizations have limited scope and concerns only part of the OE so the corresponding locks and changes are limited. As a result, multiple reorganizations can be concurrently executed in a LSDS.

6. CONCLUSIONS

This article proposes ParaMoise as a variant of the $\mathcal{M}$oise$^+$ organization and reorganization model which uses Workflow Specification and Workflow together with corresponding sets of execution rules to form a novel Functional Specification. The workflow-based FS creates the possibility of parallel execution of arbitrary acyclic goals structures. To enhance its capabilities, the mechanisms of alternatives (among precedences incoming to a goal or outgoing from it) and goals repetitions are adopted. In addition, nested workflows are made possible using goals representing another workflow. Thanks to goal and precedence relations labeling, the progress of a workflow can be easily tracked.

The workflow is used also by a novel reorganization model, capable to change any aspect (Structural, Functional, Deontic) of a MAS at runtime. The reorganization is scalable, fault tolerant and strongly parallel: it can be executed by multiple agents, and multiple reorganizations can occur at the same time thanks to the proposed lock mechanism. The reorganization is started by a trigger that can be exogenous or endogenous. There is no exact definition of the creation of a reorganization plan: it can be created at runtime by agents or it can be predefined. As a result, agents can autonomously decide what and how to reorganize.

The future work directions include usage of the ParaMoise model to fully describe a real CC management system, implementation of the proposed model, the design of effective and efficient model distribution strategies, and the development of reorganization design and selection mechanisms for ParaMoise.

7. ACKNOWLEDGEMENTS

M. Guzek acknowledges the support of the National Research Fund of Luxembourg (FNR) and Tri-ICT, with the AFR contract no. 1315254. This work was completed with the support of the FNR GreenIT project (C09/IS/05).

8. REFERENCES

[1] D. J. Brown and C. Reams. Toward energy-efficient computing. *Queue*, 8(2):30:30–30:43, Feb. 2010.

[2] V. Dignum. *A model for organizational interaction: based on agents, founded in logic.* PhD thesis, Proefschrift Universiteit Utrecht, 2003.

[3] E. Feller, L. Rilling, and C. Morin. Snooze: A scalable and autonomic virtual machine management framework for private clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 482 –489, may 2012.

[4] M. Ghijsen, W. Jansweijer, and B. Wielinga. Towards a framework for agent coordination and reorganization, agentcore. In *Proceedings of the 2007 international conference on Coordination, organizations, institutions, and norms in agent systems III*, COIN'07, pages 1–14, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] J. Hübner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20:369–400, 2010.

[6] J. Hübner, J. Sichman, and O. Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In G. Bittencourt and G. Ramalho, editors, *Advances in Artificial Intelligence*, volume 2507 of *Lecture Notes in Computer Science*, pages 439–448. Springer Berlin / Heidelberg, 2002.

[7] J. Hübner, J. Sichman, and O. Boissier. Using the $\mathcal{M}$oise$^+$ model for a cooperative framework of mas reorganisation. In A. Bazzan and S. Labidi, editors, *Advances in Artificial Intelligence, SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, pages 481–517. Springer Berlin / Heidelberg, 2004.

[8] J. F. Hübner, J. S. Sichman, and O. Boissier. Developing organised multi-agent systems using the $\mathcal{M}$oise$^+$ model: Programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.

[9] S. Hwang and C. Kesselman. Grid workflow: a flexible failure handling framework for the grid. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 126 – 137, june 2003.

[10] R. Kota, N. Gibbins, and N. R. Jennings. Decentralised structural adaptation in agent organisations. In *AAMAS-OAMAS*, pages 54–71, 2008.

[11] KVM [online]. `http://www.linux-kvm.org`.

[12] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. N. Prasad, A. Raja, R. Vincent, P. Xuan, and X. Q. Zhang. Evolution of the gpgp/tæms domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9:87–143, 2004.

[13] A. Sorici, G. Picard, O. Boissier, A. Santi, and J. F. Hübner. Multi-Agent Oriented Reorganisation within the JaCaMo infrastructure. In *Proceedings of The Third International Workshop on Iinfraestructures and tools for multiagent systems: ITMAS 2012*, pages 135–148, Valencia, Espagne, 2012.

[14] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, Dec. 2008.

[15] Xen [online]. `http://xen.org`.

[16] W. Zheng-guang and L. Xiao-hui. A graph based simulation of reorganization in multi-agent systems. In *Intelligent Agent Technology, 2006. IAT '06. IEEE/WIC/ACM International Conference on*, pages 129 –132, dec. 2006.