Pre-proceedings of the International Workshop on

COORDINATION, ORGANIZATION, INSTITUTIONS AND NORMS in AGENT SYSTEMS

COIN@AAMAS2009

Held as a satellite event of the
Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)

Budapest, Hungary, 12 May 2009

Preface

Multi-agent systems (MAS) are complex artifacts in which a multitude of autonomous software agents interact, pursuing individual and/or collective goals. Such a view usually assumes some form of organization, a set of norms or conventions that articulate or restrain interactions in order to enable agents to achieve their goals. The engineering of effective coordination or regulatory mechanisms is a key problem for the design of MAS.

This workshop aims at bringing together the topics of Coordination, Organization, Institutions and Norms (COIN). A significant number of influential papers on these topics have been appearing in the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) and other conferences and workshops on software agents and multi-agent systems. The series of COIN workshops is aimed at consolidating and expanding these topics by providing focused events in which researchers from different communities participate.

MAS such as formal organizations, electronic institutions and (software) agent societies, often have to adapt in order to reflect, among others, environmental, economic and social changes in them. As the operation of such systems must not stop, adaptation cannot be achieved by restarting the system. Instead, this adaptation should be attained by adapting system components and the way they interact at run-time.

In order to explore the issue of adaptation in MAS, the COIN workshop series merged with the International Workshop on Organised Adaptation in Multi-Agent Systems (http://oamas08.iit.demokritos.gr/). Consequently, papers that explore the dynamic aspects of norms, organizations and institutions are now particularly welcome in COIN.

12 papers were accepted for presentation in COIN@AAMAS2009 as a result of a rigorous selection process on 19 submissions. These papers, along with the papers accepted for presentation at the other two editions of COIN 2009, will be published in a Springer LNAI volume, on the proviso that they go through a second round of review.

We would like to thank the members of the COIN steering committee for their guidance, the members of the programme committee for their reviews and all authors for submitting their work to COIN@AAMAS2009. We are confident that very interesting and fruitful discussions will take place during the workshop.

Budapest, May 2009

Alexander Artikis
Wamberto Vasconcelos

Workshop Organisers

Alexander Artikis	NCSR "Demokritos", Greece
Wamberto Vasconcelos	University of Aberdeen, UK

Steering Committee

Guido Boella	Università degli Studi di Torino, Italy
Olivier Boissier	Ecole Nationale Superieure des Mines de Saint-Etienne, France
Virginia Dignum	Utrecht University, The Netherlands
Nicoletta Fornara	University of Lugano, Italy
Christian Lemaitre	Universidad Autonoma Metrolitana, Mexico
Eric Matson	Wright State University, USA
Pablo Noriega	IIIA, Spain
Sascha Ossowski	Universidad Rey Juan Carlos, Spain
Julian Padget	University of Bath, UK
Jeremy Pitt	Imperial College London, UK
Jaime Sichman	University of Sao Paulo, Brazil
Wamberto Vasconcelos	University of Aberdeen, UK
Javier Vazquez Salceda	Universitat Politecnica de Catalunya, Spain
George Vouros	University of the Aegean, Greece

Programme Committee

Alexander Artikis	NCSR "Demokritos", Greece
Guido Boella	Università degli Studi di Torino, Italy
Olivier Boissier	Ecole Nationale Superieure des Mines de Saint-Etienne, France
Stephen Cranefield	University of Otago, New Zealand
Cristiano Castelfranchi	ISTC, Rome, Italy
Virginia Dignum	University of Utrecht, The Netherlands
Marc Esteva	IIIA, Spain
Nicoletta Fornara	University of Lugano, Switzerland
Jomi Fred Hubner	University of Blumenau, Brazil
Lloyd Kamara	Imperial College London, UK
Victor Lesser	University of Massachusetts Amherst, USA
Christian Lemaitre	Universidad Autonoma Metropolitana, Mexico
Eric Matson	Wright State University, USA
John-Jules Meyer	University of Utrecht, The Netherlands
Daniel Moldt	University of Hamburg, Germany
Pablo Noriega	IIIA, Spain
Tim Norman	University of Aberdeen, UK
Eugenio Oliveira	Universidade do Porto, Portugal
Sascha Ossowski	Universidad Rey Juan Carlos, Spain
Julian Padget	University of Bath, UK
Alessandro Ricci	Universita di Bologna, Italy
Antonio Carlos da Rocha Costa	UCPEL, Brazil
Juan Antonio Rodriguez-Aguilar	IIIA, Spain
Jaime Sichman	University of Sao Paulo, Brazil
Carles Sierra	IIIA, Spain

Contents

A Norm-based Organization Management System

N. Criado, V. Julián, V. Botti, E. Argente *

Grupo de Tecnología Informática - Inteligencia Artificial
Departamento de sistemas informáticos y computación
Camino de Vera S/N 46022 Valencia (Spain)
{ncriado,vingalda,vbotti,eargente}@dsic.upv.es

Abstract. Virtual organizations are conceived as an effective mechanism for ensuring coordination and global goal fulfilment of an open system, in which heterogeneous entities (agents or services) interact and might also present self-interested behaviours. Organizations describe the system functionality, structure and dynamics. However, available tools rarely give support for these organizational abstractions. The THOMAS multi-agent architecture allows the development of open multi-agent applications. It provides a useful framework for the development of virtual organizations, on the basis of a service-based approach. In this paper, the Organization Management System component of the THOMAS architecture is presented. The Organization Management System is in charge of the organization life-cycle process, including the normative management. It provides a set of structural, informative and dynamic services, which allow describing both specification and administration features of the structural elements of the organization and their dynamics. Moreover, it makes use of a normative language for controlling the service request, provision and register. In this paper, the Organization Management System and its implementation are detailed.

Key words: Multi-Agent Systems, Normative Language, Virtual Organizations, Web Services.

1 Introduction

A promising current approach in the MAS area is the development of open systems, in which heterogeneous entities can freely join and leave the system, participating inside under regulated restrictions. Organizations are conceived as an effective mechanism for coordinating the behaviour of heterogeneous agents, imposing not only structural restrictions on their relationships, but also normative restrictions on their behaviour [1, 2]. Thus, organizations describe the

* This work is supported by TIN2005-03395 and TIN2006-14630-C03-01 projects of the Spanish government, GVPRE/2008/070 project, FEDER funds and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022, FPU grant AP-2007-01256 awarded to N.Criado.

system functionality (i.e. roles, tasks, services), the norms that control agent behaviours, the formation of groups of agents, the global goals pursued by these groups and the relationships between entities and their environment. Organization dynamics need to be managed in order to clearly define the organization life-cycle, which includes its creation, performance and restructuring.

Moreover, the "computing as interaction" paradigm [3] defines computation as an inherent social activity that takes place by means of communication between computing entities. More specifically, large distributed systems are conceived in terms of service provider or consumer entities [4]. Therefore, the relevant technological approaches of this paradigm are service oriented architectures and multi-agent systems. Services provide a standard infrastructure for the interaction of heterogeneous software entities. On the other hand, multi-agent systems offer a more general and complex notion of service oriented architectures (SOA); agents, due to their intelligent and social capabilities, allow the redefinition of traditional services adding new features such as dynamic composition, negotiation about quality of service, etc. In the last years, several works have focused on the problem of integrating these two approaches, in order to model autonomous and heterogeneous computational entities in dynamic and open environments. Their main effort is directed at masking services for redirection, integration or administration purposes [5].

Taking into account this integrating view, THOMAS has been defined as an open architecture for large-scale open multi-agent systems, based on a service-oriented approach [6]. This architecture provides agents with a set of services for offering and discovering entities' functionality and for managing the organization life-cycle. With this purpose of achieving a better integration among MAS and Web Services, agents provide, require and publish their functionalities employing Web Services standards (such as OWL-S), and they can also make use of traditional Web Services.

One of the main components of the THOMAS architecture is the Organization Management System (OMS), which is responsible for the management of the organizations and their constituent entities. In order to allow this management, the OMS must provide a set of structural, informative and dynamic services for describing both specification and administration features of the structural elements of the organization and their dynamics.

With the aim of allowing the organization management, the OMS requires a normative language for controlling how services can be employed, i.e., when and how agents can request, provide or publish not only their services, but also these ones provided by the open architecture. In this sense, a normative language that imposes a deontic control over agents for requesting, providing or publishing services has been defined. The OMS is also in charge of controlling all norms applying to the organization life-cycle services.

Following, a description of the proposed Organization Management System component, in charge of the organization management by means of organizational services is detailed. Its norm management process, both with a description of its norm representation language is detailed in section 3. The implementation

of this Organization Management System component is explained in section 4. Finally, discussion and conclusions are detailed in sections 5 and 6, respectively.

2 Organization Management System

As previously mentioned, the THOMAS architecture has the aim of integrating both multi-agent systems and service-oriented computing technologies as the foundation of virtual organizations.

This open architecture for large-scale open multi-agent systems is composed by a range of services included on different modules or components[1]. In this sense, agents have access to the architecture infrastructure through the following components:

- **Service Facilitator** (SF). This component provides a mechanism and support by which organizations and agents can offer and discover services.
- **Platform Kernel** (PK). It maintains the basic management services for an agent platform. It integrates the FIPA AMS and the FIPA communication network layer.
- **Organization Management System** (OMS). This component is mainly responsible of the management of the organizations and their entities. Thus, it gives support to the organization life-cycle management.

The present paper is focused on this last component. It is in charge of controlling the organizational life-cycle process. In THOMAS, organizations are structured by means of *Organizational Units* (OU) [7] which represent a set of agents that carry out some specific and differentiated activities or tasks, following a predefined pattern of cooperation and communication. An OU is formed by different entities along its life-cycle which can be either single agents or other OUs. They represent a virtual meeting point because agents can dynamically enter and leave organizational units, by means of adopting (or leaving) roles inside. An OU has also an internal topology (i.e. hierarchical, team, plain), which imposes restrictions on agent relationships (for example, supervision, monitoring or information relationships). A more detailed explanation of the OU concept and a description of different topologies can be found in [7].

The OMS is in charge of controlling how the Organizational Units are created, which entities are participating inside them, how these entities are related and which roles they are playing through time. For this reason, the OMS offers agents a set of services for organization life-cycle management, classified in:

- **Structural services**, which enable agents to request the OMS to modify the structural and normative organization specification. They comprise services for adding/deleting norms (*RegisterNorm*, *DeregisterNorm*), adding/deleting roles (*RegisterRole*, *DeregisterRole*) and creating new organizational units or deleting them (*RegisterUnit*, *DeregisterUnit*). Publishing these services enables agents to modify the organization structure through its life-time.

[1] http://www.fipa.org/docs/THOMASarchitecture.pdf.

– **Informative services**, that provide information of the current state of the organization, detailing which are the roles defined in a OU (*InformUnitRoles*), the roles played by an agent (*InformAgentRoles*), the specific members of an OU (*InformMembers*), its member quantity (*InformQuantity*), its internal structure (*InformUnit*), and the services and norms related with a specific role (*InformRoleProfiles*, *InformRoleNorms*).
– **Dynamic services**, which allow defining how agents can adopt roles inside OUs (*AcquireRole*, *LeaveRole*) or how agents can be forced to leave a specific role (*Expulse*), normally due to sanctions. Publishing these services enables external agents to participate inside the system.

This set of services for organization life-cycle management allows defining specification and administration features of the structural components of the organization (roles, units and norms) and their dynamics (entry/exit of entities). However, a specific control on who can make use of these services and in which conditions is needed. This type of control is defined by means of norms.

3 Norm Management

Normative systems have been defined as a mechanism for enabling cooperation inside Open MAS. In this sense, norms are persuasive methods for obtaining the desired behaviour from agents. In addition, norms can be viewed as a coordination skill for organizing MAS, since they specify the desired behaviour of the society members [8]. Regarding this second conception of norms, our proposal consists on employing norms for regulating agent organizations. More specifically, this work has the purpose of applying the normative theory for defining the way in which agents may modify the structure of their organization (norms, organizational units and roles) and its execution components, in order to adapt it dynamically to environmental changes.

Recently, works on norms have focused on overcoming the gap between theoretical works on normative systems and practical MAS. They give a computational interpretation of norms that allows norm execution. However, none of them raises the normative management problem or gives an infrastructure that enables including the normative theory inside the implementation of real MAS applications. With this aim, we have developed both a normative language for controlling agent organizational dynamics and a normative management engine, which are explained in the following sections.

3.1 Norm Representation Language

Before addressing the norm controlling problem, the definition of a formal language for the representation of the normative concepts is needed. Our normative language is mainly based on previous works for implementing norms inside Electronic Institutions (EI) [9, 10]. These works define a normative language for controlling the communicative acts (illocutions) of agents inside an EI. In addition, they propose an extension of this language for allowing the definition of

norms concerning non-dialogical actions. Our language takes these approaches as a starting point and increases them in order to give support to functional and organizational management. More concretely, our normative language makes possible the definition of constraints on agent behaviours in terms of actions related to service controlling and organizational processes. The main contributions of the proposed language are: (i) it allows the definition of consequences of norms by means of sanctions and rewards; (ii) it gives support to organizational concepts such as roles or organizational units and; (iii) the definition of agents' functionality by means of the OWL-S standard increases norm expressiveness, as will be argued lately.

Following, some issues about the developed normative language are commented. For a more detailed description of this language see [11].

The proposed language is a coordination mechanism that attempts to: (i) promote behaviours satisfactory to the organization, i.e. actions that contribute to the achievement of global goals; and (ii) avoid harmful actions, i.e. actions that prompt the system to be unsatisfactory or unstable. Norm semantics is based on deontic logic since it defines obligations, permissions and prohibitions. Our approach conceives norms as expectations that may not be fulfilled. Thus, norms are not impositions (i.e. they are not automatically enforced on agents by their designer), but they are methods for persuading agents to behave correctly by means of the application of sanctions and rewards. For example, an agent would be expelled from the organization as sanction if it violates norms systematically. An analysis on the effectiveness of both sanctions and rewards as mechanisms for enforcing norms is over the scope of this article. However, this work assumes that agents are aware of norms, punishments and rewards. In this sense, a normative reasoning process for norm-aware agents has been proposed in [12].

Norms define agent rights and duties in terms of actions that agents are allowed or not to perform. Actions have been divided in two categories: actions related to the organizational aspects of MAS; and actions concerning service accessing. Hence, two main types of norms have been defined:

- **Organizational Norms**: related to services offered by the OMS to members of the organization. They establish organizational dynamics, e.g. role management (role cardinalities, incompatibility between roles) and the protocol by which agents are enabled to acquire roles.
- **Functional Norms**: related to services offered by the members of the organization or the SF. They define role functionality in terms of services that can be requested/provided, service requesting order, service conditions, protocols that should be followed, etc. They establish service management according to previous service results, environmental states, etc.

Table 1 details the reduced BNF syntax of this language. A norm is defined by means of a deontic operator (<*deontic_concept*>), an addressed entity and an action, that concerns organizational (<*organizational_action*>) or functional (<*functional_action*>) management. The <*temporal_situation*> field establishes a temporal condition for the activation of the norm. It can be expressed as a

deadline, an action or a service result. A norm may also contain a state condition for its activation (<*if_condition*>). It is a boolean condition that can be expressed over some variables, identifiers, failed or satisfactory states of norms or final result of services.

<norm>::=<deontic> <entity> <action> [<temporal>] [*IF* <if_condition>] | *norm_id*	
<ext_norm>::=<norm> [*SANCTION*(<norm>)] [*REWARD*(<norm>)]	
<deontic>::=*OBLIGED* | *FORBIDDEN* | *PERMITTED*	$\phi : FORBIDDEN\ \alpha \equiv [\alpha]V$ $\phi : OBLIGGED\ \alpha \equiv [\neg\alpha]V$ $\phi : PEMITTED\ \alpha \equiv [\alpha]\neg V$
<entity>::=<agent>: <role> [− <unit>] | <role> [− <unit>] | <entity_id>	$\phi : \phi' SANCTION\ \alpha \equiv \phi' \wedge [V]DO(\alpha)$ $\phi : \phi' REWARD\ \alpha \equiv \phi' \wedge [\neg V]DO(\alpha)$
<agent>::=?*variable* | *agent_id* <role>::=?*variable* | *role_id* <unit>::=?*variable* | *unit_id* <entity_id>::=*agent_id* | *role_id* | *unit_id*	$\phi : \phi' BEFORE\ \alpha \equiv \phi' \vee DONE(\alpha)$ $\phi : \phi' AFTER\ \alpha \equiv [\alpha]\phi'$ $\phi : \phi' BETWEEN(\alpha_1, \alpha_2) \equiv [\alpha_1]\phi' \vee DONE(\alpha_2)$
<action>::=<functional_action> | <organizational_action>	$\phi : \phi' IF \beta \equiv \beta \rightarrow \phi'$
<temporal>::=*BEFORE* <sit> | *AFTER* <sit> | *BETWEEN*(<sit> , <sit>)	

Table 1. On the left side, BNF syntax of norms is detailed. On the right side, its semantics expressed by means of dynamic logic is given. α is an action description. β is an state description. V, $DO(\alpha)$ and $DONE(\alpha)$ are the well-known predicates for representing violation states, an action α that will be done next and an action α that has been performed. Finally, ϕ represents a norm.

As an example, norm 1 authorizes an agent to request acquisition of the *Subordinated* role. The addressed agent is free to perform this action. Usually, obligations and prohibitions have sanctions and rewards as persuasive methods. Sanctions and rewards are represented through norms addressed to entities that will act as norm defenders or promoters. The definition of sanctions and rewards recursively as norms can create an infinite chain of norms. Thus, not only addressed agents might be controlled by norms, but also their controllers (defenders or promoters). Following M. Luck et al. proposal [13], our normative model does not impose any restriction on this fact, so it is the norm designer who is in charge of specifying when to stop this recursive process, i.e. when a controller is trustworthy enough. For example, norm 2 obliges a *Supervisor* agent to request *AddUnit* service; if the *Supervisor* agent does not respect the norm, then it will be expelled from the organization by the OMS as sanction. Norm 2 contains a state condition and a temporal condition also. In this case, these conditions indicate that the agent is obliged to request *AddUnit* service before 10 seconds if it is the only *Supervisor* inside the organization.

$$\begin{aligned} &agent : ?role\ PERMITTED\ REQUEST\ AcquireRole \\ &MESSAGE(CONTENT(RoleID\ \text{“}Subordinated\text{”})) \end{aligned} \tag{1}$$

$$
\begin{aligned}
?agent : Supervisor\, OBLIGED\, REQUEST\, AddUnit \\
IF\, InformQuantity(\text{“}Supervisor'') = 1 \\
BEFORE(10'') \\
SANCTION\, oms\, OBLIGED\, PROVIDE\, Expulse \\
MESSAGE(CONTENT(?agent, \text{“}Supervisor''))
\end{aligned}
\tag{2}
$$

Organizational norms are related to actions that allow agents to request organizational services (<*org_service*>) for adopting roles, registering new norms, etc. These services are provided by the OMS. The BNF syntax of organizational actions is detailed in Table 2.

<organizational_action>::=REQUEST <org_service> MESSAGE(<msg_cont>)
<org_service>::=<structural_service> | <dynamic_service> | <informative_service>
<structural_service>::=RegisterNorm | RegisterRole | DeregisterNorm | DeregisterRole | DeregisterUnit | RegisterUnit
<informative_service>::=InformUnitRoles | InformAgentRoles | InformUnit | InformMembers | InformRoleProfiles | InformRoleNorms | InformQuantity
<dynamic_service>::=AcquireRole | LeaveRole | Expulse

Table 2. BNF syntax of organizational actions

Functional norms are defined in terms of actions related to the publication (REGISTER), provision (SERVE) or usage (REQUEST) of services. The BNF syntax of this type of actions is detailed in Table 3. Norm 3 contains an example of a functional norm. It obliges a service *Provider* agent to register its own *SearchService* service.

<functional_action>::=<serv_publication> | <serv_provision> | <serv_usage>
<serv_publication>::=REGISTER *service_name* PROFILE <profile_desc> [PROCESS<process_desc>]
<service_provision>::=SERVE *service_name* PROCESS <process_desc> [MESSAGE(<msg_cont>)]
<service_usage>::=REQUEST *service_name* MESSAGE(<msg_cont>)

Table 3. BNF syntax of functional actions

$$
\begin{aligned}
?agent : Provider\, OBLIGED \\
REGISTER\, SearchService \\
PROFILE \\
INPUT(ServiceDescription : String) \\
OUTPUT(ServiceID : Identifier)
\end{aligned}
\tag{3}
$$

As previously mentioned, the specification of functionalities by means of OWL-s standard allows defining functionality more expressively: representing service preconditions and effects; global functionalities are described as complex services that are composed of atomic services, so a complex service specification describes how agent behaviours are orchestrated; and functionality is detailed in two ways: services that entities perform and services that entities need. Thus, Service Oriented Computing (SOC) concepts such as ontologies, process models, choreography, facilitators, service level agreements and quality of service measures can be applied to MAS. Our proposal of normative language offers support for specifying knowledge about a service following OWL-s ontology. The profile

(<*profile_desc*>) for advertising and discovering services contains input and output parameters of the service and its preconditions and postconditions. The process model (<*process_desc*>) gives a detailed description of a service's operation. It details the sequence of actions carried out by the service. These actions are linked through each other by means of different control constructs: CONNECTS indicates a sequential ordering between two actions; JOIN indicates a concurrence between actions and a final synchronization of them; IF-THEN-ELSE and WHILE-DO define the classical control structures. Finally, the grounding provides details on how to interoperate with a service, via messages <*msg_cont*>). Table 4 contains syntax of service processes, profiles and requesting messages.

<profile_desc>::=[INPUT(<param_list>)] [OUTPUT (<param_list>)]
[PRE(<cond_exp>)][POST(<cond_exp>)] | *profile_id*

<process_desc>::=*process_id* | ?*variable* | <action> CONNECTS <process_desc> |
<action> JOIN <process_desc> |
IF <cond_exp> THEN(<process_desc>) [ELSE (<process_desc>)] |
WHILE <cond_exp> DO(<process_desc>)

<msg_cont>::=[SENDER(<entity>)] [RECEIVER (<entity>)]
[PERFORMATIVE (*performative_id*)] CONTENT (<args>)

<action>::=*task_id*(<param_list>) | <service_usage>

<param_list>::=*variable* : *type* [,<param_list>]

Table 4. BNF syntax of service profile and process

In this section, a general language for controlling agent service access has been briefly described. For further details and examples see [11].

3.2 Norm Management Process

Once the general characteristics of our proposed normative language for controlling agent service access have been described, some aspects related to the management of organizational norms are commented in this section. As previously mentioned, our formalism allows representing constraints over organizational dynamics. Thus, the controlled norms define access to the organizational services provided by our architecture.

Recently, the line of research on norm implementation is based on the Electronic Institution (EI) proposal [9, 10]. The EIs provide a framework for heterogeneous agent cooperation. However, they are not an open environment in its broadest sense, because agents interact inside the institution through the infrastructure provided by the EI. Therefore, the behaviour of external agents is completely controlled by the institution, which allows or not agents to pronounce certain illocutions. Thus, norms are pre-imposed on agents. The institutional mediation prevents agent behaviour from deviating from desired behaviour. Moreover, the fact that all communications are made through the institution allows an easy implementation and enforcement of norms. In this sense, the existence of a middleware for mediating the agent communication avoids the need to take into consideration the limitations that exist in open environments. Such limitations are related to the detection of fact occurrence and the extra capabilities needed in order to impose norms upon other agents.

Our proposed virtual organization architecture is completely different since it does not have any mediator layer. On the contrary, agents are allowed to interact freely. In this sense, our architecture is more related to the notion of *Partially Controlled MAS* [14], in which agents may deviate from ideal behaviour. As a consequence, there has to be a control mechanism for motivating agents to obey the norms. However, our architecture offers a set of services for the management of the organizations life-cycle. In this sense, the OMS is not a centralized entity that controls all the interactions performed by agents. On the contrary, it is a *controllable* entity (which is directly controlled by the system designer [14]) which offers a set of organizational services in order to give support to agent cooperation. One of these coordination mechanisms is the definition of norms that regulate agent behaviour. The OMS does not control the agent communications, it is only in charge of enforcing the norms that regulate access to OMS services. The verifiability issue of a norm is a crucial aspect in the normative management process since there is not any mediator middleware that controls all the communications.

Norm Verifiability Works on norm implementation conceive norms as a mechanism for enabling coordination and cooperation inside open MAS. Nevertheless, none of them faces with one of the main problems inside open systems, which is the existence of limitations. The term *limitation* refers to the fact that an entity needs extra information and capabilities in order to act as a norm supervisor or controller. Therefore, an analysis of normative verifiability is needed, before dealing with norm implementation.

The OMS can control norms which are related to the provision of organizational services. Therefore, the set of OMS verifiable norms is a subset of the organizational norms. Verifiable norms are *regulative* norms that define ideal behaviour by means of obligations, prohibitions and permissions. More specifically, permissive and prohibition norms concerning the access to OMS services are controllable, since the OMS checks whether the client agent is allowed to perform such request before providing it. On the other hand, obligation norms can not be imposed by the OMS as it has not capabilities to force agents to carry out an action. However, the OMS can detect the violation of an obligation norm related to an OMS service and perform the associated sanction. On the contrary, if the norm has been fulfilled then the OMS will carry out the actions specified by the reward. Logically, obligation norms should be active for a certain period of time, i.e. normative activation and deactivation events must be defined in order to allow the OMS to determine the norm fulfilment.

Table 5 contains BNF syntax for the definition of verifiable conditions. They are related to informative services provided by the OMS (<*variant*>) such as role cardinalities, information about roles played by agents, etc.

Regarding detectable events, their syntax is detailed in Table 6. Basically, they are the request (<*ServiceRequest*>) and provision (<*ServiceProvision*>) of services offered by the OMS. Both verifiable condition and detectable events syntax are a refinement of the general condition syntax proposed in [11].

Finally, sanctions and rewards can be defined by means of norms that oblige the OMS to request or provide a specific service, as shown in norm 2.

In this section, issues concerning the formal language for representing norms and syntax of verifiable organizational norms have been detailed. Not only an abstract component in charge of the management of organizations has been proposed, but also an implementation of the OMS component has been made. Next, this implementation is described.

<if_condition>::=NOT(<cond_exp>) | < cond_exp >

<cond_exp>::=<condition> | NOT <condition> | (<condition> AND <cond_exp>) | (<condition> OR <cond_exp>)

<condition>::=<variant> <operator> <variant>

<operator>::=> | < | >=| <= | =

<variant>::=<informative_service>(<args>) | *value*

Table 5. Verifiable conditions syntax

<sit>::=<event> (<event> AND <sit>) | (<event> OR <sit>)

<event>::=<ServiceRequest> | <ServiceProvision> | *deadline*

<ServiceRequest>::=[<entity>] REQUEST <organizational_action>

<ServiceProvision>::=(<ServiceRequest>) = <ServiceResult>

<ServiceResult>::=Not-Allowed | Error | Provided

Table 6. Verificable events syntax

4 Organization Management System Implementation

As previously argued, the OMS is a *controllable* entity which offers a set of organizational services. In this proposal, these services have been developed as an agent implemented as a rule-based system programmed in Jess [2]. This system maintains a fact base representing the organizational state, and also it is in charge of controlling verifiable norms. The implementation has been included in a prototype of the THOMAS architecture[3]. Following, the specific implementation of both service and norm management provided by the OMS is detailed.

4.1 Service Management Implementation

The implementation of services should consider the maintenance of the organizational state as well as the existence of norms that regulate access to organizational services. When a new service request is sent to the OMS, it registers this fact in the organizational rule-based system and carries out the service provision. As previously stated, the set of services provided by the OMS are those ones related to the organizations life-cycle management, i.e. the management of their structural components (unit, roles and norms) and the organizational dynamics.

Figure 1 contains a portion of the source code of the function that performs the service request management process:

[2] http://herzberg.ca.sandia.gov/

[3] http://www.dsic.upv.es/users/ia/sma/tools/Thomas/

– (i) Firstly, the normative context must be analyzed in order to determine whether the client agent is allowed to request this service according to the organizational state.
– (ii) If so, then the OMS provides the requested service.

Mainly, the service provision process consists of updating facts and rules belonging to the rule system. Following, as an example, the source code of the service that allows registering a new norm is shown in Figure 2. The process of registering of a new norm involves:

```
(deffunction serviceRequest (?agentid ?serviceid $?info)
 i) check normative context
(if (not(isAllowed ?agentid ?serviceid ?info))
 then
   (assert(servicerequest(agentid ?agentid)(serviceid ?serviceid)(info ?info)
    (state not-allowed)))
   (return "not-allowed"))
ii) service execution
(if (not(apply ?serviceid ?agentid ?info))
 then
   (assert(servicerequest(agentid ?agentid)(serviceid ?serviceid)(info ?info)(state error)))
   (return "error"))
(assert(servicerequest(agentid ?agentid)(serviceid ?serviceid)(info ?info)(state provided)))
(return "provided"))
```

Fig. 1. Service Request Management Function

– (i) Verification of input data, i.e. checking the adequacy of norm syntax as well as the uniqueness of the provided norm identifier.
– (ii) Norm verifiability detection, i.e. determine if the norm control should be carried out by the OMS. Verifiability checking is done by means of an interpreter, which has been automatically built from the BNF syntax of our organizational normative language employing the JavaCC[4] tool.
– (iii) Inconsistency checking. Taking [15, 16] as a starting point, an inconsistent situation is that one in which an action is forbidden and permitted or forbidden and obliged simultaneously. The off-line detection of all norm inconsistencies is not possible, since the norm activation conditions are based on the detection of events and facts that may occur during execution. For the moment, the inconsistency detection mechanism is restricted to the determination of *static inconsistencies*, which are situations in which the same action is defined unconditionally as permitted and forbidden to the same entity. As future work, we will employ results of theoretical works on norm change[5] as well as conflict resolution techniques for solving *dynamic inconsistencies* among norms.
– (iv) The fact corresponding to the new norm is registered in the rule system.

Due to lack of space, only the organizational service for registering a new norm has been explained in detail. However, all services needed for the management of units, roles, norms and organizational dynamics, have been implemented in a similar way.

[4] https://javacc.dev.java.net/
[5] http://icr.uni.lu/normchange07/

4.2 Norm Management Implementation

The OMS is responsible for the provision of organizational services and the performance of the norm management process. Norm implementation covers the detection of norm activation and deactivation as well as the determination of the allowed actions according to the defined normative context.

```
(deffunction addnorm (?agentid ?normid ?norminfo)
 (if (> 0 (count-query-results getnorm ?normid)) then(return FALSE))
 i) check norm syntax
 (bind ?interpreter (new NormativeLanguageParser.NormativeInterpreter))
 (bind ?content (call ?interpreter checkNorm ?norminfo))
 (assert(norm (normid ?normid)(norminfo ?norminfo)))
 (if (eq ?content "") then(return FALSE))
 ii) checks norm verificable
 (bind ?interpreter (new omsNormativeLanguageParser.NormativeInterpreter))
 (bind ?content (call ?interpreter checkNorm ?norminfo))
 (if (eq ?content "") then(return TRUE))
 iii)check inconsistencies
 (if (eq (checkConsistency ?content) FALSE) then (return FALSE))
 iv)assert new organizational norm
 (assert-string (str-cat (organizationalnorm (normid " ?normid ") "?content")"))
 (return TRUE))
```

Fig. 2. AddNorm Function

Regarding detection of norm activation and deactivation, this functionality has been implemented through a set of rules that detect the occurrence of the activation and deactivation events. This implementation of norms by means of rules is based on a previous work aimed at implementing norms for Electronic Institutions [9]. Next, some details about the implementation of each type of norm are commented:

– *Service access norms.* As previously mentioned, these norms allow the definition of prohibitions and permissions concerning the use of organizational services. More formally, the semantics of permissions and prohibition norms, expressed as Event-Condition-Action rules (ECA-rules), is:

$$\begin{aligned} \textbf{on } event_{start} \textbf{ if } if_{condition} \textbf{ do } \oplus permitted(a) \\ \textbf{on } a \textbf{ if } permitted(a) \textbf{ do } \oplus provided(a) \\ \textbf{on } event_{end} \textbf{ do } \ominus permitted(a) \\ \textbf{if } not(if_{condition}) \textbf{ do } \ominus permitted(a) \end{aligned} \tag{4}$$

$$\begin{aligned} \textbf{on } event_{start} \textbf{ if } if_{condition} \textbf{ do } \oplus forbidden(a) \\ \textbf{on } a \textbf{ if } forbidden(a) \textbf{ do } \oplus notAllowed(a) \\ \textbf{on } event_{end} \textbf{ do } \ominus forbidden(a) \\ \textbf{if } not(if_{condition}) \textbf{ do } \ominus forbidden(a) \end{aligned} \tag{5}$$

The first four ECA-rules describe semantics of a permission (4), whereas the last four ones represent a prohibition norm semantics (5). According to this, a general service access norm is controlled by means of the definition of four new rules in the expert system. The first rule detects norm activation and asserts the permission ($\oplus permitted(a)$) or prohibition ($\oplus forbidden(a)$). If the action (a) occurs and it is allowed, then the service is provided

($\oplus provided(a)$). On the contrary, if a is forbidden, then *notAllowed* fact is asserted. The last two rules retract the norm from the expert system when the norm is deactivated, i.e when the condition ($if_{condition}$) is not true or the completion event is detected ($event_{end}$). Fig. 3 contains source code of the function that registers a new service access norm. The first rule (i) detects norm activation whereas the last two ones (ii and iii) deactivates the norm if the end event is detected or the activation condition is not true.

```
(defrule newNorm
 ?f<-(norm (normid ?normid) (deonticConcept ?deon)...)
 (test(or (eq ?deon forbidden)(eq ?deon permitted)))
=>
 i) Activation Norm
 (bind ?rule (str-cat "(defrule activateNorm"?normid"
  "?condition" "?after"
    =>
  (assert(activenorm(normid "?normid")...)
 (build ?rule)
 ii) Deactivacion Norm
 (bind ?rule (str-cat "(defrule deactivateNormIF"?normid"
  ?f<-(activenorm(normid "?normid")...)
  (not "?condition")
    =>
  (retract ?f))"))
 (if (not(eq ?condition ""))then(build ?rule))
 iii) Event Deactivacion Norm
 (bind ?rule(str-cat "(defrule deactivateNormBEFORE"?normid"
  ?f<-(activenorm(normid "?normid")...)
  "?before"
    =>
  (retract ?f))"))
 (if (not(eq ?before ""))then(build ?rule)))
```

Fig. 3. New Norm Rule

– *Obligation norms.* They cannot be directly implemented by the OMS, since it is not able to force another agent to carry out a specific action. However, it might persuade agents to behave correctly by performing sanctions and rewards. Following, a formal description of obligation semantics is shown:

$$\begin{aligned} \textbf{on } event_{start} \textbf{ if } if_{condition} \textbf{ do } \oplus expected(a) \\ \textbf{on } a \textbf{ if } expected(a) \textbf{ do } \ominus expected(a) \bullet reward \\ \textbf{on } event_{end} \textbf{ if } expected(a) \textbf{ do } \ominus expected(a) \bullet sanction \end{aligned} \tag{6}$$

Thus, the implementation of obligation norms consists on controlling the activation of the obligation. Then the OMS waits for the fulfilment of the expected action (a), i.e the OMS asserts a new expectation ($\oplus expected(a)$). If the action is performed before the deadline ($event_{end}$) then the reward is carried out. Otherwise the OMS will perform the sanction.

The determination of the allowed actions is made by means of the analysis of the normative context. This checking is performed by the OMS each time it receives a new service request. Then the OMS checks whether it exists a norm addressed to the client agent that forbids such service request. Thus, an action is considered as allowed when there is not any norm that forbids it explicitly. Norms can be addressed to any agent that plays a specific role or they can affect

a specific agent also. Consequently, a criterion for norm precedence is needed. In this case, a rule known as *lex specialis* has been employed [17]. Therefore, the normative analysis begins with checking agent addressed norms. If there is not any norm, norms related to the roles played by the agent are considered.

This section has illustrated some details of the implementation of the organizational management system entity. Following, conclusions and a discussion on related works are presented.

5 Discussion

Organizations represent an effective mechanism for activity coordination, not only for humans but also for agents. They have recently been used in agent theory to model coordination in open systems and ensure social order in MAS applications [18].

Virtual Organizations include the integration of organizational and individual perspectives, and the dynamic adaptation of models to organizational and environmental changes [1]. The necessity of managing the organizational life-cycle has been taken into account in several approaches such as the S-Moise+ middleware [19] (based on Moise+ [20] organizational model), the Brain system [21] and the ORA4MAS proposal [22]. In our architecture the OMS entity acts in a similar way, receiving the organizational service requests and providing them, accordingly to the established organizational norms. However, all this functionality has been designed following a Service Oriented approach. Thus, services for controlling organizations life-cycle have been described employing Web Service standards as OWL-S. In addition, these services are registered and published by the architecture in order to allow agents to discover them and to know how to make use of them. Our architecture is aimed at supporting the development of open virtual organizations. Thus, a normative mechanism for controlling how agents make use of the provided services is supplied.

Regarding works on norms, they have traditionally a theoretical point of view. Recently, norms have received a more practical conception in order to allow the regulation of open distributed systems. More specifically, traditional approaches based on deontic logics have evolved to a more operational conception of norms that allows their employment inside the design and execution of real MAS applications as a coordination mechanism. These approaches must allow reasoning about the global system performance as well as the agent individual reasoning. The *normative learning* term has been defined as the process by which agents take into consideration norms inside their decision making [23]. Works concerning the normative reasoning are related to the norm emergence [24, 25] and the norm acceptance [26, 27]. Taking these later works as a starting point, the OMS component described in this paper is aware of the existence of norms that regulate access to its services. Moreover, it is able to internalize norms as beliefs, in order to take its decisions according to its expected behaviour.

On the other hand, proposals for norm implementation are based on the e-institution metaphor [9, 10]. They assume the existence of an "special" entity,

which is the institution itself, that acts as a middleware for agent communications. Therefore, the institutional entity has an unlimited knowledge about occurrence of facts as well as extra capabilities for enforcing norms. Because of this, these works do not provide an effective mechanism for normative reasoning. In fact, they provide an implementation of norm theory for allowing the institution to maintain the coherency of the normative state.

In order to allow the definition of MAS as Open Systems, in its broadest sense, a new normative implementation that gives support to agent normative reasoning is needed. The normative reasoning term is defined as the process by which an agent decides to accept a norm and incorporates it to its decision making behaviour [23]. With this aim, we have proposed a general normative language for expressing norms and a normative implementation that allows agents to take into consideration the existence of norms.

This proposal of normative language is a more general and expressive formalism mainly focused on controlling service registering and usage, making it possible a better integration of both MAS and Web Service Technologies. The main and new aspects of the proposed language are: (i) it allows the definition of norms that cover organizational dynamics; and (ii) norms define agent functionality in terms of services requested and provided by each role. Both organizational performance and functionality of the system are established by means of norms defined in terms of service requesting, provision or registering.

6 Conclusions

This work belongs to a higher project whose goal is to develop models, frameworks, methods and algorithms for constructing large-scale open distributed computer systems. Our aim is to employ this architecture for building demonstrators on e-procurement, e-healthcare and water conflict resolution. Thus, the Organizational Management System implementation has been included in a prototype of the THOMAS architecture. This component is mainly responsible for the management of organizations and their entities. The OMS allows the creation and management of both norms and organizations. Therefore, our normative implementation has been employed for controlling access to the organizational services. In this sense, norms can be conceived as a method for regulating the dynamical organizational adaptation to the environmental changes.

References

1. V. Dignum, F. Dignum. A landscape of agent systems for the real world. Tech. report 44-cs-2006-061, Inst. Information and Computing Sciences, Utrecht University, 2006.
2. O. Boissier, B. Gâteau. Normative multi-agent organizations: Modeling, support and control. In *Normative Multi-agent Systems*, Dagstuhl Seminar, 2007.
3. M. Luck, P. McBurney, O. Shehory, S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*. AgentLink, 2005.
4. M. Luck, P. McBurney. Computing as interaction: Agent and agreement technologies. In *IEEE SMC Conference on Distributed Human-Machine Systems*: 1–6, 2008.

5. D. Greenwood, M. Calisti. Engineering web service - agent integration. In *IEEE Int. Conf. on Systems, Man and Cybernetics*, 2: 1918– 1925, 2004.
6. C. Carrascosa, A. Giret, V. Julian, M. Rebollo, E. Argente, V. Botti. Service Oriented MAS: An open architecture. In *AAMAS*, In Press, 2009.
7. E. Argente, J. Palanca, G. Aranda, V. Julian, V. Botti, A. García, A. Espinosa. Supporting agent organizations. In *CEEMAS'07*, 4696: 236–245, 2007.
8. G. Boella, L. van der Torre, H. Verhagen. Introduction to the special issue on normative multiagent systems. *Auton. Agents Multi-Agent Syst.*, 17:1–10, 2008.
9. A. García-Camino, P. Noriega, J.A. Rodríguez-Aguilar. Implementing norms in electronic institutions. In *EUMAS*: 482–483, 2005.
10. V. Torres. Implementing norms that govern non-dialogical actions. In *COIN III*, LNCS:4870. Springer, 2008.
11. E. Argente, N. Criado, V. Julian, V. Botti. Designing Norms in Virtual Organizations. In *CCIA*, 184:pages 16–23. IOS Press, 2008.
12. N. Criado, V. Julian, E. Argente. Towards the Implementation of a Normative Reasoning Process. In *PAAMS*, In Press, 2009.
13. F. López, M. Luck. Modelling norms for autonomous agents. In *ENC*: 238–245. IEEE Computer Society, 2003.
14. R. I. Brafman, M. Tennenholtz. On Partially Controlled Multi-Agent Systems. Journal of Artificial Intelligence Research, 4:477–507, 1996.
15. M. J. Kollingbaum, W. W. Vasconcelos, A. García-Camino, T. J. Norman. Conflict resolution in norm-regulated environments via unification and constraints. In *DALT*, *LNCS* 4897:158–174. Springer, 2007.
16. W. Vasconcelos, M. J. Kollingbaum, T. J. Norman. Resolving conflict and inconsistency in norm-regulated virtual organizations. In *AAMAS*: 632–639, 2007.
17. G. Boella, L. van der Torre. Permissions and obligations in hierarchical normative systems. In *ICAIL*, 2003.
18. V. Dignum, J. Meyer, H. Weigand, F. Dignum. An organization-oriented model for agent societies. In *RASTA*: 31–50, 2002.
19. J. Hubner, J. Sichman, O. Boissier. S-moise+: A middleware for developing organised multi-agent systems. In *EUMAS*: 64–78, 2006.
20. J. Hubner, J. Sichman, O. Boissier. MOISE+: towards a structural, functional, and deontic model for MAS organization. In *AAMAS*: 501–502, 2002.
21. G. Cabri, L. Leonardi, F. Zambonelli. BRAIN: A Framework for Flexible Role-Based Interactions in Multiagent Systems. In *CoopIS/DOA/ODBASE*:145-161, 2003
22. R. Kitio, O. Boissier, J.F. Hbner, A. Ricci. Organisational Artifacts and Agents for Open Multi-Agent Organisations: "Giving the Power Back to the Agents". In *COIN*: 171-186, 2007
23. H.J.E. Verhagen. *Norm Autonomous Agents*. PhD thesis, The Royal Institute of Technology and Stockholm University, 2000.
24. A. Walker, M. Wooldridge. Understanding the emergence of conventions in multi-agent systems. In *ICMAS*: 384–390, June 1995.
25. B.T.R. Savarimuthu, S. Cranefield, M. Purvis, M. Purvis. Role model based mechanism for norm emergence in artificial agent societies. In *COIN*: 1–12, 2007.
26. C. Castelfranchi, F. Dignum, C. M. Jonker, J. Treur. Deliberative normative agents: Principles and architecture. In *ATAL*, 1757: 364–378, 1999.
27. F. Dignum, D. Morley, L. Sonenberg, L. Cavedon. Towards socially sophisticated BDI agents. In *ICMAS*: 111–118. IEEE Computer Society, 2000.

Building Multi-Agent Systems for Workflow Enactment and Exception Handling*

Joey Lam, Frank Guerin, Wamberto Vasconcelos, and Timothy J. Norman
{j.lam, f.guerin, w.w.vasconcelos, t.j.norman}@abdn.ac.uk

Department of Computing Science
University of Aberdeen, Aberdeen, U.K. AB24 3UE

Abstract. Workflows represent the coordination requirements of various distributed operations in an organisation; workflows neatly capture business processes, and are particularly suitable for cross-organisational enterprises. Typical workflow management systems are centralised and rigid; they cannot cope with the unexpected flexibly. Multi-agent systems offer the possibility of enacting workflows in a distributed manner, via software agents which are intelligent and autonomous, and respect the constraints in a norm-governed organisation. Agents should bring flexibility and robustness to the workflow enactment process. In this paper, we describe a method for building a norm-governed multi-agent system which can enact a set of workflows and cope with exceptions. We do this by providing agents with knowledge of the organisation, the domain, and the tasks and capabilities of agents. This knowledge is represented with Semantic Web languages, and agents can reason with it to handle exceptions autonomously.

1 Introduction

The Workflow Management Coalition defines a workflow as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" [21]. Workflows can be formalised and expressed in a machine readable format, and this makes it possible for them to be employed in service-oriented computing scenarios. In such scenarios we may be dealing with open heterogeneous computing systems, where errors and exceptions are likely to occur. We would like the computing systems to cope with these exceptions. Ideally we would like to be able to deal with the unexpected; while we could write specific exception handling routines to deal with some common exceptions which we expect to arise, it will be difficult to anticipate all possible exceptions. Hence we need intelligence to deal with the unexpected. Typical workflow management systems (e.g., [12, 13]) are centralised and rigid; they cannot cope with the unexpected flexibly. Moreover, they have not been designed for dynamic environments requiring adaptive responses. To overcome this we argue that it is necessary to use agents to control the enactment of a workflow in a distributed manner; agents can be endowed with sufficient intelligence to allow them to

* This work is funded by the European Community (FP7 project ALIVE IST-215890).

manage exceptions autonomously. This should bring flexibility and robustness to the process of enacting workflows.

We are interested in building multi-agent systems (MASs) which simulate the operations in large organisations, and so must adhere to constraints defined by the organisation[1]. In this paper we explore the issue of building agent systems which can enact workflows in a distributed fashion, and which can cope with exceptions as they arise. We propose a method for building such agent systems, given the appropriate knowledge as an input. The knowledge input to the system is divided into three main components, as illustrated in Figure 1. Firstly there

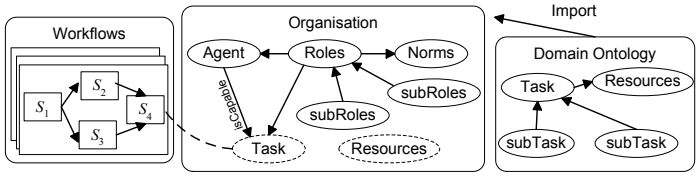


Fig. 1: Overview of the Proposed Approach

is the *organisational knowledge*, consisting of such things as roles, norms, role classification, and resources available. Secondly there are the *workflows*, describing the tasks to be executed, and appropriate flow of control, and also including variable definitions which are used to control flow. Thirdly there is the *domain ontology*, describing concepts of the world including tasks and resources. Some tasks are atomic and can be directly executed by agents, but some tasks require a workflow to be executed. In this case the name of the task matches the name of a workflow. As indicated by the dashed line there are links between the workflow tasks and the organisation and imported domain ontology. Tasks and resources are not described in the organisation directly; the domain ontology is imported to the organisation ontology. This knowledge allows us to firstly allocate tasks in an agent system where workflow tasks are distributed among the agents in a way which is consistent with the organisational constraints. Secondly, it allows the agents to enact the workflows, updating the organisation as appropriate. Thirdly it allows agents to cope with exceptions as they arise, because the agents can query the ontology to find alternative agents or tasks when problems arise.

In our system the workflow specifications and the ontologies are available to be queried by any of the agents in the system, and the ontologies can also be updated as events add or modify instances in the ontology. This requires a centralised service which maintains the knowledge, and updates it, when instructed to by an agent. This centralised approach is somewhat undesirable in a distributed agent system (lack of robustness, and scalability), however additional robustness could be introduced by having multiple copies of the knowledge, and some synchronisation processes to maintain consistency. Both of these possibilities entail challenges which go beyond the scope of the current paper; we will merely assume the knowledge is available.

[1] Our focus for the moment is simulation, but eventually we envisage that our agents will support real human users in executing tasks as part of human-agent teams.

This paper is structured as follows. In Section 2 we briefly introduce Semantic Web languages. In Section 3 we describe norm-governed organisations represented in Semantic Web languages. We describe the representation of workflows and explain allocation of tasks in Sections 4 and 5 respectively. The details of agents enacting workflows and dealing with exceptions are given in Sections 6 and 7 respectively. Section 8 looks at related work and Section 9 concludes and proposes future work.

2 Semantic Web Languages

The OWL-DL [6] ontology language is a variant of the $\mathcal{SHOIN}(\mathbf{D})$ Description Logic [7], which provides constructs for full negation, disjunction, a restricted form of existential quantification, cardinality restrictions, and reasoning with concrete datatypes. We make use of the open world assumption, which requires that something is false if and only if it can be proved to contradict other information in the ontology. Since we assume a MAS as an open system, its knowledge of the world is incomplete, and the knowledge is extendable. If a formula cannot be proved true or false, we draw no conclusion[2].

Formally, an ontology $\mathcal{O}$ consists of a set of *terminology* axioms $\mathcal{T}$ (TBox), *role* axioms (RBox) and assertional axioms $\mathcal{A}$ (ABox), that is, $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$. An axiom in $\mathcal{T}$ is either of the form $C \sqsubseteq D$ or $C \doteq D$, where C and D are arbitrary concepts (aka. classes in OWL); the RBox contains assertions about roles (such as functional, transitive roles) and role hierarchies; an axiom in $\mathcal{A}$ is either of the form $C(a)$ (where C is a concept and a is an individual name; a belongs to C), or of the form $R(a, b)$ (where a, b are individual names (aka. instances in OWL) and R is a role name (aka. a datatype or object property in OWL); b is a filler of the property R for a). The meaning of concepts, roles and individuals is given by an interpretation. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set of individuals (the *domain* of the interpretation) and an *interpretation function* $(\cdot^{\mathcal{I}})$, which maps each atomic concept $\mathsf{CN} \in \mathbf{C}$ ($\mathbf{C}$ is a set of concept names) to a set $\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each atomic role $R \in \mathbf{R}$ ($\mathbf{R}$ is a set of role names) to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function can be extended to give semantics to concept descriptions. An interpretation $\mathcal{I}$ is said to be a model of a concept C, or $\mathcal{I}$ models C, if the interpretation of C in $\mathcal{I}$ is not empty. A concept A is *unsatisfiable* w.r.t. a terminology $\mathcal{T}$ if, and only if, $A^{\mathcal{I}} = \emptyset$ for all models of $\mathcal{I}$ of $\mathcal{T}$. An ontology $\mathcal{O}$ is *inconsistent* if it has no models.

OWL DL benefits from many years of DL research, leading to well defined semantics, well-studied reasoning algorithms, highly optimised systems, and well understood formal properties (such as complexity and decidability) [3].

The Semantic Web Rule Language (SWRL)[3] extends the set of OWL axioms to include Horn-Clause-like rules that can be expressed in terms of OWL classes and that can reason about OWL instances. SWRL provides deductive reasoning capabilities that can infer new knowledge from an existing OWL knowledge base. However, OWL DL extended with SWRL is no longer decidable. To make the

[2] We can reason in an inconsistent ontology by tolerating a limited number of contradictions. A formula is *undefined* (or *undetermined*) if it entails neither true nor false; a formula is *overdefined* (or *over-determined*) if it entails both true and false.

[3] http://www.w3.org/Submission/SWRL/

extension decidable, Motik et al. [15] propose DL-safe rules where the applicability of a rule is restricted to individuals explicitly named in a knowledge base. For example: $\mathsf{parent(x,y)} \wedge \mathsf{brother(y,z)} \wedge \mathcal{O}(\mathsf{x}) \wedge \mathcal{O}(\mathsf{y}) \wedge \mathcal{O}(\mathsf{z}) \rightarrow \mathsf{uncle(x,z)}$ where $\mathcal{O}(x)$ must hold for each explicitly named individual x in the ontology. Hence, DL-safe rules are SWRL rules that are restricted to known individuals.

The language SPARQL-DL [18] supports mixed TBox/RBox/ABox queries for OWL-DL ontologies. Throughout the paper we will use qnames to shorten URIs with rdf, rdfs, and owl prefixes to refer to standard RDF (Resource Description Framework), RDF-S and OWL namespaces, respectively. We also use the prefix dom and org to refer to the namespace of the Domain and Organisation ontologies. Our agents will be able to query the ontology to access the knowledge. For example, agents can search for all roles working in the finance department which are obliged to perform task "WriteReport" by using this query[4]:

Type(_:x, ?role), PropertyValue(_:x, org:worksIn, _:y), Type(_:y, org:FinanceDept), PropertyValue(_:x, org:isObliged, _:z), Type(_:z, dom:WriteReport)

3 Norm-Governed Organisations

In this section, we describe how to represent roles, role classification, and norms using OWL and SWRL. Our specification in this section is adequate to allow agents to query an organisation at a certain point in time and ask questions such as "is agent x prohibited from doing task t", or "is agent y empowered to do task b". However, we do not provide a specification for how the organisation is changed as a result of agents performing speech acts, or as a result of other events. We assume that the system provides other specifications for this purpose, and we focus only on the specifications necessary for workflow enactments.

We represent the agent organisation using Semantic Web languages. The agent literature has many examples of different approaches to the specification of norm-governed organisations, using languages such as the event calculus or C+, for example [2]. Semantic Web languages are not as expressive as these, but they have the advantage of having very efficient DL reasoners, and being standardised. To specify the organisational knowledge relevant to our workflows we can get by without the expressiveness of more sophisticated languages; for example we only need to do static queries on current knowledge, and we do not require the ability to reason over different time intervals.

3.1 Roles and their Constraints

The ontology we propose in this paper models the concepts of a role, its role classification(s), restrictions on roles (such as mutually exclusive roles, cardinality, prerequisite roles) [16], and other aspects of the organisation. Roles are modelled in a classification to reflect the subsumption of role descriptions. Sub-roles inherit the properties from the super-roles; the properties of a sub-role override those of its super-roles if the sub-role has more restrictive properties (the sub-role cannot be less restrictive or the ontology would be inconsistent). Cardinality restrictions can be used for example to restrict the number of agents a task can be assigned to. Disjointness axioms can represent separation of duty restrictions.

[4] Type(?a,?C) gives the most specific classes an instance belongs to.

We now give an example specification to illustrate these ideas. In Figure 2, a role classification is shown. Sub-roles inherit the properties from super-roles. For example, Staff are obliged to work from 9am to 5pm during weekdays; its sub-roles inherit this obligation. The properties of a sub-role override those of its super-role. Manager is permitted to employ staff; its sub-roles inherits this property. However, this property of the AccountingManager is more restrictive; it is only permitted to employ AccountingStaff (see axioms 4 and 5 below). For the cardinality restrictions, we can model that only one agent can fill the role of the general manager (see axioms 11 and 12 below); a member of staff works in exactly one department (see axiom 6 below). An example of mutually exclusive roles is that a department manager cannot be a general manager simultaneously (see axiom 8 below). An example of separation of duty is that a staff submitting a project proposal is prohibited from approving the proposal (see axiom 9 below). Prerequisite roles means that a person can be assigned to role $r1$ only if the person already is assigned to role $r2$ (see axiom 10).

(1) Programmer $\sqcup$ Manager $\sqcup$ Secretary $\sqsubseteq$ Staff

(2) DeptManager $\sqcup$ GeneralManager $\sqsubseteq$ Manager

(3) AccountingManager $\sqcup$ ITManager $\sqsubseteq$ DeptManager

(4) Manager $\sqsubseteq \exists$ isPermitted.($\exists$ employs.Staff) $\sqcap \forall$ isPermitted.($\exists$ employs.Staff)

(5) AccountingManager $\sqsubseteq \exists$ isPermitted.($\exists$ employs.AccountingStaff) $\sqcap \forall$ isPermitted.($\exists$ employs.AccountingStaff)

(6) Staff $\sqsubseteq =1$ worksIn

(7) range(worksIn) $=$ Department

(8) DeptManager $\sqsubseteq \neg$ GeneralManager

(9) Staff(x) $\wedge$ ProjectProposal(p) $\wedge$ ApproveProjectProposal(act) $\wedge$ submits(x,p) $\wedge$ approves(act,p) $\wedge\mathcal{O}$(x) $\wedge\mathcal{O}$(p) $\wedge\mathcal{O}$(act) $\rightarrow$ isProhibited(x,act)

(10) AccountingManager $\sqsubseteq \exists$ prerequisites.Accountant

(11) GeneralManager $\sqsubseteq =1$ takenBy

(12) range(takenBy) $=$ Agent

(13) canDelegate $\sqsubseteq$ hasPower

(14) Staff $\sqsubseteq \exists$ isObliged.($\exists$ works.(Weekdays $\sqcap$ OfficeHour)) $\sqcap \forall$ isObliged.($\exists$ works.(Weekdays $\sqcap$ OfficeHour))

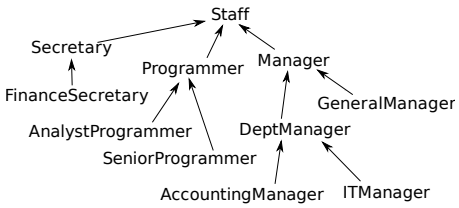


Fig. 2: Roles and a Role Classification

3.2 Normative Notions

We firstly describe norms concerning agents performing some task Task. We model permission[5], obligation, prohibition and power as isPermitted, isObliged,

[5] Explicitly defined permission means *strong* permission in our system. Undefined permission axioms represent *weak permission*.

isProhibited and hasPower OWL object properties to relate roles in the organisation and tasks. Their domain and range is Role and Task respectively. Permissions allow the agent to achieve a state of affairs or perform an action (see for example axioms 4 and 5 above). Permission and prohibition are distinct from power because a member may be empowered to do something even though he is prohibited from doing it. Prohibitions forbid the agent from achieving a state of affairs or performing an action (see for example axiom 9 above). An obligation indicates that some act has to be done (see for example axiom 14 above). It is common to specify a time-limit or a condition for obligations. The axiom 14 above states a conditional obligation, such that staff have to work during office hours and weekday. Due to limited space, we will not describe time-limit constraints in the paper. We now model the relations between the basic notions; the relations can be equivalence, compatibility or incompatibility (or conflict) [20]. The following rules list some of these relations.

(1) If an act is permitted and prohibited then there is a conflict.
isPermitted(x,act) ∧ isProhibited(x,act) ∧$\mathcal{O}$(x) ∧$\mathcal{O}$(act) → owl:Nothing(x)

(2) If an act is obligatory, then it is permitted.
isObliged(x,act) ∧$\mathcal{O}$(x) ∧$\mathcal{O}$(act) → isPermitted(x,act)

(3) If an act is obligatory and prohibited then there is a conflict.
isObliged(x,act) ∧ isProhibited(x,act) ∧$\mathcal{O}$(x) ∧$\mathcal{O}$(act) → owl:Nothing(x)

(4) If a prohibited act is performed then there is a violation.
performed(x,act) ∧ isProhibited(x,act) ∧$\mathcal{O}$(x) ∧$\mathcal{O}$(act) → violated(x,act)

Compared to standard deontic logic, here these deontic notions are being given quite a different interpretation by the Semantic Web languages. For example in deontic logic we could say that "obliged" is equivalent to "not permitted not to", however in Semantic Web languages we cannot express this. SWRL does not allow us to negate the atoms within the scope of isPermitted(...). Nevertheless, the Semantic Web version of these norms seems to be adequate for representing simple agent scenarios, as illustrated in our examples below. One thorny issue is contraposition; a DL axiom such as $A \sqsubseteq B$ entails $\neg B \sqsubseteq \neg A$. This entailment is not desirable in exceptional situations where there is a special condition such that some type of A is not a B. The only way we can deal with this is to explicitly state all exceptions in the axioms, for example Bird ⊓ ¬ Penguin ⊓ ¬ Ostrich ⊑ CanFly.

In this paper we distinguish between institutional tasks (such as authorising a purchase), and physical tasks (such as printing a document). An institutional task can only be performed by an agent which has power to do that task. For example we say that action 'manager x employs staff y' is valid if x is empowered to employ a staff at that time, therefore y is now a member of staff. Otherwise, it is an invalid action due to its lacking of institutional power. The following axioms mean that a manager has the power to employ staff; when the manager performs EmployStaff, the person will become a staff:

EmployStaff ≐ ∃ employ.Staff ⊓ ∀ employ.Staff

Manager ⊑ ∃ hasPower.EmployStaff

Manager(m) ∧ Person(p) ∧ EmployStaff(act) ∧ hasPower(m,act) ∧ employ(act,p) ∧ performed(m,act) ∧$\mathcal{O}$(m) ∧$\mathcal{O}$(p) ∧$\mathcal{O}$(act) → Staff(p)

4 Workflows

We now introduce a representation for workflows. A common way to represent a workflow is using Petri Nets [19] or BPEL (Business Process Execution Language) [1]. In this paper we represent a workflow as a digraph, which is a simplified and minimalistic way to capture the basic concepts of workflows.

Definition 1. *A workflow is a tuple* $\langle N, \mathbf{S}, \mathbf{E}, s_0, \mathbf{S}_f \rangle$*, where*

1. N *is the name of the workflow,*
2. $\mathbf{S}$ *is a finite set of states of the form* $\langle id, T \rangle$*, where* id *is the number identifying this state, and* T *is the task;*
3. $\mathbf{E}$ *is a set of edges linking states. Edges take the form* $\langle id_1, l, v, id_2 \rangle$*, where* id_1 *is the state this edge leaves from,* id_2 *is the state this edge arrives at,* v *is the variable associated with the edge, and* l *is a label indicating the type of edge. There are four types of edge,* $l \in \{AND, OR, JOIN\text{-}AND, JOIN\text{-}OR\}$ *where AND-edges and OR-edges describe exclusive-or branches, and JOIN-AND-edges and JOIN-OR-edges describe joins. For any pair of states with multiple edges linking them, the edges must be of the same type;*
4. s_0 *is the initial state of the workflow, and* $\mathbf{S}_f \subseteq \mathbf{S}$ *is the set of final states.*

We consider a travel request workflow example in a company. Figure 3 graphically depicts the example. Figure 4 shows the representation of the workflow. The figure annotates edges with the name of the variable associated with the edge. We refer to *input* and *output* variables of the workflow; for example the state ⟨2,checkRequestForm⟩ has input variable "TravelRequestForm" and output variable "CorrectTravelRequestForm". In this example, we assume the workflow is triggered by an agent who issues a travel request. Firstly a travel request form is issued; the request form should be checked to have correct information. The checked form is then passed to be approved. The output of "approveTravelRequest" is either "ApprovedForm" or "RejectedForm". If the output variable is "ApprovedForm", the order for the travelling can be placed; otherwise, the request is rejected.

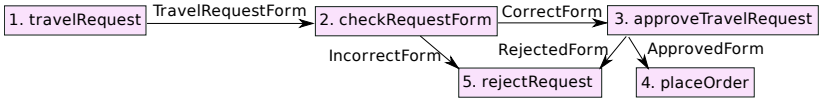


Fig. 3: Travel Request Workflow Example

$W = \langle$travelRequest_WF$, \mathbf{S}, \mathbf{E}, 1, \{4, 5\}\rangle$
$\mathbf{S} = \{\langle$1,travelRequest$\rangle$, $\langle$2,checkRequestForm$\rangle$, $\langle$3,approveTravelRequest$\rangle$, $\langle$4,placeOrder$\rangle$, $\langle$5,rejectRequest$\rangle\}$
$\mathbf{E} = \{\langle$1,AND,TravelRequestForm ,2$\rangle$, $\langle$2,OR,CorrectTravelRequesForm,3$\rangle$,
$\langle$2,OR,IncorrectTravelRequesForm,5$\rangle$ $\langle$3,OR,ApprovedForm,4$\rangle$, $\langle$3,OR,RejectedForm,5$\rangle\}$

Fig. 4: Travel Request Workflow as a Digraph

5 Allocating Tasks to Agents

In this section we describe how we allocate tasks to software agents which, together, will enact a set of workflows. Agents are parameterised by the roles they take up – these roles dictate the tasks agents become responsible for.

The input to the ontological creation of agents is a set of workflows $\mathcal{W}$ and an ontology $\mathcal{O}$, and the output is an updated ontology with a set of software agents introduced as subclasses of the Agent concept, with roles and tasks associated with them. In Figure 5 we show how we create agents in our ontology. The

algorithm *agent_creation*($\mathcal{W} = \{W_1, \ldots, W_n\}$, $\mathcal{O}$)
  $\mathcal{T} = \bigcup_{i=1}^{n} \mathbf{S}_i$, $\langle N_i, \mathbf{S}_i, \mathbf{E}_i, s_{0_i}, \mathbf{S}_{f_i} \rangle \in \mathcal{W}$
  for each role R_i in $\mathcal{O}$ **do**
    for each $\langle id, T \rangle \in \mathcal{T}$ **do**
      if $R_i \sqsubseteq \exists$ isObliged.T **then**
        $\mathcal{T} := \mathcal{T} \setminus \{\langle id, T \rangle\}$; $\mathcal{T}_i := \mathcal{T}_i \cup \{T\}$
      else if $R_i \sqsubseteq \exists$ hasPower.$T \sqcap \exists$ isPermitted.T **then**
        $\mathcal{T} := \mathcal{T} \setminus \{\langle id, T \rangle\}$; $\mathcal{T}_i := \mathcal{T}_i \cup \{T\}$
      else if $R_i \sqsubseteq \exists$ isPermitted.T **then**
        $\mathcal{T} := \mathcal{T} \setminus \{\langle id, T \rangle\}$; $\mathcal{T}_i := \mathcal{T}_i \cup \{T\}$
  if $\mathcal{T} \neq \emptyset$ **then fail** // org. cannot enact a workflow
  else
    for each role R_i in $\mathcal{O}$ with $\mathcal{T}_i = \{T_0^i, \ldots, T_m^i\}$ **do**
      $\mathcal{O} := \mathcal{O} \cup \{Ag_i \sqsubseteq$ Agent$\}$
      $\mathcal{O} := \mathcal{O} \cup \{Ag_i \doteq \exists$ hasRole.$(R_i) \sqcap \exists$ isCapable.$(T_0^i \sqcup \cdots \sqcup T_m^i)\}$
return $\mathcal{O}$;

Fig. 5: Creation of Agents in $\mathcal{O}$

algorithm collects in $\mathcal{T}$ all tasks of the workflows and distributes them among the roles of the organisation. The distribution gives priority to *i)* obligations, then *ii)* institutionalised power and permissions, and finally *iii)* permissions, captured in the algorithm by the order of the nested **if** constructs. All tasks should be distributed among roles, otherwise the algorithm fails, that is, the organisation represented in the ontology cannot enact one of the workflows. If all tasks have been assigned to roles, then for each role R_i we create in $\mathcal{O}$ a subclass Ag_i of Agent, with tasks $\mathcal{T}_i = \{T_0^i, \ldots, T_m^i\}$ associated to the agent via isCapable.

For each Ag_i in $\mathcal{O}$ we start up an independent computational process – a software agent – which will support the enactment of workflows. Each software agent, upon its bootstrapping, will use the definition of the subclass as the parameterisation of its mechanisms: the role and tasks associated to the agent will guide its behaviour, explained in Section 6. For simplicity, in our algorithm above, we assume that each agent will enact exactly one role; however this could easily be changed if required.

6 Enactment of Workflows

After allocation, the next step is that agents take up roles in the organisation and enact workflows. Agents plan their actions in real-time. The workflow provides an outline plan, but many of the details need to be decided by agents. During the enactment, an ontology is used by the agents to check what actions they can perform. There is a relationship between the tasks and variables in the workflows, and the concepts and instances in the ontology. Each time the agents perform workflow tasks or assign values to variables, they update the instances in

the ontology. Some agent actions will involve the consumption of organisational resources, in which case the agent will update the instances in the ontology. Thus the ontology maintains a record of the current status of the workflow enactment, as well as relevant aspects of the organisation. In this paper we avoid details of how the implementation could work, but the update of the ontology can be done by the agents whenever they are about to do a task; the agent sends an update instruction to the service which maintains the ontology.

We will detail the relationship between the ontology and the workflow enactment; this is easiest to illustrate by referring to an example. We continue with the travel request example from Section 4, and we add to it an ontology (see the axioms below) which describes roles, norms, and descriptions of tasks. The following axioms state the norms governing agents and the tasks to be executed.

(1) Secretary $\sqsubseteq \exists$ isObliged.checkRequestForm
(2) Manager $\sqsubseteq \exists$ hasPower.approveRequest
(3) DeptManager $\sqsubseteq \exists$ hasPower.approveTravelRequest
(4) Manager(m) $\wedge$ TravelRequestForm(f) $\wedge$ approveTravelRequest(act) $\wedge$ requestedFrom(f,m) $\wedge$ approves(act,f) $\wedge\mathcal{O}$(m) $\wedge\mathcal{O}$(m) $\wedge\mathcal{O}$(f) $\rightarrow$ isProhibited(m,act)
(5) checkRequestForm $\doteq \exists$ checks.(TravelRequestForm $\sqcap \exists$ isCorrect.xsd:boolean)
(6) Staff(s) $\wedge$ requestedFrom(f,s) $\wedge$ hasName(s,n) $\wedge$ filledName(f,n) $\wedge$ hasStaffID(s,id) $\wedge$ filledStaffID(f,id) $\wedge\mathcal{O}$(s) $\wedge\mathcal{O}$(f) $\wedge\mathcal{O}$(n) $\wedge\mathcal{O}$(id) $\wedge \cdots \rightarrow$ isCorrect(f, {"true"^^⟨ xsd:boolean⟩})
(7) CorrectTravelRequestForm $\doteq$ TravelRequestFrom $\sqcap \exists$ isCorrect.{"true"^^⟨ xsd:boolean⟩}
(8) InCorrectTravelRequestForm $\doteq$ TravelRequestFrom $\sqcap \exists$ isCorrect.{"false"^^⟨ xsd:boolean⟩}
(9) checkRequestForm $\sqsubseteq \exists$ hasInput.TravelRequestForm $\sqcap$ $\exists$hasOutput.(CorrectTravelRequestForm $\sqcup$ InCorrectTravelRequestForm)
(10) functional(isCorrect)
(11) ApprovedForm $\doteq$ TravelRequestForm $\sqcap \exists$ isApproved.{"true"^^⟨ xsd:boolean⟩}

Each state in a workflow is mapped to a task in the domain ontology by matching the same name (i.e. each task is a concept in the ontology). For example, state 2 in the workflow (Figure 4) is mapped to checkRequestForm in the domain ontology. When a task in a workflow is about to be executed by an agent, an instance of the corresponding task in the ontology is created by the agent (in real-time).

Similarly, each input (or output) variable from a workflow task maps to a concept in the ontology. For example, the variable "ApprovedForm" in the workflow (Figure 4) is mapped to the concept ApprovedForm (see axiom 11 above) in the domain ontology. Every time an agent assigns a value to an input (or output) from a workflow task, then a new instance is also created in the ontology, corresponding to the value of the variable.

The creation of instances in the ontology allows an agent to check if its next action complies with the constraints of the organisation. The agent who is about to execute a workflow task first tentatively creates an instance in the ontology, and then calls the DL reasoner to check the ontology's consistency and also to check if violations have been introduced. If the ontology is inconsistent, then the agent knows that the task it was about to execute is an error; on the other hand, if the ontology entails violated(x,task) for some agent "x", then the agent knows that the task it was about to execute would cause it to violate norms. Thus the agent should not carry out the task, and should revert to the ontology before the instance was added. This type of check can pick up on such things as an agent performing a prohibited action (see axiom 4 in subsection 3.2) or axiom (4) above which forbids a manager from approving his own travel request. Of course an autonomous agent may choose to execute the task regardless, in which case

the instance is added, and the ontology may now have recorded violations (in the case of broken prohibitions) or may be inconsistent (in the case of an agent updating wrong information). In the second case, the inconsistent ontology can still be used thereafter for agents to check proposed actions. Ontology reasoners can reason with inconsistent ontologies by selecting consistent subsets [8]. In our case, the reasoner can identify the set of "Minimal Unsatisfied Preserving Sub-Ontologies" (MUPSs) in an inconsistent ontology [17]; each MUPS is a minimal set of problematic axioms. Thus, given an inconsistent ontology, an agent can add an instance and check if the number of MUPSs has increased; if so, then this instance has caused further inconsistencies, otherwise the instance (and hence the proposed workflow task) is acceptable.

We now describe how the agent's behaviour is related to the ontology and the workflow using the "travel request" workflow in Figure 4. Let us look at the second state of the workflow in Figure 4, i.e., the "checkRequestForm" task. When control passes to this state there already exists an instance of a TravelRequestForm in the ontology, say this is TF124. Now the above axioms (1) states that the Secretary is obliged to perform the "checkRequestForm" workflow task. The secretary is going to perform this action, so the secretary agent creates an instance of checkRequestForm, say this is CRF54. Axiom (5) states that checkRequestForm is defined as having at least one checks relationship, and therefore the secretary agent must also add an ABox axiom for the relationship checks(CRF54, TF124); the agent can also deduce that the form TF124 should be correct or incorrect (i.e. boolean). Now due to axiom (6), assuming the form has been filled correctly, then its isCorrect property should have a true value; this corresponds to ABox axiom isCorrect(TF124, {"true"^^⟨ xsd:boolean⟩}), which can now be inferred from the ontology. Now from axiom (7) it can be inferred that the travel request form TF124 is an instance of the concept CorrectTravelRequestForm. Finally axiom (9) tells us that this instance TF124 is the value to be assigned to the output variable "CorrectTravelRequesForm" of this workflow task. However, the secretary might erroneously decide to assign the output TF124 as the output value "CorrectTravelRequesForm" or "IncorrectTravelRequesForm" in the workflow. As mentioned above, whenever an agent assigns a value to a workflow input (or output) variable, the agent must add an instance of the corresponding concept to the ontology. Thus the secretary's choice is between adding the ABox axiom CorrectTravelRequestForm(TF124) or InCorrectTravelRequestForm(TF124). Of course if the latter choice is made, then the ontology becomes inconsistent, because the form passed all the tests of axiom (6), hence CorrectTravelRequestForm(TF124) can already be inferred. Thus the secretary knows that declaring the form incorrect breaks the organisation's constraints. Likewise, if the form was incorrectly filled, the secretary would break the organisation's constraints by declaring it to be correct. Axiom (10) states that the isCorrect property can only have one value (i.e. the form's correctness cannot be both true and false).

7 Dealing with Exceptions

During the enactment of workflows, exceptions may occur easily, for example due to unavailable resources, or agent failures. One way to deal with exceptions is to classify exceptions into classes and pre-define rules or policies to handle each case; specialised agents then perform defined remedial actions [9]. How-

ever, exceptions are difficult to predict during design, especially in open and dynamic environments. It is preferable to program agents with intelligence and adaptivity so they can accommodate unexpected changes in their environment. To satisfy this requirement we have associated the workflow tasks with semantic information in the OWL ontology, and we have also represented the background knowledge for the organisation. This allows agents to reason about the description of tasks and agents in the organisation and find alternative ways to deal with workflow tasks when exceptions arise.

Exceptions often involve the inability to execute some particular task in the workflow. This can be repaired by breaking off from the execution of the workflow at that point, and executing some sequence of actions which can act as a substitute for the problematic task. The appropriate sequence of actions may itself involve another workflow which is nested inside the original workflow. A simple example of this could be when a member of an organisation is absent and unable to execute a task in a workflow; then another member of the organisation may repair this problem by invoking a delegation workflow to delegate the absent member's duties to another suitable member of staff.

We provide exception handlers for the following two types of exception: (1) an agent exception, where an agent may have crashed, or is not performing for some reason; and (2) a task exception, where a task is unachievable.

The "Agent_Exception_handler" in Figure 6 handles Exceptions regarding unavailable agents. The input to this routine consists of the problematic workflow state and the agent who is handling the exception (Ag_H); this is the agent who completed the preceding workflow state, and was unable to pass control to the next agent. This routine first tries to find an alternative agent which has the appropriate capability (and institutional power if necessary), and to delegate the task to that agent. If no suitable agent can be found, then a substitute task is sought; the subroutine "Find_Alternative_Task" (Figure 8) tries to find a task with the same inputs and outputs. If no suitable task can be found, then the Agent_Exception_handler tries to procure additional staff, and this is done via a nested workflow for staff procurement. The agent uses its own internal procedure "callWorkflow" to initiate a workflow to procure a new member of staff who can do task T_p; this procedure returns true if the workflow successfully procures new staff. We do not detail the procurement workflow, but it will make a selection between either hiring contract staff or recruiting new staff, depending on the organisational rules governing that class of staff.

The "Task_Exception_handler" in Figure 6 deals with unachievable tasks. Its inputs are the problematic workflow state and the agent who is handling the exception (Ag_H); in this case this is the agent who attempted to execute the problematic task, and was unable to. The routine begins by checking if the task has failed due to the unavailability of a required resource. If so, an alternative resource is sought. This is done by finding siblings of the original resource in the ontology. This could for example replace a black and white laser printer with a colour laser printer for a simple print job; the colour printer is less desirable as it is more costly, but it can do the job. If no suitable substitute resource can be found, then a substitute task is sought; the subroutine "Find_Alternative_Task" (Figure 8) tries to find a task with the same inputs and outputs. If no suitable task can be found, then the Task_Exception_handler tries to procure the resource,

and this invokes a nested workflow for resource procurement. This workflow is shown in Figure 9, it will make a selection between either hiring the equipment or purchasing it, depending on the organisational rules governing that type of equipment (we do not give the details of these rules).

algorithm Agent_Exception_handler($\langle id, T_p \rangle, Ag_H$)
  // Try to find an alternative agent who has the capability to do T_p
  if Find_Alternative_Agent($\langle id, T_p \rangle, Ag_H, \mathcal{O}$) **return** true;
  // Try to find an alternative task (or workflow) with the same input/output as T_p
  else if Find_Alternative_Task($\langle id, T_p \rangle, Ag_H, \mathcal{O}$) **return** true;
  else if callWorkflow(procureStaff_WF,T_p) **return** true; // initiate the staff procurement workflow
  else return false;

algorithm Task_Exception_handler($\langle id, T_p \rangle, Ag_H$)
  // if something is missing try alternative resources
  if there exists some resource R such that
        $T_p \sqsubseteq \exists$uses.R and ! available (R)
    then for each R_i, where sibling(R, R_i)
      let r be an instance of R
      let r_i be an instance of R_i
      let t be an instance of T_p
      $\mathcal{O}' := \{\mathsf{uses}(t, r_i)\} \cup \mathcal{O} \setminus \{\mathsf{uses}(t, r)\}$
      if consistent($\mathcal{O}'$) then $\mathcal{O} := \mathcal{O}'$; **return** true;
    end for
  // Try to find an alternative task (or workflow) with the same input/output as T_p
  if Find_Alternative_Task($\langle id, T_p \rangle, Ag_H, \mathcal{O}$) **then return** true;
  // If resources are missing
  if there exists some resource R such that $T_p \sqsubseteq \exists$ uses.R and ! available (R)
    then if callWorkflow(procureResource_WF,R) **return** true
    // initiate the resource procurement workflow
  return false;

Fig. 6: Dealing with Exceptions

algorithm Find_Alternative_Agent($\langle id, T_p \rangle, Ag_H, \mathcal{O}$)
  $Ag_H = \langle \mathcal{R}_H, \mathcal{T}_H \rangle$
  // if T_p is a type of institutional task, then it needs institutional power
  if $T_p \sqsubseteq$ Institutional_Task $\in \mathcal{O}$
  // then find agents having the capability and power to do T_p
  then Agent_list := RunQuery("Type(_:y, ?agent), PropertyValue(_:x, org:isPermitted, _:z),
        PropertyValue(_:x, org:hasPower, _:z), PropertyValue(_:y, org:isCapable, _:z),
        PropertyValue(_:y, org:hasRole, _:x), Type(_:z,dom:T_p)")
  // else find agents having the capability to do T_p
  else Agent_list := RunQuery("Type(_:y, ?agent), PropertyValue(_:y, org:isCapable, _:z), Type(_:z,dom:T_p)")
  for each Ag_i in Agent_list do
    // if Ag_H has power to delegate to Ag_i, then Ag_H gives him the order to do it
    if $\exists Role \in \mathcal{R}_H$ such that canDelegate($Role, Ag_i$)
      then SpeechAct(Ag_H, Ag_i,order, T_p) **return** true;
    else // Ag_H requests Ag_i to take the obligation
      if callWorkflow(request_WF, Ag_H, Ag_i, T_p) **then return** true;
  end for
  return false;

Fig. 7: Finding Alternative Agents

We now describe dealing with exceptions with examples. We consider the workflow example shown in Figure 9 which deals with printing a finance report. When a finance report is written, its format is then checked. Next, the report is to be approved, and printed locally, finally the report is posted. Below we list some of the ontology axioms which are relevant to this workflow.

algorithm Find_Alternative_Task($\langle id, T_p \rangle$, Ag_H, $\mathcal{O}$)
  $Ag_H = \langle \mathcal{R}_H, \mathcal{T}_H \rangle$
  // find any task T with same input/output
  Task_list := RunQuery("Type(_:t, ?task),Type(_:z, dom:T_p), PropertyValue(_:z, dom:hasInput, _:xi),
      PropertyValue(_:z, dom:hasOutput, _:xo), PropertyValue(_:t, dom:hasInput, _:xi),
      PropertyValue(_:t, dom:hasOutput, _:xo)")
  for each $T \in$ Task_list // check the consistency for each alternative task
    let t be an instance of T_p
    $\mathcal{O}' := \{T(t)\} \cup \mathcal{O} \setminus \{T_p(t)\}$
    if consistent($\mathcal{O}'$) then $\mathcal{O} := \mathcal{O}'$
      if T is a workflow, then Ag_H initiates T
      else if T is a task // then check if the Ag_H can do T
        if $T_p \sqsubseteq$ Institutional_Task $\in \mathcal{O}$ // then find agents having the capability and power to do T_p
          if $\exists Role \in \mathcal{R}_H$ such that isCapable(Ag_H, T)$\wedge$ hasPower($Role, T$)$\wedge$ isPermitted($Role, T$)
          then $Ag_H := \langle \mathcal{R}_H, \mathcal{T}_H \cup \{T\} \rangle$; **return** true;
        else if $\exists Role \in \mathcal{R}_H$ such that isCapable(Ag_H, T)
          then $Ag_H := \langle \mathcal{R}_H, \mathcal{T}_H \cup \{T\} \rangle$; **return** true;
        // otherwise call routine for finding alternative agent
        if Find_Alternative_Agent($\langle id, T \rangle$, Ag_H, $\mathcal{O}$) **then return** true;
  end for
  return false;

Fig. 8: Finding Alternative Task

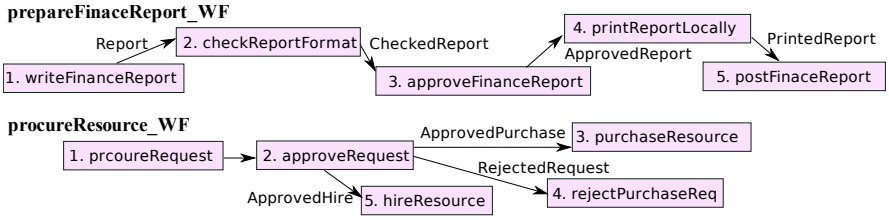


Fig. 9: Finance Report and Purchase Workflows

(1) ApproveReport $\sqsubseteq$ Institutional_Task
(2) Manager $\sqsubseteq \exists$ hasPower.ApproveReport $\sqcap \forall$ hasPower.ApproveReport $\sqcap \exists$ isCapbale.ApproveReport $\sqcap$ $\exists$ isPermitted.ApproveReport
(3) ITManager $\sqsubseteq \exists$ hasPower.ApproveITReport $\sqcap \forall$ hasPower.ApproveITReport
(4) FinanceManager $\sqsubseteq \exists$ hasPower.ApproveFinanceReport $\sqcap \forall$ hasPower.ApproveFinanceReport
(5) FinanceManager $\sqcup$ ITManager $\sqcup$ GeneralManager $\sqsubseteq$ Manager
(6) ApproveFinanceReport $\sqcup$ ApproveITReport $\sqsubseteq$ ApproveReport
(7) ApproveFinanceReport $\sqsubseteq \neg$ ApproveITReport
(8) Secretary $\sqsubseteq \exists$ isObliged.checkReportFormat

The FinanceManager has the power to do ApproveFinanceReport. If the FinanceManager is not available, then control returns to state 2 of the workflow, where the secretary must handle the exception using the routine in Figure 6. This leads to the routine "Find_Alternative_Agent" (in Figure 7) being run. After running the query within "Find_Alternative_Agent", the agent whose role is GeneralManager is returned as an appropriate candidate to carry out the approval task. This is because GeneralManager inherits power from its superclass Manager, while ITManager is restricted to approve IT reports only. The final stage of the exception handling is for the secretary to initiate a "request" workflow, to request that the general manager take on the obligation to approve the finance report. If this is successful, then the workflow can resume with the new substitute agent.

The next step is to print the report. To show an example of an unachievable task, let us assume no printer or toner is available; therefore printReportLocally cannot be implemented. As the secretary is responsible for this unachievable task, the secretary must execute of the routine "Task_Exception_handler" (in Figure

6), which reveals that a resource is missing, and no suitable alternative resource exists in the organisation. The routine then searches for alternative tasks and finds that the task printReportCommercially is a sibling task of printReportLocally, has the same input ApprovedReport and output PrintedReport. However, if the report is sensitive, it is not allowed to print it commercially (see axiom 6 below). Assume that printRpt322 is the instance of printReportLocally, which is to be replaced by printReportCommercially; according to "Find_Alternative_Task" (in Figure 8) we move the instance printRpt322 from printReportLocally to printReportCommercially. If the report is sensitive, i.e., printRpt322 is also an instance of SensitiveReport, then from axiom 6 below we could infer that the agent performing the commercial print act is violating norms. Hence, printReportCommercially should not be executed. If the agent chooses not to violate the norms, then "Find_Alternative_Task" fails and control returns to "Task_Exception_handler" (in Figure 6). Having failed to find an alternative resource or task, this routine now tries to procure the resource required for the task. This means that the secretary must initiate the "procureResource_WF" workflow (in Figure 9), and if successful, the printing resource is available, and the workflow can progress.

(1) printReportLocally $\doteq$ $\exists$ print.(Report $\sqcap$ $\exists$ printedBy.(Printer $\sqcap$ $\exists$ hasToner.(Toner $\sqcap$ $\exists$ hasAmt.GreaterThanZero)))
(2) Printer $\sqcup$ Toner $\sqsubseteq$ Resource
(3) procureRequest $\doteq$ $\exists$ requests.(Resource $\sqcap$ $\exists$ hasAmt.LessThanOne $\sqcap$ $=$ 1 hasPrice)
(4) printReport $\sqsubseteq$ $\exists$ hasInput.ApprovedReport $\sqcap$ $\exists$ hasOutput.PrintedReport
(5) printReportLocally $\sqcup$ printReportCommercially $\sqsubseteq$ printReport
(6) Staff(s) $\wedge$ SensitiveReport(r) $\wedge$ printReportCommercially(act) $\wedge$ print(act,r) $\wedge$ performed(s,act) $\wedge\mathcal{O}$(s) $\wedge\mathcal{O}$(r) $\wedge\mathcal{O}(act)$ $\rightarrow$ violated(s,act)
(7) Secretary $\sqsubseteq$ $\exists$ isObliged.printReportLocally

8 Related Work

Various works use agents to enact workflows. Buhler and Vidal [4] proposed to integrate agent services into BPEL4WS-defined workflows. The strategy is to use the Web Service Agent Gateway to slide agents between a workflow engine and the Web services it calls. Thus the workflow execution is managed centrally rather than by the agents. On the other hand Guo et al. [5] describe the development of a distributed multi-agent workflow enactment mechanism from a BPEL4WS specification. They proposed a syntax-based mapping between some of main BPEL4WS constructs to the Lightweight Coordination Calculus (LCC). This work however does not address organisational or normative aspects of an agent system; we believe that these high level aspects are important for agent systems that are to model real processes in human organisations. Furthermore we have shown how the use of ontologies to describe aspects of the organisation and domain can be valuable in exception handling, as agents are part of an organisation and will be unable to deal with exceptions entirely on their own.

Klein and Dellarocas [9] explicitly deal with the issue of exceptions; they propose the use of specialised agents that handle exceptions. The exception handling service is a centralised approach, in which a coordination doctor diagnoses agents' illnesses and prescribes specific treatment procedures. Klein and Dellarocas [10] identified an exception taxonomy which is a hierarchy of exception types, and then described which handlers should be used for what exceptions. Klein et al. [11] describe a domain-independent but protocol-specific exception handling services approach to increasing robustness in open agent systems. They focus on

"agent death" in the Contract Net protocol. We would argue that our approach is more generic in that it is neither domain-specific, nor protocol-specific. When seeking alternative ways to achieve workflow tasks, our agents can use the same handling routine regardless of the workflow in progress. A further distinction between our work and the above related works use some device which is added into the system to deal with exceptions, for example: specialised agents, an exception repository, or a directory to keep track of agents. In contrast, our approach aims to endow the agents of the system themselves with the ability to deal with exceptions by querying ontologies to find alternative ways.

Perhaps the closest approach to our work in the literature is from Mallya and Singh [14]. Building on the commitment approach, they have proposed novel methods to deal with exceptions in a protocol. They distinguish between expected and unexpected exceptions. Unexpected exceptions are closest to the types of exceptions we tackle here. Mallya and Singh's solution makes use of a library of sets of runs (sequences of states of an interaction) which could be spliced into the workflow at the point where the exception happens. This is similar to the way our exception handling can sometimes include a nested workflow in place of a failed task, to repair the workflow. However, they do not describe how these sets of runs can be created, but it is likely that one would need access to observed sequences from previous enactments of similar workflows. The aim of the commitment approach is in line with our work, as it endows the agents with some understanding of the meaning of the workflow they are executing, by giving them knowledge of the commitments at each stage. This would make it possible for agents to find intelligent solutions when exceptions arise. Similarly, in our approach, agents are endowed with semantic knowledge (represented in an ontology) about the capabilities and norms of the other roles so that they can find suitable candidates to execute tasks in the case of exceptions.

9 Conclusions & Future Work

In this paper we have described a method by which an agent system could be constructed to enact a set of given workflows, while respecting the constraints of a given organisation. We have shown how Semantic Web languages can be used to describe the organisational knowledge, as well as domain knowledge which can be used by the agents if exceptions arise during the enactment of a workflow; the agents can then use this knowledge to make intelligent decisions about how to find alternative ways to complete the workflow.

Some issues which have not been addressed include the updating of the organisational knowledge by agent activities outwith the workflows, for example speech acts that may change norms in the system. Also, we have not included any mechanism to detect when an obligation is violated. This could be addressed by associating time limits with obligations and including timer events which are triggered when obligations time out, and then checking if they have been fulfilled; this remains for future work.

Our aim has been to allow agents to deal with unexpected exceptions, rather than coding specific exception handlers for a predefined set of expected exceptions. Nevertheless we have had to define exception handling routines for some predefined situations, such as agent exceptions, or task exceptions. However, our

predefined situations cover a broad class of exceptions, and there can be many possible solutions if the ontological knowledge is suitably rich.

References

1. IBM, BEA Systems, Microsoft, SAP AG and Siebel Systems, business Process Execution Language for Web Services version 1.1. Technical report, July 2003.
2. A. Artikis. *Executable Specification of Open Norm-Governed Computational Systems*. PhD thesis, Imperial College London, 2003.
3. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
4. P. Buhler and J. M. Vidal. Integrating agent services into BPEL4WS defined workflows. In *Proceedings of the Fourth International Workshop on Web-Oriented Software Technologies*, 2004.
5. L. Guo, D. Robertson, and Y.-H. Chen-Burger. Using multi-agent platform for pure decentralised business workflows. *Web Int. and Agent Systems*, 6(3), 2008.
6. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2nd International Semantic Web Conference (ISWC 2003)*, number 2870, pages 17–29. Springer, 2003.
7. I. Horrocks and U. Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence*, pages 448–453, 2005.
8. Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies. In Kaelbling and Saffiotti, editors, *IJCAI'05*, 2005.
9. M. Klein and C. Dellarocas. Exception handling in agent systems. In *AGENTS '99: 3rd Annual Conference on Autonomous Agents*, pages 62–68, 1999.
10. M. Klein and C. Dellarocas. Towards a systematic repository of knowledge about managing multi-agent system exceptions. Technical Report ASES Working Report ASES-WP-2000-01, Massachusetts Institute of Technology, 2000.
11. M. Klein, J. Rodriguez-Aguilar, and C. Dellarocas. Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *Auton. Agents and Multi-Agent Systems*, 7(1-2):179–189, 2003.
12. A. Lanzén and T. Oinn. The taverna interaction service: enabling manual interaction in workflows. *Bioinformatics*, 24(8):1118–1120, 2008.
13. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.
14. A. U. Mallya and M. P. Singh. Modeling exceptions via commitment protocols. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 122–129. ACM, 2005.
15. B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In *Proc. of the 3rd Int. Semantic Web Conf.*, pages 549–563. Springer, 2004.
16. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.
17. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 355–362, 2003.
18. E. Sirin and B. Parsia. SPARQL-DL: SPARQL query for OWL-DL. In *3rd OWL Experiences and Directions Workshop (OWLED-2007)*, 2007.
19. W. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
20. G. H. von. Wright. Deontic logic. *Mind, New Series*, 60(237):1–15, January 1951.
21. WfMC. Workflow management coalition terminology and glosary. Technical Report WFMC-TC-1011, Workflow Management Coalition, 1999.

A Reputation Model for Organisational Supply Chain Formation *

Roberto Centeno[1], Viviane Torres da Silva[2], Ramón Hermoso[1]
{roberto.centeno, ramon.hermoso}@urjc.es, and
viviane.silva@ic.uff.br

[1] Centre for Intelligent Information Technologies (CETINIA)
University Rey Juan Carlos Madrid (URJC) - Spain
[2] Universidade Federal Fluminense (UFF) - Brazil

Abstract. The use of organisational concepts and the design of reputation mechanisms have been proved as successful methods to build multi-agent systems where agents' decision-making processes to select partners are crucial for the system functioning. In supply chain domains this latter issue becomes crucial in the phase of *formation* since entities participating need to establish business relationships as soon as possible in order to maximise their profits. In this work we propose: *i)* a novel formalisation of supply chains using organisational concepts and, *ii)* a reputation model based on those organisational concepts and on *personal norms* with which agents define their preferences about potential interactions. To conclude, we present a case study pointing out the stronger points of our work.

1 Introduction

A supply chain combines multiple enterprises to collaboratively provide customers with products or services [1]. A supply chain life-cycle comprises two main processes: the supply chain formation and the supply chain management. This paper focuses on the supply chain formation that concerns the selection of the participants to the supply chain and the agreement about the terms of the exchange [2]. In particular, the main goal of the paper is to contribute to the selection of the enterprises that will participate in the supply chain in order to establish stronger relationships.

Due to the need to quickly respond to changes in market requirements and to rapidly create or reconfigure supply chains, we present an approach that contributes to dynamic supply chain formation. A supply chain is viewed as an organisation composed of different enterprises, each represented by an agent playing roles in the organisation.

* The present work has been partially funded by the Spanish Ministry of Education and Science under project TIN2006-14630-C03-02 (FPI grants program) and by the Spanish project "Agreement Technologies" (CONSOLIDER CSD2007-0022, INGENIO 2010)

Agents and organisations to represent enterprises and supply chains, respectively, have been used because, in recent years, multi-agent systems (MAS) have been recognised as a promising technology for the automation of supply chains [1]. In addition, organisational approaches are more and more used since they allow facing complex problems using simple abstractions [3]. Those abstractions can be concepts that structure relationships among organisation members, such as roles that agents can play, and also constraints, such as norms that attempt to regulate agents' behaviour.

The ability to select suitable partners is one of the keys to build successful supply chains [4], so this issue should be adequately supported by our model. As has been demonstrated by several studies [5–7], trust and reputation contribute significantly to the formation of suitable partnerships and of stable supply chains. Therefore, we propose the use of *a reputation mechanism based on norms to give support to the dynamic supply chain formation.* The reputations of the agents are evaluated according to the (organisational and personal) norms that they violate and fulfil. Norms define the actions agents are prohibited, permitted or obligated [8] to perform and the sanctions/reward to be applied in the case of violations/fulfilment [9]. The organisational norms are the ones defined by supply chains (or organisations) while the personal norms are defined by the agents themselves. Note that the reputation mechanism should be used together with other mechanisms to investigate the most appropriate enterprises to participate in the supply chain being formed by analysing, not only the enterprises reputations, but also the products or resources provided, their cost, etc.

The remainder of the paper is organised as follows. Section 2 formalises our organisation model and Section 3 our reputation model. In Section 4 we present an overview of supply chain formation and Section 5 uses the formalisation presented in Section 2 to formalise supply chains. Section 6 illustrates our approach by using a supply chain case-study. Finally, Section 7 states some related work and Section 8 concludes and introduces some future work.

2 Organisational Model

In this section we present the formalisation of our organisational model. This model relies on three basic entities, namely: *roles*, *organisational norms* and *agents* endowed with *personal norms*. We define them in the next sections.

2.1 Organisation definition

Following the framework proposed in [10], our work focuses on a particular type of organised multiagent system - from now on *organisation* - which is endowed with two different organisational mechanisms: *organisational norms* and *roles*. An organisational mechanism is a mechanism which tries to influence the agents' behaviour; usually towards more effectiveness with regard to some objective. Taking into account these elements, we formally define an organisation as follows:

Definition 1 *An organisation* $\mathcal{O}$ *is a tuple* $\langle \mathcal{A}g, \mathcal{A}, \mathcal{X}, \phi, x_0, \varphi, \{\mathcal{ON}^{om}, \mathcal{R}^{om}\}\rangle$ *where:*

- $\mathcal{A}g$ is a set of agents participating in the organisation, $|\mathcal{A}g|$ denotes the number of agents;
- $\mathcal{A}$ is a possibly infinite action space that includes all possible actions that can be performed in the system. $\mathcal{A}$ includes an action a_{skip}; the action of doing nothing[3];
- $\mathcal{X}$ is the environmental state space;
- $\phi : \mathcal{X} \times \mathcal{A}^{|\mathcal{A}g|} \times \mathcal{X} \rightarrow [0..1]$ is the transition probability distribution, describing how the system evolves as a result of agents' actions;
- $x_0 \in \mathcal{X}$ stands for the initial state of the system;
- $\varphi : \mathcal{A}g \times \mathcal{X} \times \mathcal{A} \rightarrow \{0, 1\}$ is the agents' capability function describing the actions agents are able to perform in a given state of the environment. $\varphi(ag, x, a) = 1$ means that agent ag is able to perform action a in the state x (0 otherwise);
- $\mathcal{ON}^{om}$ is an organisational mechanism based on organisational norms that regulates agents' behaviour. It will be formalised in section 2.5 definition 6;
- $\mathcal{R}^{om}$ stands for an organisational mechanism based on roles that defines the positions agents may enact in the organisation. It will be formalised in section 2.3 definition 4.

The proposed model adopts the view in which an agent's environment is everything that surrounds it, i.e., any existing entity including other agents. Accordingly, when agents perform actions that are executed in the environment, these actions may influence other agents. The model assumes that the organisation evolves at discrete time steps. In each time step, all agents participating in the organisation perform an action and the new state of the system is produced, with a given probability, through the joint actions of all agents. We also assume that agents can select a "skip" action (a_{skip}), which allows for modelling asynchronous behaviours. The agents' capability function defines what actions the agents are able to perform in a given state. It defines the environmental limitations imposed by the environment (e.g. if there does not exist a hotel in the environment, agents cannot book a room). These environmental limitations could be considered as hard constraints, in the sense that cannot be avoided, i.e. an agent either has the capability to perform an action or not.

2.2 Agents

Agents are considered independent, autonomous software components that are able to perceive observations about their environment and, based on these observations, take actions. This could be a general definition about agents. However in our work we add the concept of *personal norms*. They are individual norms which agents have and apply to situations in which they are involved. Therefore, we formally define an agent as follows:

[3] This action allows for modelling asynchronous behaviours.

Definition 2 *An agent $\mathcal{A}g$ is a tuple $\langle \mathcal{S}, \mathcal{O}, g, t, per, \delta, s_0, \mathcal{PN}^{im} \rangle$ where:*

- $\mathcal{S}$ is the (possible) infinite set of internal states of the agent;
- $\mathcal{O}$ is the set of possible observations an agent is able to perceive from the system;
- $g : \mathcal{O} \times \mathcal{S} \rightarrow \mathcal{S}$ is the agent's state transition function;
- $t : \mathcal{S} \rightarrow \mathcal{A}$ is the agent's decision function describing the next action it will choose given an internal state;
- $per : \mathcal{X} \rightarrow \mathcal{O}$ is a perception function assigning an observation to an environmental state;
- $\delta : \mathcal{A} \rightarrow \{0, 1\}$ stands for the agent's capability function which assigns 1 to an action when the agent has the capability to perform it, or 0 in other case;
- s_0 is the agent's initial internal state;
- $\mathcal{PN}^{im}$ stands for a mechanism based on *personal norms* that internally regulates the agent's behaviour. It will be formalised in section 2.6 definition 8.

At each time step, agents perceive an observation from the current environmental state. Agents transit from that observation to a new internal state, and then they will select their next action by using their own decision function taking into account their new internal state. Furthermore, agents regulate their own behaviour according to their personal norms (they are described in Section 2.6).

2.3 Roles as Organisational Mechanisms

In order to better describe how roles act as an organisational mechanism we need first to formally define a role:

Definition 3 *Let $\mathcal{RI}$ be a set of role identifiers. A role is a pair $\langle r_{id}, \omega \rangle$ where*

- $r_{id} \in \mathcal{RI}$ is the role name;
- $\omega : \mathcal{A} \rightarrow \{0, 1\}$ is a function that represents if the role is suitable to perform an action. The assignment of 0 means the action cannot be performed with the current role.

This definition embraces some different aspects about the role semantics: *i)* holds the sense of being used as a first-order block to build MAS. It does not lose any semantics regarding this issue; *ii)* represents a group of functionalities, since every role will be qualified for the set of possible actions in the system. If the value associated with an action is 0, it means the role cannot be used for doing the specified action; *iii)* the role entails an expected behaviour that the agent enacting it has to fulfil.

As we have pointed out in Def.1 we propose to endow organisations with roles which are defined as an organisational mechanism [10]. Thus, we formally define roles as organisational mechanisms as follows:

Definition 4 *A role-based organisational mechanism $\mathcal{R}^{om}$ is a tuple $\langle \mathcal{R}$, can, isPlaying$\rangle$ where:*

– $\mathcal{R}$ stands for a set of roles;
– *can* defines if an agent is able to perform an action and it is defined as follows:

$$can : \mathcal{A}g \times \mathcal{R} \rightarrow \begin{cases} 1 & if \quad \delta_{ag_i}(a_j) = 1 \wedge \exists r \in \mathcal{R} \mid \omega(a_j) = 1 \wedge \varphi(ag_i, x_k, a_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where

- $ag_i \in \mathcal{A}g$ represents an agent and $a_j \in \mathcal{A}$ is an action;
- $\delta_{ag_i}(a_j)$ stands for the agent's ag_i capability function performing the action a_j (see Def. 2);
- $\varphi(ag_i, x_k, a_j)$ represents the environmental capability for the agent ag_i to perform action a_j given an environmental state x_k;
- $\omega(a_j)$ defines if the role $r \in \mathcal{R}$ is suitable to perform the action a_j (see Def. 3);

– $isPlaying : \mathcal{A}g \times \mathcal{R} \rightarrow \{0, 1\}$ is a function that returns 1 if an agent *can* play a role and is currently playing it and 0 in any other case. This function is necessary in order to determine which roles agents play at any time.

2.4 Agents, Roles and Action Dynamics

As stated previously, during the lifetime of the organisation, agents must select and perform an action at each time step[4] while playing a role. We define this fact as a situation, so then agents will be involved in several situations during a finite time period. Formally, we define a situation as follows:

Definition 5 *A situation $\mathcal{S}it$ is a tuple $\langle \mathcal{A}g, \mathcal{R}, \mathcal{A}, T \rangle$ where:*

– $\mathcal{A}g$ is the agent involved in the situation;
– $\mathcal{R}$ is the role which is being played by the agent $\mathcal{A}g$ in that situation;
– $\mathcal{A}$ stands for the action which is being performed by the agent $\mathcal{A}g$ playing the role $\mathcal{R}$;
– T is the time period in which the situation happened.

Different types of situations can be defined following the definition above. Situations in which there are no roles or actions or even an agent are valid; for instance, a situation in which an agent is performing an action, regardless of the role it is playing at the current time; or an agent playing a role without performing any action; or a situation can even describe the fact that all agents are playing a particular role and are performing a particular action during a time period. Therefore, the components agent, role or action could be empty in a situation as well. In this work, situations are a key concept to deal with norms.

[4] Including the action of doing nothing (a_{skip})

2.5 Organisational Norms as Organisational Mechanisms

Organisational norms regulate agents' behaviour by defining the actions agents are prohibited, permitted or obligated [8] to perform, and the sanctions/rewards to be applied in the case of violations/fulfilments [9]. Hence, organisational norms can be classified as organisational mechanisms, particularly as incentive mechanisms [10], since they can introduce rewards and/or penalties trying to influence agents' behaviour. In this sense, we define organisational norms as an organisational mechanism as follows:

Definition 6 *An organisational mechanism based on organisational norms* $\mathcal{ON}^{om}$ *is a tuple* $\langle \mathcal{ON}, monitor, act \rangle$ *where:*

- $\mathcal{ON}$ stands for a set of organisational norms defined in the organisation;
- $monitor : \mathcal{X} \times \mathcal{ON} \rightarrow \{0, 1\}$ represents a function which is able to monitor the state of an organisation, determining when an organisational norm has been violated (assigning 1) or fulfilled (assigning 0);
- $act : \mathcal{X} \rightarrow \mathcal{A}$ is a function that applies the consequences of the violation or fulfilment of a norm - perceived by the *monitor* function, so imposing a punishment and/or a reward or nothing (a_{skip}).

On the other hand, we define organisational norms as norms which regulate situations, that is, they regulate when a situation is prohibited, permitted or obligated to perform. Formally we define organisational norms as follows:

Definition 7 *An organisational norm* $\mathcal{ON}_i(Org, \mathcal{S}it)$ *regulates a situation in an organisation where:*

- Org stands for an organisation and $\mathcal{S}it$ is a situation.

Organisational norms entail constraints in the system. They can be seen as a second regulation layer imposed by the organisation. Even if an agent *can* perform an action, it does not mean that the agent should do it. Of course the agent can do the action, but organisational norms should also influence the agent somehow if the selected action is violating one of them. Thus, the agents are firstly constrained by its capability to do actions from a physical perspective (what can I do?) and, secondly, by the organisational norms in the organisation (what should I do?).

2.6 Personal Norms as Individual Mechanisms

These norms regulate the situation in which an agent is involved from an individual point of view. The difference to organisational norms is that the former have a private scope, since organisational norms must be known by all participants in the organisation, whilst personal norms could only be known by the owner. Another important difference between personal and organisational norms is how their violation/fulfilment are handled. In the case of organisational norms the violation/fulfilment of them may be checked by a third independent party (an

authority) not involved in the situation in which the violation/fulfilment occurs. Meanwhile, the only entity able to apply a penalty or a reward when a personal norm is violated/fulfiled is the agent that is the owner of the norm. Hence, following Def. 6, the functions *monitor* and *act* should be defined in the agent owner of the personal norm; that is, this agent will be in charge of monitoring and acting when a personal norm is violated or fulfilled. Formally, personal norms are defined as follows:

Definition 8 *A personal norm-based individual mechanism* $\mathcal{PN}^{im}$ *is a tuple* $\langle \mathcal{PN}, monitor, act \rangle$ *where:*

- $\mathcal{PN}$ stands for a set of personal norms defined by an agent;
- $monitor : \mathcal{S} \times \mathcal{PN} \rightarrow \{0, 1\}$ represents a function which is able to monitor the agent's internal state, determining when a personal norm has been violated (assigning 1) or fulfilled (assigning 0);
- $act : \mathcal{S} \rightarrow \mathcal{A}$ is a function that selects an action in order to impose a reward and/or a punishment, when the *monitor* function determines if a personal norm has been violated/fulfilled in an agent's internal state.

Therefore, we formally define personal norms as follows:

Definition 9 *A personal norm* $\mathcal{PN}(\mathcal{A}g, \mathcal{S}it)$ *is defined by an agent and regulates a situation in which the agent is involved:*

- $\mathcal{A}g$ stands for the agent, which is the owner of the personal norm;
- $\mathcal{S}it$ is the situation regulated by the personal norm.

These norms represent the preferences of an agent relative to a situation. Thus, a personal norm describes what should happen in the situation that is regulated by the norm, from the point of view of the agent that defines the norm. Since personal norms are relative to agent's preferences they are related to the agent's utility function as well. Then we could say that personal norms somehow tune an agent's utility assessment. Organisational and personal norms are related; on the one hand, they may regulate the same situation and; on the other hand, they are defined using the same ontology, shared by all agents participating in the organisation.

3 Reputation Model

In this section we propose a reputation mechanism based on organisations, using the concepts put forward before. Reputation mechanisms are well-known techniques to keep agents from unexpected behaviour (i.e. norm violations) since they provide agents with relevant information about the trustworthiness of others. Most of the reputation systems use quantitative values (opinions) to indicate the reputation of agents [11]. However, such information is not sufficient to understand the behaviour of the agents since such values are subjective, i.e. the

same norm violation or fulfilment can be differently evaluated by two different agents. The subjective opinion of each agent about the same third party behaviour could entail the problem of interpreting the meaning of the agent reputation. In order to tackle this issue, we propose a reputation model that not only takes into account a numerical value as the opinion an agent provides about a third party behaviour, but also the set of norms that the latter has violated or fulfilled and the facts associated with them as a justification of the former's evaluation. This could be viewed as a single-step argumentation about how an agent evaluates the opinion about others.

Thus, agents may perform a reputation request to other agents at any time. A *reputation request* is related to a situation, i.e., an agent may request the reputation of another agent playing a role and performing a particular action along a time period. Hence, we formalise a *reputation request* as follows:

Definition 10 *A reputation request* $\mathcal{R}^{req}_{\mathcal{A}g_1 \rightarrow \mathcal{A}g_2}$ *is a request performed by an agent* $\mathcal{A}g_1$ *to another agent* $\mathcal{A}g_2$ *about the reputation of agent* $\mathcal{A}g$ *in a particular situation,* $\mathcal{R}^{req}_{\mathcal{A}g_1 \rightarrow \mathcal{A}g_2}(\langle \mathcal{S}it \rangle)$ *where:*

- $\mathcal{A}g_1$ is the agent which is requesting the reputation request;
- $\mathcal{A}g_2$ stands for the agent which receives the reputation request;
- $\mathcal{S}it$ represents the situation.

When an agent receives a *reputation request* it may reply with the reputation value it has assigned to the agent participating in that situation. So far, this is the common way to use a reputation mechanisms. However, we propose reply with a subjective value which indicates the reputation of a third party, as well as the organisational norms which were violated by the agent, the facts which produced the violation of those norms and how many times the agent violated them. Hence, we define a reply to a *reputation request* as follows:

Definition 11 *A reply to a reputation request* $\mathcal{R}^{reply}_{\mathcal{A}g_2 \rightarrow \mathcal{A}g_1}$ *is a tuple* $\langle \mathcal{S}it, RepVal, \mathcal{ON}, \mathcal{F}, r \rangle$ *which represents the message sent by agent* $\mathcal{A}g_2$ *to agent* $\mathcal{A}g_1$ *as a reply to a reputation request, where:*

- $\mathcal{S}it$ stands for the situation related to the reputation request;
- $RepVal \in [0..1]$ is the reputation value that agent $\mathcal{A}g_2$ sent to agent $\mathcal{A}g_1$ about the agent which is involved in the situation;
- $\mathcal{ON}$ is the set of organisational norms which were violated by the agent involved in the situation;
- $\mathcal{F}$ stands for the set of facts that constitute proof/evidences of violations of organisational norms;
- r represents the number of times that the agent involved in the situation violated the organisational norms.

At this point, agents could evaluate a third party only with the reputation gathered by all the *reputation request*s asked to different agents. In addition, our model proposes the possibility of asking about the personal norms that

the requested agent has in relation to the same situation. Allowing agents to ask about personal norms that other agents have is important to find affinity with other participants or even to build different profiles which allow agents "better" partner selections. Hence, an agent may ask for personal norms about a particular situation to another agent. It is formalised as follows:

Definition 12 *A personal norms request* $\mathcal{PN}^{req}_{\mathcal{A}g_1 \rightarrow \mathcal{A}g_2}$ *is a request performed by an agent* $\mathcal{A}g_1$ *to another agent* $\mathcal{A}g_2$ *about the personal norms that agent* $\mathcal{A}g_2$ *has related to a particular situation,* $\mathcal{PN}^{req}_{\mathcal{A}g_1 \rightarrow \mathcal{A}g_2}(\langle \mathcal{S}it \rangle)$ *where:*

- $\mathcal{A}g_1$ is the agent which is requesting the personal norms request;
- $\mathcal{A}g_2$ stands for the agent which receives the personal norms request - the owner of the personal norm;
- $\mathcal{S}it$ is the situation.

Agents, as autonomous entities, may or may not reply with their personal norms. If they decide to reply to a personal norms request, they will send a set of personal norms they have related to the situation asked. They can send all the personal norms related to the situation or only a subset of them, this is a decision they have to make. Thus, we can formalise a reply to a personal norms request as follows:

Definition 13 *A reply to a personal norms request* $\mathcal{PN}^{reply}_{\mathcal{A}g_2 \rightarrow \mathcal{A}g_1}$ *is a tuple* $\langle \mathcal{S}it, \mathcal{PN}, \mathcal{F}, r \rangle$ *sent by the agent* $\mathcal{A}g_2$ *to agent* $\mathcal{A}g_1$ *informing about its personal norms related to a situation where:*

- $\mathcal{S}it$ is the situation referenced by the personal norms request;
- $\mathcal{PN}$ stands for a set of personal norms;
- $\mathcal{F}$ stands for the set of facts which violated the personal norms;
- r represents the number of repetitions that the agents involved in the situation violated the personal norms.

4 Supply Chains Overview

A supply chain is the link among a company, its suppliers and its customers [5]. Most of the companies involved in a supply chain are both customers (that select supplies to buy goods) and suppliers (that sell their finished goods to customers) [12]. The two main processes of a supply chain are: supply chain formation and supply chain management. On one hand, supply chain formation is the problem of deciding who will supply what, who will do what and who will buy what. On the other hand, supply chain management concerns the coordination among the different operations across the supply chain [13].

In this paper we focus on a bottom-up scenario of the supply chain formation. In the bottom-up scenario, there is not a principal enterprise in charge of the formation (as is the case of the top-down scenario) but every supply chain partner is able to contribute to the formation of the supply chain by using its knowledge

about the partners [14]. In this scenario, the selection of participants is made on the fly and not defined *a priori*.

With the aim to contribute to the formation of stable supply chains and to the establishment of strong relationships, the use of trust and reputation have been encouraged [5–7]. Reputation reflects an aggregation value incorporating multiple factor: quality of the product, quality of the service provided, reliability of financial transaction, etc. [15].

5 Supply Chains as Organisations

As we pointed out in sections 1 we claim that supply chains can be modelled as organisational multiagent systems, where agents represents stake-holders in the supply chain and organisational abstractions, such as roles or norms may significantly help agents to easily form a stable supply chain in terms of reliability and profitability. On the other hand, market rules will be represented by organisational norms, those norms shared and accepted for any actor - any agent - in the supply chain, whilst roles represent different functionalities and capacities that different agents have in order to perform different actions. Using both concepts (norms and roles), an organisational flavour endows supply chains with a larger capacity for their participants to reason about others and then better decide what to do next.

In this section we formally define a supply chain using the concepts and properties presented in section 2. We consider the following as the common model that any type of supply chain should fulfil.

Definition 14 *A supply chain $\mathcal{SC}$ is a tuple $\langle \mathcal{A}g, \mathcal{A}, \mathcal{X}, \phi, x_0, \varphi, \{\mathcal{ON}^{om}, \mathcal{R}^{om}\}\rangle$ where:*

- In the literature we can find different minimum sets of roles involved in a supply chain flow. We adhere to the definition of roles participating in any supply chain presented in [13]:

$$\mathcal{RI} = \{Provider, Manufacturer, Purchaser, Carrier, Customer\}^5.$$

- Similarly to roles, we can also distinguish a common set of possible actions $\mathcal{A}$ available in any supply chain. We propose the following:

$$\mathcal{A} = \{Sell, Buy, Transform, Store, Transport\}$$

  Other actions could be considered, such as: *PlaceOrder*, *AcceptOrder* or *Manufacture_Good*. Nevertheless, the first two actions could be deemed as part of the process of selling and buying, so they are decompositions of the actions contained in the former set, while the third one is an specialisation of *Transform* action.

[5] For the sake of simplicity this represents only the set of role names described in Section 2.3

– In our definition of organisation, an organisational mechanism, based on organisational norms, is composed of a set of organisational norms. We claim that some of these norms are not case dependent, i.e., particular to an specific supply chain, but they are common for any type of supply chain - they all are applicable for the same domain. Thus we could consider the set $\mathcal{ON} = \mathcal{ON}^d \cup \mathcal{ON}^c$ as a union of the norms present in an organisational domain, such as, in this case, supply chain domain ($\mathcal{ON}^d$), and those norms that are case-dependent ($\mathcal{ON}^c$), such as specific cases of supply chains (i.e. car manufacturing supply chains).
An example of norms in $\mathcal{ON}^d$ are:
  - $\mathcal{ON}_1$: *A Provider cannot deliver an order after the delivery deadline.*
  - $\mathcal{ON}_2$: *A Provider cannot deliver less quantity of a good than the one that was ordered.*
  - $\mathcal{ON}_3$: *A Customer cannot pay less to a Provider than the price fixed in the order.*

  Examples of case-dependent norms - those in the set $\mathcal{ON}^c$- are given in section 6.

6 Case Study: A Computer Assembly Supply Chain

In this work we have focused on a particular type of supply chain, which represents the computer industry, as illustrated in figure XXX. It is usually called *computer assembly supply chain* [16] – from now on *CASC*. Following the approach presented in section 5, we formalise this kind of supply chain as a particular organisation of multiagent system, as follows:

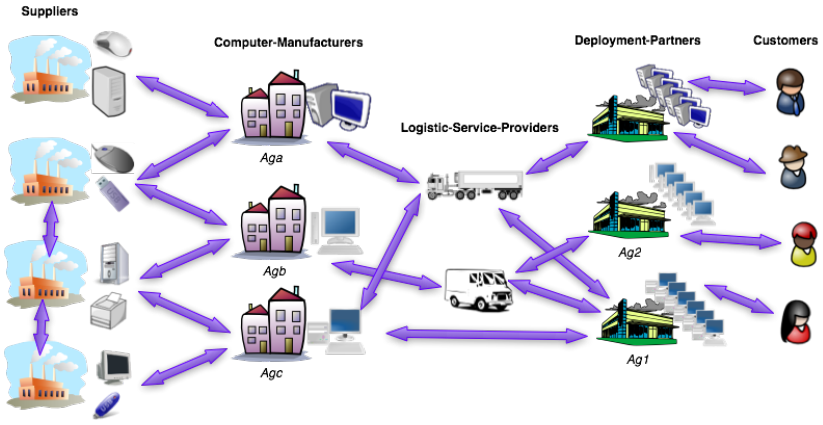


Fig. 1. An example of computer assembly supply chain

Definition 15 *A* CASC *is an organisation defined by the tuple* $\langle \mathcal{A}g, \mathcal{A}, \mathcal{X}, \phi, x_0, \varphi, \{\mathcal{ON}^{om}, \mathcal{R}^{om}\}\rangle$*, where:*

- the set of roles in $\mathcal{R}^{om}$ is the following:

$$\mathcal{RI} = \{\ \textit{supplier}, \textit{computer-manufacturer}, \textit{logistic-service-provider}, \textit{deployment-partner}, \textit{customer}\ \}$$

  Agents playing the role *supplier* are able to supply the products needed to assembly a computer while *computer-manufacturers* are able to perform the assembly of the computers. The *logistic-service-providers* transport the product from different centres and the *deployment-partners* are able to prepare the orders made by the customers. *Customers* can be corporate customers, market stores, small customers, etc. By computer we mean consumer PCs, professional PCs, servers and notebooks.
- the set of *actions* is composed of:

$$\mathcal{A} = \{\textit{Supply-Components}, \textit{Assembly-Computer}, \textit{Transport-Products}, \textit{Prepare-Order}, \textit{Buy-Computer}\ \}^{6}$$

- the set of *organisational norms* in $\mathcal{ON}^{om}$ is composed of the norms shared by all agents, representing the rules of the CASC. Examples of those norms are:
  - $\mathcal{ON}_1$: *"the number of controller cards may not exceed the number of extension slots of the system board"*
  - $\mathcal{ON}_2$: *"the processor type must be compatible with the system board"*
  - $\mathcal{ON}_3$: *"if the computer has not got a DVI port and the monitor is DVI, then the computer will be supplied with a DVI to VGA adaptor"*

In order to illustrate how the reputation model proposed in section 3 works in an example of a CASC, consider the CASC exemplified in the table below. Let us put ourselves in the place of the agent $\mathcal{A}g_1$ that is playing the role *deployment-partner (d-p)*. In order to form the supply chain $\mathcal{A}g_1$ has to select at least one *computer-manufacturer (c-m)* to be able to perform the action *prepare-order*[7]. Imagine that the agent has not interacted with any *computer-manufacturers* in the system yet. Therefore, the agent decides to use the reputation mechanism to select one of them. Below there is one possible sequence of interactions between $\mathcal{A}g_1$ and $\mathcal{A}g_2$:

[6] The action *supply-components* includes external units (printers, monitors, ..), accessories (keyboard, mouse, ..), software, etc. The action *assembly-computer* can be divided in *assembly-system-board* and *assembly-system-unit*. The action *transport-products* means transporting any kind of products.

[7] This action includes placing an order with a *computer-manufacturer*. The computer being ordered will be then transported by a *logistic-service-provider*

Org. CASC-1		
Organisational Norms		
$\mathcal{ON}_1$, $\mathcal{ON}_2$ and $\mathcal{ON}_3$		
Participants		
Agents	Role	Personal Norms
$\mathcal{A}g_1$	*deployment-partner (d-p)*	$\mathcal{PN}_{1-1}$: *"the number of empty extension slots must not be greater than one"* $\mathcal{PN}_{1-2}$: *"if the computer has not got a PS2 port and the mouse is USB, it will have an adaptor USB to PS2 and PS2 to USB"*
$\mathcal{A}g_2$	*deployment-partner (d-p)*	$\mathcal{PN}_{2-1}$: *"the number of empty extension slots must be zero"*
$\mathcal{A}g_a$	*computer-manufacturer (c-m)*	$\mathcal{PN}_{a-1}$: ...
$\mathcal{A}g_b$	*computer-manufacturer (c-m)*	$\mathcal{PN}_{b-1}$: ...

1. $\mathcal{R}^{req}_{\mathcal{A}g_1 \rightarrow \mathcal{A}g_2}(\langle \mathcal{A}g_a, \textit{c-m}, \textit{prepare-order}, [0 - now] \rangle)$: agent $\mathcal{A}g_1$ performs a reputation request to agent $\mathcal{A}g_2$ asking about the situation in which the agent $\mathcal{A}g_a$ has been involved playing the role *c-m* (computer-manufacturer) and performing the action *prepare-order* along the time period $[0 - now]$.
2. $\mathcal{R}^{reply}_{\mathcal{A}g_2 \rightarrow \mathcal{A}g_1} = (\langle \mathcal{A}g_a, \textit{c-m}, \textit{prepare-order}, [0 - now] \rangle, 0.2, \mathcal{ON}_2, \mathcal{F}_1, 2)$: agent $\mathcal{A}g_2$ replies to agent $\mathcal{A}g_1$ informing that the agent $\mathcal{A}g_a$, in the situation explained before, has a reputation of 0.2 from his point of view. It also informs that such reputation is due to two violations of the organisational norm $\mathcal{ON}_2$ by the fact $\mathcal{F}_1$. Agent $\mathcal{A}g_1$ knows that this fact describes that agent $\mathcal{A}g_a$ prepared a computer with the wrong processor.
3. Due to the low reputation value, agent $\mathcal{A}g_1$ decides to ask $\mathcal{A}g_2$ the reputation of agent $\mathcal{A}g_b$, another *computer-manufacturer*.
4. $\mathcal{R}^{req}_{\mathcal{A}g_1 \rightarrow \mathcal{A}g_2}(\langle \mathcal{A}g_b, \textit{c-m}, \textit{prepare-order}, [0 - now] \rangle)$: the agent $\mathcal{A}g_1$ performs a new reputation request to agent $\mathcal{A}g_2$ looking for the reputation of agent $\mathcal{A}g_b$.
5. $\mathcal{R}^{reply}_{\mathcal{A}g_2 \rightarrow \mathcal{A}g_1} = (\langle \mathcal{A}g_b, \textit{c-m}, \textit{prepare-order}, [0 - now] \rangle, 0.3, \_, \_, 0)$: agent $\mathcal{A}g_2$ replies to agent $\mathcal{A}g_1$ about the reputation of the agent $\mathcal{A}g_b$ in the specified situation. Although, the agent has not violated any organisational norm, the reputation of agent $\mathcal{A}g_b$ is low, 0.3.
6. Agent $\mathcal{A}g_1$ knows that such a low reputation must be due to the violation of at least one *personal norm* defined by agent $\mathcal{A}g_2$. Thus, $\mathcal{A}g_1$ decides to perform a *personal norms request* to agent $\mathcal{A}g_2$ in order to try to understand the reasons for such a low reputation.
7. $\mathcal{PN}^{req}_{\mathcal{A}g_1 \rightarrow \mathcal{A}g_2}(\langle \mathcal{A}g_b, \textit{c-m}, \textit{prepare-order}, [0 - now] \rangle)$: agent $\mathcal{A}g_1$ sends a personal norms request to agent $\mathcal{A}g_2$ asking about its personal norms which regulate the situation in which the agent $\mathcal{A}g_b$ has been involved playing the role *computer-manufacturer* and performing the action *prepare-order*.
8. Once agent $\mathcal{A}g_2$ receives the request, it may decide whether it answers or not. Suppose that the agent decides to answer with a *reply to a personal norms request*.

9. $\mathcal{R}^{reply}_{\mathcal{A}g_2 \rightarrow \mathcal{A}g_1} = (\langle \mathcal{A}g_b, c\text{-}m, prepare\text{-}order, [0-now] \rangle, \mathcal{PN}_{2-1}, \mathcal{F}_{2-1}, 3)$: the agent $\mathcal{A}g_2$ sends the reply to agent $\mathcal{A}g_1$ informing about the personal norms which regulate the situation requested ($\mathcal{PN}_{2-1}$) and the number of times they have been violated (3 times) by agent $\mathcal{A}g_b$.
10. When agent $\mathcal{A}g_1$ receives the reply it realises that: *i)* the low reputation is due to a violation of a personal norm ($\mathcal{PN}_{2-1}$); *ii)* $\mathcal{PN}_{2-1}$ is a more restricted norm than ($\mathcal{PN}_{1-1}$), its own personal norm; and *iii)* there exists some kind of affinity between them - this situation is important for them, because of it is regulated by a personal norm; thus, agent $\mathcal{A}g_2$ might be a potential "good" reputation source for agent $\mathcal{A}g_1$ for future partner selections.

7 Related Work

Although there are several studies [5–7] that demonstrate that trust and reputation contribute significantly to the formation of stable supply chains, there are several authors that have not considered trust/reputation while proposing their supply chain formation models, mechanism and protocols, such as [17, 18, 2].

In [12] the authors investigate the impact of reputation on supply chains. They demonstrate that a weighting of the agent's decision to choose a supplier in favour of the reputation component at the expense of the price component leads to the formation of stable supply chains that increase the tendency of monopoly formation. An important drawback of their analysis is that they focus on the individual dimension of reputation occurring in direct interactions between two agents. They have not considered that individuals can offer different opinions about the reputation of others.

A prototype for an agent-based electronic marketplace using reputations was proposed in [19]. In this prototype agents are able to evaluate the behaviour of others, store the reputations and distribute them to others. The two main disadvantages of this approach are: *i)* it is not clear how the agents evaluate the reputations of others, i.e., it is not clear what influences the reputation of an agent; and *ii)* the information that is distributed among agents is only a reputation value, no other information is communicated about agents' past behaviour.

The paper presented in [20] proposes a decentralised negotiation protocol based on trust for a supply chain environment. One important advantage of such an approach is the use of incentive-compatible mechanisms to avoid exploitation by competitors, i.e., it is able to deal, for instance, with the unwillingness of actors to reveal sensitive but (in terms of system optimisation) valuable information. The main drawback of this approach is that the trust accounting mechanism used in this model does not employ a composed reputation index using, for example, indirect reputation like most MAS reputation applications do. The authors claim that this is not necessary due to the equilibrium property of the system introduced by the negotiation process. On the other hand, the use of indirect trust would help identify regularly cheating agents earlier at the risk of further increasing information flow and negotiation effort.

8 Conclusions

In this paper we propose *i)* an organisational model based on roles and (organisational and personal) norms to define supply chains; and *ii)* a reputation mechanism to help on the supply chain formation. We advocate the use of reputations to support the selection the supply chain participants. In order to do so, supply chains are viewed as organisations and the enterprises as agents playing roles in the organisations. We assume that the enterprises are able to evaluate the behaviour of others by considering the organisational and personal norms that they fulfil and violate, to store such evaluation as reputations and to provide the reputations when requested. Thus, while forming a new supply chain, we stimulate the use of the available reputations to help on the selection of the new partners. It is important to state that there are some works [5–7] that affirm that trust and reputation contribute significantly to the formation of suitable partners and of stable supply chains.

We are in the process of implementing a simulator to validate our approach. By using the simulator we will be able to evaluate the real benefits that come from the availability of an enterprise's reputations during supply chain formation. As future work we also intend to apply our approach in other supply chain domains to demonstrate that our approach is domain-independent. In addition, we plan to extend the reputation mechanism to reflect a hybrid model. The approach presented in this paper uses a decentralised model. There is not any centralised entity able to store and provide reputations; instead it is the agents themselves that evaluate the behaviour of others, store and provide the reputations. As stated in [11], a hybrid reputation model seems to be the best-suited approach.

References

1. Tian, J., Tianfield, H.: Multi-agent based dynamic supply chain formation in semimonopolized circumstance. In: Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues. Volume 4601 of LNCS. Springer-Verlag (2007) 179–189
2. Walsh, W., Wellman, M.: Decentralized supply chain formation: A market protocol and competitive equilibrium analysis. Journal of Artificial Intelligence Research **19** (2003) 513–567
3. Omicini, A., Ossowski, S.: Objective versus subjective coordination in the engineering of agent systems. In Klusch, M., Bergamaschi, S., Edwards, P., Petta, P., eds.: Intelligent Information Agents: An AgentLink Perspective. Volume 2586 of LNAI. Springer-Verlag (2003) 179–202
4. Wang, M., Wang, H., Vogel, D., Kumar, K., Chiu, D.K.: Agent-based negotiation and decision making for dynamic supply chain formation. Engineering Applications of Artificial Intelligence **In Press, Corrected Proof** (2008) –
5. Dan, W., Ying, F., Feng, W.: Study on trust among supply chain companies. In: International Conference on Management Science and Engineering (ICMSE). (2006) 2266–2271

6. Franke, J., Stockheim, T.: An analysis of the impact of reputation on supply webs. In: Proceedings of the 11th European Conference on Information Systems (ECIS). (2003)
7. Spekman, R., Kamauff, J., Myhr, N.: An empirical investigation into supply chain management: A perspective on partnerships. International Journal of Physical Distribution and Logistics Management **28**(8) (1998) 630–650
8. Vázquez-Salceda, J., Aldewereld, H., Dignum, F.: Implementing norms in multiagent systems. LNAI **3187** (2004) 313–327
9. y López, F.L.: Social power and norms: Impact on agent behavior. PhD thesis, Univ. of Southampton, Faculty of Engineering and Applied Science, Department of Electronics and Computer Science (2003)
10. Centeno, R., Billhardt, H., Hermoso, R., Ossowski, S.: Organising mas: A formal model based on organisational mechanisms. In: 24th Annual ACM Symposium on Applied Computing (SAC2009), Hawaii, USA, March 8-12. (2009) to appear
11. Silva, V., Hermoso, R., Centeno, R.: A hybrid reputation model based on the use of organization. In Hubner, J., Matson, E., Boissier, O., Dignum, V., eds.: Coordination, Organizations, Institutions, and Norms in Agent Systems IV. Volume 5428 of LNAI. Springer-Verlag (2009)
12. Franke, J., Stockheim, T., Knig, W.: The impact of reputation on supply chains: An analysis of permanent and discounted reputation. Journal of Information Systems and e-Business Management (2004)
13. Giovannucci, A.: Computationally manageable combinatorial auctions for supply chain automation. PhD thesis, Universitat Autonoma de Barcelona, Artificial Intelligence Research Institute (IIIA) Spain (2008)
14. Yang, Z., Zhang, D., Xu, J.: The simulation of service supply chain formation based on mobile agent's searching. In: IEEE International Conference on E-Commerce Technology for Dynamic E-Business. (2004) 175–178
15. Marsh, S., La, F.: Trust and reliance in multi-agent systems: a preliminary report. In: Proceedings of the 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World. (1992)
16. Stadtler, H., Kilger, C., eds.: Supply Chain Management and Advanced Planning. Springer-Verlag (2000)
17. Norman, T., Preece, A., Chalmers, S., Jennings, N., Luck, M., Dang, V., Nguyen, T., Deora, V., Shao, J., et al., W.G.: Agent-based formation of virtual organisations. Knowledge-Based Systems **17**(2-4) (2004) 103–111
18. Gaonkar, R., Viswanadham, N.: Strategic sourcing and collaborative planning in internet-enabled supply chain networks producing multigeneration products. IEEE Transactions on Automation Science and Engineering **2**(1) (2005) 54–66
19. Padovan, B., Sackmann, S., Eymann, T., Pippow, I.: A prototype for an agent-based secure electronic marketplace including reputation tracking mechanisms. International Journal of Electronic Commerce **6**(4) (2002) 93–113
20. Stockheim, T., Wendt, O., Schwind, M.: A trust-based negotiation mechanism for decentralized economic scheduling. In: Proceedings of the 38th Hawaii International Conference on System Sciences. (2005)

Implementing Collective Obligations in Human-Agent Teams using KAoS Policies

Jurriaan van Diggelen[1], Jeffrey M. Bradshaw[2], Matthew Johnson[2], Andrzej Uszok[2], Paul Feltovich[2]

[1]Institute of Information and Computing Sciences
Utrecht University, the Netherlands
[2]Florida Institute for Human and Machine Cognition (IHMC),
40 S. Alcaniz, Pensacola, FL 32502, USA

jurriaan@cs.uu.nl; {jbradshaw,mjohnson,auszok,pfeltovich}@ihmc.us

Abstract. Obligations can apply to individuals, either severally or collectively. When applied severally, each individual or member of a team is independently responsible to fulfill the obligation. When applied collectively, it is the group as a whole that becomes responsible, with individual members sharing the obligation. In this paper, we present several variations of teamwork models involving the performance of collective obligations. Some of these rely heavily on a leader to ensure effective teamwork, whereas others leave much room for member autonomy. We strongly focus on the implementation of such models. We demonstrate how KAoS policies can be used to establish desired forms of cooperation through regulation of agent behavior. Some of these policies concern invariant aspects of teamwork, such as how to behave when a leader is present, how to ensure that actions are properly coordinated, and how to delegate actions. Other policies can be enabled or disabled to regulate the degree of autonomy of the team members. We have implemented a prototype of a Mars-mission scenario that demonstrates varying results when applied across these different teamwork models.

Keywords: Human-agent teams, Policies, Collective Obligations

1. Introduction

Autonomy is perhaps the most fundamental property of an agent. Generally speaking, we might say that the more control an agent has over its own actions and internal state, the greater its autonomy. By this definition, collaboration almost always entails a reduction in autonomy. In collaboration, we are willing to give up some degree of autonomy in the service of achieving joint objectives [15].

Obligations can be either voluntarily adopted or imposed. Researchers who study *norms* generally focus on the ways in which agents learn, recognize, and adopt such obligations through their own deliberation, including the consideration of incentives and sanctions [5]. Our research interest has been to understand similar issues with

respect to *policies*, constraints that are imposed and enforced prescriptively on agents [2]. Constraining an agent's collaborative activities in this way is often accomplished by virtue of the organizations to which it belongs [7][13]. The purpose of this paper is to report on the latest developments within the KAoS policy and services framework, in particular w.r.t. teamwork and collective obligations.

A KAoS policy is defined as "an enforceable, well-specified constraint on the performance of a machine-executable action by a subject in a given situation" [2]. There are two main types of polices; authorizations and obligations. Authorization policies specify which actions are permitted (*positive authorizations*) or forbidden (*negative authorizations*) in a given situation. Obligation policies specify which actions are required (*positive obligations*) or waived (*negative obligations*) in a given situation. KAoS uses OWL (Web Ontology Language: http://www.w3.org/2004/OWL) to represent policies.

KAoS policies have already been successfully applied to important aspects of joint activity in the context of human-robot teamwork [11]. In this paper, we extend this research by adding the notion of a *collective obligation* [4]. The difference between an individual obligation (IO) and a collective obligation (CO) is that in IO's each individual or member of a team is independently responsible to fulfill the obligation. On the other hand, in CO's, it is the group as a whole that becomes responsible, with individual members sharing the obligation. CO's are especially useful in governing complex abstract behavior—in our case, for example, the obligation that agents have to ensure safety. The difficulty of writing individual obligations for *ensure-safety* is that it is probably not an action that can be directly executed by any one agent. Most likely, a plan must be created to decompose *ensure-safety* into more concrete actions. It is also difficult to decide, beforehand, who is the best candidate to carry out the plan, as a different plan might be adopted in different circumstances. Moreover, agents may have different capabilities, enabling them to contribute individually or jointly in particular roles. For such reasons, constraints requiring the performance of abstract team actions like *ensure-safety* are usually better implemented as collective, as opposed to individual, obligations.

Because a CO often does not direct activity at the level of the single agent's behavior, we must find a way to translate the CO to the individual level. Our research aim in this paper can thus be described: to develop general policies to fulfill collective obligations, and to map these obligations to individuals based on the current context.

Inspired by previous theoretical groundwork on these issues [4][12], we follow a very practical approach. First, we demonstrate how to represent and reason about collective obligations in OWL. Second, we describe three sets of KAoS policies that we defined to govern agent behavior in the execution of collective obligations. Third, we provide a configuration policy set that is used to adjust specific aspects of the teamwork model for use in a given situation. Finally, we present a prototype we have implemented to demonstrate the use of these policies in the context of a Mars mission scenario [16].

We claim several benefits for developers of agent teams. The first concerns *reusability*. Because the policies describe near-universal teamwork aspects, they are domain independent and can apply to many kinds of applications, thus saving development time. The second benefit concerns *sharedness*. Because teamwork requires maintaining common ground among the participants [15], agents benefit

when the code that generates team behavior can be shared by all agents. By introducing a shared collection of teamwork policies for the whole system, in conjunction with KAoS monitoring and enforcement capabilities, newly added agents fit easily into the team, no matter who developed them or which language they are programmed in. The policies accommodate even the most primitive agents by eliminating the requirement that each agent be capable of sophisticated deliberation in order to collaborate. Next, there is the benefit of separation of concerns. By using KAoS policies, the code that implements teamwork is cleanly segregated from the rest of the agent code. This avoids the typical clutter experienced when teamwork code is scattered in arbitrary locations among all agents. Finally, KAoS policies are very straightforward to read and understand, making them more suitable to implement this kind of behavior than generic rule languages or more low-level programming languages.

In addition to the benefits for agent developers, we also believe that this approach is more conducive to scientific progress towards the much more ambitious goal of human and machine joint activity [18][8]. Although the policies described in this paper are relatively simple and elementary, they are fundamental in human teamwork. Hence, when agents adopt important aspects of human teamwork, people may find them more predictable and understandable.

The remainder of the paper is outlined as follows. Section 2 explains the basic teamwork model. Section 3 provides an overview of the KAoS policy services framework. In Sections 4, 5 and 6, we describe how we used KAoS to implement the teamwork model: ontological aspects in Section 4; policies in Section 5; an implemented prototype with agents in a Mars-mission scenario in Section 6. Related work is discussed in Section 7, followed by conclusions in Section 8.

2. Team Design

Teamwork is a topic of great complexity and breadth. Here, our focus is only on one aspect of teamwork, i.e., collective obligations. Collective obligations require teams to perform some action whenever some event or state triggers the obligation. Performing such actions typically involves planning, delegation and coordination. The aim of team design is to ensure that this process is adequately supported. Three primary aspects of team design are pertinent to the issues discussed in this paper: leadership assumption, task allocation, and plan coordination. Each of these aspects can vary, resulting in different team behavior. Figure 1 depicts these aspects in three dimensions, where each combination of aspects represents a different kind of team.

Along the x-axis, two possibilities for leadership assumption are shown. We can appoint someone as a leader beforehand (i.e. pre-established leadership), or we can defer the choice and allow leaders to volunteer on demand (i.e. *ad hoc* leadership assumption). Whereas "pre-established" and "*ad hoc*" qualify as two extremes on the leadership assumption dimension, there are, of course, intermediate options possible that we do not consider here. One example is that of a predefined line of succession which is used to determine leadership if all higher-ranking leaders are unavailable.

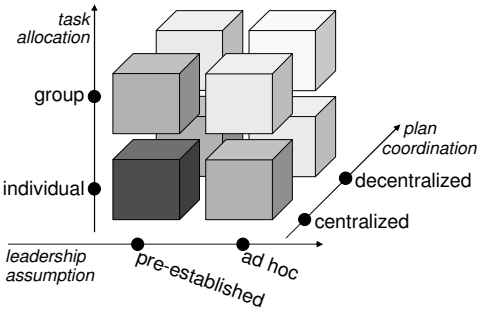


Figure 1 Three dimensions in team design

The task allocation dimension is shown along the y-axis. Individual task allocation means that requests are directed at individual agents. In group task allocation, the request is directed to the group as a whole, without specifying which individual must perform the task.

Plan coordination is depicted on the z-axis, with the two alternatives being centralized and decentralized. Figure 2 depicts the communication pattern for these two ways of coordinating plans. The left side of the figure depicts centralized coordination, i.e. the requester agent (the grey agent) is responsible for making sure that the actions are executed in the right order. The right side of the figure shows decentralized coordination, i.e. the agents executing the plan take care of the coordination themselves. In the latter case, the requester delegates plan coordination. It may do so by sending a request for action *a,* together with information about who will perform the subsequent action *b*. In the figure, this is written as "creq a,b."

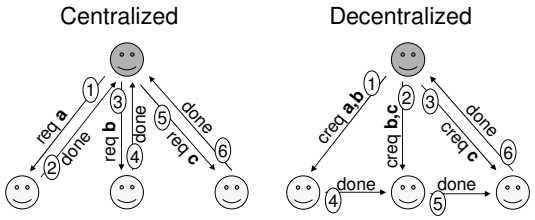


Figure 2 Centralized and decentralized coordination patterns

With centralized coordination, the requested agents may not be aware that their actions are part of a larger plan. With decentralized coordination, the requested agents require more knowledge about the action's context, i.e. they must know which agent is responsible for performing the next action in the plan.

2.1 Considerations for team design

The three dimensions outlined above can be regarded as different aspects of the dichotomy between *central authority* and *member autonomy* [3]. Pre-established leadership means that one central authority remains in charge of the team, whereas *ad hoc* leadership allows for more member autonomy because each team member may become a leader under certain circumstances. Centralized plan coordination allocates

the task of coordinating plans to one central authority, whereas decentralized plan coordination allows each agent to make its contribution to coordination, i.e. reflecting more member autonomy. Individual task allocation implies that one central authority decides who performs the tasks; whereas group task allocation yields more member autonomy as the team members decide this among themselves.

In Figure 1, the team with most central authority is represented as the black cube. For the other teams, we can say that the further away the cube is from the black cube, the more member autonomy exists in the team. The white cube represents the team with most member autonomy. Which of these eight team configurations is the best one depends on the circumstances and cannot be decided in general. Below, we outline some general considerations when choosing between central authority and member autonomy; the discussion is not intended to be exhaustive. An advantage of using a central authority might be that it allows the team designer to select the best agent for the most important tasks. In this way, the team can be better adapted to the different qualities of agents. Another advantage of a central authority approach might be accountability: that is, that it would be easier to identify the responsible agent when things go wrong.

A disadvantage of a central authority might be that it would be less robust in certain circumstances, e.g., when the leader becomes unavailable, the entire team becomes dysfunctional. Another disadvantage of central authority might arise when not every team member has the same access to the situation. For example, it may be better to have a crisis operation led by someone on site than by a predefined leader who is far away. As a last disadvantage, we mention the potentially increased response time of strongly hierarchical teams. For example, when an incident happens, this must communicated all the way up to a leader, after which the leader makes a decision and communicates it all the way down to those carrying out the work. A faster response may be obtained by allowing the observer of the incident to take immediate action.

Before we explain how these teamwork models can be implemented, we will first give some background on the KAoS policy framework.

3. KAOS POLICY FRAMEWORK

KAoS [2] provides a general framework for regulation of a variety of systems, including agent-based and robotic systems [2], web services, grid services, and traditional distributed systems. It also provides the basic services for distributed computing, including message transport and directory services, as well as more advanced features like domain and policy services.

Two important requirements for the KAoS architecture are modularity and extensibility. These requirements are supported through a framework with well-defined interfaces that can be extended, if necessary, with the components required to support application-specific policies. The basic elements of the KAoS architecture are shown in Figure 3; its three layers of functionality correspond to three different policy representations.

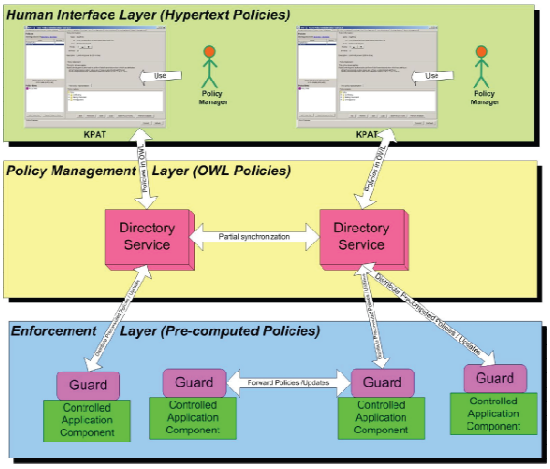


Figure 3 Notional KAoS Policy Services Architecture

- *Human Interface layer:* This layer uses a hypertext-like graphical interface for policy specification in the form of very natural English sentences, composed from pop-up menus. The vocabulary is automatically provided from the relevant ontologies, consisting of highly reusable core concepts augmented by application-specific ones.
- *Policy Management layer:* Within this layer, OWL is used to encode and manage policy-related information. The Distributed Directory Service (DDS) encapsulates a set of OWL reasoning mechanisms.
- *Policy Monitoring and Enforcement layer*: KAoS automatically "compiles" OWL policies to an efficient format that can be used for monitoring and enforcement. This representation provides the grounding for abstract ontology terms, connecting them to the instances in the runtime environment and to other policy-related information.

Maintaining consistency among these layers is handled automatically by KAoS.

3.1 System development in KAoS

Multi-agent system development in KAoS takes place at different locations, in different languages, using different tools, as summarized in the following table.

	Language	**Development Tool**
Agents or other Applications	E.g., Java	E.g., Eclipse
Policies	KAoS Policies (OWL)	KPAT
Ontologies	OWL	E.g., Protégé, COE

Figure 4 KAoS system development components

OWL ontologies provide the vocabulary used in specifying policies. They define all actions, action properties, and actor types and can be developed directly in OWL

or using an ontology editor, such as Protégé (http://protege.stanford.edu/) or COE (Cmap Ontology Editor).

Policies are also represented in OWL. They can be created using the KAoS Policy Administration Tool (KPAT). KPAT hides the complexity of OWL from the human users and allows the user to create, modify and manage policies in a very natural hypertext interface. Policies can be ranked in terms of their *priorities.* In case two conflicting policies are applicable at the same moment, the policy with the highest priority takes precedence.

The policies are used to govern the actions of agents (or other applications) within the system being developed. We use Java and Eclipse (http://www.eclipse.org/) to implement the agents for our prototype, although any other combination of a programming language and IDE could be used. KAoS includes a number of features that can be exploited in the development of agent-based systems.

As an example of system development in KAoS, suppose that we have a set of robots and we want to obligate them to beep before they move, in order to alert any nearby people of the pending movement. First, we would specify the terms `Robot`, `Beep` and `Move` in an ontology. Then, we would create a policy using KPAT, which would look like the following:

```
1 Robot is obligated to start performing Beep
2        which has any attributes
3 before Robot starts performing Move
4        which has any attributes
```

Figure 5 KAoS policy example

Once the policy has been created, it is sent by KPAT to the Directory Service for analysis and deconflication, before it is ”compiled” and distributed to the guards for run-time enforcement. Since the policy applies only to robots, it is automatically distributed only to the guards responsible for governing robots. Local enforcement mechanisms on each platform intercept movements as appropriate and check with the guard resident on that platform for policy constraints. With the new policy in place, an obligation to beep would be applied prior to each movement.

An important part of building systems in this way is deciding where to implement a given behavior. In general, there are three possible places; in the agent, in the policies, or in the ontology (cf. Figure 4). Each has advantages and disadvantages in different situations. Without policy, we would be forced to represent everything in the agent itself, so, for our beep example, the beep action might simply be coded in Java within the move method. This is not very flexible and is hidden from those unfamiliar with the code. In situations where the source code is unavailable, it simply cannot be implemented at all. A second option is to implement the behavior by adapting the ontology, i.e. by defining a move as a beep that is followed by a physical move, and having the agents query the ontology for the definition of the action. This would amount to redefining the commonly accepted meaning of *move* into something else entirely – not be a good idea either. The third option is to add the policy of Figure 5. This seems to us the cleanest method. The policy is defined external to the robot’s program and thus is viewable and editable by anyone using the system. To give an example that pushes some knowledge back into the robots, suppose that we modify our policy to state “robots must warn before they move.” The main idea is still

modeled in policy, though less specific. The ontology could be used to model the knowledge that beeping and flashing lights are both appropriate methods to warn. Finally, the robot could chose the appropriate warning method based on its own capabilities and preferences.

In the following three sections we will explain how the teamwork model described in Section 3 can be implemented by developing ontologies, policies, and agents.

4. ONTOLOGY

Extending KAoS so it can handle collective obligations posed some additional requirements to the core ontology. The first issue concerned the representation of teams. The property `teamMemberOf` was used to assert that an agent (represented by an individual in class `agent`) is a member of some team (represented by an individual in class `team`). To represent the collective obligation of a team, the property `HasCollectiveObligation` was used to refer to the instance representation[1] of the action that constitutes the CO.

The second issue concerned the representation of plans. Because a plan typically consists of multiple actions, we can represent that an action contains subactions by using the properties `subAction1` and `subAction2`. The property `subActionRelation` specifies whether the two subactions are composed in parallel or sequence. In this way, composite actions can be represented as an AND-OR graph, or planning tree [17].

The last ontological issue concerned the relationship between the plan and the action the plan seeks to achieve. Because different circumstances require different plans, we specify this as a context-dependent relation, using a rule of the form "X *counts-as* Y in context C." These so-called *counts-as* rules can be used in an ontology to translate between actions of different levels of abstraction [11]. For example, the sequence of actions *bring-to-habitat* and *nurse* (the plan) counts as *ensure-safety* (what the plan is designed to achieve) in the context of *spacesuit-failure-of-Benny-at-11:00am* (the context). An action and its associated context are related by the property `hasContext`. To represent the fact that an action has been performed, the property `hasStatus` is set to `performed`. Because we represent *counts-as* rules as subclass relations (e.g. "X subClassOf Y" represents the fact that X counts as Y), the OWL reasoner automatically derives that if X `hasStatus performed`, then also Y `hasStatus performed`.

The issues discussed above are important when monitoring policy compliance. An agent complies with an obligation to do action X, if X has the status performed before the deadline set by the obligation. This definition has two important consequences. First, the agent to which the obligation applies is not required to perform the action itself, but may also delegate the action to another agent. Second, the agent can choose to perform a plan which counts as action X (in the current context), because performance of the plan entails performance of X. Both of these two issues play a fundamental role in our approach to teamwork and are therefore implemented at the ontology level.

[1] Because OWL-DL does not allow the use of classes as property values, we created a prototypical instance for every action class (e.g. `ensureSafety`). This prototypical instance represents the same (e.g. `ensureSafetyPrototypicalInstance`). In this way, we can refer to actions both at the class level, and at the instance level.

```
1 Leader is obligated to start performing Action which has attributes:
2  all prototypicalInstance values equal the Trigger action's
3    triggerOfCollectiveObligation of the prototypicalInstance values
4  the performedBy value equals the Trigger action's performedBy values
5 after Leader finishes performing Action which has attributes:
6  any prototypicalInstance values are in the set of this action's
7    HasCollectiveObligationTrigger of the teamMemberOf of the
8    performedBy values
```

Figure 6 KAoS hypertext statement representing the policy of Definition 1.1

5. POLICIES FOR AGENT TEAMS

The general pattern of the teamwork described in this paper consists of three steps. First, the collective obligation is triggered. Second, a plan is created. Third, this plan is carried out. The policies described in this section serve to support this process by governing issues such as: how is the CO-trigger communicated to the agent creating the plan? Who creates the plan? Who carries out the plan? How is the plan coordinated to ensure the right order of actions?

5.1 Leader Policy Set

If there is a team leader, it has a special responsibility and must be treated by the other agents in a distinct way. The purpose of the Leader Policy Set is to lay down these responsibilities, managing both task allocation and plan coordination.

Definition 1 Leader Policy Set

1. *The leader of a team should adopt the collective obligations of its team as its own individual obligations*
2. *Team members should notify their leader when the collective obligation of their team is triggered*
3. *The leader of a team may request members of its team to perform actions*
4. *The leader of a team may create plans*

The first policy captures the intuition that leaders must take responsibility for their team. Definition 1.1 states more precisely what this means for collective obligations. The policy as implemented in KAoS is shown in Figure 6. The trigger of the policy is implemented at line 5,6,7 and 8 using a role-value map [1] which compares the values of two properties of the `Action` which the agent has just finished performing. It states that the property `prototypicalInstance` must have a value in common with the concatenation of the properties `performedBy`, `teamMemberOf` and `HasCollectiveObligationTrigger`. As an example of an action that would trigger the obligation, consider agent `Herman` performing the action `observeSpaceSuitFailure` (i.e. `observeSpaceSuitFailure performedBy Herman`) and that `Herman` is `teamMemberOf MecaTeam` and that `MecaTeam HasCollectiveObligationTrigger observeSpaceSuitFailurePrototypicalInstance`. The obligation is described in lines 1, 2, 3 and 4 of Figure 6. Lines 2-3 is a role-value map which describes that the actor must do the action which is given by the property `triggerOfCollectiveObligation` of the action that triggered the obligation. In our example, `observeSpaceSuitFailure` is `triggerOfCollectiveObligation` of

`ensureSafety`. Hence, the actor is obliged to perform `ensureSafety`. Line 4 describes that the agent that must fulfill the obligation is the same agent that has triggered the obligation.

The second policy of Definition 1 ensures that, in case nobody else in the team triggers the collective obligation (for example by observing a spacesuit failure), this agent will notify the leader about the event. This captures the intuition that team members must help their leader. This policy is implemented in a similar fashion to policy 1.1 (Figure 6).

The third policy in the leader policy set states that leaders do not have to do the work all by themselves, but they are authorized to request actions from their team members.

The fourth policy states that the leader is authorized to create a plan. Plan creation is done by adding a *counts-as* rule to the ontology (see Section 4). The effect of this is that all agents may perform a different action than the action they were initially obliged to do. Therefore, the right to create new plans is not self-evident. It is, however, a right that belongs to a leader.

5.2 Coordination Policy Set

The coordination policy set describes how actions in a plan should be coordinated. We consider two coordination patterns (as depicted in Figure 2), which are both governed by this policy set.

Definition 2 Coordination Policy Set

1. *An agent should notify the requester after it has performed a requested action*
2. *If the agent knows who will perform the subsequent action, it should notify that agent after it finishes performing its own action*
3. *If the agent knows who will conduct the subsequent action, it is not required to notify the requester after it finishes performing its action*

The first policy ensures that, in case of centralized coordination, the requester knows when the subsequent action may begin. This is due to the "done" messages 2, 4 and 6 on the left side of Figure 2. In case of decentralized coordination, the requester is notified after the plan is finished, i.e. by "done" message 6 on the right side of the figure.

The second policy of Definition 2 concerns the case of decentralized coordination. When an agent has received a request for a coordinated action, it knows who will perform the subsequent action, and must notify that agent after it has finished its action.

The third policy is enforced with high priority, and can be regarded as an exception to the first policy of Definition 2. This policy prevents requested agents from notifying their requester when the plan is only partially completed. As can be seen on the right hand side of Figure 2, the two agents that are requested to perform action *a* and action *b* of the plan do not send a "done" message to their requester. The rationale behind this is that, in the decentralized case, partially-finished notifications are not needed for *plan coordination*, which is the purpose of this policy set. There may be other reasons why this may be desirable, e.g., to monitor plan progress to respond to unexpected events in a timely way [8]. This can always be implemented in

an additional higher priority policy set, which is specially designed for that purpose. However, issues such as dealing with plan failure or replanning are issues of future research.

5.3 Leader Absence Policy Set

What if the agents find themselves in a leaderless team? This may happen either because nobody has been appointed as a leader or else the leader is (temporarily) unavailable. In this case, the other agents in the team must take care of the collective obligation themselves. This issue is handled by ensuring that one agent assumes the leader role, and thereby becomes subject to the leadership policies of Definition 1.

Definition 3 Leader Absence Policy Set

1. *When no leader is present, the CO is triggered, and the agent knows it can fulfill the CO, it should assume the leader role*
2. *When no leader is present, the CO is triggered, but the agent cannot fulfill the CO, it should notify the whole team of the CO trigger*
3. *An agent should not notify its team about a CO trigger, when it has been notified itself by another team member about that CO trigger*

The first policy ensures that a capable leader will volunteer in case the collective obligation is triggered in a leaderless team. An agent may assume leadership by registering with the KAoS directory-service, which only accepts such a registration when there are no other leaders already currently available. In this way, we prevent multiple agents from taking leadership at the same time, on a first come, first served basis.

The second policy is a variation on the policy of Definition 1.2, adapted to the leaderless scenario. For example, when an agent observes a safety critical event (the CO is triggered), but the agent is not capable of ensuring safety, the agent should notify all of its team members about it, so someone else in the team can fulfill the CO.

The third policy is an exception to the second rule, and prevents agents from repeatedly notifying one another about the same collective obligation trigger.

5.4 Configuration Policy Set

The policies discussed so far are the same for all eight different kinds of teams depicted in Figure 1. In this section, we will discuss the configuration policy set which states which of the eight team strategies the agents must follow.

Definition 4 Configuration Policy Set

1. *Do not request distributed coordinated actions*
2. *Do not request actions to a team*

In contrast to the policy sets we discussed earlier, these policies are optional, and can be switched on and off depending on the way the team designer wishes to configure the team. If the first policy is switched on, the team will apply centralized plan coordination. If it is switched off, the team will apply decentralized plan coordination.

If the second policy is switched on, the team will apply individual task allocation. If it is switched off, the team applies group task allocation. Group task allocation can be implemented using collective obligations that are dealt with using the policies described in the previous sections. For example, to request action *a* to a group, the action *a* is added as a collective obligation to that group. The leader absence policy set (Definition 3) ensures that a leader which is capable of performing action *a* stands up, after which the leader policy set (Definition 1) ensures that this agent performs action *a*.

To implement pre-established leadership assumption, a leader must be appointed beforehand, using KPAT. To implement *ad hoc* leadership assumption, no leader should be defined beforehand, such that the policy in Definition 3.1 ensures that a leader will volunteer at runtime if needed.

6. MECA SCENARIO

We tested the policies using a Mars mission scenario developed in the Mission Execution Crew Assistant (MECA) project [16]. This long-term project aims at enhancing the cognitive capacities of human-machine teams during planetary exploration missions by means of an electronic partner. The e-partner helps the crew to assess a situation and determine a suitable course of actions when problems arise. A large part of the project is devoted to developing a requirements baseline, taking into account human factors knowledge, operational demands, and envisioned technology. Developing new prototypes using emerging technologies, such as this one, is a continuous activity in the project.

One of the major themes is dealing with the long communication delays between Earth and Mars. This has led researchers to consider new forms of mission control that are less centralized on Earth, allowing greater autonomy to the astronauts on Mars [10]. We believe that our work on policies and team strategies is a useful contribution to this problem.

One of the use-cases that has driven the development of MECA's requirements baseline concerns an astronaut suffering from hypothermia. The initial situation is depicted in Figure 7.

Herman is in the Habitat; Anne, Albert and two rovers are in team A; Benny and Brenda are in team B. Benny and Brenda are on a rock-collecting procedure. Suddenly, Benny's space suit fails. Brenda and the MECA system diagnose the problem together and predict hypothermia. Immediate action is required. A rover from team B comes to pick Benny up and brings him to the habitat. Someone with surgery skills and someone with nursing skills await him there and take care of Benny, after which he safely recovers.

One of the requirements of MECA is that safety of the crew must be ensured at all times. We implemented this requirement using a collective obligation of the MECA team to *EnsureSafety*. The trigger of this collective obligation is *Observe-SafetyCriticalEvent.* Within the scenario, both of these actions are added in a specific MECA-action ontology which extends the KAoS core action ontology. The ontology also specifies several subconcepts of *ObserveSafetyCriticalEvent*, such as *Observe-SpaceSuitFails.* This causes *ObserveSpaceSuitFails* to trigger the collective obligation.

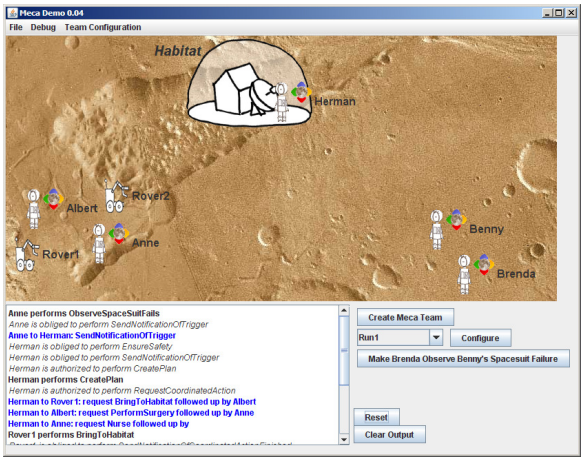


Figure 7 MECA prototype

The seven agents in the example (five astronauts and two rovers) are implemented in Java. Because most of the agent behavior in this demonstration is implemented by the policies, the Java implementation could remain very simple. We used Java to implement how the actions, such as *BringToHabitat*, are performed. For the purposes of this demonstration, a simple screen animation was sufficient. We also implemented in Java how the agents remain policy-compliant. This means that they consult the KAoS guard to check which obligations and authorization policies apply. They fulfill an obligation by simply executing the code that implements the action concerned. It fulfills a negative authorization by refraining from executing the corresponding piece of code.

The most important aspect of this demonstration is the unfolding of the scenario after the action *ObserveSpaceSuitFails* is performed. This is driven exclusively by KAoS policies. By applying the different team configurations described in Section 0, we obtain different event traces which demonstrate the functioning of the team. The event trace for the most centrally organized team (represented by the black cube in Figure 1) is shown below.

Brenda performs ObserveSpaceSuitFails
Brenda is obliged to perform SendNotificationOfTrigger
Brenda to Herman: SendNotificationOfTrigger
Herman is obliged to perform EnsureSafety
Herman is authorized to perform CreatePlan
Herman performs CreatePlan
Herman is not authorized to perform RequestCoordinatedAction
Herman is authorized to perform RequestAction
Herman to Rover1: request BringToHabitat
Rover1 performs BringToHabitat
Rover1 is obliged to perform SendNotificationOfRequestedActionFinished
Rover1 to Herman: SendNotificationOfRequestedActionFinished
Herman to Albert: request PerformSurgery
Albert performs PerformSurgery
Albert is obliged to perform SendNotificationOfRequestedActionFinished
Albert to Herman: SendNotificationOfRequestedActionFinished
Herman to Anne: request Nurse

Anne performs Nurse
Anne is obliged to perform SendNotificationOfRequestedActionFinished
Anne to Herman: SendNotificationOfRequestedActionFinished

Figure 8 Event trace of MECA team with maximal central authority

The events printed in bold are actions; the underlined events are communication actions; the italicized events represent policies that were triggered. Typical to this event trace is that Brenda immediately knows that she must contact Herman after she observed the spacesuit failure. This is due to the pre-established leadership of Herman. Furthermore, Herman delegates the parts of the plan to individual agents (i.e. individual task allocation), and he waits until the requested agent is finished before he requests the next action in the plan (i.e. centralized plan coordination).

The event trace for the team with most member autonomy (represented by the white cube in Figure 1) is shown in Figure 9.

Brenda performs ObserveSpaceSuitFails
Brenda is obliged to perform SendNotificationOfTrigger
Brenda to Rover1: SendNotificationOfTrigger
Brenda to Anne: SendNotificationOfTrigger
Brenda to Albert: SendNotificationOfTrigger
Brenda to Rover2: SendNotificationOfTrigger
Brenda to Herman: SendNotificationOfTrigger
Brenda to Benny: SendNotificationOfTrigger
Anne is obliged to perform AssumeLeaderRole
Anne is obliged to perform EnsureSafety
Anne is authorized to perform CreatePlan
Anne performs CreatePlan
Anne is authorized to perform RequestCoordinatedAction
Anne is authorized to perform TeamRequestAction
Anne to MecaTeam: request BringToHabitat
Anne to MecaTeam: request PerformSurgery after BringToHabitat
Anne to MecaTeam: request Nurse after PerformSurgery
Rover1 is obliged to perform AssumeLeaderRole
Rover1 performs BringToHabitat
Rover1 is obliged to perform SendNotificationOfTrigger
Rover1 to MecaTeam: SendNotificationOfTrigger
Albert is obliged to perform AssumeLeaderRole
Albert performs PerformSurgery
Albert is obliged to perform SendNotificationOfTrigger
Albert to MecaTeam: SendNotificationOfTrigger
Herman is obliged to perform AssumeLeaderRole
Herman performs Nurse

Figure 9 Event trace of MECA team with maximal member autonomy

Typical to this event trace is that Brenda notifies the whole team about the CO trigger, after which Anne becomes a leader (i.e. *ad hoc* leadership assumption). Furthermore, Anne delegates her actions to the MECA team (i.e., group task allocation). Also, she delegates all actions at once and instructs the agents how to coordinate the actions (i.e., decentralized plan coordination).

7. RELATED WORK

A similar approach to teamwork, based on electronic institutions, is reported in [9]. This framework captures coordination aspects by dynamically composing existing teamwork components, s.a. communication protocols and operational descriptions, to meet the current problem requirements. Our approach is more centered around the

idea of constraining autonomy, i.e. by using computational policies as basic teamwork components.

The pioneering research of Cohen and Levesque [4] introduced the notion of a *joint persistent goal* as the ultimate driving force behind teamwork. In our framework, a collective obligation serves a similar purpose. A difference is that Cohen and Levesque based their approach on mentalistic notions, such as goals, beliefs and intentions, whereas our approach is based on institutional notions, such as obligations and authorizations. This allows the approach to be used by both simple and sophisticated agents, of heterogeneous varieties.

A similar difference can be observed when comparing our implementation with other teamwork model implementations, such as STEAM [1]. STEAM is based on Soar, a general cognitive architecture for intelligent systems, whereas our approach is based on KAoS, which is a policy framework. A correspondence between our implementation and STEAM is that both approaches heavily rely on plans in the teamwork process. A crucial requirement for effective teamwork is maintaining a sufficient level of common ground [15]. By adopting the KAoS framework, some important aspects of common ground were naturally ensured. The common ontology, which is maintained by the directory service and distributed to the guards, ensures that every agent shares understanding of the domain terms. Also the collective obligations of the team, which are represented in the ontology, are mutually known.

8. CONCLUSION

In this paper, we have proposed a policy-based approach for human-agent teams. We have implemented a variety of teamwork models in KAoS. These models have demonstrated their value in a simulation of a Mars-mission scenario, where a delicate decision must be made between central authority and member autonomy.

We believe that our approach to teamwork has considerable benefits in terms of reusability, clarity, and generality. Although the types of teamwork we support are still elementary, we believe that more complex teamwork can be implemented by utilizing additional policies on top of the policies we have proposed here.

In the future, we plan to extend the teamwork model to deal with unexpected events. This requires a leader to monitor his or her plan, and to perform replanning if the plan does not go as expected. Also, the team members can be of help here by notifying their leaders when their requested actions fail (cf. [8]). Such policies can be implemented in KAoS, in a similar fashion as we have described in this paper.

REFERENCES

[1] Baader, F., Calvanese D., McGuinness, D. L., Nardi D., and Patel-Schneider, P. F. Eds. (2003). The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press.

[2] Bradshaw, J. M., et al. (2003). Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads. Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS), ACM.

[3] R.M. Burton, G. DeSanctis, B. Obel (2006), Organizational Design, Cambridge University Press.

[4] Cohen, P.R. and H.J. Levesque. (1991). Teamwork. Menlo Park,CA: SRI International.

[5] Davidsson, P. (2000): Emergent Societies of Information Agents. Klusch, M, Kerschberg, L. (Eds.): *Cooperative Information Agents IV*, LNAI 1860, Springer, 2000, pp. 143–153.
[6] Dignum, F. and Royakkers, L. (1998). Collective Obligation and Commitment, In Proceedings of 5th Int. conference on Law in the Information Society, Florence
[7] Dignum, V. (2003). A Model for Organizational Interaction. SIKS Dissertation Series.
[8] Feltovich, P.J., Bradshaw, J.M., Clancey, W.J., Johnson, M., & Bunch, L (2008). Progress appraisal as a challenging element of coordination in human and machine joint activity. In Engineering Societies in the Agents' World VIII. Lecture Notes in Computer Science Series. Heidelberg Germany: Springer.
[9] Gómez, Mario; Plaza, Enric (2008). Dynamic Composition of Electronic Institutions for Teamwork.Coordination, Organizations, Institutions, and Norms in Agent Systems III. (COIN)., LNAI, Vol. 4870, pp. 155-170. Springer Verlag.
[10] Grant, T., Soler, A. O., Bos, A., Brauer, U., Neerincx, M., and Wolff, M. 2006. Space Autonomy as Migration of Functionality: The Mars Case. In *Proceedings of the 2nd IEEE international Conference on Space Mission Challenges For information Technology* (SMC-IT). IEEE, 195-201.
[11] Grossi, D. (2007). Designing Invisible Handcuffs. Formal Investigations in Institutions and Organizations for Multi-agent Systems. SIKS Dissertation Series 2007-16, Utrecht University.
[12] Grossi, D., Dignum, F., Royakkers L., Meyer, J-J. Ch., (2004). Collective Obligations and Agents: Who Gets the Blame? Proc. of DEON'04, 7th Int. Workshop on Deontic Logic in Computer Science. Springer. LNCS 3065
[13] Hübner, J. F., Sichman, J. S., and Boissier, O., (2002). A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems. In Proc. of the 16th Brazilian Symposium on AI, LNCS, vol. 2507. Springer-Verlag, London, 118-128.
[14] Johnson, M. J., Intlekofer, K., Jr., Jung, H., Bradshaw, J. M., Allen, J. Suri, N. & Carvalho, M., (2008). Coordinated operations in mixed teams of humans and robots. In Proceedings of the 2008 IEEE International Conference on Distributed Human-Machine Systems (DHMS 2008)., pp. 63-68.
[15] Klein, G. Woods, D., Bradshaw, J.M. Hoffman, R.R. , Feltovich, P.J. (2004). Ten Challenges for Making Automation a Team Player. In Joint Human-Agent Activity," IEEE Intelligent Systems, vol. 19, no. 6
[16] Neerincx, M.A. Bos, A., Olmedo-Soler, A. Brauer, U. Breebaart, L., Smets, N., Lindenberg, J., Grant, T., Wolff, M. (2008). The Mission Execution Crew Assistant: Improving Human-Machine Team Resilience for Long Duration Missions. Proc. of the 59th International Astronautical Congress (IAC2008)
[17] Steffik, M. (1995). Introduction to Knowledge Systems, Morgan Kaufmann Publishers.
[18] Sycara, K., and Lewis, M. 2004. Integrating intelligent agents into human teams. In Team Cognition: Understanding the Factors that Drive Process and Performance,, 203-232. Washington, DC: American Psychological Association.
[19] Tambe, M. (1997). Towards Flexible Teamwork, Journal of Artificial Intelligence Research, pp. 83-124

Monitoring Social Expectations in Second Life

Stephen Cranefield and Guannan Li

Department of Information Science
University of Otago
PO Box 56, Dunedin 9054, New Zealand
scranefield@infoscience.otago.ac.nz

Abstract. Online virtual worlds such as Second Life provide a rich medium for unstructured human interaction in a shared simulated 3D environment. However, many human interactions take place in a structured social context where participants play particular roles and are subject to expectations governing their behaviour, and current virtual worlds do not provide any support for this type of interaction. There is therefore an opportunity to adapt the tools developed in the MAS community for structured social interactions between software agents (inspired by human society) and adapt these for use with the computer-mediated human communication provided by virtual worlds.

This paper describes the application of one such tool for use with Second Life. A model checker for online monitoring of social expectations defined in temporal logic has been integrated with Second Life, allowing users to be notified when their expectations of others have been fulfilled or violated. Avatar actions in the virtual world are detected by a script, encoded as propositions and sent to the model checker, along with the social expectation rules to be monitored. Notifications of expectation fulfilment and violation are returned to the script to be displayed to the user. This utility of this tool is reliant on the ability of the Linden scripting language (LSL) to detect events of significance in the application domain, and a discussion is presented on how a range of monitored structured social scenarios could be realised despite the limitations of LSL.

1 Introduction

Much of the research in multi-agent systems addresses techniques for modelling, constructing and controlling open systems of autonomous agents. These agents are taken to be self-interested or representing self-interested people or organisations, and thus no assumptions can be made about their conformance to the design goals, social conventions or regulations governing the societies in which they participate. Inspired by human society, MAS researchers have adopted, formalised and created computational infrastructure allowing concepts from human society such as trust, reputation, expectation, commitment and narrative to be explicitly modelled and manipulated in order to increase agents' awareness of the social context of their interactions. This awareness helps agents to carry out their interactions efficiently and helps preserve order in the society, e.g. the existence of reputation, recommendation and/or sanction mechanisms discourages anti-social behaviour.

As the new ‘Web 2.0’ style Web sites and applications proliferate, people’s use of the Web is moving from passive information consumption to active information sharing and interaction within virtual communities; in other words, for millions of users, the Web is now a place for social interaction. However, while Web 2.0 applications provide the middleware to enable interaction, they generally provide no support for users to maintain an awareness of the social context of their interactions (other than basic presence information indicating which users in a ‘buddy list’ online). There is therefore an opportunity for the software techniques developed in MAS research for maintaining social awareness to be applied in the context of electronically mediated human interaction, as well as in their original context of software agent interaction.

This paper reports on an investigation into the use of one such social awareness tool in conjunction with the Second Life online virtual world. Second Life is a ‘Web 3D’ application providing a simulated three dimensional environment in which users can move around and interact with other users and simulated objects [1]. Users are represented in the virtual world by animated avatars that they control via the Second Life Viewer client software. Human interaction in virtual worlds is essentially unconstrained—the users can do whatever they like, subject to the artificial physics of the simulated world and a few constraints that the worlds support, such as the ability of land owners to control who can access their land. However, many human interactions take place in a structured social context where participants play particular roles and there are constraints imposed by the social or organisational context, e.g. participants in a meeting should not leave without formally excusing themselves, and students in an in-world lecture should remain quiet until the end of the lecture. Researchers in the field of multi-agent systems have proposed (based on human society) that the violation of social norms such as these can be discouraged by publishing explicit formal definitions of the norms, building tools that track (relevant) events and detect any violations, and punishing offenders by lowering their reputations or sanctioning them in some other way [2]. Integrating this type of tool with virtual worlds could enhance the support provided by those worlds for social activities that are subject to norms.

In this research we have investigated the use of a tool for online monitoring of ‘social expectations’ [3] in conjunction with Second Life. The mechanism involves a script running in Second Life that is configured to detect and record particular events of interest for a given scenario, and to model these as a sequence of state descriptions that are sent to an external monitor along with a property to be monitored. The monitor sends notifications back to the script when the property is satisfied so that the user can be informed. This technique is not intended to provide a global surveillance mechanism for Second Life, but rather, to allow specific users and communities to model and track the social expectations that apply in particular types of structured interaction ocurring within a limited area.

The rest of this paper is structured as follows. Section 2 describes how we have used the Linden Scripting Language to detect avatars in Second Life and create a sequence of propositional state models to send to the monitor. The architecture for communication between this script and the monitor is presented in Section 3. Section 4 discusses the concept of conditional social expectations used in this work, and the model checking tool that is used as the expectation monitor. Section 5 presents some simple scenarios

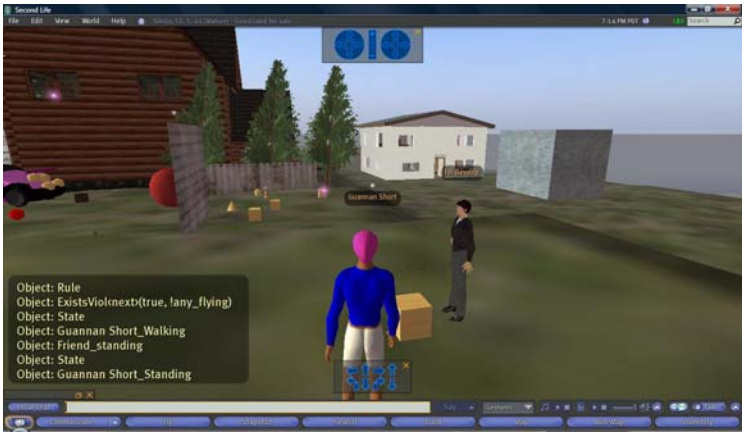


Fig. 1. The Second Life Viewer

of activities in Second Life being monitored, and Section 6 discusses some issues arising from limitations of the Linden Scripting Language and the temporal logic used to express rules. Some related work is described in Section 7, and Section 8 concludes the paper.

2 Detecting events in Second Life

As shown in Figure 1, the Second Life Viewer provides, by default, a graphical view of the user's avatar and other objects and avatars within the view. The user can control the 'camera' to obtain other views. Avatars can be controlled to perform a range of basic animations such as standing, walking and flying, or predefined "gestures" that are combinations of animation, text chat and sounds. Communication with other avatars (and hence their users) is via text chat, private instant messages, or audio streaming. The user experience is therefore a rich multimedia one in which human perception and intelligence is needed to interpret the full stream of incoming data. However, the Linden Scripting Language (LSL [4]) can be used to attach scripts to objects (e.g. to animate doors), and there are a number of sensor functions available to detect objects and events in the environment. These scripts are run within the Second Life servers, but have some limited ability to communicate with the outside world.

LSL is based on a state-event model, and a script consists of defined states and handlers for events that it is programmed to handle. Certain events in the environment automatically trigger events on a script attached to an object. These include collisions with other objects and with the 'land', 'touches' (when a user clicks on the object), and money (in Linden dollars) being given to the object. Some other types of event must be explicitly subscribed to by calling functions such as `llSensor` and `llSensorRepeat` for scanning for avatars and objects within a given arc and range (up to 96 metres), `llListen` for detecting chat messages from objects or avatars within hearing range, and `llSetTimerEvent` for setting a timer. These functions take parameters

that provide some selectivity over what is sensed, e.g. a particular avatar name or object type can be specified in `llListen`, and `llListen` can be set to listen on a particular channel, for a message from a particular avatar, and even for a particular message.

In this paper we focus on the detection of other avatars via the function `llSensorRepeat`, which repeatedly polls for nearby avatars (we choose not to scan for objects also) at an interval specified in a parameter. A series of `sensor` events are then generated, which indicate the number of avatars detected in each sensing operation. A loop is used to get the unique key that identifies each of these avatars (via function `llDetectedKey`) and the avatar's name (via `llDetectedName`). The key can then be used to obtain each avatar's current basic animation (via `llGetAnimation`). Our script can be configured with a filter list specifying which avatar/animation observations should be either recorded or ignored, where the specified avatar and animation can refer to a particular value, or "any". Detected avatar animations are filtered through this list sequentially, resulting in a set of $(avatar\_name, animation)$ pairs that comprise a model of the current state of the avatars within the sensor range. Another configuration list specifies the optional assignment of avatars to named groups or roles such as "Friend" or "ClubOfficial". There is currently no connection with the official Second Life concept of a user group (although official group membership can be detected). Group names can also be included in the filter list, with an intended existential meaning, i.e. a pair $(group\_name, animation)$ represents an observation that *some* member of the group is performing the specified animation. The configuration lists provide scenario-specific relevance criteria on the observed events, and are read from a 'notecard' (a type of avatar inventory item that is commonly used to store textual configuration data for scripts), along with the property to be monitored.

When the script starts up, it sends the property to be monitored to the monitor. It then sends a series of state descriptions to the monitor as sensor events occur. However, we choose not to send a state description if there is no change since the previous state, so states represent periods of unchanging behaviour rather than regularly spaced points in time. State descriptions are sets of proposition symbols of the form *avatar_animation* or *group_animation*.

This process can easily be extended to handle other types of Second Life events that have an obvious translation to propositional (rather than predicate) logic, such as detecting that an avatar has sent a chat message (if it is not required to model the contents of the message). Section 6 discusses this further.

3 Communication between Second Life and the monitor

Second Life provides three mechanisms for communication with entities outside their own server or the Second Life Viewer: scripts can send email messages, initiate HTTP requests, or listen for incoming XML-RPC connections (which must include a parameter giving the key for a channel previously created by the script). To push property and state information to the monitor we use HTTP. However, instead of directly embedding the monitor in an HTTP server, to avoid local firewall restrictions we have chosen to use Twitter [5] as a message channel. An XML-RPC channel key, the property to be monitored and a series of state descriptions are sent to a predefined Twitter account as

direct messages using the HTTP API[1]. The Twitter API requires authentication, which can be achieved from LSL only by including the username and password in the URL in the form http://username:password@…..

The monitor is wrapped by a Java client that polls Twitter (using the Twitter4J library [7]) to retrieve direct messages for the predetermined account. These are ignored until a pair of messages containing an XML-RPC channel key and a property to be monitored (prefixed with "C:" and "P:" respectively) are received, which indicates that a new monitoring session has begun. The monitoring session then consists of a series of messages beginning with "S:", each containing a list of propositions describing a new state. The monitor does not currently work in an incremental 'online' mode—it must be given a complete history of states and restarted each time a new state is received[2]; therefore, the Java wrapper must record the history of states. It also generates a unique name for each state (which the monitor requires).

Each time a state is received, the monitor (which is implemented in C) is invoked using the Java Native Interface (JNI). The rule and state history are written to files and the names passed as command-line arguments. An additional argument indicates the desired name of the output file. The output is parsed and, if the property is determined to be true in any state, that information is sent directly back to the Second Life script via XML-RPC.

Figure 2 gives an overview of the communication architecture.

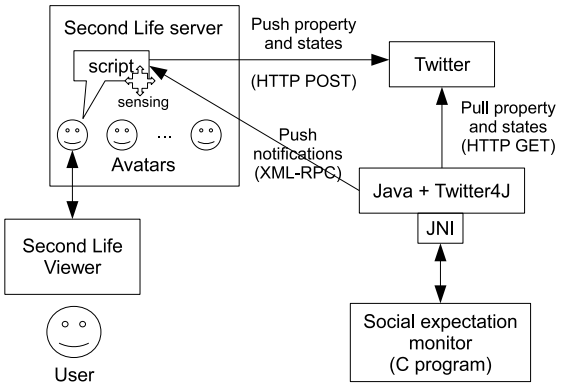


Fig. 2. The communications architecture

[1] Twitter messages are restricted to 140 characters and calls to the Twitter API are subject to a limit of 70 requests per hour, which is sufficient for testing our mechanism. For production use an alternative HTTP-accessible messaging service could be used, such as the Amazon Simple Queue Service [6].

[2] Work is in progress to add an online mode to the monitor.

4 Monitoring social expectations

4.1 Modelling social expectations

MAS researchers working on normative systems and electronic institutions [2] have proposed various languages for modelling the rules governing agent interaction in open societies, including abductive logic programming rules [8], enhanced finite state machine style models, [9], deontic logic [10], and institutional action description languages based using formalisms such as the event calculus [11].

The monitor used in this work is designed to track rules of *social expectation*. These are temporal logic rules that are triggered by conditions on the past and present, resulting in *expectations* on present and future events. The language does not include deontic concepts such as obligation and permission, but it allows the expression of social rules that impose complex temporal constraints on future behaviour, in contrast to the simple deadlines supported by most normative languages. It can also be used to express rules of social interaction that are less authoritative than centrally established norms, e.g. conditional rules of expectation that an agent has established as its personal norms, or rules expressing learned regularities in the patterns of other agents' behaviour. The key distinction between these cases is the process that creates the rules, and how agents react to detected fulfilments and violations.

Expectations become active when their condition evaluates to true in the current state. These expectations are then considered to be fulfilled or violated if they evaluate to true in a state without considering any future states that might be available in the model[3]. If an active expectation is not fulfilled or violated in a given state, then it remains active in the following state, but in a "progressed" form. Formula progression involves partially evaluating the formula in terms of the current state and re-expressing it from the viewpoint of the next state [12]. A detailed explanation is beyond the scope of this paper, but a simple example is that an expectation $\bigcirc\phi$ (meaning that ϕ must be true in the state that follows) progresses to the expectation ϕ in the next state.

4.2 The social expectation monitor

The monitoring tool we have used is an extension [3] of a model checker for hybrid temporal logics [13]. Model checking is the computational process of evaluating whether a formal model of a process, usually modelled as a Kripke structure (a form of nondeterministic finite state machine), satisfies a given property, usually expressed in temporal logic. For monitoring social expectations in an open system, we cannot assume that we can obtain the specifications or code of all participating agents to form our model. Instead our model is the sequence of system states recorded by a particular observer, in other words, we are addressing the problem of *model checking a path* [14]. The task of the model checker is therefore not to check that the overall system *necessarily* satisfies

[3] This restriction is necessary, for example, when examing an audit trail to find violations of triggered rules in *any* state. The standard temporal logic semantics would conclude that an expectation "eventually p" is fulfilled in a state s even if p doesn't become true until some later state s'.

a given property, but just that the observed behaviour of the system has, to date, satisfied it. The properties we use are assertions that a social expectation exists or has been fulfilled or violated, based on a conditional rule of expectation, expressed in temporal logic.

The basic logic used includes these types of expression, in addition to the standard Boolean constants and connectives (true, false, $\wedge$, $\vee$ and $\neg$):

- Proposition symbols. In our application these represent observations made in Second Life, e.g. *avatar_name_*sitting.
- $\bigcirc\phi$: formula ϕ is true when evaluated in the next state
- $\Diamond\phi$: ϕ is true in the current or some future state
- $\Box\phi$: ϕ is true in all states from now onwards
- $\phi \,\mathsf{U}\, \psi$: ψ is true at the current or some future state, and ϕ is true for all states from now until just before that state

$\Diamond$ and $\Box$ can be expressed in terms of U and are abbreviations of longer expressions.

The logic also has some features of Hybrid Logic [15], but these are not used in this work except for the use of a *nominal* (a proposition that is true in a unique state) in the output from the model checker to 'name' the state in which a fulfilled or violated rule of expectation became active.

Finally, the logic includes the following operators related to conditional rules of expectation, and these are the types of expression sent from the Second Life script to the model checker:

- $\mathsf{ExistsExp}(Condition, Expectation)$
- $\mathsf{ExistsFulf}(Condition, Expectation)$
- $\mathsf{ExistsViol}(Condition, Expectation)$

where $Condition$ and $Expectation$ can be any formula that does not include ExistsExp, ExistsFulf and ExistsViol.

The first of these operators evaluates to true if there is an expectation existing in the current state that results from the rule specified in the arguments being triggered in the present or past. The other two operators evaluate to true if there is currently a fulfilled or violated expectation (respectively) resulting from the rule.

Formal semantics for this logic can be found elsewhere [3].

The input syntax to the model checker is slightly more verbose than that shown above. In particular, temporal operators must indicate the name of the "next state modality" as it appears in the input Kripke structure. In the examples in this paper, this will always be written as "`<next>`". Writing "`<next>`" on its own refers to the operator $\bigcirc$.

5 Two Simple scenarios

A simple rule of expectation that might apply in a Second Life scenario is that no one should ever fly. This might apply in a region used by members of a group that enacts historical behaviour. To monitor this expectation we can use the following property:

```
ExistsViol<next>(true, !any_flying)
```

This is an unconditional rule (it is triggered in every state) stating the expectation that there will not be any member of the group "Any" (comprising all avatars) flying.

If this is the only animation state to be tracked, the script's filter list will state that the animation "Flying" for group "Any" should be recorded, but otherwise all animations for all avatars and other groups should be discarded. On startup, the script sends the property to be monitored to the monitor, via Twitter, and then as avatars move around in Second Life and their animations are detected, it sends state messages that will either contain no propositions (if no one is flying) or will state that someone is flying:

```
S: any_flying
```

These states are accumulated, and each time a new state is received, the monitor is called and provided with the property to be monitored and the model (state history), e.g. $s_1 : \{\}$, $s_2 : \{\}$, $s_3 : \{\texttt{any\_flying}\}$ (the model is actually represented in XML—an example appears below).

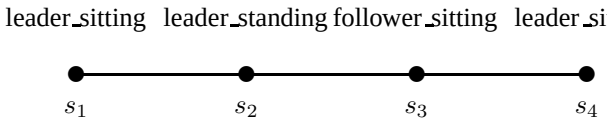
For this model, the monitor detects that the property is satisfied (i.e. the rule is violated) in state s_3 and a notification is sent back to the script. How this is handled is up to the script designer, but one option is for the script to be running in a "head-up-display" object, allowing the user to be informed in a way that other avatars cannot observe.

We now consider a slightly more complex example where there are two groups (or roles) specified in the script's group configuration list: `leader` (a singleton group) and `follower`. We want to monitor for violations of the rule that once the leader is standing, then from the next state a follower must not be sitting until the leader is sitting again. This is expressed using the following property:

```
ExistsViol<next>(
  leader_standing,
  <next>(U<next>(!follower_sitting,
                 leader_sitting))
)
```

The filter list can be configured so that only the propositions occurring in this rule are regarded as relevant for describing the state.

Suppose the scenario begins with the leader sitting and then standing, followed by the follower sitting, and finally the leader sitting again. This causes the following four states to be generated:



This is represented in the following XML format to be input to the model checker:

```
<hl-kripke-struct name="M">
  <world label="s1"/>
  <world label="s2"/>
  <world label="s3"/>
  <world label="s4"/>
  <modality label="next">
    <acc-pair to-world-label="s2"
              from-world-label="s1"/>
    <acc-pair to-world-label="s3"
              from-world-label="s2"/>
    <acc-pair to-world-label="s4"
              from-world-label="s3"/>
  </modality>
  <prop-sym label="leader_standing"
            truth-assignments="s2"/>
  <prop-sym label="leader_sitting"
            truth-assignments="s1 s4"/>
  <prop-sym label="follower_sitting"
            truth-assignments="s3"/>
  <nominal label="s1" truth-assignment="s1"/>
  <nominal label="s2" truth-assignment="s2"/>
  <nominal label="s3" truth-assignment="s3"/>
  <nominal label="s4" truth-assignment="s4"/>
</hl-kripke-struct>
```

The output of the model checker is:

```
s3: (s2, U<next>(!(follower_sitting),
                 leader_sitting))
```

This means that a violation occurred in state s_3 from the rule being triggered in state s_2. The violated expectation (after progression to state s_3) is:

```
U<next>(!(follower_sitting), leader_sitting)
```

This information is sent to the script.

6 Discussion

As mentioned in Section 2, our detection script currently only detects the animations of avatars within sensor range. This limits the scenarios that can be modelled to those based on (simulated) physical action. However, it is straightforward to add the ability to detect other LSL events, provided that they can be translated to a propositional representation. Thus we could detect that an avatar has sent a chat message, but we can't provide a propositional encoding that can express all possible chat message contents. However, the addition of new types of configuration list would allow additional flexibility. For example, regular expressions or other types of pattern could be defined along

with a string that can be appended to an avatar or group name to generate a proposition meaning that that avatar (or a member of that group) sent a chat message matching the pattern.

A significant limitation of the Linden Scripting Language is that the events that a script can detect are focused on the scripted object's own interactions with the environment—there is no facility for observing interactions between other agents, except for what can be deduced from their animations and chat. For many scenarios, it would be desirable to detect these interactions, for example, passing a certain object or sending money from one avatar to another might be a significant event in a society. One way around this problem would be to add additional scripted objects to the environment and set up the social conventions that these objects must be used for certain purposes. For example, an object in the middle of a conference table might need to be touched in order to request the right to speak next. These objects would generate appropriate propositions and send them to the main script via a private link.

The logic used currently is based on a discrete model of time, which can cause problems in some scenarios. For example, in the leader/follower scenario, it would be reasonable to allow the follower some (short) amount of time to stand after the leader stands. However, if a follower stands and another does not stand within the granularity of the same sensor event, then that second follower will be deemed in violation. It would be useful to be able to model some aspects of real time. This could be done by moving to a real-time temporal logic (which would involve some theoretical work on extending the model checker), or by some pragmatic means such as allowing the configuration parameters to define a frequency for regular "tick" timer events.

7 Related work

There seems to be little prior work that has explored the use of social awareness technology from multi-agent systems or other fields to support human interaction on the Internet in general, and in virtual worlds in particular.

A few avatar rating and reputation systems have been developed [16] to replace Second Life's own ratings system, which was disestablished in 2007. These provide various mechanisms to allow users to share their personal opinions of avatars with others.

Closer to our own work, Bogdanovych et al. [17, 18] have linked the AMELI electronic institution middleware [19] with Second Life. However, their aim is not to provide support for human interactions within Second Life, but rather to provide a rich interface for users to participate in an e-institution mediated by AMELI (in which the other participants may be software agents). This is done by generating a 3D environment from the institution's specification, e.g. *scenes* in the e-institution become rooms and transitions between scenes become doors. As a user controls their avatar to perform actions in Second Life, this causes an associated agent linked to AMELI to send messages to other agents, as defined by an action/message mapping table. Moving the avatar between rooms causes the agent to make a transition between scenes, but doors in Second Life will only open when the agent is allowed to make the corresponding scene transition according the rules of the institution.

This approach could be used to design and instrument environments that support structured human-to-human interaction in Second Life, but the e-institution model of communication is highly stylised and likely to seem unnatural for human users. In our work we are aiming to provide generic social awareness tools for virtual world users while placing as few restrictions as possible on the forms of interaction that are compatible with those tools. However, as discussed in Section 6, the limitation of the sensing functions provided by virtual world scripting languages may mean that some types of scenario cannot be implemented without providing specific scripted coordination objects that users are required to use, or the use of chat messages containing precise prespecified words or phrases.

Scripted objects acting as 'proximity sensors' have been developed as a tool for recording land use metrics in Second Life, such as the number and identities of avatars visiting a region over a period of time [20]. There are at least two companies selling proximity sensors in Second Life. Through the use of multiple sensors, large multi-region 'estates' can be monitored, which suggests that there are no inherent limitations in the use of LSL sensors that would prevent our approach from scaling.

8 Conclusion

This paper has reported on a prototype application of a model checking tool for social expectation monitoring applied to monitoring social interactions in Second Life. The techniques used for monitoring events in Second Life and allowing communication between a Second Life script and the monitor have been described, and these have been successfully tested on some simple scenarios. A discussion was presented on some of the limitations imposed by the LSL language and the logic used in the model checker, along with some suggestions for resolving these issues. Further work is needed to explore more complex scenarios and to test the scalability of the approach.

References

1. Linden Lab: Second Life home page. `http://secondlife.com/` (2008)
2. Boella, G., van der Torre, L., Verhagen, H.: Introduction to normative multiagent systems. In Boella, G., van der Torre, L., Verhagen, H., eds.: Normative Multi-agent Systems. Number 07122 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
3. Cranefield, S., Winikoff, M.: Verifying social expectations by model checking truncated paths. In: Coordination, Organizations, Institutions, and Norms in Agent Systems IV. Lecture Notes in Computer Science, 5428. Springer (2009) 204–219
4. Linden Lab: LSL portal. `http://wiki.secondlife.com/wiki/LSL_Portal` (2008)
5. Twitter: Twitter home page. `http://twitter.com/` (2008)
6. Amazon Web Services: Amazon simple queue service. `http://aws.amazon.com/sqs/` (2008)
7. Yamamoto, Y.: Twitter4j. `http://yusuke.homeip.net/twitter4j/en/` (2008)
8. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Compliance verification of agent interaction: a logic-based software tool. In Trappl, R., ed.: Cybernetics and Systems 2004. Volume II. Austrian Society for Cybernetics Studies (2004) 570–575

9. Esteva, M., de la Cruz, D., Sierra, C.: ISLANDER: an electronic institutions editor. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems, ACM (2002) 1045–1052
10. Vázquez-Salceda, J., Aldewereld, H., Dignum, F.: Implementing norms in multiagent systems. In: Proceedings of the Second German Conference on Multiagent System Technologies (MATES). Lecture Notes in Computer Science, 3187. Springer (2004) 313–327
11. Farrell, A.D.H., Sergot, M.J., Sallé, M., Bartolini, C.: Using the event calculus for tracking the normative state of contracts. International Journal of Cooperative Information Systems **14**(2 & 3) (2005) 99–129
12. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence **116**(1-2) (2000) 123–191
13. Dragone, L.: Hybrid logics model checker. http://luigidragone.com/hlmc/ (2005)
14. Markey, N., Schnoebelen, P.: Model checking a path. In: CONCUR 2003 – Concurrency Theory. Lecture Notes in Computer Science, 2761. Springer (2003) 251–265
15. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
16. Second Life: Removal of ratings in beta. `http://blog.secondlife.com/2007/04/12/removal-of-ratings-in-beta/` (2007)
17. Bogdanovych, A., Berger, H., Sierra, C., Simoff, S.J.: Humans and agents in 3D electronic institutions. In: Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems, ACM (2005) 1093–1094
18. Bogdanovych, A., Esteva, M., Simoff, S.J., Sierra, C., Berger, H.: A methodology for 3d electronic institutions. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, IFAAMAS (2007) 358–360
19. Esteva, M., Rosell, B., Rodrguez-Aguilar, J.A., Arcos, J.L.: AMELI: An agent-based middleware for electronic institutions. In: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems. Volume 1. IEEE Computer Society (2004) 236–243
20. Kezema, K.: Further analysis of parcel data collection. Blog post. `http://jeffkurka.blogspot.com/2009/03/further-analysis-of-parcel-data.html` (2009)

Directed Deadline Obligations in Agent-based Business Contracts

Henrique Lopes Cardoso and Eugénio Oliveira

LIACC, DEI / Faculdade de Engenharia, Universidade do Porto
R. Dr. Roberto Frias, 4200-465 Porto, Portugal
{hlc,eco}@fe.up.pt

Abstract. There are B2B relationships that presume cooperation in contract enactment. This issue should be taken into account when modeling, for computational handling, contractual commitments through obligations. Deadline obligations have been modeled by considering that reaching the deadline without compliance brings up a violation. When modeling commitments in business contracts, directed obligations have been studied for identifying two agents: the obligation's bearer and the counterparty, who may claim for legal action in case of non-compliance. We argue in favor of a directed deadline obligation approach, taking inspiration on international legislation over trade procedures. Our proposal to model contractual obligations is based on authorizations granted in specific states of an obligation lifecycle model, which we formalize using temporal logic and implement in a rule-based system. The performance of a contractual relationship is supported by a model of flexible deadlines, which allow for further cooperation between autonomous agents. As a result, the decision-making space of agents concerning contractual obligations is enlarged and becomes richer. We discuss the issues that agents should take into account in this extended setting.

1 Introduction

In cooperative B2B Virtual Organizations, agents (representing different enterprises) share their own competences and skills in a regulated way, through commitments expressed as norms in contracts. The importance of successfully proceeding with business demands for flexibility of operations: agents should try to facilitate the compliance of their partners. This common goal of conducting a multiparty business is based on the fact that group success also benefits each agent's private goals. These goals are not limited to the ongoing business relationship, but also concern future opportunities that may arise.

While addressing this problem with norms and multi-agent systems, we find that many approaches to normative multi-agent systems are abstracted away from their potential application domain. As such, deontic operators are often taken to have a universal semantics. For instance, deadline obligations are violated if the obliged action or state is not obtained until the deadline is reached.

We argue that in some domains – such as in business contracts – such an approach is not desirable. For instance, the United Nations Convention on Contracts for the International Sale of Goods (CISG) [1] establishes what parties may do in case of deadline violations. In some cases they are allowed to fulfill their obligations after the deadline (Article 48), or even to extend the deadlines with the allowance of their counterparties. Furthermore, a party may extend his counterparty's deadlines (Articles 47 and 63), which denotes a flexible and even cooperative facet of trade contracts.

In this paper we propose a different approach (in comparison with [2][3][4][5]) to the use of obligations in MAS in the domain of business contracts. Following a cooperative business performance posture, we argue that obligations should be directed, and that deadlines should be flexible. We start by reviewing, in section 2, the most typical variations regarding the formalization of obligations, after which we propose an approach based on directed obligations with deadlines. The flexibility required when handling temporal restrictions of obligations is addressed in section 3. The proposed approach is based on authorizations, and we present a lifecycle for directed obligations with temporal restrictions. In section 4 we investigate the decision-making process of agents concerning authorizations. Implementation of the proposed model in a rule-based system is discussed in section 5. Section 6 discusses related work and section 7 concludes.

2 Contractual Obligations

The use of norms in MAS makes use of the well-known deontic operators of obligation, permission and prohibition [6]. In theoretical deontic logic approaches, these operators are sometimes used to represent abstract general principles (e.g. it is forbidden to kill). In more applied research, deontic operators are ascribed either to roles or to particular agents in a system; e.g. $O_b(f)$ indicates that agent b is obliged to bring about fact f (a state of affairs or an action) – in this case agent b is said to be the *bearer* of the obligation.

Also, deontic operators are often made conditional and time constrained. Considering obligations, the conditional aspect has taken two different perspectives: conditional obligations of the form $O_b(f/s)$, meaning that agent b is obliged to bring about f when situation s arises; and conditional norms of the form $s \rightarrow O_b(f)$, meaning that if s then b is obliged to bring about f. As for the temporal aspect of deontic operators, deadlines (either time references or more generally defined as states of affairs) are typically employed for stipulating the validity of the operator: $O_b(f, d)$ is a *deadline obligation* indicating that agent b is obliged to bring about f before d.

We will base the following discussion on the *obligation* deontic operator, as it is the most important operator to represent trade relationships in B2B contracts. The meaning of deontic operators has been studied, mainly regarding the use of

deadlines (e.g. [2]). Regarding deadline obligations, the usual approach to their semantics is to consider the following entailments[1]:

- $O_b(f, d) \wedge (f\ B\ d) \models \mathit{Fulf}_b(f, d)$ — If the fact to bring about occurs before the deadline, the agent has fulfilled his obligation.
- $O_b(f, d) \wedge (d\ B\ f) \models \mathit{Viol}_b(f, d)$ — If the deadline occurs before the fact to bring about, the agent has violated his obligation.

The introduction of *Fulf* and *Viol* enables reasoning about the respective situations. The implementation of this semantics using forward-chaining rules has been studied in [3]. Although intuitive, this semantics is quite rigid in that violations are all defined in a universal way (discounting the fact that different norms can respond to violations in different ways).

The analysis of contracts brings into discussion the notion of *directed obligations* [8]. Obligations are seen as directed from a *bearer* (responsible for fulfilling the obligation) to a *counterparty*. Some authors [4] define the very notion of *contractual obligation* as an obligation with an "obligor" (bearer) and an "obligee" (counterparty). The relationship between these two roles in a directed obligation has been studied, giving rise to two different theories. The *benefit theory* promotes the fact that the counterparty of an obligation is intended to benefit from its fulfillment (see [8] for a benefit theory perspective of directed obligations). A more relevant approach in which contracts are concerned – the *claimant theory* – takes the stance that obligations are interpreted as claims from counterparties to bearers (see [5] for a claimant theory support).

In general, claimant approaches are based on the following definition for directed obligation (adapted from [5]): $O_{b,c}(f) =_{def} O_b(f) \wedge (\neg f \Rightarrow P_c(la_b))$. A directed obligation from agent b towards agent c to bring about f means that b is obliged to bring about f and if b does not bring about f then c is *permitted* to initiate legal action against b. The concept of legal action is rather vague. A similar approach is taken in [9], where agent c is said to be *authorized* to repair the situation in case b does not fulfill his obligation. Repair actions include demanding further actions from b; e.g., c may demand compensation for damages. It is interesting to note that such definitions are careful enough to base the claims of the counterparty on the *non-fulfillment* of the obligation, not on its violation. In fact, these definitions do not include deadlines, which are the basis for violation detection. Another significant issue is the discretionary nature of the counterparty's reaction (he is permitted or authorized), instead of an automatic response based on the non-fulfillment of the bearer[2].

[1] In the following formulae we will follow linear temporal logic (LTL) [7], with a discrete time model. Let $x = (s_0, s_1, s_2, ...)$ be a timeline, defined as a sequence of states s_i. The syntax $x \models p$ reads that p is true in timeline x. We write x^k to denote state s_k of x, and $x^k \models p$ to mean that p is true at state x^k. We use a weak version of the *before* LTL operator B, where q is not mandatory: $x \models (p\ B\ q)$ *iff* $\exists_j(x^j \models p \wedge \forall_{k<j}(x^k \models \neg q))$.

[2] As in automatic violation detection approaches based on deadlines, complemented with the definition of violation reaction norms.

We propose the use of *directed deadline obligations* as the basis for defining contractual obligations: $O_{b,c}(f, d)$. In section 3 we describe a model for flexible obligation violation, based on the principle that the deadline is meant to indicate when the counterparty is authorized to react to the non-fulfillment of an obligation directed to him. A possible reaction is to declare the obligation as violated, but there are other means to settle the matter, to the benefit of both involved parties. An extension of directed (contractual) obligations with temporal restrictions is also introduced in [4], but that approach is based on a rigid model of violations, in that they are automatically obtained at the deadline.

2.1 Directed Deadline Obligations

Our proposal combines directed [5][8] and deadline [2] obligations. Although this has been done in the past (e.g. [4]), in our approach deadlines have a distinct role in the definition of obligations. In section 3 we detail such a role.

Directed deadline obligations take the form $O_{b,c}(f, d)$, meaning that agent b is obliged towards agent c to bring about f before d. We do not make obligations conditional (as in [4]), because we assume they are obtained from conditional norms: rules prescribing obligations when certain situations arise.

We consider that if fact f is not yet the case when deadline d arises, the obligation is not yet violated, but is in a state where the counterparty is authorized to take some action. We emphasize the case for a *deadline violation* (as opposed to obligation violation). This comprises a flexible approach to handling non-ideal situations: each deadline violation is different, as each may have a different impact on the ongoing business, and each occurs between a specific pair of agents with a unique trust relationship.

Some evidence from the CISG convention [1] led us to this approach:

> *Article 48: (1) [...] the seller may, even after the date for delivery, remedy at his own expense any failure to perform his obligations, if he can do so without unreasonable delay [...]; (2) If the seller requests the buyer to make known whether he will accept performance and the buyer does not comply with the request within a reasonable time, the seller may perform within the time indicated in his request. [...]*

This means that even though a deadline has been violated, the bearer may still be entitled to fulfill *the same* obligation. This kind of delay is also called a *grace period*: a period beyond a due date during which an obligation may be met without penalty or cancellation.

Figure 1 illustrates the intuitive semantics of a directed deadline obligation. The shaded area represents the period of time within which the achievement of f will certainly bring a fulfillment of the obligation. The region to the right of d indicates that counterparty c is entitled to react if f is not accomplished; however, as long as no reaction is taken, b can still fulfill his obligation.

Therefore, a deadline violation brings a counterparty authorization. Authorizations are taken into account in the normative system by having rules and

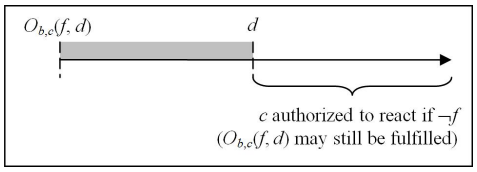


Fig. 1. Directed obligation with deadline.

norms that are based on the materialization of such authorizations. The available options are discussed in section 3.

2.2 Livelines and Deadlines

The deadline approach is often taken to be appropriate for specifying temporal restrictions on obligations. However, in certain cases a time window should be provided. In international trade transactions, for instance, storage costs may be relevant. Also, perishable goods should be delivered only when they are needed, not before. This is why in CISG [1] we have:

> *Article 52: (1) If the seller delivers the goods before the date fixed, the buyer may take delivery or refuse to take delivery.*

Therefore, anticipated fulfillments are not always welcome. We find it necessary to include a variation of directed deadline obligations, to which we add a *liveline*: a time reference after which the obligation should be fulfilled. In this case we have $O_{b,c}(f, l, d)$: agent b is obliged towards agent c to bring about f between l (a liveline) and d (a deadline). Figure 2 illustrates the intuitive semantics of this kind of obligation. The shaded area represents the period of time within which the achievement of f will certainly bring a fulfillment of the obligation. If f is accomplished before l, however, it may be the case that c is not willing to accept such a fulfillment, or at least that he may not be happy about it – the region to the left of l entitles c to react if f is accomplished. The region to the right of d is as with (simple) directed deadline obligations.

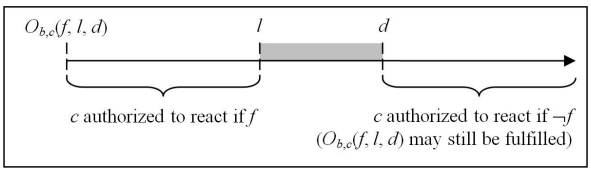


Fig. 2. Directed obligation with liveline and deadline.

We escape from an approach with a fixed time reference for obligation fulfillment (an obligation for bringing about f at time t), which would be suggested

by the term "date fixed" in CISG's Article 52 transcription above. We find it more convenient to define a fixed date as an interval, say, from the beginning till the end of a specific date[3].

3 Managing Liveline and Deadline Violations

After we have advocated, in the preceding section, a counterparty authorization approach to deadline violations, in this section we address the issue of what kind of actions the counterparty may take in such situations, and what are their effects on the obligation whose deadline has been violated. The same accounts to directed obligations with both livelines and deadlines.

The successful enactment of a contract is dependent on the need to make contractual provisions performable in a flexible way. The importance of having flexible trade procedures is apparent, once again, in the CISG convention [1]:

> *Article 47: (1) The buyer may fix an additional period of time of reasonable length for performance by the seller of his obligations.*
>
> *Article 63: (1) The seller may fix an additional period of time of reasonable length for performance by the buyer of his obligations.*

These articles emphasize, once more, the need for flexible deadlines. Note that the counterparty's benevolence on conceding an extended deadline to the bearer does not prescribe a new obligation; instead, *the same* obligation may be fulfilled within a larger time window. Furthermore, it is also in the counterparty's best interest that this option is available, given the importance of reaching success in the performance of the contract.

In some other cases, a party may decide that the non-fulfillment of an obligation should be handled in a more strict way. The CISG convention specifies conditions for cancelling a contract in case of breach:

> *Article 49: (1) The buyer may declare the contract avoided: (a) if the failure by the seller to perform any of his obligations [...] amounts to a fundamental breach of contract; [...]; (2) However, in cases where the seller has delivered the goods, the buyer loses the right to declare the contract avoided unless he does so: (a) in respect of late delivery, within a reasonable time after he has become aware that delivery has been made; [...]*
>
> *Article 64: (1) The seller may declare the contract avoided: (a) if the failure by the buyer to perform any of his obligations [...] amounts to a fundamental breach of contract; [...]; (2) However, in cases where the buyer has paid the price, the seller loses the right to declare the contract avoided unless he does so: (a) in respect of late performance by the buyer, before the seller has become aware that performance has been rendered; [...]*

[3] This is actually a matter of time granularity.

These articles allow contract termination in both non-performance and late performance cases. However, the second case is limited to the awareness of the offended party.

From these excerpts we can distinguish two types of reactions to non-fulfillments: a smoother one (from articles 47, 48 and 63), in which parties are willing to recover from an initial failure to conform to an obligation; and a stricter one (articles 49 and 64), where the failure is not self-containable anymore. Based on these options, we propose a model for a directed deadline obligation lifecycle.

3.1 Authorizations on Violations

Following the discussion above, we identify the possible states for an obligation, together with the elements we shall use to signal some of those states (when obtained, these elements are supposed to persist over time):

- *inactive*: the obligation is not yet in effect, but will eventually be prescribed by a norm;
- *active*: the obligation was prescribed by a norm – $O_{b,c}(f, d)$ or $O_{b,c}(f, l, d)$
- *pending*: the obligation may be fulfilled from now on;
- *liveline violation*: the fact being obliged has been brought ahead of time – $LViol_{b,c}(f, l, d)$
- *deadline violation*: the fact being obliged should have been brought already – $DViol_{b,c}(f, d)$ or $DViol_{b,c}(f, l, d)$
- *fulfilled*: the obligation was fulfilled – $Fulf_{b,c}(f, d)$ or $Fulf_{b,c}(f, l, d)$
- *violated*: the obligation was violated and cannot be fulfilled anymore – $Viol_{b,c}(f, d)$ or $Viol_{b,c}(f, l, d)$

Starting with the simpler case of directed deadline obligations, we identify the (absolute) fulfillment case:

- $O_{b,c}(f, d) \wedge (f\ B\ d) \models Fulf_{b,c}(f, d)$

Then we state the consequence of reaching a deadline with no achievement of the obligated fact:

- $O_{b,c}(f, d) \wedge (d\ B\ f) \models DViol_{b,c}(f, d)$

Note that, differently from the usual approach, we set the obligation to have a violated deadline – $DViol_{b,c}(f, d)$ – but not to be violated in itself.

The counterparty's reaction to a deadline violation will only change the obligation's state if the option is to deem the obligation as violated, by *denouncing* this situation. For this we introduce the element $Den_{c,b}(f, d)$, which is a denounce from agent c towards agent b regarding the failure of the latter to comply with his obligation to bring about f before d. Since we consider the achievement of facts to be common knowledge, a party may only denounce the non-fulfillment of an obligation while that obligation is not fulfilled yet[4]:

[4] This is a simplification of what articles 49 and 64 of CISG suggest.

– $DViol_{b,c}(f, d) \wedge (f\ B\ Den_{c,b}(f, d)) \models Fulf_{b,c}(f, d)$
– $DViol_{b,c}(f, d) \wedge (Den_{c,b}(f, d)\ B\ f) \models Viol_{b,c}(f, d)$

Figure 3 illustrates, by means of a state transition diagram, the lifecycle of directed deadline obligations. We take obligations as being prescribed from conditional norms; the confirmation of the norm's condition will change the prescribed obligation's state from inactive to active. The obligation is also automatically pending, since it may be legitimately fulfilled right away.

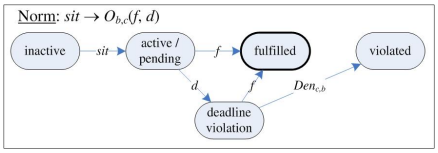


Fig. 3. Lifecycle of a directed deadline obligation.

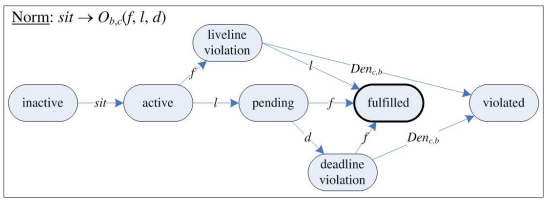


Fig. 4. Lifecycle of a directed obligation with liveline and deadline.

Figure 4 contains the state transition diagram for directed obligations with livelines and deadlines. In this case, the obligation will only be pending when l arises, since only then it may be fulfilled in a way that is compliant with the terms of the contract. We define the following relations:

– $O_{b,c}(f, l, d) \wedge (f\ B\ l) \models LViol_{b,c}(f, l, d)$
– $LViol_{b,c}(f, l, d) \wedge (l\ B\ Den_{c,b}(f, l, d)) \models Fulf_{b,c}(f, l, d)$
– $LViol_{b,c}(f, l, d) \wedge (Den_{c,b}(f, l, d)\ B\ l) \models Viol_{b,c}(f, l, d)$
– $O_{b,c}(f, l, d) \wedge (l\ B\ f) \wedge (f\ B\ d) \models Fulf_{b,c}(f, l, d)$
– $O_{b,c}(f, l, d) \wedge (d\ B\ f) \models DViol_{b,c}(f, l, d)$
– $DViol_{b,c}(f, l, d) \wedge (f\ B\ Den_{c,b}(f, l, d)) \models Fulf_{b,c}(f, l, d)$
– $DViol_{b,c}(f, l, d) \wedge (Den_{c,b}(f, l, d)\ B\ f) \models Viol_{b,c}(f, l, d)$

We have now two kinds of temporal violations: liveline violations of the form $LViol_{b,c}(f, l, d)$ and deadline violations of the form $DViol_{b,c}(f, l, d)$. In both cases a denounce may establish the obligation as violated, if issued before l or f, respectively.

3.2 Smoother Authorizations on Violations

The diagrams in figures 3 and 4 only include events that produce a change in an obligation's state. The denouncement of the non-fulfillment of an obligation, making it violated and consequently not fulfillable any longer, denotes a situation in which a bearer's attempt to fulfill the obligation will no longer be significant to the counterparty, and thus a consummated violation should be handled according to applicable norms. These may bring sanctions, further obligations or ultimately a contract cancellation, as in articles 49 and 64 of CISG.

In order to accommodate less strict situations (see articles 47, 48 and 63 of CISG), we consider that in liveline and deadline violation states, while the obligation can still be fulfilled, the counterparty may react to the non-ideal situation. These possibilities are not illustrated in figures 3 and 4, since they do not bring state changes. For instance, in international trade transactions storage costs may be relevant. The counterparty may therefore be authorized to demand for payment of storage costs from an early compliant bearer. Another example for the deadline violation case:

> *Article 78: If a party fails to pay the price or any other sum that is in arrears, the other party is entitled to interest on it [...]*

While obligation state transitions are processed with appropriate rules (including rules that take denounces into account), authorizations expressing the counterparty's right to demand for compensation are handled by the system through appropriate norms, which may be defined in a contract basis.

4 Decision-making on Directed Deadline Obligations

The authorization approach described above enriches the decision-making space of agents concerning norms. Since commitments can be violated, agents (as human delegates) may decide whether to fulfill them or not. Furthermore, because the violation state is determined by the counterparty's choice to denounce this situation, both parties associated with a directed obligation are in a position to decide over it after the deadline.

In order to model the decision making process, we need to assess each agent's valuations on the obligation states and facts they are able to bring about. We will write $v_a(f)$ and $v_a(S)$ to denote the valuation agent a makes of fact f or state S, respectively (similarly to the valuation model used in [10]). When valuating an obligation's state (namely a fulfillment or a violation), agents should take into account two different sorts of effects. First, since an obligation is taken to be a part of a wider contract that should benefit all participants, the obligation cannot be taken in isolation, as its fulfillment or violation may trigger further commitments. Second, an agent's reputation is affected by whether or not he stands for his commitments. In the following we assume that an agent is capable of anticipating and evaluating the consequences of his actions within a contract.

For an obligation $O_{b,c}(f, d)$ we have the following valuation constraints for b:

$v_b(O_{b,c}(f,d)) < 0$: an obligation is a burden to its bearer
$v_b(f) < v_b(O_{b,c}(f,d))$: there is a heavier cost associated with bringing about f
$v_b(\mathit{Fulf}_{b,c}(f,d)) > 0$: b gains from fulfilling his obligation
$v_b(\mathit{Viol}_{b,c}(f,d)) < 0$: b loses from violating his obligation

The notions of gain and loss for the bearer extend to outside this obligation. For instance, fulfilling an obligation may bring an entitlement (a new obligation where the bearer becomes the counterparty). Violating an obligation will potentially bring penalties to the bearer, hence the negative valuation. In both cases, the reputation of agent b is affected (positively or negatively). Unlike in [10], we do not impose that $v_b(\mathit{Viol}_{b,c}(f,d)) < v_b(f) + v_b(\mathit{Fulf}_{b,c}(f,d))$. An agent may be able to exploit a contract flaw by considering that in a specific situation he is better off violating his obligation than fulfilling it. Of course that even if the above condition holds, agent b may still choose to violate his obligations, because of other conflicting goals: he may lose with respect to the outcome of this contract, but may possibly win across contracts.

As for the counterparty c, we have:

$v_c(O_{b,c}(f,d)) > 0$: an obligation is an asset for the counterparty
$v_c(f) > v_c(O_{b,c}(f,d))$: c benefits from f
$v_c(\mathit{Fulf}_{b,c}(f,d)) \leq 0$: c may acquire obligations after fulfillment
$v_c(\mathit{Viol}_{b,c}(f,d)) \geq 0$: c may obtain compensations after violation

Note that both fulfillments and violations may bring no value if they have no further consequences in the contract.

In a rough attempt to model the decision making process of a counterparty of an obligation whose deadline was violated, we could state that he should denounce (and thus obtain the obligation's violation) if[5]

$$v_c(f) + v_c(\mathit{Fulf}_{b,c}(f,d)) < v_c(\mathit{Viol}_{b,c}(f,d)).$$

We consider that valuations may possibly vary with time. Were that not the case, the above condition would only need to be checked right after d, at which point the counterparty would either denounce or decide to wait indefinitely for the bearer to fulfill his obligation. For instance, we believe that it makes sense to think of $v_c(f)$ as possibly decreasing with time (like a resource that should be available but is not yet). Even when the above condition does not hold, the counterparty may still prefer to tolerate the less preferred situation of failure for matters of conflicting goals (just as with the bearer).

Until now we have discussed the possibility of agents (both bearers and counterparties) deciding on breach over compliance (either by assessing intra-contract consequences or by inter-contract conflicts). But in scenarios enriched with social features agents can exploit, it may be the case that agents decide to behave cooperatively even when they have to bear a contained disadvantage. In such settings, more than being altruistic, agents may try to enhance their trust awareness in the community, from which they will benefit in future interactions or contracts.

[5] We assume there is no cost associated with the denouncing action.

5 Implementation and Practical Issues

The logical relationships expressed above provide us a formalism to define directed deadline obligations. However, in order to monitor contracts at run-time, we need to ground this semantics into a reasoning engine capable of responding to events in a timely fashion. That is, elements describing obligation states should allow us to reason about those states *as soon as they occur*.

A natural choice we have made before [3] is the use of a rule-based inference engine, with which the following (forward-chaining) rules can be defined to implement the semantics of directed obligations with livelines and deadlines[6]:

- $O_{b,c}(f, l, d) \wedge f \wedge \neg l \rightarrow LViol_{b,c}(f, l, d)$
- $LViol_{b,c}(f, l, d) \wedge l \wedge \neg Den_{c,b}(f, l, d) \rightarrow Fulf_{b,c}(f, l, d)$
- $LViol_{b,c}(f, l, d) \wedge Den_{c,b}(f, l, d) \wedge \neg l \rightarrow Viol_{b,c}(f, l, d)$
- $O_{b,c}(f, l, d) \wedge l \wedge \neg LViol_{b,c}(f, l, d) \wedge f \wedge \neg d \rightarrow Fulf_{b,c}(f, l, d)$
- $O_{b,c}(f, l, d) \wedge d \wedge \neg f \rightarrow DViol_{b,c}(f, l, d)$
- $DViol_{b,c}(f, l, d) \wedge f \wedge \neg Den_{c,b}(f, l, d) \rightarrow Fulf_{b,c}(f, l, d)$
- $DViol_{b,c}(f, l, d) \wedge Den_{c,b}(f, l, d) \wedge \neg f \rightarrow Viol_{b,c}(f, l, d)$

With this approach, we assume an immediate assertion of facts and deadlines when they come into being. Furthermore, rules are expected to be evaluated in every working memory update (e.g. right after a fact is asserted), in order to produce the indicated conclusions, which are added to the normative state in a cumulative fashion. To detect the moment at which the *before* relation holds, we translated terms of the form $(e_1\ B\ e_2)$ into a conjunction $e_1 \wedge \neg e_2$. The fourth rule demanded for a more careful construction, since we had two consecutive before relations – we needed to ensure that there was no liveline violation when having both l and f.

5.1 Reasoning with Time

In business contracts it is common to have deadlines that are dependent on the fulfillment date of other obligations. Therefore, instead of having fixed (absolute) dates, these may at times be relative, calculated according to other events. CISG [1] expresses this by saying that dates can be *determinable* from the contract:

> *Article 33: The seller must deliver the goods: (a) if a date is fixed by or determinable from the contract, on that date; (b) if a period of time is fixed by or determinable from the contract, at any time within that period [...]*
>
> *Article 59: The buyer must pay the price on the date fixed by or determinable from the contract [...]*

[6] The simpler case of directed deadline obligations is a simplification over these rules.

It is therefore useful to timestamp each event: facts, fulfillments and violations. For that purpose, $Fulf_{b,c}(f, l, d)^t$ will be used to indicate that b has fulfilled at time point t its obligation towards c to obtain f between l and d; similarly for $Viol_{b,c}(f, l, d)^t$. Since a fact itself has now a timestamp attribute, for ease of reading we will write fact f achieved at time point t as $Fact(f)^t$. A denounce will also be written $Den_{c,b}(f, l, d)^t$.

Norms will be based on these elements and on their time references in order to prescribe other obligations with relative deadlines. For instance,

$$Fulf_{b,c}(Deliver(x, q), \_, \_)^t \rightarrow O_{c,b}(Pay(price), t, t + 10)$$

means that once agent b has fulfilled his obligation to deliver q units of x to agent c, the latter is obliged to pay the former within a period of 10 time units.

5.2 Re-implementing Rules

We also need to update our rules in order to stamp each generated event. In fact, having timestamps also allows us to implement such rules in a way that has a closer reading to the LTL *before* operator:

- $O_{b,c}(f, l, d) \wedge Fact(f)^t \wedge t < l \rightarrow LViol_{b,c}(f, l, d)$
- $LViol_{b,c}(f, l, d) \wedge l \wedge \neg(Den_{c,b}(f, l, d)^u \wedge u < l) \rightarrow Fulf_{b,c}(f, l, d)^l$
- $LViol_{b,c}(f, l, d) \wedge Den_{c,b}(f, l, d)^u \wedge u < l \rightarrow Viol_{b,c}(f, l, d)^u$
- $O_{b,c}(f, l, d) \wedge Fact(f)^t \wedge l < t \wedge t < d \rightarrow Fulf_{b,c}(f, l, d)^t$
- $O_{b,c}(f, l, d) \wedge d \wedge \neg(Fact(f)^t \wedge t < d) \rightarrow DViol_{b,c}(f, l, d)$
- $DViol_{b,c}(f, l, d) \wedge Fact(f)^t \wedge \neg(Den_{c,b}(f, l, d)^u \wedge u < t) \rightarrow Fulf_{b,c}(f, l, d)^t$
- $DViol_{b,c}(f, l, d) \wedge Den_{c,b}(f, l, d)^u \wedge \neg(Fact(f)^t \wedge t < u) \rightarrow Viol_{b,c}(f, l, d)^u$

This kind of approach has the benefit of relaxing the rule evaluation policy: rules do not have to be evaluated after each working memory update, since we are checking the timestamps of each event (see also [3]).

5.3 Example Contract

Considering a two-party business scenario, a contract should be beneficial for both involved parties. Therefore, both are obliged to bring about certain facts (e.g. payments or deliveries) in specific situations, and those facts should benefit the obligations' counterparties. The contract will typically specify remedies for breach situations (such as those pointed out at CISG). For the sake of illustration, we present a possible buyer-supplier contract: agent S commits to supply agent B, whenever he orders, good X for 7.5 per unit. The norms below define this particular contractual relationship. Agent S is supposed to deliver the ordered goods between 3 to 5 days after the order (norm $n1$), and agent B shall pay within 30 days (norm $n2$). Furthermore, if agent B does not pay in due time, he will incur in a penalty consisting of an obligation to pay an extra 10% on the order total (norm $n3$). Finally, if agent S violates his obligation to deliver, the contract will be canceled (norm $n4$).

(n1) $Fact(Order(X, q))^w \rightarrow O_{S,B}(Deliver(X, q), w + 3, w + 5)$
(n2) $Fulf_{S,B}(Deliver(X, q), l, d)^w \rightarrow O_{B,S}(Pay(q * 7.5), w, w + 30)$
(n3) $DViol_{B,S}(Pay(p), l, d) \rightarrow O_{B,S}(Pay(p * 0.10), d, d + 30)$
(n4) $Viol_{S,B}(Deliver(X, q), l, d)^w \rightarrow Cancel\_contract$

Note that the interest applied on payments is automatic once a deadline violation is detected (norm $n3$). On the other hand, a contract cancellation (norm $n4$) requires that agent B denounces the inability of agent S to fulfill the delivery. It is therefore up to agent B whether to wait further and accept a delayed delivery or not. If the agreed upon contract conditions are important enough, allowing a counterparty deviation (and hence taking a cooperative attitude regarding the compliance of the contract) may be a good decision.

Different kinds of situations may be easily modeled using this kind of norms. Moreover, using flexible deadlines also ensures a degree of freedom for agents to make decisions in the execution phase of contracts, which is important for dealing with business uncertainty.

6 Related Work

Most implementations of norms in multi-agent systems ignore the need for having directed obligations from bearers to counterparties. The most likely reason for this is that in those approaches obligations are seen as (implicitly) directed from an agent to the normative system itself. It is up to the system (e.g. an electronic organization [11] or an electronic institution [12]) to detect violations and to enforce the norms which are designed into the environment (in some cases they are even regimented in such a way that violation is not possible). On the contrary, our flexible approach towards an Electronic Institution allows agents to define the norms that will regulate their mutual commitments.

Other authors have proposed different lifecycles for commitments and deontic operators. Directed social commitments are modeled in [13], in the context of dialogical frameworks. Violated commitments resort to their cancellation, which may bring sanctions. An interesting issue that is explicit in the model is the possibility for the bearer to cancel his commitment, allowing the counterparty to apply sanctions; also, updating is allowed through cancellation of the commitment and creation of a new one. A more compact model is presented in [14], also considering the possibility to update commitments. However, fulfillment and violation are not dealt with explicitly in this model; instead, a commitment is discharged when fulfilled, or else may be canceled.

Taking a cooperative approach to contract fulfillment, in [15] an obligation lifecycle model includes states that are used in a contract fulfillment protocol. Agents communicate about their intentions to comply with obligations, and in this sense an obligation can be refused or accepted. After being accepted, the obligation may be canceled or complied with. These states are obtained according to the performance of a contractual relationship. Our model should also require that agents communicate their intentions regarding an obligation

with a violated deadline. In fact, CISG's Article 48 seems to go in this direction, in order to protect the bearer's efforts toward a late fulfillment of the obligation.

The need to identify two opposite roles in deontic operators is not exclusive of obligations. In [5] the concept of directed permission is described on the basis of *interference* and *counter-performance*. If a party is permitted by another to bring about some fact, the latter is not allowed to interfere with the attempt of the former to achieve that fact. The authors also sustain a relation between directed obligations and directed permissions: $O_{b,c}(f) \rightarrow P_{b,c}(f)$, that is, if an agent b has an obligation towards an agent c, then b is permitted (by c) to bring about the obliged fact and c is not permitted to interfere. This is very important in international trade transactions, especially when storage costs can be high. Some evidence from CISG [1] brings us once more the same insight:

> *Article 53: The buyer must pay the price for the goods and take delivery of them [...]*
>
> *Article 60: The buyer's obligation to take delivery consists: (a) in doing all the acts which could reasonably be expected of him in order to enable the seller to make delivery; and (b) in taking over the goods.*

In this case the permission is described in terms of an obligation of the counterparty (the buyer).

Our model of directed obligations with livelines and deadlines has some connections with research on real-time systems, where a time-value function valuates a task execution outcome depending on the time when it is obtained. *Soft* real-time systems use soft deadlines: obtaining the result after the deadline has a lower utility. In contrast, for *hard* real-time systems the deadline is crisp: after it, the result has no utility at all, and missing the deadline can have serious consequences. Our approach seems to be soft with a hard-deadline discretionally declared by the counterparty of the task to achieve. Deadline goals are also analyzed in [16] in the context of goal-directed and decision-theoretic planning. Goals are given a temporal extent and can be partially satisfied according to this temporal component. The authors propose a *horizon* time point somewhere after the deadline, after which there will be no benefit in achieving the goal. In our case the horizon is not static, but can be defined by the counterparty.

A model for commitment valuations, on which we have based our decision-making prospect, has been proposed in [10]. However, while their work is centered on checking correctness of contracts, we focus on valuations in the course of a contract execution. We do not assume that a contract is correct from a fairness point of view. This difference in concerns has brought divergent considerations when valuating fulfillment and violation states.

Other authors have studied agent decision-making regarding norm compliance. For instance, violation games, put in perspective of a game-theoretic approach to normative multi-agent systems in [17], model the interactions between an agent and the normative system that is responsible to detect violations and sanction them accordingly. That line of research analyses how an agent can violate obligations without being sanctioned. In our case, while we assume that

temporal violations are always detected, we explore decision-making from the point of view of both the bearer and the counterparty of a directed obligation.

7 Conclusions

In cooperative B2B Virtual Organizations, contracts specify, through obligations, the interdependencies between different partners, and provide legal options to which parties can resort in case of conflict. However, when this joint activity aims at pursuing a common goal, the successful performance of business benefits all involved parties. Therefore, when developing automated monitoring tools, one should take into account that agents may be cooperative enough to allow counterparties' deviations.

Taking this into account, in this paper we have presented a novel model for contractual obligations – *directed deadline obligations*. Following a claimant theory approach, the directed aspect concerns the need to identify the agent who will be authorized to react in case of non-fulfillment. We started from previous theoretical approaches to model such authorizations, and developed a more concrete formalization by linking authorizations with a flexible model of deadlines. Obligation violations are now dependent on the counterparty motivation to claim them. We have also considered in our model smoother authorizations.

Our approach is based on real-world evidence from business contracts (namely the United Nations Convention on Contracts for the International Sale of Goods), which denotes a flexible and even cooperative facet of trade contracts. This facet extends to the concept of B2B Virtual Organizations, wherein different parties come together to share a business goal that is achievable through the cooperative fulfillment of a common contract.

We addressed the important issue of agent decision-making, which is enriched by our model of authorizations. Both parties involved in a directed deadline obligation may have a say regarding its violation. When considering obligations as interlinked through norms in a contract, agents should evaluate the consequences of fulfillment and violation states as prescribed in the contract. Furthermore, in "socially rich" environments, agents should explore the value of future relationships by enhancing their perceived trustworthiness and predisposition to facilitate compliance, something that is made possible by our directed deadline obligations approach.

Acknowledgments. The first author is supported by FCT (Fundação para a Ciência e a Tecnologia) under grant SFRH/BD/29773/2006.

References

1. UNCITRAL: United nations convention on contracts for the international sale of goods (cisg) (1980)

2. Broersen, J., Dignum, F., Dignum, V., Meyer, J.J.C.: Designing a deontic logic of deadlines. In Lomuscio, A., Nute, D., eds.: 7th International Workshop on Deontic Logic in Computer Science. Volume 3065 of LNCS., Madeira, Portugal, Springer Verlag, Heidelberg (2004) 43–56
3. Lopes Cardoso, H., Oliveira, E.: A context-based institutional normative environment. In Hübner, J.F., Boissier, O., eds.: AAMAS'08 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN), Estoril, Portugal (2008) 119–133
4. Ryu, Y.U.: Relativized deontic modalities for contractual obligations in formal business communication. In: 30th Hawaii International Conference on System Sciences (HICSS). Volume 4., Hawaii, USA (1997) 485–493
5. Tan, Y.H., Thoen, W.: Modeling directed obligations and permissions in trade contracts. In: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences, Volume 5, IEEE Computer Society (1998)
6. von Wright, G.: Deontic logic. Mind **60** (1951) 1–15
7. Emerson, E.A.: Temporal and modal logic. In Leeuwen, J.v., ed.: Handbook of Theoretical Computer Science. Volume B: Formal Models and Sematics. North-Holland Pub. Co./MIT Press (1990) 995–1072
8. Herrestad, H., Krogh, C.: Obligations directed from bearers to counterparties. In: Proceedings of the 5th international conference on Artificial intelligence and law, College Park, Maryland, United States, ACM (1995) 210–218
9. Dignum, F.: Autonomous agents with norms. Artificial Intelligence and Law **7**(1) (1999) 69–79
10. Desai, N., Narendra, N.C., Singh, M.P.: Checking correctness of business contracts via commitments. In: Proc. 7th Intl. Joint Conf. on Autonomous Agents and Multiagent Systems, Estoril, Portugal, IFAAMAS (2008) 787–794
11. Vázquez-Salceda, J., Dignum, F.: Modelling electronic organizations. In Marik, V., Muller, J., Pechoucek, M., eds.: Multi-Agent Systems and Applications III: 3rd Int. Central and Eastern European Conf. on Multi-Agent Systems (CEEMAS'03). Volume 2691 of LNAI., Prague, Czech Republic, Springer Verlag (2003) 584–593
12. Esteva, M., Rodríguez-Aguilar, J.A., Sierra, C., Garcia, P., Arcos, J.L.: On the formal specifications of electronic institutions. In Dignum, F., Sierra, C., eds.: Agent-mediated Electronic commerce: The European AgentLink Perspective. Volume 1991 of LNAI. Springer (2001) 126–147
13. Pasquier, P., Flores, R.A., Chaib-Draa, B.: Modelling flexible social commitments and their enforcement. In Gleizes, M.P., Omicini, A., Zambonelli, F., eds.: Engineering Societies in the Agents World V. Volume 3451 of LNAI. Springer Verlag (2005) 139–151
14. Wan, F., Singh, M.P.: Formalizing and achieving multiparty agreements via commitments. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, The Netherlands, ACM (2005) 770–777
15. Sallé, M.: Electronic contract framework for contractual agents. In Cohen, R., Spencer, B., eds.: Advances in AI: 15th Conf. of the Canadian Soc. for Computational Studies of Intelligence. Springer (2002) 349–353
16. Haddawy, P., Hanks, S.: Utility models for goal-directed, decision-theoretic planners. Computational Intelligence **14**(3) (1998) 392–429
17. Boella, G., van der Torre, L.: A game-theoretic approach to normative multi-agent systems. In Boella, G., van der Torre, L., Verhagen, H., eds.: Normative Multi-agent Systems (NorMAS07). Volume 07122 of Dagstuhl Seminar Proceedings. (2007)

An Approach for Virtual Organizations' Dissolution

Nicolás Hormazábal[1], Henrique Lopes Cardoso[2], Josep Lluis de la Rosa[1], and Eugénio Oliveira[2]

[1] Universitat de Girona, Agents Research Lab,
Av. Lluis Santaló S/N, Campus Montilivi, Edifici PIV, 17071 Girona, Spain
nicolash@eia.udg.edu, peplluis@eia.udg.edu,
[2] Universidade do Porto, LIACC, DEI / Faculdade de Engenharia,
R. Dr. Roberto Frias, 4200-465 Porto, Portugal
hlc@fe.up.pt, eco@fe.up.pt

Abstract. Current research on Virtual Organizations focuses mainly on their formation and operation phases, devoting only short references to the dissolution phase. These references typically suggest that dissolution should be obtained when the organization has fulfilled all its objectives or when it is no longer needed. This last definition is quite vague and hard to tackle with, as the need for an organization is not always easy to measure.
We believe that, besides fulfillment of objectives, more causes should be considered for the dissolution of a Virtual Organization; not always is an organization capable of achieving its goals, neither continuing operating. Organizations can change during their operation as well as the environment they operate in, and these changes may affect on their performance to the point that they should not continue operating, and the causes that should lead to the dissolution could affect also to future organizations' formation. Considering Virtual Organizations correspondence to real society organizations, some features from real-world commercial law related to dissolution can be applied to their virtual counterpart too. In this paper we introduce the different causes that should be considered for Virtual Organization dissolution, and a case study focused on one of these causes is presented as a way to emphasize the significance of the dissolution process.

1 Introduction

Generally speaking, Virtual Organizations (VO) are composed of a number of autonomous agents with their own capabilities (on problem-solving, task execution and performance) and resources. Being autonomous, agents usually pursue individual goals, but in some cases these goals can be achieved with better performance or higher benefits inside a cooperation environment with other agents, where the resulting organization can even offer new services through the combination of complementary competences. For example, in an economic environment, agents may represent different units or enterprises that come together

in response to new market opportunities requiring a combination of resources that no partner alone can fulfill [1]. These cooperative organizations have been researched mainly from the point of view of their formation and operation. However, their lifecycle has been outlined as having an additional phase, therefore comprising *formation*, *operation* and *dissolution*.

Although the automation of a dissolution process has been mentioned as a research and development challenge on the study of VOs [2], there is not much work addressing dissolution. This phase is often overlooked without getting into deeper research. Yet, in economic terms, if an organization's dissolution is not properly managed, it can generate tremendous costs [3]. The timeliness of dissolution is dictated by the existing agents and resource availability. If a VO is underperforming without a chance for reconfiguring itself (or the possible reconfiguration is not enough to improve the performance), then it should dissolve in order to free assigned resources and members.

Under normal circumstances, the dissolution should happen after the VO has fulfilled its objectives [4]. Some researchers also mention that such partnerships should dissolve when they are not longer sustainable [5] or the VO is no longer needed. The main topic of this paper is the clarification of these terms, through an identification of the causes that should be considered for the dissolution of a VO.

The paper is organized as follows. Section 2 briefly describes some real life organizations and their normative environment used to set the basis for the dissolution process in Virtual Organizations. Section 3 describes the normative framework used for supporting the dissolution process. Section 4 explains the dissolution process, describing the steps needed for the dissolution and the causes that a VO should dissolve. Section 5 present a case study focused on one of the dissolution causes presented. Finally in section 6 the conclusions of the current work are presented.

2 Real-World Organizations

In virtual environments, agent societies enable interactions between agents and are therefore the virtual counterpart of real-life societies and organizations [6]. As such, when seeking to support VO dissolution, issues related with real life organizations' dissolution should be considered.

The most common type of regulated organization that exists in society is the commercial organization, such as limited or public limited companies. These organizations are regulated by the law, and therefore they live inside a normative environment enforced by its respective institution. Every country has its own laws, but there are several common key features among western countries that can be used for reference. We shall use Spanish Commercial Law ([7], [8]) as a starting point, specifically in what concerns the dissolution of this type of commercial organization.

The dissolution of a commercial organization is divided in two phases. First there is the identification of a *dissolution cause*. In some cases, the agreement of

the organization's members is also needed to move forward to the next phase. The second phase is the *liquidation*, where once a dissolution cause is identified, the society moves forward to perform the tasks needed to its final extinction, having as output a dissolution report which summarizes the organization's activity.

From the above, the dissolution causes can be classified in two different groups:

- Causes that when identified, dissolve automatically the society without the need of the members' (or the board) agreement.
- Causes that when identified, need an agreement from the members (or the board) for going on to the next step, the *liquidation*.

These causes depend, besides on the law itself, also on the contents of the society's articles of association, their statutes (where, for example, the duration of the society is specified, in case the partners decide to have a fixed one) or the society's assets. The law may also include slightly different legislations on some aspects depending on the society's scope.

Institutions regulate interactions between the members of a society, defining the "rules of the game", on what is permitted and what is forbidden and in what conditions [9]. Similarly, a VO needs to operate within a normative environment, enforced in this case by an Electronic Institution (EI from now on) which is the electronic counterpart of the real-life institutions.

3 Normative Framework

Commercial organizations are restricted by a legal context where they operate in, and internally by the statutes or articles of association created during the organization's formation. There are, then, different normative layers related to the organizations' activities, first a common set of norms for every organization (the law) and then specific norms for each one of them (the statutes or articles of association). An institutional normative framework should therefore include a hierarchical organization of norms. Borrowing from [11], we consider norms organized in three levels (see Figure 1).

The EI aims to support agent interaction as a coordination framework and provides a level of trust by offering an enforceable normative environment. This means that the EI will facilitate both the creation and the enforcement of contracts among agents [12]:

- Institutional norms, at the higher level, influence the formation of VO constitutional and operational contracts; they setup the normative background on which cooperation commitments can be established. Regulations on general contracting activities and the behavior of every agent in the EI are included in this level.
- Constitutional norms represent the core of the cooperation agreement between the agents. The agreement is represented with norms that regulate the created coalition, which usually exists for a period of time. Norms at this level only affect the agents that participate in the VO.

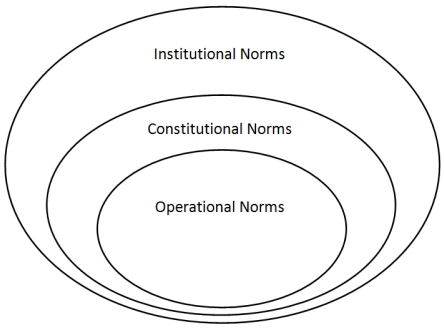


Fig. 1. Normative Framework.

– Operational Norms indicate the actions to be performed by contractual agents by specifying operational contracts, which may be established among a subset of the VO's agents.

Drawing a parallel between the real life organizations (like commercial organizations) and the EI framework, institutional norms map the commercial law, constitutional norms correspond to the organization's articles of association or statutes, and the operational norms represent the individual task commitments inside the organization (table 1).

Table 1. Parallel between societies and EI

Real Life Societies	Electronic Institution Framework
The Law	Institutional Norms
Statutes	Constitutional Norms
Task Commitments	Operational Norms

The VO activity is therefore governed by norms placed on different layers on the institutional normative framework. Focusing on the dissolution phase of a VO lifecycle, there should be some norms related to the identification of when a VO has to be dissolved, thus helping to identify the dissolution causes.

4 Dissolution Process

Inspired by the commercial law, in this work we suggest a two step dissolution process. First the dissolution activation (which will be called *activation*), consisting on the identification of a dissolution cause for the VO, and then the execution of the dissolution process, where the needed tasks for the dissolution have to be ran (this step will be called *liquidation*).

4.1 Activation

On the current literature, the causes for VO dissolution are mainly on the successful achievement of all its goals or a decision of involved partners to stop the operation [10]. But if the partners decide to stop the operation of the VO, they should someway specify the cause of the decision; if the organization is ending its activities before fulfilling its goals, this could be considered as an unsuccessful state. This information should be used for future organization formation and partner selection.

Before dissolve, VOs can attempt to adapt themselves to environmental changes or perform a reorganization in order to maintain or improve the performance due to different causes. This means that not always the right choice is to move forward to the dissolution, or otherwise in some cases maybe is better to dissolve instead of trying to make a VO reorganization.

We suggest then to distinguish two type of dissolution causes: First the causes that need the decision of the involved members for moving onto the dissolution which will be called *Necessary Causes*, as they are necessary for the dissolution, but not sufficient as they need the member's agreement.

Additionally, there are some causes that should automatically dissolve the organization without the need for the partners' decision. These causes are the *Sufficient Causes*.

During the VO operation, *necessary* or *sufficient* causes could be identified, which could lead the VO to different dissolution sub-states (figure 2). If a *sufficient cause* is identified, the VO goes directly for the *liquidation*, the mandatory step before the complete dissolution where the organization enters into an *on-liquidation* sub-state until it finishes the related tasks. But if a *necessary cause* is identified, the VO goes to a *pending dissolution* sub-state, where the VO waits for the partners' confirmation for the dissolution, or for the VO modifications (the adaptation or evolution of the VO) that will avoid the dissolution and make the VO return to the operation phase. If no measures are taken for returning to the operation phase after a period of time defined by the EI, the VO dissolves going to the *on-liquidation* sub-state.

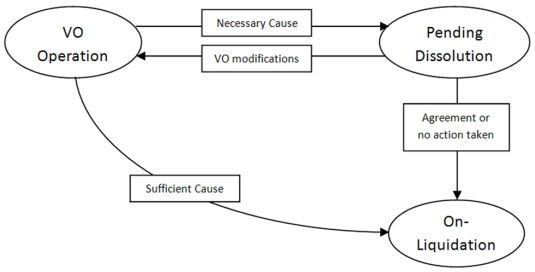


Fig. 2. Dissolution Sub-States.

In short during the dissolution, if a *sufficient cause* is detected, the organization goes for the *liquidation*. If a *necessary cause* is detected and no actions on the VO are taken to solve the issues related to the dissolution cause, the VO goes for the *liquidation*.

Sufficient Causes. *Sufficient causes*, once identified, are sufficient for the automatic dissolution of the VO. The causes of this type that we identify are:

Deadline: On the VO cooperation agreement created during its formation, the duration of the organization can be specified. During the operation of a VO, partners can modify their own norms, their cooperation agreement, so as well they can extend the lifespan of the organization, but once it is reached, the organization should dissolve as it was created for this duration.

Reduction: During the formation of a VO, the agents specify on the cooperation agreement the resources they are willing to devote for the organization. This is what defines the organization's assets: The total amount of resources an organization has. The EI should establish the minimum required resources for a VO to be considered as such. If for some reason the VO suffers a reduction of its resources below the minimum, the VO dissolves. For example, in a football (soccer) team the minimum resources for a team are 7 players, below that number it is not a team anymore.

Agreement: As we cannot discard the case where the VO partners arbitrarily decide to dissolve the organization, the agreement for the dissolution should be considered too. For that, a minimum percentage (typically over 50%) of partners must decide to dissolve the organization.

Necessary Causes. *Necessary causes* are necessary, but not sufficient. For being sufficient, they need the agreement of the VO partners. Putting it another way, the partners have to take actions to prevent the dissolution.

Fulfillment: As mentioned before, the dissolution can be reached by the successful achievement of all the VO goals. During the formation of the VO, agents must define the organization's goals on the cooperation agreement. Once they are fulfilled, the Institution can be dissolved. The reason that this is a *necessary cause* and not a *sufficient* one is that once the goals have been achieved the agents can evaluate if they want to set new ones based on the performance and continue operating.

Unfeasibility: There are some cases when a VO cannot fulfill its goals. This could happen by internal issues like the lost of key resources for achieving all the goals, or could happen by external causes like changes on the environment that affects the organization, like for example the arrival of a new organization that competes for the same goals. The VO can make changes to improve its performance, change its goals or add new resources, among other measures, to prevent the dissolution.

Inactivity: For any reason, it could happen that the VO could show no activity during a period of time; after a specified period, the organization could be considered as idle or dead, after that it could go on to the dissolution phase.

Loss: This dissolution cause has sense only when the benefits of the VO are measurable and in the same unit that the assets specified on the VO formation (see the *Reduction sufficient cause* above). On the cooperation agreement, the organizational assets are specified based on the resources that each member is willing to spend on. If, during the operation of the VO, instead of benefits there are losses and these losses are over the half of the organizational assets, the VO can be dissolved as it can be considered unviable.

Some examples of possible action to take on the VO for avoiding the dissolution after a *necessary cause* has been identified:

- New goal definition or reallocation of the resource and agent assignment for the tasks.
- Addition of new agents to the VO or replacing partners.
- Force the resume of the VO activities after a period of inactivity.
- Modify the VO assets by adding new resources (or removing them).

In short, there are seven different dissolution causes, grouped by *sufficient causes* and *necessary causes* (table 2).

Table 2. Dissolution Causes

Sufficient Causes	Deadline Reduction Agreement
Necessary Causes	Fulfillment Unfeasibility Inactivity Loss

Activation on Electronic Institution Framework In the different layers of the EI normative framework (from section 3), we should have norms that support the VO dissolution, more specifically at the institutional and constitutional levels. Institutional norms should contain at least four values for the dissolution support, which will be called *dissolution support elements*:

- Minimum Resources (R): The minimum resource requirements that a VO needs to have to be considered as such. The VO assets have to be greater than this value.
- Time of inactivity (Ti): The time that a VO has to be inactive before considering its dissolution.

- Maximum loss over assets (Ml): The maximum percentage of loss over the VO's initial assets before considering its dissolution.
- Minimum votes for the majority (V): Here is specified the default value for the minimum percentage of the total number of participants need to agree on the dissolution can be specified.

These values in the top-level of the norms hierarchy (Institutional Norms) can be context-dependent. The grouping of predefined norms through appropriate contexts mimics the real-world organization of legislations applicable to specific activities [13]. So depending on the type of organization, they could have some different *dissolution support elements*.

The following is an abstraction of concepts that should be included in a VO contract. Regarding the constitutional norms, the VO contract should include at least the VO duration D; a starting and ending dates for the VO operation. The contract structure should contain the Cooperation Effort each agent has committed to as a result of the negotiation process prior to the VO formation. For each agent A_i, with the assigned resources R_k, based on the cooperation effort structure specified on [11]:

$$CoopEff = \{\langle A_i, R_k, W\rangle\}$$
$$W = \langle MinQt, MaxQt, Freq, UnitPr\rangle$$

W represents the workload for each participant agent A_i specified between a minimum ($MinQt$) and a maximum value ($MaxQt$), with a frequency ($Freq$) during the lifetime of the organization and the unit price ($UnitPr$) the agent has assigned to perform the assigned workload.

The frequency depends on the unit used for measuring the VOs' duration (i.e. days, weeks, computer cycles), which depends on the VO's scope. For example when the duration unit is *days*, if the workload is specified for each week then the frequency $Freq$ is 7 (every seven days).

The significance of the cooperation effort for the dissolution is that with it, the organizational asset Oa of the organization can be calculated, given the total duration of the organization D for each agent A_i in the VO:

$$Oa = \sum_{A_i} MaxQt * UnitPr * \frac{D}{Freq}$$

This organizational asset will be used to evaluate the *Reduction* and *Loss* dissolution causes.

Each one of the dissolution causes depends on one normative level (table 3) except for *Reduction* and *Loss* which depend on both Institutional and Constitutional norms, as they depend on the initial VO assets (on the constitutional norms) and on a minimum value specified on the institutional norms in the case that the VO has not redefined it for itself.

Table 3. Dependence between dissolution causes and normative framework levels

Normative Level	Dissolution Cause
Institutional Norms	Agreement Inactivity Reduction Loss
Constitutional Norms	Deadline Fulfillment Reduction Loss

Unfeasibility is a different case. Although it can be considered as a constitutional norms dependent cause, the truth is that is more complicated to identify than observing the assigned resources for each VO goal. A VO could find itself in a situation where it cannot fulfill its objectives for causes beyond the organization itself. Sometimes for external causes, the VO performance could decrease and the organization should adapt to the environment, making modifications by reconfiguring itself (some authors introduce a separate phase for adaptation and others mention the adaptation as a part of the operation phase) or dissolve. Tools for monitoring the VO are needed for identifying cases like the *Unfeasibility* one, which once identified can avoid a useless extension of the operation time of the VO if the expected results are to be negative.

4.2 Liquidation

The *liquidation* is the last step before the complete dissolution of the VO. Every running task must be stopped and 'freeze' the VO activity for realizing the *liquidation* step. The organization goes to an *on-liquidation* sub-state inside the dissolution phase (see figure 2).

During the organization's operation, a profit and expenses log must be maintained, which will allow creating the final balance during this step. Some of the other main aspects that should be supported [10] are:

- Definition of general liabilities upon the dissolution of the VO.
- Keeping track of the individual contributions to a product/service that is jointly delivered (namely in terms of the quality and product life cycle maintenance).
- Redefinition/ceasing information access rights after ceasing the cooperation.
- Assessing the performance of partners, generating information to be used by partner selection tools in future VO creation.

This last item is especially relevant, as it does not only support the formation of future VOs, but can also support the identification of dissolution causes based (such as *unfeasibility*) on past experiences. An organization can use this information to identify if it is possible to fulfill its objectives given its status on a specific time.

For evaluating the partners' performance it is better not to make a single evaluation at the dissolution time, but at several times during the organization's lifespan in order to have a complete picture of the performance evolution. Three fixed times are recommended for evaluating the organization: At the moment of its formation, at the half of its expected lifespan and at the end, before dissolving [14]. Additionally, new evaluations should be made if key elements are changed on the VO like the Cooperation Agreement.

The evaluation of performance depends on the VO's scope. A suggestion of the evaluation elements is:

$$Ev = \langle Time, CA, Ben, Exp, Wf, Wr \rangle$$

Where:

- $Time$: The time when the evaluation has been made.
- CA: The VO Cooperation Agreements.
- Ben, Exp: A balance of the VO's benefits and expenses.
- Wf: The workload (in time or price unit) used for the fulfilled tasks.
- Wr: The expected workload needed for fulfilling the remaining tasks.

The output of the *liquidation* process should be a Dissolution Report (DR), which will contain all the evaluations made during the organization's lifespan Evs, together with the dissolution cause DC. Additionally, it can contain an assessment Sc (a score between 0 and 1) from each agent A_i evaluating the VO's performance based on the fulfillment on their individual goals. A suggestion for the content of the dissolution report DR:

$$\begin{aligned} DR &= \langle Evs, DC, Vals \rangle \\ Evs &= \{Ev_1, Ev_2, ..., Ev_n\} \\ Vals &= \{Val_1, Val_2, ..., Val_n\} \\ Val_i &= \langle A_i, Sc \rangle \\ DC \in \{&Deadline, Reduction, Agreement, \\ &Fulfillment, Unfeasibility, Inactivity, Loss\} \end{aligned}$$

This dissolution report, stored on a knowledge base, will allow supporting future VO formation and partner selection, giving information on the performance (from the benefits and expenses) and evaluation of each agent , and for the cases that the VO has not fulfilled its objectives, provides also information on the causes of that.

5 Unfeasibility Case Study

We developed a simple digital environment for simulating the creation of agent's organizations, for testing a way to identify the *unfeasibility* dissolution cause.

In this environment, agents form organizations (as the idea is to focus only on the dissolution, the organization formation process is done automatically) with a fixed duration (in time steps), after then the organization dissolves.

The mechanism is simple: agents move and interact asynchronously through a grid space (which represents the environment), and when they find another agent in their neighborhood (nearer than two cells), they send a message proposing the creation of an organization. In the next time step, asked agents answer if they accept or not. Every agent in the system offer a single (not unique) service, where the advantage of forming an organization lies in that two agents together can offer their own service plus their service combination, expanding their own markets.

The idea is to demonstrate the utility of supporting tools to automate the dissolution causes identification, and how the dissolution can affect in the overall system performance comparing the results with cases without the *unfeasibility* cause. Also here, agents have a transitional step between the *dissolution activation* and the *liquidation* for deciding if to proceed or not based on the evaluation results on the organization's performance.

Each organization, at the moment of their dissolution will generate a *dissolution report* containing evaluations of the organization at different time periods. Each evaluation will contain only the benefits from the last evaluation (or the benefits so far if it is the first evaluation), the diversity of the offered services and the timesteps passed from the last evaluation. These evaluations will be generated at three time periods of the VO's lifespan: At the first third of its expected lifespan, at the second third, and at the moment of its dissolution when the *dissolution report* containing the evaluations is created (i.e. if a VO has a fixed lifespan of 30 steps, the report will contain evaluations of the VO's benefits at steps 10, 20 and 30). If an organization decides to extend its lifespan, new evaluations will be added to the report.

A knowledge base with previous cases will be used to identify when the agents' expectations will probably not be fulfilled, at first this knowledge base will be empty and it will be filled with the *dissolution reports* each dissolved organization generates.

For the simulation, the following hypotheses related to the agents have been made:

1. Each agent offers a single service.
2. Agents who coalesce are more probable to get benefits. To the extreme that, for this case, single agents get no benefits.
3. When agents coalesce, there are three options related to the organization's lifespan: a) set a fixed lifespan, b) do not fix a lifespan and c) set an initial lifespan which can be changed.
4. In the specific negotiation scenario, at least two agents coalesce; one agent who makes an offer for creating an organization and one or more who receive the offer. Each offer has a 50% of chances for being accepted. This is for simplifying the negotiation process while still having the chance of offer refusals.

As for the calculated benefits and organization services, it is assumed that:

1. Two or more agents offering the same service can't be part of the same organization.
2. The benefits are calculated based on the services an organization offers and the demand they have.
3. The organizations will offer the individual services of each member agent, as well the combination of these services. For example, if an organization is composed of two agents, which respectively offer the services A and B, the organization will offer the services A, B and A+B (figure 3)
4. Every service has the same base demand, as well as the combined services.
5. The demand of a service depends of the competition that this service offering has (how many organizations offer the same service). For example, if an organization offers the services A, B and A+B, and another active organization offers the services A, C and A+C, there will be two competitors for the service A.

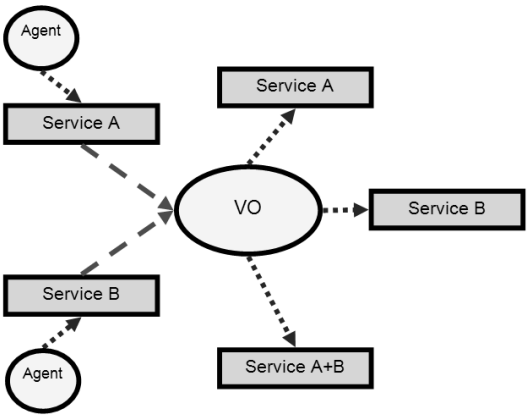


Fig. 3. Services on an organization.

Benefits for each time step are calculated by the following equation:

$$E = \sum_i (\frac{B}{C_i} + N)$$

Where:

- E are the total earnings or benefits of the organsiation at each step.
- B is the base earning for each service i.
- C_i is the number of organizations that offer the same service i (including the organization which its earnings are being calculated).

– N is a random number from a normal distribution with average 0 and variance (B/2).

This implies that the greater the diversity on the services an organization offers, the lower competition and the higher benefits it will likely get.

The organization's goal is to get at each timestep a minimum "'acceptable"' benefit E above B/5; if it identifies that the goal is not achievable, *unfeasibility* cause is detected. On the other hand, if the organization estimates that its expected benefits can be over B/2, it considers to extend its liefetime as the expected benefits are good.

For supporting the dissolution cause identification, a knowledge base with previous cases will be used. In this experiment we will use a case based algorithm (which from now on will be referred as the algorithm) to identify cases when it's better to dissolve the organization if the goal cannot be fulfilled, which means that it founds itself in an *unfeasibility* case. The same algorithm will be used when the organization's lifespan is about to reach its end, identifying if it's better to extend it as the benefit expectancies are good, instead of proceeding to the *liquidation*.

As said before, during the organization's *dissolution*, a *dissolution report* will be created and stored on the knowledge base with different evaluation cases containing the VO's benefits, services diversity and the timestep when the evaluation was made.

The algorithm, in its retrieving step, will identify pairs of consecutive evaluations similar to the current and last evaluations. Once a similar case is found, the algorithm will try to predict the following state based on the past case and evaluate, reusing the past similar case, which is the best action to do for the organization: if it is better to continue operating by extending its lifespan or dissolve.

The similarity for the algorithm is calculated by:

$$Sim = (Div_k * w_1 + Ben_k * W_2) + (Div_{k-1} * w_1 + Ben_{k-1} * W_2)$$

Where:

– Div is the diversity similarity at a time k and a time $k-1$. This value is calculated by the percentage difference on the amount of different agent types (identified by the service they offer) that are member of the organizations. For example: Having in one case 4 different agents in an organization, and in another 5, then the diversity similarity will be $4/5 = 0,8$.
– Ben are the benefits similarity per time step at a time k and a time $k-1$. Is calculated by the same method above, but using the benefits per step instead of the number of different agents.
– w_n are the respective weights for the similarity values. For this case, the weight will be equal for every similarity value.

In the knowledge base there must be an evaluation at a time $k+1$ in order to estimate the future benefits given the current state.

To identify positive cases (when it seems the goal can be fulfilled for the next timesteps) from the negative ones (when the goal cannot be fulfilled), the algorithm will compare the earning expectations with the benefits found in the similar past cases from the knowledge base, reusing them.

5.1 Setup

The simulation environment has been developed in RePast[3]. RePast is an open source agent modeling toolkit developed in Java which provides different tools for tracking and displaying agents' and environment values. And the tests were done in a grid of 50x50 cells, with 500 different agents that can offer one of the 10 different services. The base earning for each service was fixed in 1, and the default duration time of an organization was 15 time steps. It was tested during 10.000 time steps through three different experiments:

Experiment 1: Organizations start with a defined lifespan, which it can be extended or reduced, supported by the algorithm.

Experiment 2: Organizations have an unlimited lifespan, so new organizations can never be dissolved, as agents only get benefits when they are part of an organization (from hypothesis 2), this could be a reasonable strategy to guarantee benefits for each agent at each timestep once they have formed an organization, as opposite the other experiments where due the organization's dissolution, there are agents without organizations wandering in the grid without getting benefits more often.

Experiment 3: Organizations have a fixed lifespan which cannot be modified, so they dissolve always when the expected deadline is reached.

5.2 Results

After ten runs of 10.000 steps for each experiment, the results on the average benefits at each step can be seen on figure 4. After the step 8.600 the benefits per step seem to stabilize and reach the 98% of the steady value, so for the results will be considered for the average benefits, from the step 8.600 onward. The average benefits per step are in the table 4.

Table 4. Average benefits per timestep from the step 8600 onward

	Average Benefits	Std. Deviation
Experiment 1	1.530,04	12,69
Experiment 2	997,21	13,35
Experiment 3	543,77	16,26

There is a significant improvement when the algorithm supports the identification of the unfeasibility dissolution cause of the organization and when the

[3] `http://repast.sourceforge.net`

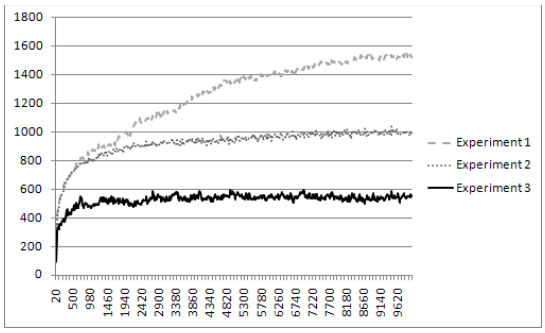


Fig. 4. Average organization's benefits per step, 3 experiments, 10 runs, 10.000 steps each.

organization is allowed to modify its own lifespan (experiment 1). In the experiment 2 there are not many agents out of a organization, and in consequence most of them are getting benefits, but this does not guarantee that they are in the best possible organization, as maybe they are better out of the organization searching for new ones without making benefits instead being part of a bad performing one. The unfeasibility dissolution cause in this case, not only helps to prevent the organizations operate when the goals cannot be achieved, but also if the goals are related to the benefits, helps to improve the overall performance.

6 Conclusions

VOs have been approached from different perspectives, but most of these approaches are focused mainly on the first phases of their lifecycle (*formation* and *operation*), leaving the *dissolution* as an unresolved issue pending for future work. The current paper makes an approach for this phase, presenting it as a two steps phase (*activation* and *liquidation*), with two sub-states (*pending dissolution* and on *liquidation*).

One of the main contributions of this work is in the description of the causes originating dissolution, besides the VO goal fulfillment or the partner's decision to dissolve. We also provide elements from this dissolution process for supporting future VO formation, as the resulting dissolution report from the *liquidation* step, which could be significant for the future partner selection and for the future identification of dissolution causes, like the unfeasibility which can be identified by experience from past similar cases (see section 5).

Dissolution prevents the operation of bad performing or unnecessary organizations, and can improve the overall performance by correctly identifying when an organization should no longer operate.

Not all the dissolution causes are mandatory for dissolving the VO; some of them need the partners' approval for going on to the dissolution as they can be also a cause for the VO reconfiguration. The VO formation phase should consider

new issues during the negotiation process, related to the norms for *dissolution* phase.

Finally, the basis for the dissolution process has been inspired from real-world organizations' dissolution, and because of this a normative framework is needed for supporting the dissolution process with a similar structure from real life norms (the law at a higher level, and the organizations' statutes below). Although the commercial law is used as an inspiration, this approach is not restricted to economic-based organizations; assets, costs or benefits are not restricted to economical approaches, as they can be identified within the amount of workload inside a VO.

The *dissolution* phase is not trivial, here is an approach to it and hopefully this work will fulfill the goal of emphasize its significance and provide a good reference for contributing on the formalization of the VO. Future work will be focused on completing the formalization of the dissolution phase.

References

1. Dignum, F., Dignum, V.: Towards an Agent-based Infrastructure to Support Virtual Organisations, PRO-VE '02: Proceedings of the IFIP TC5/WG5.5 Third Working Conference on Infrastructures for Virtual Enterprises, vol: 213, 363–370, (2002)
2. Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing), AgentLink, (2005)
3. Parunak, H. Van Dyke: Technologies for Virtual Enterprises, Agility Journal, (1997)
4. Katzy, B., Zhang, C., Löh, H.: Reference Models for Virtual Organisations Virtual Organizations Systems and Practices, 45–58, Springer US, (2005)
5. De Roure, D., Jennings, N.R., Shadbolt, N.R.: The Semantic Grid: Past, Present, and Future, Proceedings of the IEEE, vol: 93/3, 669–681, (2005)
6. Dignum, V., Dignum F.: Modelling Agent Societies: Co-ordination Frameworks and Institutions, Progress in Artificial Intelligence, vol: 2258/2001, 7–21, (2001),
7. Ley de Sociedades Anónimas, Texto Refundido de la Ley de Sociedades Annimas, Aprobado por el RDLeg 1564/1989, de 22 de diciembre, BOE del 27/12/1989, (1989)
8. Ley de Responsabilidad Limitada, Ley 2/1995, de 23 de marzo, BOE del 24/03/1995, (1995)
9. Esteva, M., Rodrguez-Aguilar, J. A., Sierra, C., Garcia, P., Arcos, J. L.: On the Formal Specification of Electronic Institutions, Agentmediated Electronic commerce: The European AgentLink Perspective, LNAI 1991, 126–147, (2001)
10. Camarinha-Matos, L.M. and H. Afsarmanesh, H.: Virtual Enterprise Modeling and Support Infrastructures: Applying Multi-Agent Systems Approaches, Lecture Notes in Artificial Intelligence LNAI 2086, 335–364, (2001)
11. Lopes Cardoso, H., Oliveira, E.: Virtual Enterprise Normative Framework Within Electronic Institutions, Engineering Societies in the Agents World V, 14–32, (2005)
12. Lopes Cardoso, H., Oliveira, E.: Electronic institutions for B2B: dynamic normative environments, Artificial Intelligence and Law, vol: 16, num: 1, 107–128, (2007)
13. Lopes Cardoso, H., Oliveira, E.: A Contract Model for Electronic Institutions, Proceedings of The AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN@AAMAS'07), 73–84, (2007)
14. Collier, B., DeMarco, T., Fearey, P.: A Defined Process For Project Postmortem Review, IEEE Software, vol: 13, num: 4, (1996)

A Normative Multiagent Approach to Requirements Engineering

Guido Boella[1] and Leendert van der Torre[2] and Serena Villata[1]

[1] Dipartimento di Informatica, University of Turin, Italy.
{boella,villata}@di.unito.it
[2] Computer Science and Communication, University of Luxembourg.
leendert@vandertorre.com

Abstract. In this paper we present a new model for the requirements analysis of a system. We offer a conceptual model defined following a visual modeling language, called dependence networks. TROPOS [9] uses dependence networks in the requirements analysis, in this paper we propose to extend them with norms. This improvement allows to define a new type of dependence networks, called conditional dependence networks, representing a new modeling technique for the requirements analysis of the system. Our model, moreover, allows the definition of the notion of coalition depending on the different kinds of network. We present our model using the scenario of virtual organizations based on a Grid network.

1 Introduction

The diffusion of software applications in the fields of e-Science and e-Research underlines the necessity to develop open architectures, able to evolve and include new software components. In the late years, the process of design of these software systems became more complex. The definition of appropriate mechanisms of communication and coordination between software components and human users motivates the development of methods with the aim to support the designer for the whole development process of the software, from the requirements analysis to the implementation.

The answer to this problem comes from software engineering that provided numerous methods and methodologies allowing to treat more complex software systems. One of these methodologies is the TROPOS methodology [9], developed for agent-oriented design of software systems. The intuition of the TROPOS methodology [9] is to couple, together with the instruments offered by software engineering, the multiagent paradigm. In this paradigm, the entities composing the system are agent, autonomous by definition, characterized by their own sets of goals, capabilities and beliefs. TROPOS covers five phases of the software development process: early requirements allowing the analysis and modeling of the requirements of the context in which the software system will be inserted, late requirements describing the requirements of the software system, architectural and detailed design of the system and, finally, the code implementation.

The TROPOS methodology [9] is based on the multiagent paradigm but it does not consider the addition of a normative perspective to this paradigm. Since twenty years, the design of artificial social systems is using mechanisms like social laws and norms

to control the behavior of multiagent systems [3]. These social concepts are used in the conceptual modeling of multiagent systems, for example in requirements analysis, as well as in formal analysis and agent based social simulation. For example, in the game theoretic approach of Shoham and Tennenholtz [19], social laws are constraints on sets of strategies. In this paper, we propose to add norms, presented thanks to the normative multiagent paradigm, both to the requirements analysis phases and to the conceptual meta-model. This paper addresses the following research question:

– How to apply a normative multiagent approach to the early and late requirements analysis?

Our approach is based, following the approach of TROPOS [9], on a semiformal language of visual modeling and it is composed by the following components. First, as shown in the UML diagram of Figure 4, we present our ontology that defines the set of concepts composing our conceptual metamodel. The elements composing the ontology are agents, goals, facts, skills, dependencies, coalitions with the addition of the normative notions of roles, institutional goals, institutional facts, institutional skills, dynamic dependencies and obligations, sanctions, secondary obligations and conditional dependencies. Second, our model is defined as a directed labeled graph whose nodes are instances of the metaclasses of the metamodel, e.g., agents, goals, facts, and whose arcs are instances of the metaclasses representing relationships between them such as dependency, dynamic dependency, conditional dependency. Finally, we have a set of rules and constraints to guide the building of the main concepts of the metamodel, e.g. the formation of coalitions and their stability is constrained to the kind of dependencies linking its members. In TROPOS [9], the requirements analysis is split in two main phases, the early requirements and the late requirements. In our methodology, these two phases share the same conceptual and methodological approach, thus we will refer to them just as requirements analysis.

We introduce the normative issue of obligations, representing them directly in dependence networks. This introduction allows the definition of a third kind of modeling called conditional dependency modeling based on the structure of conditional dependence networks. Conditional dependence networks represent obligations as particular kinds of dependencies and these obligations are related to notions by means of sanctions if the obligation is not fulfilled and contrary to duty when the primary obligation, not fulfilled, actives a secondary obligation. Moreover, we introduce the notion of coalition and we propose to use methods of social order such as obligations and sanctions to efficiently achieve the maintenance of the stability and the cohesion of these groups. Our model is intended to support the requirements specification for high level open interaction system where heterogeneous and autonomous agents may interact.

Our model is not intended to support all analysis and design activities in software development process, from application domain analysis down to the system implementation as in the TROPOS methodology [9], but only the requirements analysis phases which involve dependence networks. This paper is organized as follows. Section 2 describes a Grid computing scenario . In Section 3, we present the dependency and the dynamic dependency modeling while in Section 4 we present a new kind of dependence network, called conditional dependence network. Related work and conclusions end the paper.

2 The Grid Scenario

The Grid Computing paradigm provides the technological infrastructure to facilitate e-Science and e-Research. Grid technologies can support a wide range of research including amongst others: seamless access to a range of computational resources and linkage of a wide range of data resources. It is often the case that research domains and resource providers require more information than simply the identity of the individual in order to grant access to use their resources. The same individual can be in multiple collaborative projects, each of which is based upon a common shared infrastructure. This information is typically established through the concept of a virtual organization (VO) [14]. A virtual organization allows the users, their roles and the resources they can access in a collaborative project to be defined. In the context of virtual organizations, there are numerous technologies and standards that have been put forward for defining and enforcing authorization policies for access to and usage of virtual organizations resources. Role based access control (RBAC) is one of the more well established models for describing such policies. In the RBAC model, virtual organization specific roles are assigned to individuals as part of their membership of a particular virtual organization.

As presented by Zhao et al. [25], obligations are requirements and tasks to be fulfilled, which can be augmented into conventional systems to allow extras information to be specified when responding to authorization requests. For example in [25], administrators can associate obligations with permissions, and require the fulfillment of the obligations when the permissions are exercised. The general idea of the RBAC model is that, permissions are associated with functional roles in organizations, and members of the roles acquire all permissions associated with the roles. Allocation of permission to users is achieved by assigning roles to users. Failure of the fulfilling an obligation will incur a sanction.

Some of the main features of a node in a Grid are reliability, degree of accepted requests, computational capabilities, degree of faults and degree of trust for confidential data. These different features set up important differences among the nodes and the possible kinds of coalitions that can be formed and maintained. Reciprocity-based coalitions can be viewed as a sort of virtual organizations in which there is the constraint that each node has to contribute something, and has to get something out of it. The scenario of virtual organizations based on Grid networks represents a case study able to underline the benefits of a normative multiagent paradigm for requirements analysis. First of all, in the normative multiagent paradigm as well as in the common multiagent one, the autonomy of agents is the fixed point of all representations, i.e., the Grid philosophy imposes the autonomy of the nodes composing it. Second, the normative multiagent paradigm allows a clear definition of the notion of role and its associated permissions, i.e. the role based access control policy needs a design able to assign roles and represents to all the consequent constraints based on them. Third, the normative multiagent paradigm allows the introduction at requirements analysis level of obligations able to model the system. Fourth, the concept of coalition and the constraints introduced by this concept can model the concept of ”local network“ in virtual organizations. Finally, the presented modeling activities depict the system using structures similar to the Grid network itself.

3 Dependency and Dynamic Dependency Modeling

Figure 1 shows the ontology on which is based our model containing a number of concepts related to each other. We divide our ontology in three submodels: the agent model, the institutional model, and the role assignment model, as shown in Figure 1. Roughly, the institutional model represents sets of agents regulated by social norms. For more details, see [4]. The Figure depicts, following the legend of Figure 2, the three submodels which group the concepts of our ontology.

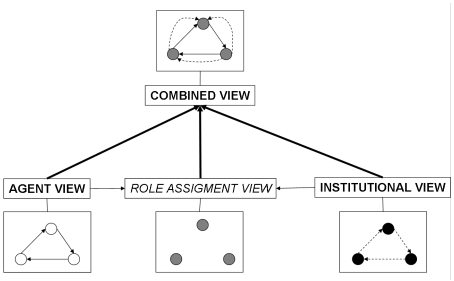


Fig. 1. The conceptual metamodel.

Such a decomposition is common in organizational theory, because an organization can be designed without having to take into account the agents that will play a role in it. Also, if another agent starts to play a role, for example if a node with the role of simple user becomes a VO administrator, then this remains transparent for the organizational model. Likewise, agents can be developed without knowing in advance in which institution they will play a role.

As shown in Figure 1, we add to the notions of agent, goal and capability composing the agent view, those related to the institutional view such as the notion of role and all its institutional goals, capabilities and facts. These notions are unified in the combined view and to each agent it is possible to assign different roles depending on the organization in which the agent is playing. In this way, early and late requirements can be based both on agents and on roles. Models are acquired as instances of a conceptual metamodel resting on the concepts presented in the following sections. For more details on the three conceptual submodels, see Boella et al. [5] and Boella et al. [4].

3.1 Dependency Modeling

Figure 2 shows the components of our model. Our model is a directed labeled graph whose nodes are instances of the metaclasses of the metamodel, e.g., agents, goals, facts, and whose arcs are instances of the metaclasses representing relationships between them such as dependency, dynamic dependency, conditional dependency.

Dependence networks [20] represent our first modeling activity consisting in the identification of the dependencies among agents and among roles. In the early requirements phase, we represent the domain stakeholders using these networks while in the

late requirements phase, the same kind of approach is followed representing the agents of the future system involved in the dependence network. Figure 2-(a) shows the graphical representation of the model obtained following this modeling activity, the *dependency modeling*. The legend describes the agents (depicted as white circles), the roles (depicted as black circles), the agents assigned to roles (depicted as grey circles), the agents'/roles' goals (depicted as white rectangles) and the dependency among agents (one arrowed line connecting two agents with the addition of a label which represents the goal on which there is the dependency). The legend considers dependencies among agents but they can be also among roles or agents assigned to roles.

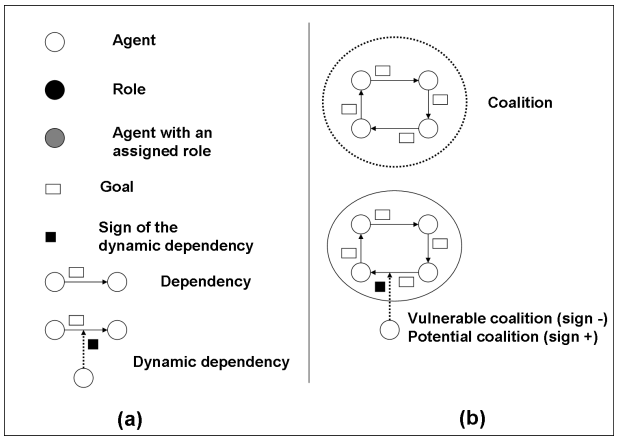


Fig. 2. Legend of the graphical representation of our model.

3.2 Dynamic Dependence Networks

Dynamic dependence networks have been firstly introduced by Caire et al. [11] and then treated in Boella et al. [5] in which the existence of a dependency depends on the actions of the agents which can delete it. Here, as shown in Figure 2-(a), we distinguish "negative" dynamic dependencies where a dependency exists unless it is removed by a set of agents due to removal of a goal or ability of an agent, and "positive" dynamic dependencies where a dependency may be added due to the power of a third set of agents. *Dynamic dependency modeling* represents our second modeling activity for requirements analysis. A formal definition of dynamic dependence networks is given in Boella et al. [4].

The legend of Figure 2-(a) describes the sign of the dynamic dependency (depicted as a black square) and the dynamic dependency among agents (depicted as one arrowed line connecting two agents with the addition of a label which represents the goal on which there is the dependency and another arrowed dotted line with the sign's label connecting an agent to the arrowed plain line that can be deleted or added by this agent). Figure 3 presents an example of dynamic dependence network on the Grid.

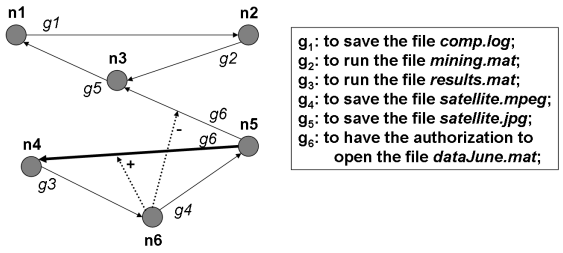


Fig. 3. An example of dynamic dependence network.

A coalition can be defined in dependence networks, based on the idea that to be part of a coalition, every agent has to contribute something and has to get something out of it. The graphical representation of coalitions is depicted in Figure 2-(b) which describes coalitions (depicted as sets of agents and dependencies included in a dotted circle) and vulnerable and potential coalitions (depicted as sets of agents and dependencies in a circle in which one or more of these dependencies can be added or deleted by another agent with a labeled dynamic dependency). Definition 1 makes a distinction between *coalitions* which are actually formed, *vulnerable coalitions* which can be destroyed by the deletion of dynamic dependencies and, *potential coalitions*, which can be formed depending on additions and deletions of dynamic dependencies.

Definition 1 (Coalition). *Let A be a set of agents and G be a set of goals. A coalition function is a partial function $C : A \rightarrow 2^A \times 2^G$ such that $\{a \mid C(a, B, G)\} = \{b \mid b \in B, C(a, B, G)\}$, the set of agents profiting from the coalition is the set of agents contributing to it. Let $\langle A, G, dyndep^-, dyndep^+, \geq \rangle$ be a dynamic dependence network, and dep the associated static dependencies.*

1. *A coalition function C is a coalition if $\exists a \in A, B \subseteq A, G' \subseteq G$ such that $C(a, B, G')$ implies $G' \in dep(a, B)$. Coalitions which cannot be destroyed by addition or deletion of dependencies by agents in other coalitions.*
2. *A coalition function C is a vulnerable coalition if it is not a coalition and $\exists a \in A, D, B \subseteq A, G' \subseteq G$ such that $C(a, B, G')$ implies $G' \in \cup_D dyndep^-(a, B, D)$. Coalitions which do not need new goals or abilities, but whose existence can be destroyed by removing dependencies.*
3. *A coalition function C is a potential coalition if it is not a coalition or a vulnerable coalition and $\exists a \in A, D, B \subseteq A, G' \subseteq G$ such that $C(a, B, G')$ implies $G' \in \cup_D (dyndep^-(a, B, D) \cup G' \in dyndep^+(a, B, D))$ Coalitions which could be created or which could evolve if new abilities or goals would be created by agents of other coalitions on which they dynamically depend.*

Figure 3 presents two different coalitions. On the one hand, we have the a *real* coalition composed by agents n_1, n_2 and n_3. On the other hand, we have a potential coalition, such as a coalition which could be formed if agent n_6 really performs the dynamic addition, making agent n_5 dependent on agent n_4.

4 Conditional Dependency Modeling

In this section, we answer to the question *how to introduce obligations in dependence networks* by defining the conditional dependency modeling. Normative multiagent systems are "sets of agents (human or artificial) whose interactions can fruitfully be regarded as norm-governed; the norms prescribe how the agents ideally should and should not behave. [...] Importantly, the norms allow for the possibility that actual behavior may at times deviate from the ideal, i.e., that violations of obligations, or of agents' rights, may occur" [6]. The notion of conditional obligation with an associated sanction is the base of the so called regulative norms. Obligations are defined in terms of goals of the agent and both the recognition of the violation and the application of the sanctions are the result of autonomous decisions of the agent.

A well-known problem in the study of deontic logic is the representation of contrary-to-duty structures, situations in which there is a primary obligation and what we might call a secondary obligation, coming into effect when the primary one is violated [18]. A natural effect coming from contrary-to-duty obligations is that obligations pertaining to a particular point in time cease to hold after they have been violated since this violation makes every possible evolution in which the obligation is fulfilled inaccessible. A classical example of contrary-to-duty obligations is given by the so called "gentle murder" by Forrester [13] which says "do not kill, but if you kill, kill gently".

The introduction of norms in dependence networks is based on the necessity to adapt the requirements analysis phases to model norm-based systems. An example of application of this kind consists in the introduction of obligations in virtual Grid-based organizations [25] where obligations, as shown in Section 2, are used to enforce the authorization decisions. On the one hand, in approaches like [25], obligations are considered simply as tasks that have to be fulfilled when an authorization is accepted/denied while, on the other hand, in approaches like [17], the failure in fulfilling the obligation incurs a sanction but there is no secondary obligation.

The introduction of obligations brings us to introduce a new kind of goal, the normative one. These goals originate from norms and they represent the obligation itself. We define a new set of normative concepts, based on Boella et al. [2] model of obligations, and we group them in a new view, called the normative view. The normative view is composed by a set of norms N and three main functions, $oblig$, $sanct$ and ctd representing obligation, sanctions and contrary-to-duty obligations. The UML diagram of Figure 4 provides a unified vision of the presented concepts of the ontology representing our conceptual metamodel.

Definition 2 (Normative View). *Let the agent view* $\langle A, F, G, X, goals : A \rightarrow 2^G, skills : A \rightarrow 2^X, rules^1 : 2^X \rightarrow 2^G \rangle$ *and the institutional view* $\langle RL, IF, RG, X, igoals : RL \rightarrow 2^{RG}, iskills : RL \rightarrow 2^X, irules : 2^X \rightarrow 2^{IF} \rangle$*, the normative view is a tuple* $\langle A, G, RG, N, oblig, sanct, ctd \rangle$ *where:*

- A *is a set of agents,* G *is a set of goals,* RG *is a set of institutional goals;*
- N *is a set of norms;*

[1] $rules$ and $irules$ associate sets of (institutional) actions with the sets of (institutional) facts to which they lead.

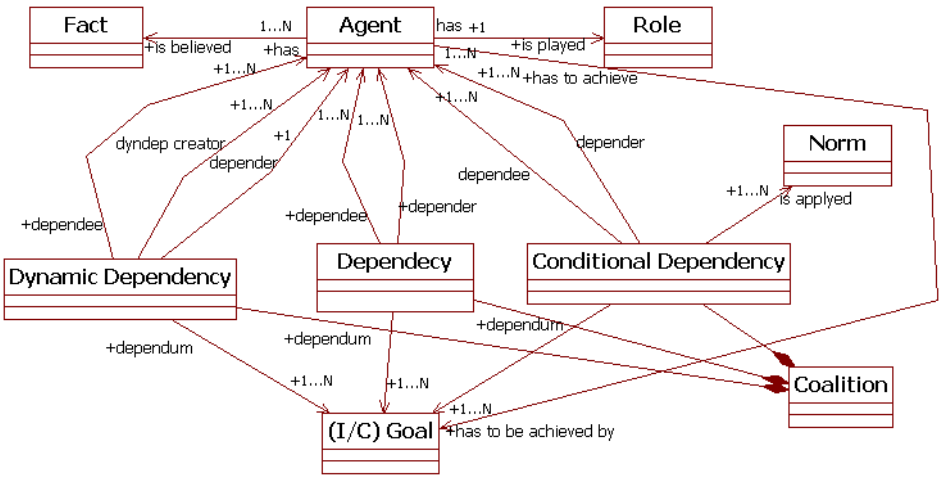


Fig. 4. The UML class diagram specifying the main concepts of the metamodel.

- *the function* $oblig : N \times A \rightarrow 2^{G \cup RG}$ *is a function that associates with each norm and agent, the goals and institutional goals the agent must achieve to fulfill the norm. Assumption:* $\forall n \in N$ *and* $a \in A$, $oblig(n, a) \in power(\{a\})$[2].
- *the function* $sanct : N \times A \rightarrow 2^{G \cup RG}$ *is a function that associates with each norm and agent, the goals and institutional goals that will not be achieved if the norm is violated by agent* a*. Assumption: for each* $B \subseteq A$ *and* $H \in power(B)$ *that* $(\cup_{a \in A} sanct(n, a)) \cap H = \emptyset$.
- *the function* $ctd : N \times A \rightarrow 2^{G \cup RG}$ *is a function that associates with each norm and agent, the goals and institutional goals that will become the new goals the agent* a *has to achieve if the norm is violated by* a*. Assumption:*$\forall n \in N$ *and* $a \in A$, $ctd(n, a) \in power(\{a\})$.

We relate norms to goals following a twofold direction. First, we associate with each norm n a set of goals and institutional goals $oblig(n) \subseteq G \cup RG$. Achieving these normative goals means that the norm n has been fulfilled; not achieving these goals means that the norm is violated. We assume that every normative goal can be achieved by the group, i.e., the group has the power to achieve it. Second, we associate with each norm a set of goals and institutional goals $sanct(n) \subseteq G \cup RG$ which will not be achieved if the norm is violated and it represents the sanction associated with the norm. We assume that the group of agents does not have the power to achieve these goals. Third, we associate with each norm (primary obligation) another norm (secondary obligation) represented by a set of goals and institutional goals $ctd(n) \subseteq G \cup RG$ that have to be fulfilled if the primary obligation is violated.

[2] Power relates each agent with the goals it can achieve.

Our aim is not to present an new theorem that, using norms semantics, checks whether a given interaction protocol complies with norms. We are more interested in considering, in the context of requirements analysis, how agents' behaviour is effected by norms and in analyzing how to constrain the modeling of coalitions' evolution thanks to a normative system. There are two main assumptions in our approach. First of all we assume that norms can sometimes be violated by agents in order to keep their autonomy. The violation of norms is handled by means of sanctions and contrary to duty mechanisms. Second, we assume that, from the institutional perspective, the internal state of the external agents is neither observable nor controllable but the institutional state or public state of these agents is note since connected to a role and it can be changed by the other agents.

We define a new modeling activity, called *conditional dependency modeling*, to support in the early and late requirements analysis the representation of obligations, sanctions and contrary-to-duty obligations. Conditional dependence networks are defined as follows:

Definition 3 (Conditional Dependence Networks (CDN)).
A conditional dependence network is a tuple $\langle A, G, cdep, odep, sandep, ctddep \rangle$ *where:*

- A *is a set of agents and* G *is a set of goals;*
- *cdep* : $2^A \times 2^A \rightarrow 2^{2^G}$ *is a function that relates with each pair of sets of agents all the sets of goals on which the first depends on the second.*
- *odep* : $2^A \times 2^A \rightarrow 2^{2^G}$ *is a function representing a obligation-based dependency that relates with each pair of sets of agents all the sets of goals on which the first depends on the second.*
- *sandep* $\subseteq (OBL \subseteq (2^A \times 2^A \times 2^{2^G})) \times (SANCT \subseteq (2^A \times 2^A \times 2^{2^G}))$ *is a function relating obligations to the dependencies which represent their sanctions. Assumption:* $SANCT \in$ *cdep and* $OBL \in$ *odep.*
- *ctddep* $\subseteq (OBL_1 \subseteq (2^A \times 2^A \times 2^{2^G})) \times (OBL_2 \subseteq (2^A \times 2^A \times 2^{2^G}))$ *is a function relating obligations to the dependencies which represent their secondary obligations. Assumption:* $OBL_1, OBL_2 \in$ *odep and* $OBL_1 \cap OBL_2 = \emptyset$.

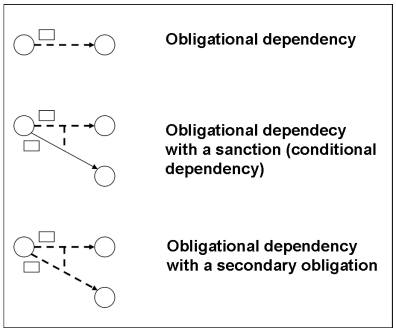


Fig. 5. Legend of the graphical representation of the *conditional dependency modeling*.

Figure 5 gives a graphical representation of the *conditional dependency modeling*. It describes the obligation-based dependency (depicted as a striped arrowed line), the obligation-based dependency with the associated sanction expressed as conditional dependency (depicted as a striped arrowed line representing the obligation connected to a common arrowed line representing the sanction by a striped line) and the obligation-based dependency with the associated secondary obligation (depicted as a striped arrowed line representing the primary obligation connected to another striped arrowed line representing the secondary obligation by a striped line). The two functions *ctddep* and *sandep* are graphically represented as the striped line connecting the obligation to the sanction or to the secondary obligation.

Example 1. Considering Grid's nodes of Figure 3, we can think to add two constraints under the form of obligations and we build the following conditional dependence network $CDN = \langle A, G, cdep, odep, sandep, ctddep \rangle$:

1. Agents $A = \{n_1, n_2, n_3, n_4, n_5, n_6\}$;
2. Goals $G = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8\}$;
3. *cdep*$(\{n_1\}, \{n_2\}) = \{\{g_1\}\}$: agent n_1 depends on agent n_2 to achieve the goal $\{g_1\}$: to save the file *comp.log*;
   dep$(\{n_2\}, \{n_3\}) = \{\{g_2\}\}$: agent n_2 depends on agent n_3 to achieve the goal $\{g_2\}$: to run the file *mining.mat*;
   dep$(\{n_3\}, \{n_1\}) = \{\{g_5\}\}$: agent n_3 depends on agent n_1 to achieve the goal $\{g_5\}$: to save the file *satellite.jpg*;
   dep$(\{n_4\}, \{n_6\}) = \{\{g_3\}\}$: agent n_4 depends on agent n_6 to achieve the goal $\{g_3\}$: to run the file *results.mat*;
   dep$(\{n_6\}, \{n_5\}) = \{\{g_4\}\}$: agent n_6 depends on agent n_5 to achieve the goal $\{g_4\}$: to save the file *satellite.mpeg*;
   dep$(\{n_5\}, \{n_4\}) = \{\{g_6\}\}$: agent n_5 depends on agent n_4 to achieve the goal $\{g_6\}$: to have the authorization to open the file *dataJune.mat*;
   odep$(\{n_2\}, \{n_1\}) = \{\{g_7\}\}$: agent n_2 is obliged to perform goal $\{g_7\}$ concerning agent n_1 : to run the file *mining.mat* with the highest priority;
   odep$(\{n_4\}, \{n_5\}) = \{\{g_8\}\}$: agent n_4 is obliged to perform goal $\{g_8\}$ concerning agent n_5 : to share results of the running of file *dataJune.mat* with agent n_5;
   odep$(\{n_4\}, \{n_6\}) = \{\{g_8\}\}$: agent n_4 is obliged to perform goal $\{g_8\}$ concerning agent n_6 : to share results of the running of file *dataJune.mat* with agent n_6;
   sandep$\{((\{n_2\}, \{n_1\}) = \{\{g_7\}\}, (\{n_1\}, \{n_2\}) = \{\{g_1\}\})\}$;
   ctddep$\{((\{n_4\}, \{n_5\}) = \{\{g_8\}\}, (\{n_4\}, \{n_6\}) = \{\{g_8\}\})\}$;

Example 1 is depicted in Figure 6 which shows the network in the step after the deletion and the insertion of the two dynamic dependencies of Figure 3. In Figure 6, following the definition of coalition, we have two coalitions composing, e.g., two local groups of a virtual organization. The first one is composed by nodes n_1, n_2, n_3 and the other one is composed by nodes n_4, n_5 and n_6. Since these two subsets of the virtual organization have to work with a good cohesion then it is possible to insert some constraints, made clear by obligations. The first obligation consists in giving the highest priority to, for example, a computation for an agent composing the same local coalition as you. This first obligation is related to a sanction if it is violated. This link is made

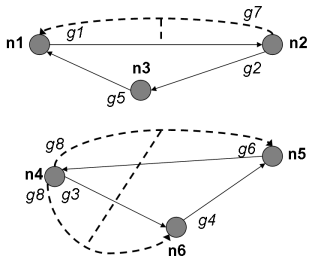


Fig. 6. Conditional Dependence Network of Example 1.

clear by the function *sandep* and it represents the deletion of a dependency concerning a goal of the agent that has to fulfill the obligation. The second obligation, instead, is related to a secondary obligation and it means that the agent has to share the results of a computation with a member of its coalition but, if it does not fulfill this obligation then it has to share these results with another member of its coalition.

Figure 7 shows the graphical representation of how an obligation in a conditional dependence network can evolve toward the application of a sanction or of a secondary obligation. In the first case, if the obligation is fulfilled and it is linked to a sanction then the obligation can be removed and also the connection among the obligation and the sanction can be removed. The only dependency that remains in the network is the one related to the sanction that passes from being a conditional dependency to a common dependency. If the obligation is not fulfilled then it is deleted and the deletion involves also the conditional dependency representing the sanction. The sanction consists exactly the deletion of this conditional dependency associated to a goal that the agent would achieve. In the second case, if the obligation is fulfilled and it is linked to a secondary obligation then the obligation is deleted and also the secondary obligation is deleted since there is no reason to already exists. If the obligation, instead, is not fulfilled then the primary obligation is deleted but the secondary obligation not. Note that in Figure 7 are depicted only the conditional dependencies and the obligational dependencies and not all the other kinds of possible dependencies present in the network.

Summarizing, we represent obligations, sanctions and contrary-to-duty obligations as tuples of dependencies related to each other. An obligation is viewed as a particular kind of dependency and it is related to dependencies due to sanctions and dependencies due to secondary obligations. In the first case, we have that sanctions are common dependencies, already existing inside the system that, because of their connection with the obligation, can be deleted. These obligations can be of different kinds depending on the involved agents. For example, we can have a primary obligation linked to two secondary obligations: a first case con involve the same agents, e.g., agent a has to pay agent b for a service but he does not do the payment thus the secondary obligation is to pay to agent b an additional cost, and second case can involve a third agent, e.g., agent a continues to not pay you thus a third agent c is obliged to punish it for example with the deletion of all the services he has to perform for this agent.

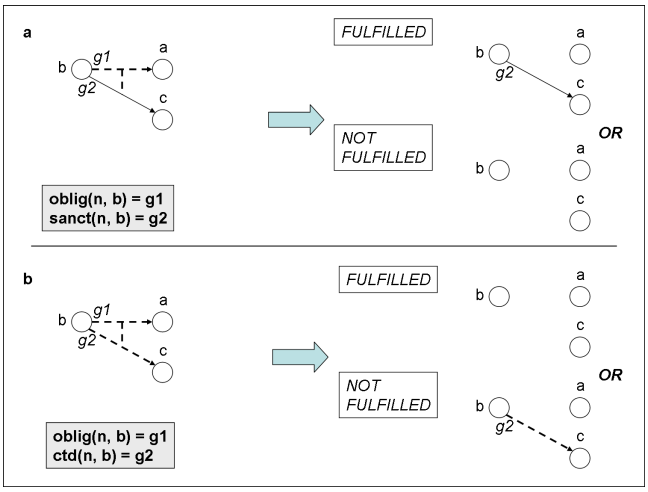


Fig. 7. The evolution of conditional dependence networks.

4.1 Two case studies: personal norms and transactions.

In the real life, everybody's life is regulated by personal norms like *not kill* and *not leave trash on the roads.* These norms are referred to every person and it seems that everyone depends on the others to achieve these goals that can be represented as goals of the whole society. It is similar to the social delegation cycle: do not do the others what you do not want them to do to you. In this case, we can represent the dependence network as a full connected graph since every agent depends on all the other agents, for example to not be killed. This case study makes explicit the necessity to simplify the dependence network with the aim to individuate the obligations. The simplification brought by the representation of obligations is relevant, as can be seen in Figure 8-(a).

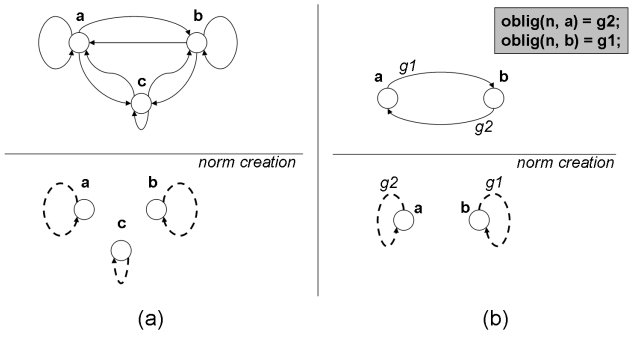


Fig. 8. Case studies: personal norms and transactions.

The second case study consists in transactions. A transaction is an agreement or communication carried out between separate entities, often involving the exchange of items of value, such as information, goods, services and money. This is the basic idea underlying norm emergence. Let us consider the case of two agents a and b, where a is a buyer and b is a seller. If we consider two goals such as $g1$: *book sent by the seller* b *to buyer* a and $g2$: *money transferred from the buyer* a *to the seller* b, we have the dependence network depicted in Figure 8-(b). The two agents depend on each other to achieve their goals, the seller is waiting for its payment and the buyer is waiting for its good. When introduced, our representation of obligations allows to obtain a simplified version of the network in which each agent depends on itself to not violate the obligation. The dependence network derived after the norm creation is much more simpler than the previous one representing however the same concepts. This simplified version can be used for the requirements analysis phase of the multiagent system allowing to individuate in a simpler way the obligations present inside the system, without the necessity to analyze all the goal-based dependencies present in the network.

4.2 Coalitions in Conditional Dependence Networks

In this section, we answer to the question: what kind of constraints are set by obligations in conditional dependence networks concerning coalitions. In Section 3, we presented a definition of coalition based on the structure of dynamic dependence networks. In these dynamic coalitions we deal with conditional goals but there is not the presence of obligations intended as sets of dependencies linked together by a relation of the kind obligation-sanction or primary obligation-secondary obligation. Conditional dependence networks have to be taken into account when a system is described in terms of coalitions, vulnerable coalitions and potential coalitions since they can change depending on the conditional dependencies set by obligations. A coalition has to consider sanctions and secondary obligations, according to these constraints:

Definition 4 (Constraints for Coalitions in Conditional Dependence Networks). *Let A be a set of agents and G be a set of goals. A coalition function is a partial function $C \subseteq A \times 2^A \times 2^G$ such that $\{a \mid C(a, B, G)\} = \{b \mid b \in B, C(a, B, G)\}$, the set of agents profiting from the coalition is the set of agents contributing to it.*

Introducing conditional dependence networks, the following constraints arise:

- $\forall(dep_1, dep_2) \in$ *sandep,* $dep_2 \notin C$ *if and only if* $dep_1 \notin C$*. If the obligation, associated to the dependency* dep_1 *is not part of the coalition* C *then also the sanction* dep_2 *associated to the obligation is not part of the coalition* C*. If the obligation, associated to the dependency* dep_1 *is part of the coalition* C *then also the sanction* dep_2 *associated to the obligation is part of the coalition* C*.*
- $\forall(dep_1, dep_2) \in$ *ctddep,* $dep_2 \in C$ *if and only if* $dep_1 \notin C$*. If the primary obligation, associated to the dependency* dep_1 *is not part of the coalition* C *then the secondary obligation* dep_2 *is part of the coalition* C*. If the primary obligation, associated to the dependency* dep_1 *is part of the coalition* C *then the secondary obligation* dep_2 *is not part of the coalition* C*.*

Example 2. Let us consider conditional dependence network of Example 1, depicted in Figure 6. Applying these constraints, we have that if the obligation on goal g_7 is fulfilled then the coalition composed by agents n_1, n_2 and n_3 already exists since the dependency associated to the sanction is not deleted. If the obligation on goal g_7 is not fulfilled then the obligation is deleted but also the sanction is deleted and the coalition does not exist any more. Concerning the second coalition, if the obligation on goal g_8 is fulfilled then both the primary and the secondary obligation are removed but if the primary obligation is not fulfilled then the secondary obligation is part of the coalition composed by agents n_4, n_5 and n_6.

5 Related work

The idea of focusing the activities that precede the specification of software requirements, in order to understand how the intended system will meet organizational goals, is not new. It has been first proposed in requirements engineering, specifically in Eric Yu's work with his i* model [24]. The rationale of the i* model is that by doing an earlier analysis, one can capture not only the what or the how, but also the why a piece of software is developed. As stated throughout the paper, the most important inspiration source for our model is the TROPOS methodology [9] that spans the overall software development process, from early requirements to implementation. Other approaches to software engineering are those of KAOS [12], GAIA [23], AAII [16] and MaSE [15] and AUML [1]. The comparison of these works is summarized in Figure 9.

	Early requirements	Late requirements	Architectural design	Detailed design
i*	X	X		
Kaos		X		
GAIA		X	X	
AAII and MaSE			X	X
AUML				X
TROPOS	X	X	X	X

Fig. 9. Comparison among different software engineering methodologies.

The main difference between these approaches and our one consists in the use at the same time of the normative multiagent paradigm based on both the notion of institution and the notion of obligation with its related concepts of contrary-to-duty and sanction and of graphical modeling language based on dependencies among agents. Moreover, these approaches do not consider the notion of coalition, as group of actors with a common set of goals and the possible constraints on their structure.

An example of normative multiagent system introducing obligations has been done by Boella and van der Torre [7]. Interesting approaches on the application of the notion of institution to multiagent systems are defined in Sierra et al. [21], Bogdanovych et al. [8] and Vazquez-Salceda et al. [22].

6 Conclusions

This paper provides a detailed account of a new requirements analysis model based on the normative multiagent paradigm, following the TROPOS methodology [9]. The paper presents and discusses the early and late requirements phases of systems design. The first part of the paper presents the key concepts of the ontology of our methodology as shown by the UML diagram of Figure 4. The second part of the paper presents the graphical representations of the three modeling activities by which our model is composed. These modeling activities are called *dependency modeling*, *dynamic dependency modeling* and *conditional dependency modeling*. The addition of normative concepts is a relevant improvement to requirements analysis since it allows, first, to constrain the construction of the requirements modeling and, second, to represent systems, as for example Grid-based systems, in which there are explicit obligations regulating the behaviour of the components composing it. Moreover, we model the requirements analysis phases also in a context in which there is the possible presence of coalitions.

Of course, this model is not intended for any type of software. For system software, e.g., a compiler, or embedded software, the operating environment of the system-to-be is an engineering artifact, with no identifiable stakeholders. In such cases, traditional software development techniques may be most appropriate. However, a large and growing percentage of software operates within open, dynamic organizational environments.

Concerning future work, we are concentrating our efforts on the definition of the notion of coalitions' stability in this model. We are interested in representing the coalitions' evolution process by means of our modeling techniques and in defining more powerful constraints on coalitions with the aim to maintain, thanks to the application of norms, coalitions' stability during this evolution process. In our opinion, this would be a relevant improvement to the studies concerning coalitions' stability because of the application, at the same time, of a social network approach, providing measures and graph-based methods, and a normative multiagent approach, providing mechanisms like social laws and norms. Moreover, we aim in addressing an evaluation analysis to specify the consistency of a system defined by means of our modeling techniques. Finally, we are improving our conditional dependency modeling by adding also the representation of prohibitions.

References

1. B. Bauer, J. P. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software systems. *Software Engineering and Knowledge Engineering*, 11(3):207–230, 2001.
2. G. Boella, P. Caire, and L. van der Torre. Autonomy implies creating one's own norms norm negotiation in online multi-player games. *KAIS*, 18:137–156, 2009.
3. G. Boella, L. van der Torre, and H. Verhagen. Introduction to normative multiagent systems. *Computational and Mathematical Organization Theory*, 12:71–79, 2006.
4. G. Boella, L. van der Torre, and S. Villata. Changing institutional goals and beliefs of autonomous agents. In Bui et al. [10], pages 78–85.
5. G. Boella, L. van der Torre, and S. Villata. Social viewpoints for arguing about coalitions. In Bui et al. [10], pages 66–77.

6. G. Boella and L. W. N. van der Torre. Regulative and constitutive norms in normative multiagent systems. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *KR*, pages 255–266. AAAI Press, 2004.
7. G. Boella and L. W. N. van der Torre. Power in norm negotiation. In N. T. Nguyen, A. Grzech, R. J. Howlett, and L. C. Jain, editors, *KES-AMSTA*, volume 4496 of *Lecture Notes in Computer Science*, pages 436–446. Springer, 2007.
8. A. Bogdanovych, M. Esteva, S. J. Simoff, C. Sierra, and H. Berger. A methodology for developing multiagent systems as 3d electronic institutions. In *AOSE*, volume 4951 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2007.
9. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
10. T. D. Bui, T. V. Ho, and Q.-T. Ha, editors. *Intelligent Agents and Multi-Agent Systems, 11th Pacific Rim International Conference on Multi-Agents, PRIMA 2008, Hanoi, Vietnam, December 15-16, 2008. Proceedings*, volume 5357 of *Lecture Notes in Computer Science*. Springer, 2008.
11. P. Caire, S. Villata, G. Boella, and L. van der Torre. Conviviality masks in multiagent systems. In L. Padgham, D. C. Parkes, J. Müller, and S. Parsons, editors, *AAMAS (3)*, pages 1265–1268. IFAAMAS, 2008.
12. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.
13. J. W. Forrester. Gentle murder, or the adverbial samaritan. *Journal of Philosophy*, 81:193–197, 1984.
14. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. of Supercomputer Applications*, 15, 2001.
15. J. C. García-Ojeda, S. A. DeLoach, Robby, W. H. Oyenan, and J. Valenzuela. O-mase: A customizable approach to developing multiagent development processes. In *Agent-Oriented Software Engineering VIII, 8th International Workshop*, volume 4951 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.
16. D. Kinny, M. P. Georgeff, and A. S. Rao. A methodology and modelling technique for systems of bdi agents. In W. V. de Velde and J. W. Perram, editors, *MAAMAW*, volume 1038 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1996.
17. N. H. Minsky and A. Lockman. Ensuring integrity by adding obligations to privileges. In *ICSE*, pages 92–102, 1985.
18. H. Prakken and M. J. Sergot. Contrary-to-duty obligations. *Studia Logica*, 57(1):91–115, 1996.
19. Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artif. Intell.*, 73(1-2):231–252, 1995.
20. J. S. Sichman and R. Conte. Multi-agent dependence by dependence graphs. In *AAMAS*, pages 483–490. ACM, 2002.
21. C. Sierra, J. Thangarajah, L. Padgham, and M. Winikoff. Designing institutional multi-agent systems. In L. Padgham and F. Zambonelli, editors, *AOSE*, volume 4405 of *Lecture Notes in Computer Science*, pages 84–103. Springer, 2006.
22. J. Vázquez-Salceda, V. Dignum, and F. Dignum. Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3):307–360, 2005.
23. M. Wooldridge, N. R. Jennings, and D. Kinny. The GAIA methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.
24. E. Yu. Modeling organizations for information systems requirements engineering. In *First IEEE International Symposium on Requirements Engineering*, pages 34–41, 1993.
25. G. Zhao, D. W. Chadwick, and S. Otenko. Obligations for role based access control. In *AINA Workshops (1)*, pages 424–431. IEEE Computer Society, 2007.

Towards a logical model of social agreement for agent societies

Emiliano Lorini[1] and Mario Verdicchio[2] *

[1] IRIT, Toulouse, France
lorini@irit.fr
[2] Università degli studi di Bergamo
mario.verdicchio@unibg.it

Abstract. Multi-agent systems (MASs), comprised of autonomous entities with the aim to cooperate to reach a common goal, may be viewed as computational models of distributed complex systems such as organizations and institutions. There have been several model proposals in the agent literature with the aim to support, integrate, substitute human organizations, but no attempt has gone beyond the boundaries of this research context to become a mainstream software engineering implementation guideline, nor has it been adopted as a universal model of multi-agent interaction in economics or social sciences. In this work we counter top-down, operational organization specifications with a logical model of a fundamental concept: agreement, with the long-term aim to create a formal model of multi-agent organization that can serve as a universally accepted basis for implementation of collaborative distributed systems.

1 Introduction

Multi-agent systems (MASs) can provide an effective computational model of autonomous individuals interacting in a complex distributed system. The models that simulate the operations of multiple entities can show how agent technology can be exploited in economics and social sciences. The lack of a breakthrough so far is possibly paralleled by some lack of generality in the proposed MAS implementations. Several research works aim at proposing operational models of multi-agent organizations in the form of templates of norms, roles, interaction patterns, and so on, that have a significant impact on the agent community, but whose adoption by a wider audience may be hindered by a discrepancy between how organizations are conceived in this research context and how they actually emerge in the real world.

In this work we begin our attempt to formalize the concept of organization starting from what we consider its most fundamental component: the agreement. We see an organization as a way to coordinate agent interaction that starts from an agreement between the relevant agents. Moreover, we adopt a bottom-up, formal approach to keep our analysis as general as possible, and, as a consequence, the application field of our current and future results as wide as possible.

* Emiliano Lorini is financed by the ANR French project ForTrust; Mario Verdicchio is financed by the FP7 project PrimeLife.

The paper is organized as follows: Section 2 illustrates more in detail the motivations to our efforts; Section 3 presents the syntax and the semantics of our logical model, and some choices made in the model are discussed in Section 4, while Section 5 presents some theorems; Sections 6 and 7 illustrate how agreements are formed and how commitments and norms can be grounded on them, respectively; Section 8 provides some pointers to significant related literature, and, finally, Section 9 concludes.

2 Motivation

MASs can be seen as conceived with two distinct purposes. In the scenarios envisioned by the pioneers of this field, whose hopes were boosted also by the unprecedented success of Internet technologies, agents were viewed as a further development of the object-oriented paradigm, leading to the implementation of goal-driven, mobile programs that could cooperate with each other autonomously to reach a common objective. In a broader interpretation going beyond the strictly technological aspects to include social, economic, legal ones, MASs would be seen as a computational model of any group of interacting entities. In this respect, agents are programs that simulate a real-life complex system whose properties are to be analyzed by means of a computer system.

The lack (so far) of a so-called 'killer application' based on MAS technology does not necessarily entail that the latter interpretation traces the only viable path for agent researchers, who thus should focus on simulation. Nevertheless, in our opinion, significant achievements in the simulation-oriented MAS research are a necessary step to finally reach a breakthrough also in the area of mainstream software development. We agree with DeLoach [5] when he states that MAS researchers have not yet demonstrated that the agent approach can yield competitive or even better solutions than other programming paradigms by providing reliable, complex, distributed systems.

The context of virtual organizations is our point of reference, and from this perspective, a very effective demonstration of the impact of MAS technologies can be provided by a believable agent-based simulation of real-life, human organizations. Once agents are proven to be capable of delivering detailed models of complex organizations, then they can become a very appealing candidate for cutting-edge software solutions aiming at supporting, or even substituting, their human counterparts.

Several models of virtual organizations have been proposed in the literature [18], [8]. In particular, Electronic Institutions [16] have been presented as a way to regulate agent interaction in open environments. We see some issues rising from this research line: How really open are these environments with respect to the constraints introduced by the proposed organizational models? How does the fact that these models are operational (as opposed to logical) affect their impact on the potential adopters? These questions can be seen as different facets of our main concern: the affinity of virtual organizations with real ones is a key factor in MAS technology's shift from research to practice. Although it is very tempting to provide detailed specifications of virtual organizations in terms of roles, scenarios, interaction patterns, communication protocols and so on, we think that such approach inevitably narrows down the scope of a proposal to the researchers' working hypotheses. The top-down specification of a predefined template is not the way organizations are born in the real world, and this distance between

theoretical research and actual organizational dynamics might correspond to the gap between the agent-based proposals and the solutions adopted in the industry.

Our work has a rather different (if not opposite) starting point. We intend to provide a logical model (as opposed to operational) that allows for the formalization of the creation of organizations in a bottom-up fashion (as opposed to top-down). It might seem surprising that researchers who call for the elimination of the gap between theory and practice opt for a logic-based approach. However, this is a research field where universal models for basic concepts (including the very concept of 'agent') are still missing. We think that theoretical definitions of general concepts might work as wider and more solid foundations for the construction of a model of organizations that can eventually provide effective implementation guidelines. This is also the idea behind the choice of a bottom-up approach: To keep a model of organizations as general as possible, instead of trying to impose a standard template (which is a surely successful approach only in monopoly contexts), we aim at shedding some light on the basic mechanisms that lead a group of independent individuals (or autonomous agents) to form an organization.

In a top-down approach, agents join an organization with pre-established rules. In our bottom-up approach, we see an organization as the product of the agreement of several agents on how their future interactions should be regulated. Thus, the aim of this work is to formally define 'agreement' as a fundamental concept for the creation of multi-agent organizations, that is, we intend to propose a logic of social agreement.

3 A modal logic of social agreement

We present in this section the syntax and semantics of the logic $\mathcal{SAL}$ (*Social Agreement Logic*). The logic $\mathcal{SAL}$ specifies the conditions under which agreements are established and annulled. Moreover it accounts for the relationships between agreement formation and agents' preferences.

3.1 Syntax

The syntactic primitives of the logic $\mathcal{SAL}$ are the following: a nonempty set of atomic formulas $ATM = \{p, q, \ldots\}$, a nonempty finite set of agents $AGT = \{i, j, \ldots\}$, a nonempty set of atomic actions $ACT = \{\alpha, \beta, \ldots\}$. We note $2^{ACT*} = 2^{ACT} \setminus \emptyset$ the set of all non-empty sets of actions, and $2^{AGT*} = 2^{AGT} \setminus \emptyset$ the set of all non-empty sets of agents.

We introduce a function REP that associates to every agent i in AGT a non-empty set of atomic actions called *action repertoire* of agent i:

$$REP : AGT \longrightarrow 2^{ACT*}.$$

For every agent $i \in AGT$ we define the set of i's *action tokens* of the form $i{:}\alpha$, that is,

$$\Delta_i = \{i{:}\alpha \mid \alpha \in REP(i)\}.$$

That is, $i{:}\alpha$ is an action token of agent i only if α is part of i's repertoire. We note

$$\Delta = \bigcup_{i \in AGT} \Delta_i$$

the pointwise union of the sets of possible action tokens of all agents.

The following abbreviations are convenient to speak about joint actions of groups of agents. For every non-empty set of agents I we note $JACT_I$ the set of all possible

combinations of actions of the agents in I (or *joint actions* of the agents in I), that is,

$$JACT_I = \prod_{i \in I} \Delta_i.$$

For notational convenience we write $JACT$ instead of $JACT_{AGT}$. Elements in every $JACT_I$ are tuples noted δ_I, δ'_I, $\delta''_I, \ldots$. Elements in $JACT$ are simply noted δ, δ', $\delta'', \ldots$. For example suppose that $I = \{1, 2, 3\}$ and $\delta_I = \langle 1{:}\alpha, 2{:}\beta, 3{:}\gamma \rangle$. This means that δ_I is the joint action of the agents $1, 2, 3$ in which 1 does action α, 2 does action β and 3 does action γ.

The language of $\mathcal{SAL}$ is the set of formulas defined by the following BNF:

$$\varphi ::= p \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{Agree}_I\varphi \mid \mathtt{Do}_{i:\alpha}\varphi$$

where p ranges over ATM, i ranges over AGT, $i{:}\alpha$ ranges over Δ_i, and I ranges over 2^{AGT*}.

The classical Boolean connectives $\wedge$, $\rightarrow$, $\leftrightarrow$ and $\top$ (tautology) are defined from $\bot$, $\vee$ and $\neg$ in the usual manner.

The operators of our logic have the following reading.

- $\mathtt{Agree}_I\varphi$: 'the agents in the group I agree that φ'.
- $\mathtt{Do}_{i:\alpha}\varphi$: 'agent i is going to do α and φ will be true afterwards' (therefore $\mathtt{Do}_{i:\alpha}\top$ is read: 'agent i is going to do α').

Operators of the form $\mathtt{Agree}_I$ enable one to express those issues on which the agents in I agree, while forming a coalition. For example, $\mathtt{Agree}_I\neg smokePublic$ expresses that the agents in I agree that people should not smoke in public spaces.

The formula $\mathtt{Agree}_I\bot$ literally means that 'the agents in I agree on a contradiction'. We assign a special meaning to this formula by supposing that 'agreeing on a contradiction' means 'not being part of the same group' (or 'not forming a coalition'). This is because we assume that functioning as members of the same coalition is (at least in a minimal sense) a rational activity, and a rational group of agents cannot agree on a contradiction. Thus, $\mathtt{Agree}_I\bot$ should be read 'the agents in I do not function as members of the same group' or 'the agents in I do not form a coalition' or 'the agents in I do not constitute a group'. Conversely, $\neg\mathtt{Agree}_I\bot$ has to be read 'the agents in I function as members of the same group' or 'the agents in I form a coalition' or 'the agents in I constitute a group'. This concept of constituted group is expressed by the following abbreviation. For every $I \in 2^{AGT*}$:

$$\mathtt{Group}(I) \stackrel{\text{def}}{=} \neg\mathtt{Agree}_I\bot.$$

Note that this definition of group demands for some form of agreement, in particular if the agents in I form a coalition (i.e. $\mathtt{Group}(I)$) then the agents in I agree that they form a coalition (i.e. $\mathtt{Agree}_I\mathtt{Group}(I)$). Indeed, as we will show in Section 3.3, our agreement operators satisfy the axiom $\neg\mathtt{Agree}_I\varphi \rightarrow \mathtt{Agree}_I\neg\mathtt{Agree}_I\varphi$.

If I is a singleton then $\mathtt{Agree}_I$ is used to express the individual preferences of agent i. That is, for every $i \in AGT$:

$$\mathtt{Agree}_{\{i\}}\varphi \stackrel{\text{def}}{=} \mathtt{Pref}_i\varphi.$$

Formula $\mathtt{Pref}_i\varphi$ has to be read 'agent i prefers that φ is possible' (semantically this means that 'φ is true in all states that are preferred by agent i').

The following additional abbreviations will be useful to make more compact our notation in the sequel of the article. For every $i \in AGT$:

$$\texttt{Sat}_i\varphi \stackrel{\text{def}}{=} \neg\texttt{Pref}_i\neg\varphi.$$

Formula $\texttt{Sat}_i\varphi$ has to be read 'φ is a satisfactory state of affairs for agent i' (semantically this means that 'there exists at least one preferred state of agent i in which φ is true').

For every $I \in 2^{AGT*}$ and $\delta_I \in JACT$:

$$\texttt{Do}_{\delta_I}\varphi \stackrel{\text{def}}{=} \bigwedge_{j\in I} \texttt{Do}_{\delta_j}\varphi.$$

Formula $\texttt{Do}_{\delta_I}\varphi$ has to be read 'the agents in I execute in parallel their individual actions δ_i in the vector δ_I and φ will be true after this parallel execution'. We shorten this to 'the joint action δ_I is going to be performed by group I and φ will be true afterwards'. In other words, we consider a weak notion of joint action δ_I as the parallel execution of the individual actions δ_i by every agent in I.

For every $I \in 2^{AGT*}$:

$$\texttt{Pref}_I\varphi \stackrel{\text{def}}{=} \bigwedge_{j\in I} \texttt{Pref}_j\varphi;$$
$$\texttt{Sat}_I\varphi \stackrel{\text{def}}{=} \bigwedge_{j\in I} \texttt{Sat}_j\varphi.$$

Formula $\texttt{Pref}_I\varphi$ has to be read 'every agent in I prefers that φ is true', whilst $\texttt{Sat}_I\varphi$ has to be read 'φ is satisfactory for every agent in I'.

3.2 Semantics

Frames of the logic $\mathcal{SAL}$ ($\mathcal{SAL}$-frames) are tuples $F = \langle W, R, A\rangle$ defined as follows.

- W is a non empty set of possible worlds or states.
- $R : \Delta \longrightarrow W \times W$ maps every possible action token $i{:}\alpha$ to a deterministic relation $R_{i:\alpha}$ between possible worlds in W.[3]
- $A : 2^{AGT*} \longrightarrow W \times W$ maps every non-empty set of agents I to a transitive[4] and Euclidean[5] relation A_I between possible worlds in W.

It is convenient to view relations on W as functions from W to 2^W; therefore we write $A_I(w) = \{w' : (w, w') \in A_I\}$ and $R_{i:\alpha}(w) = \{w' : (w, w') \in R_{i:\alpha}\}$. If $A_I(w) \neq \emptyset$ and $R_{i:\alpha}(w) \neq \emptyset$ then we say that A_I and $R_{i:\alpha}$ are defined at w.

Given a world $w \in W$, $A_I(w)$ is the set of worlds which are compatible with group I's agreements at world w. If I is a singleton $\{i\}$ then $A_{\{i\}}(w)$ is the set of worlds that agent i prefers. If $(w, w') \in R_{i:\alpha}$ then w' is the unique actual *successor* world of world w, that will be reached from w through the occurrence of agent i's action α at w. (We might also say that $R_{i:\alpha}$ is a partial function). Therefore, if $R_{i:\alpha}(w) = \{w'\}$ then at w agent i performs an action α resulting in the next state w'.

It is convenient to use $R_{\delta_I} = \bigcap_{i\in I} R_{\delta_{\{i\}}}$. If $R_{\delta_I}(w) \neq \emptyset$ then coalition I performs joint action δ_I at w. If $w' \in \bigcap_{i\in I} R_{\delta_i}(w)$ then world w' results from the performance of joint action δ_I by I at w.

Frames will have to satisfy some other constraints in order to be legal $\mathcal{SAL}$-frames. For every $i, j \in AGT$, $\alpha \in REP(i)$, $\beta \in REP(j)$ and $w \in W$ we have:

[3] A relation $R_{i:\alpha}$ is deterministic iff, if $(w, w') \in R_{i:\alpha}$ and $(w, w'') \in R_{i:\alpha}$ then $w' = w''$.

[4] A relation A_I is transitive iff for every $w \in W$, if $(w, w') \in A_I$ and $(w', w'') \in A_I$ then $(w, w'') \in A_I$.

[5] A relation A_I is Euclidean iff for every $w \in W$, if $(w, w') \in A_I$ and $(w, w'') \in A_I$ then $(w', w'') \in A_I$.

S1 if $R_{i:\alpha}$ and $R_{j:\beta}$ are defined at w then $R_{i:\alpha}(w) = R_{j:\beta}(w)$.

Constraint S1 says that if w' is the *next* world of w which is reachable from w through the occurrence of agent i's action α and w'' is also the *next* world of w which is reachable from w through the occurrence of agent j's action β, then w' and w'' denote the same world. Indeed, we suppose that every world can only have one *next* world. Note that S1 implies the determinism of every $R_{i:\alpha}$.

We also suppose that every agent can perform at most one action at each world. That is, for every $i \in AGT$ and $\alpha, \beta \in REP(i)$ such that $\alpha \neq \beta$ we have:

S2 if $R_{i:\alpha}$ is defined at w then $R_{i:\beta}$ is not defined at w.

We impose the following semantic constraint for individual preferences by supposing that every relation $A_{\{i\}}$ is serial, i.e. an agent has always at least one preferred state. For every $w \in W$ and $i \in AGT$:

S3 $A_{\{i\}}(w) \neq \emptyset$.

The following semantic constraint concerns the relationship between agreements and individual preferences. For every $w \in W$ and $I, J \in 2^{AGT*}$ such that $J \subseteq I$:

S4 if $w' \in A_I(w)$ then $w' \in A_J(w')$.

According to the constraint S4, if w' is a world which is compatible with I's agreements at w and J is a subgroup of group I, then w' belongs to the set of worlds that are compatible with I's agreements at w'.

The last two semantic constraints we consider are about the relationships between preferred states of an agent and actions. For every $w \in W$, $i \in AGT$ and $\delta_{\{i\}} \in \Delta_i$:

S5 if $R_{\delta_{\{i\}}}$ is defined at w' for every $w' \in A_{\{i\}}(w)$ then $R_{\delta_{\{i\}}}$ is defined at w.

According to the constraint S5, if action $\delta_{\{i\}}$ of agent i occurs in every state which is preferred by agent i, then the action $\delta_{\{i\}}$ occurs in the current state.

For every $w \in W$ and $i \in AGT$:

S6 if $R_{\delta_{\{i\}}}$ is defined at w then there exists $I \in 2^{AGT*}$ such that $i \in I$ and $R_{\delta_{\{i\}}}$ is defined at w' for every $w' \in A_I(w)$.

According to the constraint S6, if agent i's action $\delta_{\{i\}}$ occurs at world w then there exists a group I to which i belongs such that, for every world w' which is compatible with I's agreements at w, i's action $\delta_{\{i\}}$ occurs at w'.

Models of the logic $\mathcal{SAL}$ ($\mathcal{SAL}$-models) are tuples $M = \langle F, V \rangle$ defined as follows.

- F is a $\mathcal{SAL}$-frame.
- $V : W \longrightarrow 2^{ATM}$ is a truth assignment which associates each world w with the set $V(w)$ of atomic propositions true in w.

Given a model M, a world w and a formula φ, we write $M, w \models \varphi$ to mean that φ is true at world w in M. The rules defining the truth conditions of formulas are just standard for p, $\bot$, $\neg$ and $\vee$. The following are the remaining truth conditions for $\texttt{Agree}_I\varphi$ and $\texttt{Do}_{i:\alpha}$.

– $M, w \models \texttt{Agree}_I\varphi$ iff $M, w' \models \varphi$ for all w' such that $w' \in A_I(w)$
– $M, w \models \texttt{Do}_{i:\alpha}\varphi$ iff there exists $w' \in R_{i:\alpha}(w)$ such that $M, w' \models \varphi$

Note that $\texttt{Agree}_I$ is a modal operator of type necessity, whilst $\texttt{Do}_{i:\alpha}$ is of type possibility.

The following section is devoted to illustrate the axiomatization of the logic $\mathcal{SAL}$.

3.3 Axiomatization

The axiomatization of the logic $\mathcal{SAL}$ includes all tautologies of propositional calculus and the rule of inference *modus ponens* (**MP**).

(**MP**) From $\vdash_{\mathcal{SAL}} \varphi$ and $\vdash_{\mathcal{SAL}} \varphi \rightarrow \psi$ infer $\vdash_{\mathcal{SAL}} \psi$

We have the following four principles for the dynamic operators $\texttt{Do}_{i:\alpha}$.

(**K**$_{\texttt{Do}}$) $(\texttt{Do}_{i:\alpha}\varphi \wedge \neg\texttt{Do}_{i:\alpha}\neg\psi) \rightarrow \texttt{Do}_{i:\alpha}(\varphi \wedge \psi)$

(**Alt**$_{\texttt{Do}}$) $\texttt{Do}_{i:\alpha}\varphi \rightarrow \neg\texttt{Do}_{j:\beta}\neg\varphi$

(**Single**) $\texttt{Do}_{i:\alpha}\top \rightarrow \neg\texttt{Do}_{i:\beta}\top$ if $\alpha \neq \beta$

(**Nec**$_{\texttt{Do}}$) From $\vdash_{\mathcal{SAL}} \varphi$ infer $\vdash_{\mathcal{SAL}} \neg\texttt{Do}_{i:\alpha}\neg\varphi$

Dynamic operators of the form $\texttt{Do}_{i:\alpha}$ are modal operators which satisfy the axioms and rule of inference of the basic normal modal logic K (Axiom **K**$_{\texttt{Do}}$ and rule of inference **Nec**$_{\texttt{Do}}$). Moreover, according to Axiom **Alt**$_{\texttt{Do}}$, if i is going to do α and φ will be true afterwards, then it cannot be the case that j is going to do β and $\neg\varphi$ will be true afterwards. According to Axiom **Single**, an agent cannot perform more than one action at a time. This axiom makes perfectly sense in simplified artificial settings and in game-theoretic scenarios in which actions of agents and joint actions of groups never occur in parallel.

We have the following principles for the agreement operators and the preference operators, and for the relationships between agreement operators, preference operators and dynamic operators.

(**K**$_{\texttt{Agree}}$) $(\texttt{Agree}_I\varphi \wedge \texttt{Agree}_I(\varphi \rightarrow \psi)) \rightarrow \texttt{Agree}_I\psi$

(**D**$_{\texttt{Pref}}$) $\neg\texttt{Pref}_i\bot$

(**4**$_{\texttt{Agree}}$) $\texttt{Agree}_I\varphi \rightarrow \texttt{Agree}_I\texttt{Agree}_I\varphi$

(**5**$_{\texttt{Agree}}$) $\neg\texttt{Agree}_I\varphi \rightarrow \texttt{Agree}_I\neg\texttt{Agree}_I\varphi$

(**Create**$_{\texttt{Agree}}$) $\texttt{Agree}_I(\varphi \rightarrow \bigwedge_{J\subseteq I} \texttt{Sat}_J\varphi)$

(**Int1**$_{\texttt{Pref,Do}}$) $\texttt{Pref}_i\texttt{Do}_{\delta_i}\top \rightarrow \texttt{Do}_{\delta_i}\top$

(**Int2**$_{\texttt{Pref,Do}}$) $\texttt{Do}_{\delta_i}\top \rightarrow \bigvee_{i\in I} \texttt{Agree}_I\texttt{Do}_{\delta_i}$

(**Nec**$_{\texttt{Agree}}$) From $\vdash_{\mathcal{SAL}} \varphi$ infer $\vdash_{\mathcal{SAL}} \texttt{Agree}_I\varphi$

Operators for agreement of the form $\texttt{Agree}_I$ are modal operators which satisfy the axioms and rule of inference of the basic normal modal logic K45 [4] (Axioms $\mathbf{K}_{\texttt{Agree}}$, $\mathbf{4}_{\texttt{Agree}}$ and $\mathbf{5}_{\texttt{Agree}}$, and rule of inference $\mathbf{Nec}_{\texttt{Agree}}$). It is supposed that the agents in a coalition always agree on the contents of their agreements and on the contents of their disagreements (Axioms $\mathbf{4}_{\texttt{Agree}}$ and $\mathbf{5}_{\texttt{Agree}}$). That is, if the agents in I agree (resp. do not agree) that φ should be true then, they agree that they agree (resp. do not agree) that φ should be true.

We add a specific principle for individual preferences by supposing that an agent cannot have contradictory preferences (Axiom $\mathbf{D}_{\texttt{Pref}}$).

Axiom $\mathbf{Create}_{\texttt{Agree}}$ expresses a property about the relationship between the agreements of a group and the agreements of its subgroups. The agents of every group I agree that φ should be true only if φ is a state of affairs which is satisfactory for every subgroup J of I. Note that this axiom is based on the assumption that an agent i cannot agree with the common view of a certain group I about a certain issue φ, if φ is inconsistent with i's preferences. In our view this is a reasonable assumption for modeling situations of team activity and collaboration in which agreements identify certain solutions to coordination problems among the agents in a group which are satisfactory for all agents.

Axiom $\mathbf{Int1}_{\texttt{Pref,Do}}$ and Axiom $\mathbf{Int2}_{\texttt{Pref,Do}}$ express general principles of intentionality describing the relationship between an agent's action, his preferences, and the agreements of the group to which the agent belongs. According to Axiom $\mathbf{Int1}_{\texttt{Pref,Do}}$, if agent i prefers that he performs action $\delta_{\{i\}}$ ($\delta_{\{i\}}$ occurs in all states that are preferred by agent i) then agent i starts to perform action $\delta_{\{i\}}$. A similar principle for the relationship between individual intentions and action occurrences has been studied in [14]. According to Axiom $\mathbf{Int2}_{\texttt{Pref,Do}}$, if an agent i starts to perform a certain action $\delta_{\{i\}}$ then it means that either agents i prefers to perform this action or there exists some group I to which agent i belongs such that the agents in I agree that i should perform action $\delta_{\{i\}}$. In other terms, an agent i's action $\delta_{\{i\}}$ is intentional in a general sense: either it is driven by i's intention to perform action $\delta_{\{i\}}$ or it is driven by the collective intention that i performs action $\delta_{\{i\}}$ of a group I to which agent i belongs.

We call $\mathcal{SAL}$ the logic axiomatized by the axioms and rules of inference presented above. We write $\vdash_{\mathcal{SAL}} \varphi$ if formula φ is a theorem of $\mathcal{SAL}$ (i.e. φ is the derivable from the axioms and rules of inference of the logic $\mathcal{SAL}$). We write $\models_{\mathcal{SAL}} \varphi$ if φ is *valid* in all $\mathcal{SAL}$-models, i.e. $M, w \models \varphi$ for every $\mathcal{SAL}$-model M and world w in M. Finally, we say that φ is *satisfiable* if there exists a $\mathcal{SAL}$-model M and world w in M such that $M, w \models \varphi$. We can prove that the logic $\mathcal{SAL}$ is *sound* and *complete* with respect to the class of $\mathcal{SAL}$-frames. Namely:

Theorem 1 *$\mathcal{SAL}$ is determined by the class of $\mathcal{SAL}$-frames.*

Proof. It is a routine task to check that the axioms of the logic $\mathcal{SAL}$ correspond one-to-one to their semantic counterparts on the frames. In particular, Axioms $\mathbf{4}_{\texttt{Agree}}$ and $\mathbf{5}_{\texttt{Agree}}$ correspond to the transitivity and Euclideanity of every relation A_I. Axiom $\mathbf{D}_{\texttt{Pref}}$ corresponds to the seriality of every relation $A_{\{i\}}$ (constraint S3). Axiom $\mathbf{Alt}_{\texttt{Do}}$ corresponds to the semantic constraint S1. Axiom **Single** corresponds to the semantic constraint S2. Axiom $\mathbf{Create}_{\texttt{Agree}}$ corresponds to the semantic constraint S4. Axiom $\mathbf{Int1}_{\texttt{Pref,Do}}$

corresponds to the semantic constraint S5. $\mathbf{Int2}_{\mathtt{Pref},\mathtt{Do}}$ corresponds to the semantic constraint S6.

It is routine, too, to check that all of our axioms are in the Sahlqvist class. This means that the axioms are all expressible as first-order conditions on frames and that they are complete with respect to the defined frames classes, cf. [2, Th. 2.42]. □

4 Discussion

One might wonder why we did not include a principle of monotonicity of the form $\mathtt{Agree}_I\varphi \rightarrow \mathtt{Agree}_J\varphi$ for $J \subseteq I$ in our logic of agreement: for every sets of agents I and J such that $J \subseteq I$, if the agents in I agree that φ should be true then the agents in the subgroup J agree that φ should be true as well. We did not do include this principle because we think that it is not sufficiently general to be applied in all situations. Indeed, a minority group J of a larger group I might exist which does not have the same view than the larger group. For example, an entire community agrees that taxes should be payed even if every member of the community has a preferred state in which he does not pay taxes. Axiom $\mathbf{Create}_{\mathtt{Agree}}$ only requires that the members of the community agree that the situation in which everybody pays taxes is satisfactory for everyone.

Consider now the following principle $\mathtt{Agree}_I(\varphi \rightarrow \bigwedge_{i\in I}\mathtt{Pref}_i\varphi)$ and even the weaker $\mathtt{Agree}_I(\varphi \rightarrow \bigvee_{i\in I}\mathtt{Pref}_i\varphi)$: every group of agents I agree that φ should be true only if all of them prefer φ, and every group of agents I agree that φ should be true only if some of them prefers φ. These two principles are also too strong. Indeed, the agents in a group I might agree that φ should be true, without claiming that φ must be preferred by every agent in I and without claiming that φ must be preferred by some agent in I. For example, the members of a community I might agree that taxes should be payed by every agent in I without claiming and agreeing that tax payment must be preferred by every agent in I, and without claiming and agreeing that tax payment must be preferred by some agent in I. The members of the community just agree that tax payment must be something preferable by the whole community.

Finally, let us explain why we did not include stronger versions of Axiom $\mathbf{Int1}_{\mathtt{Pref},\mathtt{Do}}$ and Axiom $\mathbf{Int2}_{\mathtt{Pref},\mathtt{Do}}$ of the form $\mathtt{Agree}_I\mathtt{Do}_{\delta_I}\top \rightarrow \mathtt{Do}_{\delta_I}\top$ and $\mathtt{Do}_{\delta_I}\top \rightarrow \mathtt{Agree}_I\mathtt{Do}_{\delta_I}\top$ in the axiomatization of our logic $\mathcal{SAL}$ for every $I \in 2^{AGT*}$. On the one hand the principle $\mathtt{Agree}_I\mathtt{Do}_{\delta_I}\top \rightarrow \mathtt{Do}_{\delta_I}\top$ is too strong because autonomous agents should be capable to violate norms and to decide not to conform to multiparty agreements with other agents (see Section 7). For example, the agents in a population might agree at the public level that each of them should pay taxes (i.e. $\mathtt{Agree}_{\{1,\ldots,n\}}\mathtt{Do}_{\langle 1:payTaxes,\ldots,n:payTaxes\rangle}\top$) but, in private, some of them does not pay taxes (i.e. $\neg\mathtt{Do}_{\langle 1:payTaxes,\ldots,n:payTaxes\rangle}\top$). On the other hand the principle $\mathtt{Do}_{\delta_I}\top \rightarrow \mathtt{Agree}_I\mathtt{Do}_{\delta_I}\top$ is too strong because there are situations in which the agents in a set I perform a joint action δ_I without agreeing that such a joint action should be performed. Each agent in I is doing his part in δ_I without caring what the other agents in I do. For example, i might be cooking while j is reading a book without reciprocally caring what the other does, and without agreeing that the action of cooking performed by i and the action of reading performed by j should occur together. One might say that i and j do not have *interdependent reasons* for jointly preferring that i cooks while j reads a book.

5 Some $\mathcal{SAL}$-theorems

Let us now discuss some $\mathcal{SAL}$-theorems. The first group of theorems present some generalizations of Axioms $\mathbf{Alt}_{\mathtt{Do}}$ and **Single** for joint actions of groups.

Proposition 1. *For every $I, J \in 2^{AGT*}$ and $\delta_I, \delta'_I, \delta_J$ such that $\delta_I \neq \delta'_I$:*

(1a) $\quad \vdash_{\mathcal{SAL}} \mathtt{Do}_{\delta_I}\varphi \rightarrow \neg\mathtt{Do}_{\delta_J}\neg\varphi$

(1b) $\quad \vdash_{\mathcal{SAL}} \mathtt{Do}_{\delta_I}\top \rightarrow \neg\mathtt{Do}_{\delta'_I}\top$

According to Theorem 1a, if group I is going to perform the joint action δ_I and φ will be true afterwards, then it cannot be the case that group J is going to perform the joint action δ_J and φ is going to be false afterwards. According to Theorem 1b, every group of agents can never perform more than one joint action at a time.

The second group of theorems present some interesting properties of agreement. Theorems 2a and 2b are derivable from Axioms $\mathbf{4}_{\mathtt{Agree}}$, $\mathbf{5}_{\mathtt{Agree}}$ and $\mathbf{D}_{\mathtt{Pref}}$. According to these two theorems, the agents in I agree (resp. do not agree) that φ if and only if they agree that they agree (resp. do not agree) that φ. According to Theorem 2c, a group of agents I can intend to perform at most one joint action. Theorem 2d is just a variant of Axiom $\mathbf{Create}_{\mathtt{Agree}}$ (it is derivable from by means of Axiom $\mathbf{K}_{\mathtt{Agree}}$). Theorem 2e expresses an interesting property about coalition formation and coalition disintegration: if the agents in I agree that a minority part J of I agrees that φ and another minority part J' of I agrees that $\neg\varphi$, then the agents in I do not form a coalition (i.e. I is not a constituted group).

Proposition 2. *For every $I, J, J' \in 2^{AGT*}$ and δ_I, δ'_I such that $\delta_I \neq \delta'_I$ and $J, J' \subseteq I$:*

(2a) $\quad \vdash_{\mathcal{SAL}} \mathtt{Agree}_I\varphi \leftrightarrow \mathtt{Agree}_I\mathtt{Agree}_I\varphi$

(2b) $\quad \vdash_{\mathcal{SAL}} \neg\mathtt{Agree}_I\varphi \leftrightarrow \mathtt{Agree}_I\neg\mathtt{Agree}_I\varphi$

(2c) $\quad \vdash_{\mathcal{SAL}} \mathtt{Agree}_I\mathtt{Do}_{\delta_I}\top \rightarrow \neg\mathtt{Agree}_I\mathtt{Do}_{\delta'_I}\top$

(2d) $\quad \vdash_{\mathcal{SAL}} \mathtt{Agree}_I\varphi \rightarrow \mathtt{Agree}_I \bigwedge_{i\in I} \mathtt{Sat}_i\varphi$

(2e) $\quad \vdash_{\mathcal{SAL}} \mathtt{Agree}_I(\mathtt{Agree}_J\varphi \wedge \mathtt{Agree}_{J'}\neg\varphi) \rightarrow \neg\mathtt{Group}(I)$

Before concluding this section it is to be noted that at the current stage our logic does not allow to deal with situations in which I is a constituted group and I's agreements are in complete contradiction with the preferences of some agents in I. For instance, by Theorem 2e and Axiom $\mathbf{4}_{\mathtt{Agree}}$, we can prove that formula $\mathtt{Agree}_I\varphi \wedge \mathtt{Agree}_I\mathtt{Pref}_i\neg\varphi$ implies $\neg\mathtt{Group}(I)$, if $i \in I$. In other terms our logic $\mathcal{SAL}$ does not allow to deal with collective decisions which are taken sometimes in legal institutions and politics and which are based on special procedures like majority voting. For example, the agents in I might be the members of the Parliament of a certain country and form a coalition (i.e. $\mathtt{Group}(I)$). Moreover, they might collectively decide by majority voting to declare war upon another country (i.e. $\mathtt{Agree}_I war$), although they agree that there is a (pacifist) minority $i, j \in I$ in the Parliament who prefer that war is not declared upon another country (i.e. $\mathtt{Agree}_I(\mathtt{Pref}_i\neg war \wedge \mathtt{Pref}_j\neg war)$). As emphasized above, in our logic

an agreement about a proposition φ in a group I exists only if φ is satisfactory for every agent in I. As emphasized above, this is a reasonable assumption for modeling situations of team activity and collaboration in which agreements are about solutions to coordination problems in a group which are satisfactory for all agents of the group.

6 Reaching an agreement on what to do together

We can provide in our logic $\mathcal{SAL}$ the formal specification of some additional principles explaining how some agents might reach an agreement on what to do together starting from their individual preferences. We do not intend to add these principles to the axiomatization of $\mathcal{SAL}$ presented in Section 3.3. We just show that $\mathcal{SAL}$ is sufficiently expressive to capture them both syntactically and semantically so that the can be easily integrated into our formal framework. The principles we intend to characterize are specified in terms of agreements about the conditions under which a certain joint action should be performed.

In certain circumstances, it is plausible to suppose that a group of agents I agree that if there exists a unique satisfactory joint action δ_I for all agents in I, then such a joint action should occur. In other terms, the agents in a group I agree on the validity of the following general principle: 'Do together the joint action δ_I, if it is the only joint action that satisfies every agent in I!'. This criteria is often adopted by groups of agents in order to find cooperative solutions which are satisfactory for all them. For example, in a Prisoner Dilemma scenario with two agents i and j the joint action $\langle i{:}cooperate, j{:}cooperate\rangle$ is the only satisfactory solution for both agents. If the two agents i and j agree on the previous principle and face a PD game, then they will agree that $\langle i{:}cooperate, j{:}cooperate\rangle$ is the joint action that they should perform. The previous principle of agreement creation is formally expressed in our logic as follows. For every $I \in 2^{AGT*}$ and $\delta_I \in JACT_I$:

(*) $$\mathtt{Agree}_I((\mathtt{Sat}_I\mathtt{Do}_{\delta_I}\top \wedge \bigwedge_{\delta'_I \neq \delta_I} \neg \mathtt{Sat}_I\, \mathtt{Do}_{\delta'_I}\top) \rightarrow \mathtt{Do}_{\delta_I}\top)$$

Principle * corresponds to the following semantic constraint over $\mathcal{SAL}$-frames. For every $w \in W$, $I \in 2^{AGT*}$ and $\delta_I \in JACT_I$:

S6 if $w' \in A_I(w)$ and $S_i \circ R_{\delta_I}(w') \neq \emptyset$ for all $i \in I$ and, for all $\delta'_I \neq \delta_I$ there exists $i \in I$ such that $S_i \circ R_{\delta'_I}(w') = \emptyset$ then, $R_{\delta_I}(w') \neq \emptyset$

where $S_i \circ R_{\delta_I}(w')$ is defined as $\bigcup\{R_{\delta_I}(v) \mid v \in S_i(w')\}$.

If we suppose that the Principle * is valid then the following consequence is derivable for every $I \in 2^{AGT*}$ and $\delta_I \in JACT_I$:

(3) $$\mathtt{Agree}_I(\mathtt{Sat}_I\mathtt{Do}_{\delta_I}\top \wedge \bigwedge_{\delta'_I \neq \delta_I} \neg \mathtt{Sat}_I\, \mathtt{Do}_{\delta'_I}\top) \rightarrow \mathtt{Agree}_I\mathtt{Do}_{\delta_I}\top$$

REMARK. Note that in the previous Principles * and 3 of agreement creation *mutual trust* between the agents in the group is implicitly supposed, that is, it is supposed that

every agent i in I thinks it possible that the other agents in I will do their parts in the joint action δ_I. Indeed, trust between the members of the group is a necessary condition for agreement creation (on this point, see [1] for instance). We postpone to future work a formal analysis of the relationships between trust and agreement. To this aim, we will have to extend our logic $\mathcal{SAL}$ with doxastic modalities to express agents' beliefs.

Example 1. Imagine a situation of exchange of goods in EBay between two agents i and j. Agent i is the buyer and agent j is the seller. They have to perform a one-shot trade transaction. We suppose $AGT = \{i, j\}$. Agent i has the following two actions available: *pay* and *skip* (do nothing). Agent j has the following two actions available: *send* and *skip* (do nothing). That is, $\Delta_i = \{i{:}send, i{:}skip\}$ and $\Delta_j = \{j{:}pay, j{:}skip\}$. Therefore, the set of possible joint actions of the two agents is

$$JACT = \{\langle i{:}skip, j{:}skip\rangle, \langle i{:}send, j{:}skip\rangle, \langle i{:}skip, j{:}pay\rangle, \langle i{:}send, j{:}pay\rangle\}.$$

The two agents i and j agree that the situation in which i sends the product and j pays is satisfactory for both of them.

(A) $\texttt{Agree}_{\{i,j\}}\texttt{Sat}_{\{i,j\}}\texttt{Do}_{\langle i:send, j:pay\rangle}\top$.

Moreover, agent i and agent j agree that the situation in which i does nothing and j pays the product, the situation in which i sends the product and j does not nothing, and the situation in which i and j do nothing, always leave one of them unhappy. Thus we have that agent i and agent j agree that there is no other situation different from $\langle i{:}send, j{:}pay\rangle$ that is a satisfactory situation for both of them:

(B) $\texttt{Agree}_{\{i,j\}} \bigwedge_{\delta'_{\{i,j\}} \neq \langle i:send, j:pay\rangle} \neg\texttt{Sat}_{\{i,j\}}\texttt{Do}_{\delta'_I}\top$.

From items A and B, by using Principle 3, we infer that agent i and agent j agree that they should perform the joint action $\langle i{:}send, j{:}pay\rangle$:

(C) $\texttt{Agree}_{\{i,j\}}\texttt{Do}_{\langle i:send, j:pay\rangle}\top$.

Other conditions under which the agents in a group can reach an agreement on what to do together could be studied in our logical framework. For instance, one might want to have general principles of the following form which can be used to find a solution in coordination problems. Suppose that δ_I and δ'_I are both satisfactory joint actions for all agents in group I. Moreover, there are no joint actions δ''_I different from δ_I and δ'_I which are satisfactory for all agents in I. Then, either the agents in I agree that δ_I should be performed or they agree that δ'_I should be performed. In other terms, if the agents in a group I face a coordination problem then they strive to find a solution to this problem.

7 Grounding norms and commitments on agreements

The logic of agreement $\mathcal{SAL}$ presented in the previous section provides not only a formal framework in which the relationships between individual preferences of agents in a group and group agreements can be studied, but also it suggests a different perspective on concepts traditionally studied in the field of deontic logic.

Consider for instance deontic statements of the following form "within the context of group I it is required that agent i will perform action $\delta_{\{i\}}$" or "within the context of

group I it is required that agent i will perform his part in the joint action δ_I together with the other agents in I". These statements just say that i has a *directed obligation* towards his group I to do a certain action as part of a joint plan of the group I (see e.g. [12, 13] for a different perspective on directed obligations). By way of example, imagine the situation in which agent i and agent j are trying to organize a party together. After a brief negotiation, they conclude that i will prepare the cake, while j will buy drinks for the party. In this situation, "within the context of group $\{i, j\}$ it is required that agent i will prepare the cake for the party and it is required that agent j will buy drinks for the party". The following abbreviation expresses the classical deontic notion of directed obligation in terms of the concept of agreement. For every $I \in 2^{AGT*}$, $i \in I$ and $\delta_{\{i\}} \in \Delta_i$:

$$\mathtt{Oblig}_i(\delta_{\{i\}},I) \stackrel{\text{def}}{=} \mathtt{Agree}_I\mathtt{Do}_{\delta_{\{i\}}}\top.$$

Formula $\mathtt{Oblig}_i(\delta_{\{i\}},I)$ has to be read 'within the context of group I it is required that agent i will perform action $\delta_{\{i\}}$'. It is to be noted that the notion of directed obligation represents an essential constituent of the notion of *social commitment*. Thus, in our approach, an essential aspect of an agent i's commitment with respect to his group I to do a certain action $\delta_{\{i\}}$ is the fact that all agents in I agree that i should perform action $\delta_{\{i\}}$. Since all agents in the group I agree on this, they are entitled to require agent i to perform this action. Moving beyond the notion of directed obligation as an essential constituent of social commitment, our logic $\mathcal{SAL}$ can be used to provide a a formal characterization of the notion of *mutual (directed) obligation* in a group I, as 'every agent in I is required to perform its parts in a joint action δ_I of the group'. Formally, for every $I \in 2^{AGT*}$ and $\delta_I \in JACT_I$:

$$\mathtt{MutualObg}_I(\delta_I) \stackrel{\text{def}}{=} \bigwedge_{i\in I} \mathtt{Oblig}_i(\delta_{\{i\}},I).$$

Formula $\mathtt{MutualObg}_I(\delta_I)$, which is equivalent to $\mathtt{Agree}_I\mathtt{Do}_{\delta_I}\top$, has to be read 'the agents in the group I are mutually obliged to perform their parts in the joint action δ_I'. This notion of mutual (directed) obligation is an essential constituent of the notion of *mutual (social) commitment*. As already emphasized in Section 4, in our logic agents can violate obligations assigned to them (breaking their social commitments). Violation of a directed obligation is expressed in our logic by the construction $\mathtt{Oblig}_i(\delta_{\{i\}},I) \wedge \neg\mathtt{Do}_{\delta_{\{i\}}}\top$: within the context of group I it is required that agent i will perform action $\delta_{\{i\}}$, but agent i does not perform action $\delta_{\{i\}}$. The discussion on the notion of commitment will be extended in Section 8 where our approach will be compared with some formal approaches to agreement recently proposed in the MAS area.

8 Related work

The literature about the concepts this work deals with, such as agent, organization, agreement, is too vast to be given an exhaustive overview here. The best we can do is to provide pointers to some significant works that relate to our effort or that are set against it in a way that stimulates discussion. We take inspiration from Garcia et al. [9] to determine the dimensions along which multi-agent organizational concepts are developed: structural, functional, dynamic, and normative.

From a conceptual perspective, the formalization of agreements comes before any structural consideration. Let us refer to an example in the literature: in their work,

Dignum et al. [7] propose an attempt to describe minimum requirements for agents to be organized into an institution. The minimum requirements rely on an existing institution designed with the ISLANDER tool [8], and call for a middle agent. The ISLANDER platform is probably the most complete tool to date for the specification of institutions. A set of core basic notions are provided to enable a designer to specify, among other things, the roles (i.e. standardized behavioral patterns) included in the institution and the dynamics of agent interactions through scenes. The tool is kept as general as possible to allow for the widest possible variety of definable institutions. Nevertheless, the specification of an institution is meant to be entirely performed by a human designer, and agents are supposed to join an institution by assuming one or more roles. This approach does not take into account the process by which individuals look for and reach agreements that give rise to an institution. Thus, the platform is an effective means to translate an existing institution into a MAS, but the automatization of the creation of organizations is out of scope. One important consequence is that human designers must adapt to the guidelines provided by the proposed technology, which surely does not lower the aforementioned barrier between research and industry.

We call for an analogous scope shift with respect to the functional dimension of organizations, that is, their goals and how to achieve them. We follow the guidelines provided by a conceptual distinction drawn by Griffiths and Luck [11] between teamwork and coalition formation. The former is seen as focused on task assignment and action coordination among agents in the short term, whereas the latter is said to be dealing with the establishment in the long term of a group of agents with a common aim or goal. We agree with the authors in viewing multi-agent organizations as a means to achieve long-termed objectives and in considering trust as a key concept for an organization: trust undoubtedly influences an agent's decisions on whether undertaking cooperation with others. Our focus is slightly different in the context of this research: we consider an organization to be born when an agreement is made, so our efforts are on the formalization of agreements. An investigation on the relationship between trust and agreements lies ahead in our research path. Nevertheless, we share the authors' aim to determine the basic principles that lead to the creation of organizations, as opposed to several coalition formation research works, where the effort is spent in optimizing match-making algorithms between a set of tasks and a set of agent capabilities (e.g.: [17]), which is very useful, but deals only with a specific facet of the MAS organization context.

We can consider dynamic and normative dimensions as intrinsic to any attempt to formalize a concept like a multi-agent organization. Dynamic aspects include the formation and the evolution of coalitions of agents and, on a smaller scale, the preconditions and the consequences of an agent's action. When these conditions deal with deontic concepts the MAS is characterized also by a normative dimension. Human organizations have always included deontic aspects such as duties, rights, sanctions and so on, so modeling these aspects in their virtual counterparts is a rather obvious consequence. A much less obvious question is which concept or set of concepts to choose as the fundamental basis for the formalization of organizations. Dignum et al., for instance, choose violation as a fundamental concept to define deadlines in a MAS [6]. Violation is surely a very important concept in any normative context, and especially in those where

deadlines are the central focus. Nevertheless, we argue that it does not play a primary role when one wants to deal with a more general overview of organizations, especially electronic ones. As pointed out by Cardoso et al. [3], one might wonder what happens when agents ignore the sanctions attached to a violation. In real life, when an individual is not able or willing to abide by a sanction deriving by some misdemeanor, a coercive action is eventually enforced, such as confiscation or imprisonment. Such coercions are not (yet?) implementable in a distributed information system, so that effectiveness of violations and relevant sanctions is somehow diminished, and contingent upon the role played by trust in the system.

With these considerations in mind, we see agreement as a more suitable concept to equate agent-based organizations with real ones. In [19] a formal approach to multiparty agreement between agents is proposed based on the notion of social commitment. According to the authors, a multiparty agreement among the agents in $\{1, \ldots, n\}$ is given by a set of commitments $\{C_1, \ldots, C_n\}$ where C_i the commitment that agent i has towards the other agents. Thus, in this approach the notion of commitment is taken as a primitive concept and the notion of agreement is built on it. In our work we just take the opposite direction: we start with a primitive notion of agreement in a group I (which depends on the individual preferences of the agents in I), and on the top of it we built a notion of directed obligation, the essential constituent of the notion of social commitment. The logic of agreement $\mathcal{SAL}$ has some similarities with the logical framework based on the concept of acceptance we presented in [15] and [10], in which a logical analysis of the relationships between the rules and norms of an institution and the *acceptances* of these rules and norms by the members of the institution has been provided. However, in [15] and [10] the relationships between individual preferences of agents and collective acceptances (or agreements) were not investigated. This aspect has been one of the main topics of the present paper.

9 Conclusions

We consider this paper only as a starting point of a long enterprise. Top-down specifications of MAS-based organizations have so far provided stimulating discussions and promising applications which have not crossed the boundaries of the agent community. Our long-term aim is to provide a formal specification of all the basic notions that characterize organizations in general, including those in the real world, and we started with what we consider to be a fundamental concept: agreement. The formalization of this concept with a logical approach aims at analyzing in detail both its static characteristics and its dynamic properties, that is, what is meant by the term agreement and how it is supposed to influence agents' behavior when cooperation is the common goal. Once a model is universally established which is formal and general enough to abstract from particular types of organizations or specific operational details, such a model may be used as a sound basis for an agent-based implementation that can really have a significant impact on economic or social scientific contexts. The relations between our formalization of agreement and the notions of norms and commitments have been investigated in this work, but other dimensions of multi-agent interaction, such as trust, are still to be tackled, which is what we intend to pursue in the future.

References

1. G. Andrighetto, L. Tummolini, C. Castelfranchi, and R. Conte. A convention or (tacit) agreement between us. In V. F. van Benthem, J. Hendricks, J. Symons, and S. A. Pedersen, editors, *Between Logic and Intuition: David Lewis and the Future of Formal Methods*, Philosophy Synthese Library. Springer. to appear.
2. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, Cambridge, 2001.
3. H. L. Cardoso, A. P. Rocha, and E. Oliveira. Supporting virtual organizations through electronic institutions and normative multi-agent system. In J. P. Rennard, editor, *Handbook of Research on Nature Inspired Computing for Economy and Management*. Idea Group, 2006.
4. B. F. Chellas. *Modal logic: an introduction*. Cambridge University Press, Cambridge, 1980.
5. S. A. DeLoach. Moving multiagent systems from research to practice. In *Future of Software Engineering and Multi-Agent Systems (FOSE-MAS)*, 2008.
6. F. Dignum, J. Broersen, V. Dignum, and J. J. Meyer. Meeting the deadline: Why, when and how. In *Formal Approaches to Agent-Based Systems*, pages 30–40. Springer, 2005.
7. F. Dignum, V. Dignum, J. Thangarajah, L. Padgham, and M. Winikoff. Open agent systems??? In *Agent-Oriented Software Engineering VIII*, pages 73–87, 2007.
8. M. Esteva, D. de la Cruz, and C. Sierra. Islander: an electronic institutions editor. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1045–1052. ACM, 2002.
9. E. Garcia, Argente E., Giret A., and Botti V. Issues for organizational multiagent systems development. In *Sixth International Workshop From Agent Theory to Agent Implementation (AT2AI-6)*, pages 59–65, 2008.
10. B. Gaudou, D. Longin, E. Lorini, and L. Tummolini. Anchoring institutions in agents' attitudes: Towards a logical framework for autonomous mas. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, pages 728–735. ACM Press, 2008.
11. N. Griffiths and M. Luck. Coalition formation through motivation and trust. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 17–24. ACM, 2003.
12. S. Kanger and H. Kanger. Rights and parliamentarism. *Theoria*, 6(2):85–115, 1966.
13. L. Lindahl. Stig Kanger's theory of rights. In G. Holmström-Hintikka, S. Lindström, and R. Sliwinski, editors, *Collected Papers of Stig Kanger with Essays on his Life and Work*, volume 2, pages 151–171. Kluwer, Dordrecht, 2001.
14. E. Lorini and A. Herzig. A logic of intention and attempt. *Synthese*, 163(1):45–77.
15. E. Lorini, D. Longin, B. Gaudou, and A. Herzig. The logic of acceptance: Grounding institutions on agents' attitudes. *Journal of Logic and Computation*. to appear.
16. P. Noriega and C. Sierra. Electronic institutions: Future trends and challenges. In *Proceedings of the 6th International Workshop on Cooperative Information Agents*, pages 14–17, 2002.
17. O. Shehory, K. P. Sycara, and S. Jha. Multi-agent coordination through coalition formation. In *ATAL '97: Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*, pages 143–154. Springer-Verlag, 1998.
18. K. Sycara, M. Paolucci, N. van Velsen, and J. A. Giampapa. The retsina mas infrastructure. *Autonomous Agents and MAS*, 7(1-2), 2003.
19. F. Wan and M. P. Singh. Formalizing and achieving multiparty agreements via commitments. In *Proceedings of the Fourth international joint conference on Autonomous agents and multiagent systems (AAMAS 2005)*, pages 770–777. ACM Press, 2005.

Policy-driven Planning in Coalitions - a Case Study

Martin J. Kollingbaum[1,*], Joseph A. Giampapa[1,†], Katia Sycara[1,‡], and Timothy J. Norman[2,◇]

[1]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA
{*mkolling,† garof,‡ katia}@cs.cmu.edu
[2]Dept. of Computing Science, Univ. of Aberdeen, Aberdeen AB24 3UE, UK
◇t.j.norman@abdn.ac.uk

Abstract. A collaborative planning effort between partners that form coalitions may be complicated by policies that regulate what actions they may deploy in their plans and, in particular, what information they are allowed to exchange during the planning process. We are interested in situations where coalitions have to be formed ad-hoc without much co-training. For this, we investigate how agents can support human planners in producing good plans while observing the normative standards that regulate their planning and communication behavior. Based on an implementation of such norm-processing agents, we conducted a set of experiments, where human test subjects were conducting collaborative planning tasks under the guidance of these agents. A summary of experimental results is provided in the paper.

1 Introduction

Constructing joint plans within coalitions in time-stressed situations poses a particular challenge, especially in the face of individual goals, self interest and with coalition members having only limited co-training for recognizing and resolving their differences. During planning, coalition partners may also have to take into account specific policies that describe what their obligations, permissions and prohibitions are in terms of the actions comprising a plan and the communication necessary to coordinate planning activities with coalition partners. With such policies in place, human planners are under pressure to produce high-quality plans, while adhering to all their obligations and prohibitions as described by their policies. In this paper, we describe how agents can assist human planners in a monitoring and advisory capacity in their policy-driven (norm-driven) planning efforts. The scope of this work is to investigate how policies (or norms) influence collaborative planning and whether agents can ease the cognitive burden for human planners to create high-quality shared plans in the face of such policies.

In the following, we describe the development of agents that observe both communication and planning activities and provide feedback on how these actions might impact on a human planner's normative situation. We conducted a set of experiments to investigate the effectiveness and impact of such agents in

a collaborative planning scenario and provide insights about the results of these experiments.

2 Agents monitoring Policies

We consider the use of agents to monitor communication and planning activities of coalition partners and reason about possible policy violations. As the human planners are involved in a distributed planning problem, they may not be aware of potential differences and conflicts between the different policies held by the coalition partners. This may adversely impact on the planning process due to the time and effort it may take to identify and reconcile policy conflicts. Also, situations may occur that call for certain policies to be violated in order to produce a viable plan. We base our notion of policies on normative concepts, in particular, these are:

- the obligations that must be fulfilled,
- the prohibitions that constrain/forbid particular actions, and
- the permissions that define the range of actions that are allowed.

We regard policies to be relevant (or active) during a planning activity under specific conditions only. Due to this conditional nature, a human planner may not recognize that policies are in conflict with those of coalition members, that violations occurred or that policies are relevant to the present circumstances. Based on what planning actions are taken or what is communicated, this normative position may change – for example, an obligation may become fulfilled or a specific planned action violates a prohibition. These changes have to be observed and remembered by the human planner (which is a cognitively demanding task) and make collaborative planning under such norms/policies a complicated task. Agents can provide assistance to a human planner by detecting and advising a planner when policies become active, when policy violations occur and may also propose courses of actions that may resolve such conflicts.

Agents, as we utilize them, do not form coalitions themselves or are part of a virtual organization. These agents operate in a supportive role to a human planner in order to ease the cognitive burden on human planners during a collaborative planning effort. The agent is assigned solely to a specific human planner and operates in a monitoring, controlling and/or advicing capacity. In order for the agent to work effectively in tandem with the human planner, the agent must intercept any communication and planning action *before* it is actually performed in order to provide the human planner with warnings in case violations were to occur and possibly advise how to rectify such a situation. The agent, therefore, has to maintain a representation of the normative position of the human planner, which is a *potential* one (and not the actual situation). Based on this "outlook" at a potential future normative situation, the agent can reason about appropriate responses to that.

In the experiments performed, we were interested in how an agent can support a collaborative planning activity. In particular, we investigated two supporting strategies or "aiding conditions" for an agent:

– in the *critique* condition, the agent detects policy violations that are incurred by the human planners in their communication and planning behavior. In case of a violation, the agent either (a) intercepts the sending of a message or (b) interrupts the planning of actions in order to inform the human planner about the set of policies that become activated due to these intended and violating actions – the planner can then decide whether to adhere to such an advice or to ignore the agent and intentionally violate a policy;
– in the *censor* condition, the agent still monitors the activities of the human planner, but silently interferes with the communication by deleting offending parts of the exchanged messages (or blocking them completely) in order to avert policy violations; in that case, the *receiver* is informed that a message is either truncated or completely censored.

The difference between the two types of agents is in their policy-related feedback to the human planner and their subsequent interaction. The critic agent, besides reasoning about policies, also monitors plan steps committed by a human planner and reasons about the effect of policies on planned actions. The censor agent, on the other hand, is not concerned with effects of policies on planned actions int only interecpts and forbids the transmission of messages that contain policy violations. The *critique* agent does not force the user to a particular action, it merely provides advice and suggestions, which the user can accept or reject. Both conditions are compared to a third control condition (the *unaided* condition), where the test subjects did not have any agent support.

2.1 Modelling Policies

Policies are given to the human planner in a verbalized form. The following example is taken from our example domain outlined in subsequent sections:

Example 1. "IF you want to deploy an ambulance along route R on day D for a rescue operation, THEN you are obliged to obtain a commitment of escort from your coalition partner"

This policy will become relevant to the human planner in the course of planning such a rescue operation, if the deployment of this specific resouce is intended. In becoming relevant, it adds to the current "social burden" of the human planner – it has to observe this obligation (beside possible other activated norms) and see to it that it is fulfilled. This obligation will be fulfilled when such a commitment of escort is obtained. In that case, we regard the obligation to have *expired*. We, therefore need to specify these additional fulfillment or *expiration conditions*. The example above would then be complete by ammending it with the following information:

Example 2. "IF you want to deploy an ambulance along route R on day D for a rescue operation, THEN you are obliged to obtain a commitment of escort from your coalition partner. IF you have aquired a commitment of escort along route R before day D THEN this obligation is fulfilled"

Independent of whether an obligation is fulfilled, it will also be de-activated in case that the activating circumstances no longer hold. In case of the above example, if the human planner decides to discard the planned deployment of an ambulance, this obligation is no longer relevant.

The following example shows a prohibition:

Example 3. "IF you know that the route R on day D is dangerous for deployments, THEN you are prohibited to deploy an ambulance along route R on day D for a rescue operation"

This prohibition becomes relevant if there is knowledge about danger on the given route available. As is obvious, this can also be formulated as a permission: "If there is no knowledge of danger ... THEN you are permitted ...". It shows that, in the design of policies, we have to clarify the *default normative position* for a coalition partner: the point of view from which the policies are designed – are we assuming that "everything is permitted that is not explicitly prohibited" or do we take the stance that "everything is prohibited that is not explicitly permitted"? For the design of our policies, we decided that, per default, any plan and communication action is permitted and that we provide explicit prohibitions only in cases where this does not obtain (explicit permissions may be included to specify particular exceptions to a given prohibition, e.g. "You are ONLY permitted, IF ..."). In the same way as obligations, prohibitions must be augmented with conditions that indicate the circumstances under which a violation occurs. Due to the specific way the agent processes norms (as described later), prohibitions have an activation condition that describes the (set of) violating circumstances – with an activation of a prohibition, its violation is indicated, whereas in its deactivated state, it is regarded as not violated.

In accordance with definitions used in previous work [1], we describe here the normative position of the human planner, which is monitored by the agent (and, therefore, is also the normative position of the agent), as the set Ω of currently *activated* policies:

Definition 1. *The set Ω comprises the currently instantiated policies, containing the permissions given, the obligations that must be fulfilled, and the prohibitions that are potentially under threat of violation.*

Note, that the agent intercepts the actions of the monitored human planner, before these actions actually take place. This allows the agent to assess the normative consequences of an action, before harm is done and can inform the human planner accordingly. If Ω contains activated prohibitions then the agent signals *potential* violations and not actual ones. By providing information about those violations and active obligations to the human planners, the monitoring agent may be able to motivate them to correct their behavior.

With respect to an implementation of such an agent, with each occurrence of either a communication or planning action, we regard the set Ω being discarded, the activations of all policies checked afresh and a new set Ω' created. Ω' represents the *potential* normative position of the human planner that would obtain,

if the intercepted actions take place. If the coalition finishes its collaborative planning activity, the set Ω^{final}, maintained for an individual coalition partner by its monitoring agent, can have the following states:

(a) Ω^{final} is empty or contains only permissions – the human planner has a clean record with all obligations fulfilled and no prohibitions violated, or
(b) Ω^{final} still contains obligations and/or prohibitions – this indicates, that those obligations were not fulfilled and the violation of prohibitions persisted beyond the planning session.

At that point in time, Ω^{final}, represents the *actual* normative state of the human planner.

Our representation of policies follows our earler work [2]. We specify an obligation, permission or prohibition on a particular action with two condition – an activation and an expiration/fulfillment condition – determining whether a policy is relevant to the human planner. If we define the set $Expr$ as the set of all possible well-formed formulae comprising first-order predicates over terms (constants, variables and the operators $\wedge$, $\vee$ and $\neg$, then a policy can be defined in the following way:

Definition 2. *A policy, expressing an obligation, permission, prohibition is a tuple* $\langle \nu, \rho, \varphi, a, e \rangle$*, where*

- $\nu \in \{\mathcal{O}, \mathcal{P}, \mathcal{F}\}$ *is a label indicating whether this is an obligation, permission or prohibition*
- ρ *is a role identifier for a norm addressee*
- φ *describes the action regulated by this policy*
- $a \in Expr$ *is the activation condition*
- $e \in Expr$ *is the expiration condition*

This definition displays in a simple fashion the elements that characterize an implementation of our policies – they are ascribed to a specific role (in our experiments, we have the roles "Party A" as the humanitarian organization and "Party B" as the military organization) and are activated/de-activated under certain conditions. The policies themselves exist in two forms, (a) formulated in simple "IF ... THEN ..."-statements that are given to human planners, and (b) implemented as a set of rules, expressing their activation/de-activation, in order to allow agents a processing of these policies and the reasoning about their current activation state.

3 Planning and Communication Environment

For agents to become operational, they must have access to plans in development and communication activities. We use a traditional forward-chaining mechanism (expert system shell Jess [3]) to implement the policy reasoning mechanism for an agent. According to our model, a policy will experience activations and de-activations under specific circumstances. In order to correctly implement their

activation and de-activation, each policy is expressed by a set of rules and data-structures recording such an activation state.

As we noted before, the agent operates in a fixed monitoring cycle:

(a) detect the current situation changed by arriving messages expressing the coalition partners' commitments for action or revealed intelligence, as well as new planned actions,
(b) reason about these changes, and
(c) create the new set of activated policies.

The agent has to intercept both communication and planning actions in order to update an internal representation of the normative situation at hand.

3.1 Conversations during Planning

In terms of communication between human planners, we strongly simplified and restricted the way conversations within a coalition takes place. In order for agents to easily monitor communication and reason about the messages exchanged, human planners converse in writing, using a specific set of message types.

In this conversation, a planning party may request another coalition member to commit to a specific action or to provide particular information. The planning party itself may also be the target of such a request. On the other hand, planning parties may pro-actively offer information or commit to specific actions as they develop their own plan. Finally, a planner may have to change its plan and, therefore, withdraw commitments or withdraw requests or offers. By identifying these general types of conversations, we can establish a set of message types:

Performative	Type
REQUEST	commitment, information
OFFER	commitment
INFORM	information
ACCEPT	commitment
PROVIDE	commitment, information
DENY	commitment, information
WITHDRAW	commitment, request for information

Table 1. Message Types

Messages according to these types are used in conversations that follow particular transitions as shown in figure 1. Dialogs for requesting a commitment or pro-actively offering it are shown. A REQUEST (for commitment) has to be answered with either a PROVIDE (a commitment) or a DENY. In the pro-active case, an OFFER can be answered with either an ACCEPT or a DENY. Requests, commitments, offers and the acceptance of offers can be withdrawn in a separate WITHDRAW conversation. The state transistions in figure 1 show

annotations such as `A:RCx` etc. These indicate that, for example, party A sends a REQUEST for a commitment (the `x` is a placeholder for particular domain-specific information).

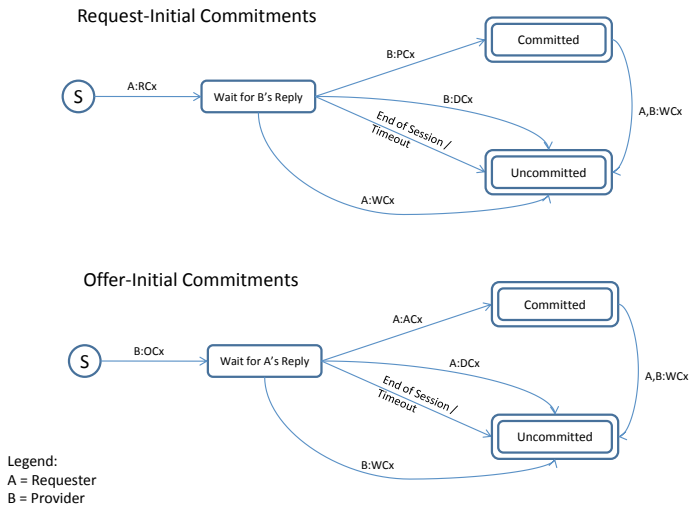


Fig. 1. Request and Offer Conversations

Not shown is INFORM, as it is a special case. It allows to disclose particular information proactively such as the set of plan steps in the planner's own plan or confidential information, without requests from another party or expecting a reply.

For planning, a human planner is provided with a set of domain-specific actions and basic manipulation operations to assemble a plan.

4 Scenario

We chose rescue missions as an example scenario and performed a set of experiments to investigate the effectiveness of agents supporting a collaborative planning effort in the context of this scenario.

In this scenario, we assume that there are two parties that form a coalition, a humanitarian relief organization with the individual goal of rescuing injured civilians from a potentially hostile region, and a military organization that has to coordinate its military objectives with the evacuation activities. In the experimental setup, the humanitarian organization is regarded as "Party A" and the

military organisation as "Party B". The goal of this coalition is to find a joint plan for rescuing as many injured people from a dangerous region to a hospital in the shortest possible time. The optimal situation for Party A would be to provide medical attention and evacuation as soon as possible. For this, party A may need support from party B, for example, an escort through a dangerous region. Party B, on the other hand, has military objectives that, potentially, may be in conflict with the support given to party A.

We assume that both parties have a set of resources such as ambulances, field hospitals/paramedic units, rescue helicopters, Jeeps, etc.. During their planning activity, the coalition partners will allocate these resources to be used in planned actions. We assume that party A and party B have a small set of capabilities they may plan to utilize in pursuing a mission. Party A can either evacuate wounded people, taking round-trips to their location or dispatch a paramedic unit to provide medical care at their location directly. Party B may either support party A by providing escort through dangerous terrain or pursue its own military goals by attacking enemy strongholds.

In support of the communication necessary during planning, a set of domain-specific speech acts is provided according to the types of performatives outlined above. Previously, we determined that there is a need for the performatives REQUEST, OFFER, INFORM, PROVIDE, ACCEPT, DENY and WITHDRAW to enable regimented conversations between human planners. We also saw that these conversations are subject to the exchange of either commitments or specific information necessary to the decision process of whether a specific action should be planned or not. The kind of commitments or information exchanged with these messages is domain-specific. In our case, we assume that party A and B exchange the following commitments:

- guarantee safety of a road (B to A)
- provide escort (B to A)
- evacuate (A to B)
- dispatch (A to B)

For example, party A can issue a REQUEST to party B asking for an escort. Party B would then either provide this commitment or deny the request. Information about the following aspects may be exchanged as well:

- intelligence
- intelligence source
- plan step
- specifics (request/provide more details regarding what road/resource/day)

With that, the parties are enabled to disclose intelligence or the source of intelligence, or to communicate planned activities and whether a road is safe for evacuating wounded etc.

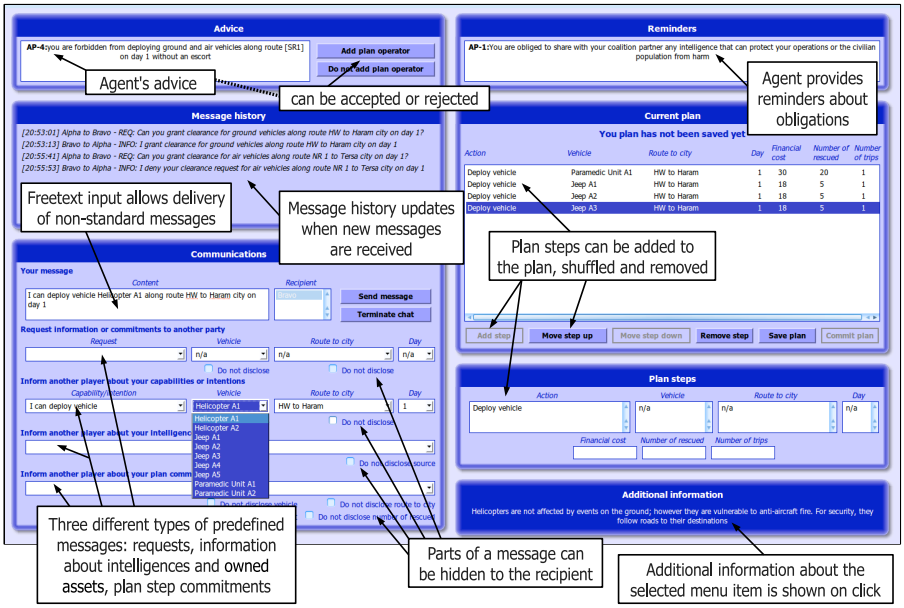


Fig. 2. Experiment Interface

5 Experiments

The human planner interacts with the agent via the application as depicted in figure 2 – the interface is shown in a mode where agents operate in the "critique" condition. The purpose of this interface is to provide the user with the possibility of sending messages to coalition partners, maintain a plan and interact with an agent. If the agent operates in the "critique" condition, it will report back on the results of its monitoring the communication and planning activities of the human planner. The following information is provided to the user in the top section of this interface (characterizing a potential normative situation):

- reminders about the current set of active obligations (right upper window)
- the set of *potential* violations of prohibitions (left upper window) – the user can either ignore this warning (intentional violation of a norm) or accept it

If the user accepts the warning of a policy violation, the offending action – sending a message or adding a particular plan step will be aborted. If the user ignores the warning these actions would go ahead.

In addition, the user manages its communication by assembling messages via pull-down menus (left part of the interface) or uses a chat window to directly converse with a coalition partner (which is not monitored by the agent), as well as its plan in the right part of the interface.

5.1 Experimental Task

The experiments conducted are characterized by the interaction of two human test subjects acting as planners in the role of the humanitarian organization (Party A) and the military organization (Party B). Both test subjects are provided with details about their private *goals*, *resources*, *intelligence*, their *capabilities*, and *policies*. Both parties are given different maps that outline locations or *destinations*, from where – in the case of Party A – injured people must be evacuated, or that represent insurgent strongholds that must be defeated by Party B. These destinations have numerical requirements – in case of Party A, a specific number of wounded to be evacuated, in case of Party B, insurgent strongholds have a specific resistance value that must be overcome by military means and incur costs.

Thirty teams, each comprising two paid subjects, were recruited to participate in the study. These teams were tested in their collaborative planning effort in one of three conditions, the unaided condition (control), the condition where the agent acted as a "critic", and the condition where the agent acted as a "censor", resulting in ten teams operating in each of the three conditions. The test subjects were forbidden to share computer screens, note sheets or other such aids and worked isolated from each other. They could only describe their intentions, commitments and planned resource deployments by using either a structured representation of messages or a free-form text chat box of the experiment software environment. The test subjects were given written documents as well as shown a video briefing them about the impending task explaining the mission objectives, resources, policies, resource deployment costs and planning constraints (e.g. a jeep can take only 5 wounded in each deployment). In a first step, a team performed a practice problem as a warmup in order to become familiar with the planning process. In particular, the practice problem of party A was: "Plan the lowest cost emergency medical evacuation to the village of Tersa on Day 1. Be sure to do so in a way that is compatible with your policies. What is the total cost of your operation?" As the second step, the team then performed the complete planning problem in one of the described experimental conditions. The total allotted time to finish the whole experiment including reading the briefing, video viewing and performing the practice problem, was 2 hours.

6 Results

The results of these experiments have shown that agents can have a positive impact on the enforcement of policies. We saw that in the unaided experiment condition (no agent monitoring and feedback), individuals would make on average from $7 - 10$ policy violations, with all individuals making at least three policy violations, per session. With reference to Figure 3, we saw that the rate of individuals deliberately violating policies dropped to a median average of one violation per session, with many individuals not making any policy violations at all.

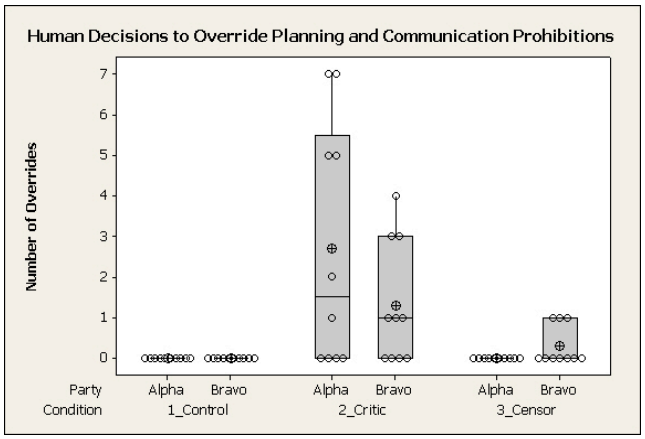


Fig. 3. Willful User Override of Agent Enforcement

This article reports experiments with two types of aiding strategies: a critic and a censor. Neither form of assistance prompted the human subjects to attempt more or fewer policy violations, as evidenced by the lack of statistical significance in differences shown in Figure 4. There was a minority of human subjects, however, that adjusted the ways in which they used the agents as a result of the type of agent intervention. For example, in the censor condition, some users would try to exhaustively generate-and-test communications for granting clearances or committing to escorts against the censorship of the agent.

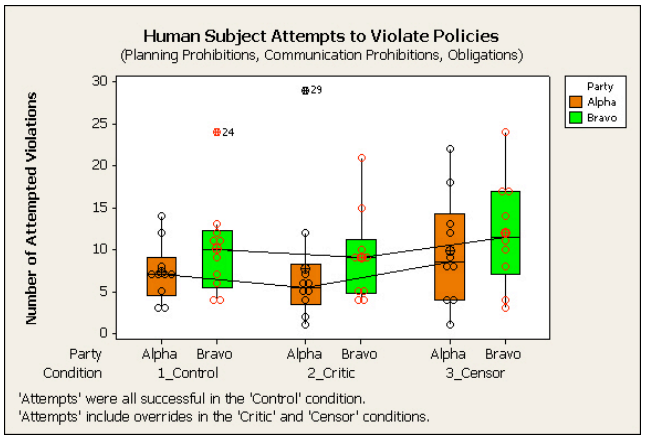


Fig. 4. All User-Committed Policy Violations

The following characteristics were observed about the critic condition.

- 60% of the subjects in both Parties A and B felt that it was necessary to override the critic agent in order to complete their plans (Figure 3).
- only one subject out of thirty actually reached the mission objective of treating 100% of the wounded on the first day of the mission. The subject did so without violating any policies, but with the assistance of 13 interventions from the critic agent.
- there is a slight degradation of performance between subjects in the control and in the critic conditions. We hypothesize that this behavior is due to the critic agent focusing the user's attention on avoiding policy violations rather than on the objectives of their task.
- the mean plan cost is slightly higher in the critic condition than in the control condition. When considering, however, that 4 out of 10 subjects in the critic condition did not violate any mission-impacting policies, it is possible that Party A's plan costs are better approximations of the true plan costs.

The following characteristics were observed about the censor condition.

- Of the two agents, critic and censor, the censor agent was the most effective at preventing policy violations. Only 3 out of 20 individuals actually circumvented the censorship of the censor agent, each one committing only 1 violation.
- The censor agent was unable to provide feedback on mission impacting policy violations (MIPVs) that were introduced as plan steps, so its performance in reducing MIPVs cannot be distinguished from the control condition.
- The Party A subjects in the censor condition were most distracted from their mission objective of treating as many wounded as possible on day 1. We hypothesize that the lack of direct feedback to the user committing the violation may be the cause.
- Similarly for both parties, the plan costs were greatest in the censor condition. We hypothesize that the lack of direct feedback to the user committing the violations may cause confusion, distracting the user from being mindful of their plan costs.

Results from these experiments provide us with feedback that informs the design of future policy-enforcing agents. For example, the censor agent was clearly the undisputed best enforcer of policy. Its enforcement reliability was countered by a significant increase of plan costs and distraction from the mission objectives for Party A. The critic agent immediately flagged policy violations, so it was possible for at least one subject in all three conditions to achieve the perfect plan (e.g. treating all the wounded on day 1), as well as enabling others to have zero mission impacting policy violations. A possible next agent to test would be a critiquing censor agent that: is capable of critiquing plan steps, provides direct feedback to the user that it is censoring, and does so without allowing the censored subject to override it.

7 Related Work

Policies have been used in disparate fields, ranging from security models of programming languages to the management of resources in distributed IT systems [4]. In this context, policies are usually regarded as "permissions" that allow the performance of specific actions such as access to data or the use of network resources. This view has its limitations: it is assumed that what is not explicitly permitted is prohibited and the concept of an obligation is not present traditionally. Recent work [5, 6] introduces richer concepts for describing policies (e.g. "obligation policies").

Our concept of a policy is strongly alligned with research into normative systems, in particular work on norm-governed agency, virtual organizations [7–10] and Electronic Institutions [11–14]. In this paper, we expand on work presented in [15] and describe a specific application of normative agents, where agents do not form virtual organisations, but observe the behaviour of human planners within a coalition. In this setting, the agent is focussed on understanding current knowledge held by the human planners – what intelligence they hold about the current state of the world, what commitments they received and made, as well as what requests they expressed and what their current plan is. With such monitoring agents in place, we share similar concerns as those posed in the context of Electronic Institutions, as outlined, for example, in [13, 14]. In a very similar fashion, we must monitor the communication behavior of coalition partners, assess the eligibility of the messages exchanged and reason about the current normative state. By using a rule-based language for encoding policies and, consequently, a rule engine such as the Jess Expert System Shell for processing these specifications, we followed a very similar implementation path as described in [13, 12], in particular implementing the reasoning about norms in Jess [16].

8 Conclusion

In this paper, we discussed difficulties in establishing joint plans within coalitions in the face of self interest, individual goals and diverse policies of the coalition members. For a planner, it would be easiest to operate without any restrictions, constraints or regulations on the operations that may be added to a plan. In a social context and, in particular, a diverse one such as coalitions of independent partners, this is not possible. As we showed, detailed and, sometimes, even conflicting policies have to be dealt with in practice, when coalitions try to engage in collaborative planning. With such policies and restrictions in place, planning becomes more complicated and optimal plans may be hard to achieve. We therefore advocate agent support for policy-based planning activities within coalitions. In this paper, we demonstrated how agents can be integrated into the dialogical process of human planners establishing a collaborative plan. We described two agent-based strategies for assisting the collaborative planning process: (a) a "critic" that provides active feedback about the fulfillment of policies and (b) a "censor" agent that silently manipulates the interaction between human planners so that their interaction and information exchanged takes place according

to given policies. We have outlined an experimental framework that allows us to evaluate the effects of these strategies in the context of a military-humanitarian scenario and presented data that shows the impact of agent support on the planning results.

9 Acknowledgement

Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

1. Vasconcelos, W.W., Kollingbaum, M.J., García-Camino, A., Norman, T.J.: Resolving Conflicts and Inconsistency in Norm-Regulated Virtual Organizations. In: AAMAS 2007. (2007)
2. Kollingbaum, M.: Norm-governed Practical Reasoning Agents. PhD thesis, University of Aberdeen (2005)
3. Friedman-Hill, E.: Jess in Action. Manning (2003)
4. Charalambides, M., Flegkas, P., Pavlou, G., Bandara, A., Lupu, E., Russo, A., Dulay, N., Sloman, M., Rubio-Loyola, J.: Policy Conflict Analysis for Quality of Service Management. In: 6th IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2005). (2005)
5. Zhao, H., Lobo, J., Bellovin, S.M.: An Algebra for Integration and Analysis of Ponder2 Policies. In: 2008 IEEE Workshop on Policies for Distributed Systems and Networks POLICY 2008. (2008)
6. Kagal, L., Finin, T.: Modeling Conversation Policies using Permissions and Obligations. In: Journal of Autonomous Agents and Multi-Agent Systems. Volume 14. Springer-Verlag (April 2007) 197–206
7. Dignum, F.: Autonomous Agents with Norms. Artificial Intelligence and Law **7** (1999) 69–79
8. Dignum, V.: A Model for Organizational Interaction: based on Agents, founded in Logic. PhD thesis, SIKS Dissertation Series 2004 (2004)
9. Lopez y Lopez, F., Luck, M., dÍnverno, M.: Normative Agent Reasoning in Dynamic Societies. In: Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2 (AAMAS2004). (2004)
10. Kollingbaum, M., Vasconcelos, W., Garcia-Camino, A., Norman, T.: Conflict Resolution in Norm-Regulated Environments via Unification and Constraints. In: DALT 2007. (2007)
11. Rodríguez-Aguilar, J.A.: On the Design and Construction of Agent-mediated Electronic Institutions. PhD thesis, Institut d'Investigació en Intel·ligència Artificial (IIIA), Consejo Superior de Investigaciones Científicas (CSIC), Campus UAB, Bellaterra, Spain, Spain (2001)

12. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Norm Oriented Programming of Electronic Institutions. In: 5th Int'l Joint Conf. on Autonomous Agents & Multiagent Systems. (AAMAS'06), Hakodate, Japan, ACM Press (May 2006)
13. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions. ACM SIGecom Exchanges **5**(5) (January 2006) 33–40
14. Vasconcelos, W.W.: Norm Verification and Analysis of Electronic Institutions. In: DALT 2004. Volume 3476 of LNAI. Springer-Verlag (2004)
15. Burnett, C., Masato, D., McCallum, M., Norman, T.J., Giampapa, J., Kollingbaum, M.J., Sycara, K.: Agent Support for Mission Planning Under Policy Constraints. In: Proceedings of the Second Annual Conference of the International Technology Alliance. (2008) 100–107
16. García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A.: Implementing Norms in Electronic Institutions. In: Procs. 4th AAMAS. (2005)

Internal agent architecture for norm identification

Bastin Tony Roy Savarimuthu, Stephen Cranefield, Maryam A. Purvis and Martin K. Purvis

Department of Information Science, University of Otago, Dunedin, P O Box 56, Dunedin, New Zealand
(tonyr,scranefield,tehrany,mpurvis)@infoscience.otago.ac.nz

Abstract. Most works on norms in the multi-agent systems field have concentrated on how norms can be applied to regulate behaviour in agent societies using a top-down approach. In this work, we describe the internal architecture of an agent which identifies what the norm of a society is using a bottom-up approach. The agents infer norms without the norms being given to them explicitly. We demonstrate how the norm associated with using a park can be inferred by an agent using the proposed architecture.

1 Introduction

Software agents that act as proxies to real world entities need to adapt to the changing needs of environments. An example would that be of virtual worlds (e.g. SecondLife [1]). Virtual environments offer a rich and expressive environment for agent interactions. Traditionally, norms have governed the behaviour of agent interactions in a closed system. In open systems such as virtual worlds, agents instead of possessing predetermined notions of what a norm is, should be able to infer and identify norms through observing patterns of interactions and their consequences.

Recognizing the norms of a society is beneficial to an agent. This process enables the agent to know what is permissible within a society and what is not. As the agent joins and leaves different agent societies, these capabilities are essential for the agent to modify its expectations of behaviour depending upon the society it is a part of. As the environment changes, the capability of recognizing the new norm helps an agent to derive new ways of achieving its intended goals.

In this work we describe an internal agent architecture for norm identification. Using a park scenario as an example, we describe the design and implementation of the internal agent architecture which aids the agent to infer what the norms of using the park are.

2 Background and related work

2.1 Background on norms

Norms are expectations of an agent about the behaviour of other agents in the society. Norms are of interest to multi-agent system (MAS) researchers as they help in sustaining social order and increase the predictability of behaviour in the society. However, software agents tend to deviate from these norms due to their autonomy. So, the study of norms has become crucial to MAS researchers as they can build robust multi-agent systems using the concept of norms and also experiment with how norms evolve and adapt in response to environmental changes.

Due to multi-disciplinary interest in norms, several definitions for norms exist. Ullman-Margalit [2] describes a social norm as a prescribed guide for conduct or action which is generally complied with by the members of the society. She states that norms are the resultant of complex patterns of behaviour of a large number of people over a protracted period of time. Coleman [3] describes *"I will say that a norm concerning a specific action exists when the socially defined right to control the action is held not by the actor but by others"*. Elster notes the following about social norms [4]. *"For norms to be social, they must be shared by other people and partly sustained by their approval and disapproval. They are sustained by the feelings of embarrassment,anxiety, guilt and shame that a person suffers at the prospect of violating them. A person obeying a norm may also be propelled by positive emotions like anger and indignation ... social norms have a grip on the mind that is due to the strong emotions they can trigger"*.

Based on the definitions provided by various researchers, we note that the notion of a norm is generally made up of the following two aspects.

- Normative expectation of a behavioural regularity: There is a general agreement within the society that a behaviour is expected on the part of an agent (or actor) by others in a society, in a given circumstance.
- A norm spreading factor: Examples of norm spreading factors include the notion of advice from powerful leaders and the sanctioning mechanism. When an agent does not follow the norm, it could be subjected to a sanction. The sanction could include monetary or physical punishment in the real world which can trigger emotions (embarrassment, guilt etc.) or direct loss of utility resulting in the agent internalising the applicable norm to avoid future sanctions. Other kind of sanctions could include agents not being willing to interact with an agent that violated the norm or the decrease of its reputation score. Other norm spreading factors include imitation and learning on the part of an agent.

It should be noted that researchers are divided on what the differences between a norm and a convention are. Our belief is that convention is a common expectation amongst (most) others that an agent adopts a particular action or behaviour. Conventions may become norms once the non-adherence of the focal action specified by the convention is sanctioned. In this paper our concern is on *norms*.

2.2 Related work

Several researchers have worked on both prescriptive (top-down) and emergent (bottom-up) approaches to norms. In a top-down approach, an authoritative leader or a normative advisor prescribes what the norm of the society should be [5, 6]. In the bottom-up approach, the agents come up with a norm through learning mechanisms [7–9]. Researchers have used sanctioning mechanisms [10] and reputation mechanisms [11, 12] for enforcing norms.

The work reported in this paper falls under the bottom-up approach in the study of norms. Many researchers in this approach have experimented with game-theoretical models for norm emergence [7,10]. Agents using these models learn to choose a strategy that maximizes utility . The agents do not possess the notion of a "normative expectation" in these works. Very few have investigated how an agent comes to know the norms of the society. Our objective in this work is to propose an architecture where agents can identify what the norms of the society are. Several researchers have proposed architectures for normative systems. For a comparison of these architectures refer to Neumann's article [13].

We note that our work parallels the work that is being carried out by the researchers involved in the EMIL project [14]. Researchers involved in the EMIL project [14] are working on a cognitive architecture for norm emergence. There have been some attempts to explore how the mental capacities of agents play a role in the emergence of norms.

The EMIL project aims to deliver a simulation-based theory of norm innovation, where norm innovation is defined as the two-way dynamics of an inter-agent process and an intra-agent process. The inter-agent process results in the emergence of norms where the micro interactions produce macro behaviour (norms). The intra-agent process refers to what goes inside an agent's mind so that they can recognize what the norms of the society are. This approach uses cognitive agents that examine interactions between agents and are able to recognize what the norms could be. The agents in this model need not necessarily be utility maximizing like the ones in the learning models. The agents in the model will have the ability to filter external requests that affect normative decisions and will also be able to communicate norms with other agents. Agents just employing learning algorithms lack these capabilities.

Researchers involved with the EMIL project [15, 16] have demonstrated how the norm recognition module of the EMIL-A platform works. In particular they have experimented with an imitation approach versus the norm recognition approach that they have come up with. The norm recognition module consists of two constructs, the normative board and a module for storing different types of modalities for norms (which they refer to as modals). Each modal represents a type of message that is exchanged between agents (e.g. the deontics modal refers to distinguishing situations as either acceptable or unacceptable). The normative board consists of normative beliefs and normative goals. They have shown that norm recognizers perform better than social conformers (imitating agents) due to the fact that the recognizers were able to identify a pool of potential norms while the imitators generated only one type of norm.

The work reported here differs from this work in three ways. Firstly, we have chosen "reaction" (positive and negative) to be a top level construct for identifying potential norms when the norm of a society is being shaped. We note that a sanction not only may imply a monetary punishment, it could also be an action that could invoke emotions (such as an agent yelling at another might invoke shame or embarrassment on another agent), which can help in norm spreading. Agents can recognize such actions based on their previous experience. Secondly, we identify three different sets of norms in agent's mind: suspected norms, candidate norms and identified norms. Thirdly, we demonstrate how our architecture allows for an agent to identify co-existing norms.

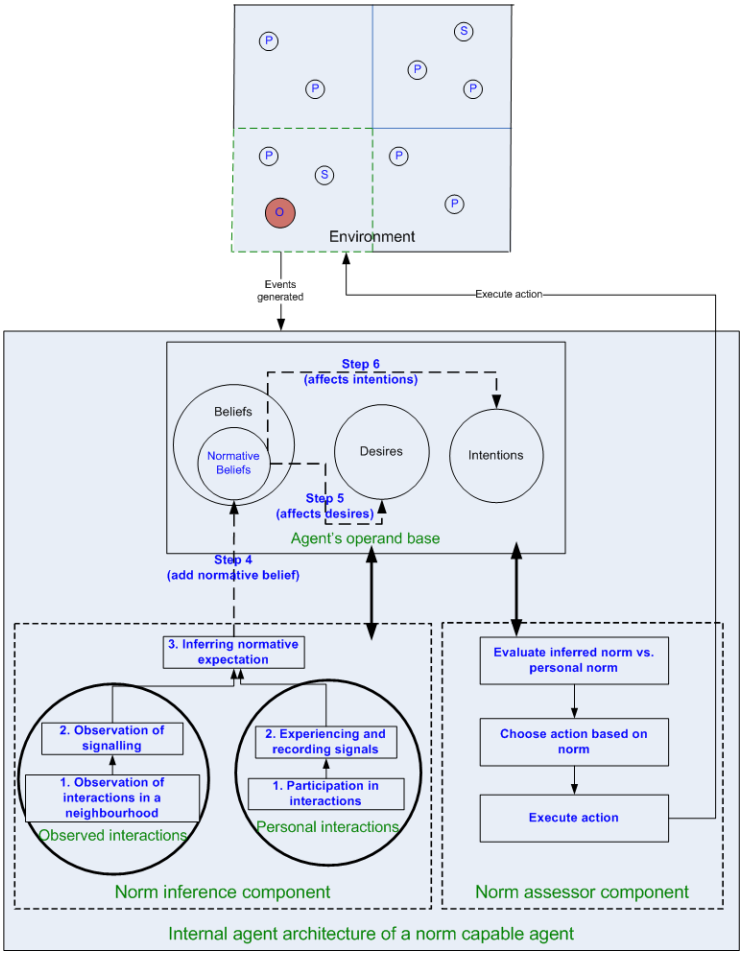


Fig. 1: Higher level architecture of norm identification

3 Architecture for norm identification

This section describes the normative inference architecture of an agent. The architecture provides a sequence of six steps that an agent goes through before it comes to know what a norm of the society is, as shown in Figure 1.

To understand the architecture let us assume that an agent society exists. Let us also assume that a norm does not exist to start with or only a few of the agents have a notion of what an appropriate action should be in a particular circumstance (a personal norm). In this architecture a typical agent would first observe the interactions that occur between the agents in the society. The interactions could be of two types. The first type of interaction is the one in which the agent itself is involved and is called a *personnel interaction* (an action that an agent does in an environment). The second type of interaction is an interaction between other agents that is observed by the observer agent, referred to as an *observed interaction*. The agent records these interactions. The top part of Figure 1 shows the types of agents in an agent society. An agent in the society can assume one or more of the three roles: a participant (P) that is involved in a personal interaction, an observer (O) and a signaller (S).

The actions observed by an observer are of two types: regular actions and signalling actions. A regular action is an event such as an agent moving to another location in a park or sitting on a bench. Signalling actions can be thought of as special events that agents understand to be either encouraging or discouraging certain behaviour.

For example, let us assume that two agents are in a public park. One agent (A) sees another agent (B) littering the park. Agent B may choose to sanction the agent A (B nods or shakes its head in disapproval and in the worst case yells at the litterer). The observer agent (C) records the signalling that takes place between these agents. The signals can either be positive or negative and it depends on one kind of norm to another. In the case of park littering, agents might issue a negative signal when an agent litters while non-littering might be considered as a normal or routine activity for which there is no positive signalling. In our architecture, signalling is a top level entity because in normative systems it is important for an agent to have an expectation of a particular behaviour. Norms do not appear from nowhere. There might be some norm entrepreneurs or norm innovators who come up with a norm (also known as personal norm (p-norm)). Though few, these agents might sanction or reward others because they violated or followed the norm.

The third step is for the agent to infer normative expectations of a society based on noted observations and signalling. An agent correlates signalling with the observations and infers what its notion of a relevant norm in the society is. A detailed description of how the norm inference works is provided in the next section.

The fourth step is to store this newly formed notion of norm in its belief set. We call the beliefs that are based on norms normative beliefs. For every signal that an agent processes, it re-evaluates its notion of the norm. Based on the inference it can modify the notion of what the norm is at any point of time

which results in dynamic creation of norms. Once the agent has a norm, its desires and intentions are influenced by the norm which might affect its goals and plans (steps 5 and 6).

Once the agent has inferred what the norm it, it will then have to decide whether to follow the norm. The norm assessor component is responsible for making this decision. The agent weighs its own personal norm against the identified norm in a given circumstance and chooses an appropriate action. The emphasis of this paper is on the norm inference component.

4 Inferring norms in a communal park

This section describes the design and implementation of a norm identification system. The context for norms is the usage of a public park.

In many human societies there exists a norm that one should not litter a communal area such as a park. However, software agents that join open societies do not come to know of the norm of a society *apriori*. Let us assume that software agents stroll through a virtual park in environments such as SecondLife [1]. Let us imagine that the virtual park is a two dimensional grid where agents move around and enjoy the park. Agents sometimes become hungry and eat food. Some agents litter (i.e. drop the rubbish on the ground) and some agents carry the rubbish with them and drop it in a rubbish bin. The actions that can be performed by agent X are *move*, *eat* and *litter*. Some agents consider littering to be an activity that should be discouraged, so they choose to sanction other agents through actions such as yelling and shaking their heads in disapproval. We assume that an agent has a filtering mechanism which categorizes actions such as *yell* and *shake-head* as sanctioning actions. These sanctioning agents can be considered as norm entrepreneurs.

Let us assume that the agents can observe each other within a certain visibility threshold (e.g. agents can only see other agents in a 3 cell neighbourhood). Agents can either be a direct participant in interactions or observers. Some participants can be sanctioning agents . The observer records another agent's actions until it disappears from its vicinity. Whenever it encounters an action of type *sanction*, it recognizes that something has gone wrong (e.g. the action is against the personal norm of the punishing agent). When such an event occurs, the agent may become emotionally charged and perform certain sanctioning action such as yelling at the litterer or shaking its head vigorously in disapproval. Hence, an agent observing this can infer that someone involved in an interaction has violated a norm. We assume that there exists a filtering mechanism in the agent that can recognize sanctioning and rewarding actions when they occur.

Let us assume that an agent perceives other agents' actions. An event that is perceived consists of an event id, an observed action, and the agent(s) participating in that event. For example an agent observing another agent eating will have the representation of *do(1,eat,A)*. This implies that the observer believes that the first event was generated by agent *A* which performs an action *eat*. A sample representation of events observed by an agent is given below.

$$\begin{pmatrix} do(1, eat, A) \\ do(2, litter, A) \\ do(3, move, B) \\ do(4, move, A) \\ do(5, sanction, B, A) \end{pmatrix}$$

Event 5 is a sanctioning event where agent B sanctions agent A. An agent records these events in its belief base. The agent has a filtering mechanism, which identifies signalling events. We can consider the filtering mechanism to be a black box that recognizes an emotionally charged event such as yelling and shaking head in disapproval and categorizes those actions to be sanctions [1]. When a sanctioning event occurs, it triggers the invocation of the norm inference module of the agent. It should be noted that signalling events can both be positive (e.g. rewards) and negative (e.g. sanctions). In this work, we have focused on the latter type of signalling.

Figure 2 shows the architecture of the norm inference component of an agent. The following sub-sections describe the four sub-components of the norm inference component.

4.1 Creating event-episodes

Agents record other agents actions in their memory. Let us assume that there are three agents A,B and C. Agent A eats, litters and moves while agent B moves and then sanctions. Agent C observes these events and categorizes them based on which agent was responsible for creating an event. $\{A\}$ followed by right arrow ($\rightarrow$) indicates the categorization of events performed by agent A as observed by agent C. A hyphen separates one event from the next.

$$\begin{pmatrix} \{A\} \rightarrow do(1, eat, A) - do(2, litter, A) - do(4, move, A) \\ \{B\} \rightarrow do(3, move, B) - do(5, sanction, B, A) \end{pmatrix}$$

When a sanction occurs, an observer agent extracts the sequence of actions from the recorded history that were exchanged between the sanctioning agent and the sanctioned agent. In the example shown above, the observer infers that something that agent A did may have caused the sanction. It could also be that something agent A failed to do might have caused a sanction. In this work we concentrate on the former of the two. Agent C then extracts the following sequence of events that take place between A and B based on the information retrieved from its history.

$$\{A, B\} \rightarrow eat(1, A) - litter(2, A) - move(4, A) - sanction(5, B, A)$$

[1] Recognizing and categorizing a sanctioning event is a difficult problem. In this paper we assume such a mechanism exists (e.g. based on an agent's past experience)

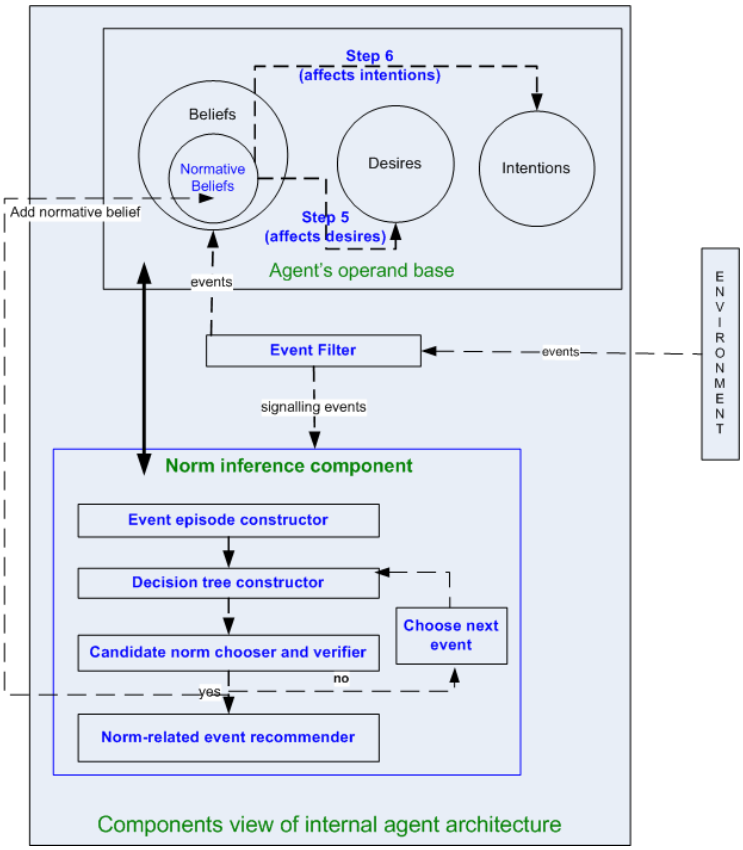


Fig. 2: Architecture of the norm inference component

To simplify the notation here afterwards only the first letter of each event will be mentioned (e.g. e for eat). The event episode for interactions between agents A and B shown above will be represented as

$$\left(\{A, B\} \rightarrow e - l - m - s\right)$$

There might be a few sanctioning events at any given point of time that an agent observes. A sample list containing ten event episodes that are observed by an agent in a certain interval of time is given below.

$$\begin{pmatrix} e-l-m-s, l-e-l-s, m-e-l-s, e-l-e-s, e-l-e-s \\ l-e-l-s, e-e-l-s, m-e-l-s, e-l-m-s, e-l-e-s \end{pmatrix}$$

4.2 Constructing an event-tree based on conditional probability

Once the event episodes are constructed, the agent creates a tree of events that occur in all episodes based on the estimation of conditional probabilities for events that might have led to sanctioning. The mechanism for constructing a decision tree is explained below.

For calculating the conditional probabilities for events that precede a sanction, an agent follows the following steps.

1. Categorizes episodes into events belonging to different levels.
2. Constructs a conditional probability tree of sub-episodes
3. Ranks sub-episodes and chooses candidate norms for verification

Categorizing episodes into event levels - Based on a certain fixed number of events that precede a sanction, an agent categorizes events of an episode into certain levels (e.g. single-level events, two-level events and three-level events). Let us assume that an agent is interested in n events in a sequence that precede a sanction. As an example let us consider *e-l-m-s*, which is the first episode from the sample list of ten episodes. The sequence of events that precede a sanction is *e-l-m* and hence the value of n is three. A single level event (level 1) is an event that precedes a sanction (i.e. m). Two-level events (level 2) are the events that are a combination of two events that precede a sanction (i.e. *e-l* and *l-m*). Three-level events (level 3) are the events that are a combination of three events that precede the sanction (i.e. *e-l-m*). Let us call each entry in these levels a sub-episode.

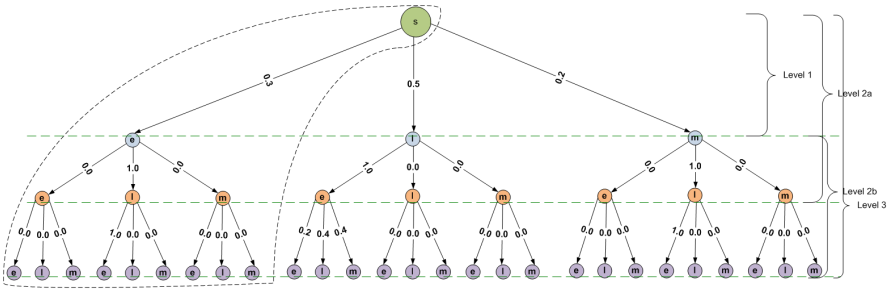


Fig. 3: Events-tree of all episodes based on conditional probability

Constructing a tree based on conditional probability - For each sub-episode in each level, the agent calculates the conditional probability. Sub-episodes for an episode *e-l-m* are *e,l,m* at level 1, *e-l* and *l-m* at level 2 and *e-l-m* at level three. The conditional probability tree of the sample list of ten events as shown in Section 4.1 is given in Figure 3.

For the sake of simplicity, let us only consider those sub-episodes that end with e in the region encompassed by a dashed line in Figure 3. In the sample list that consists of ten episodes, there are three episodes that end with event e. So, the conditional probability of event e given that a sanction has occurred is p(e|s)=0.3. One of the three events (e or l or m) could have occurred before e. The conditional probability of e occurring given that an *e-s* has occurred is p(e-e-s|e-s)=0.0 and the other two conditional probabilities are p(l-e-s|e-s)=1.0 and p(m-e-s|e-s)=0.0. Based on these, we know that p(l-e-s|s)=0.3 and the p(e-e-s|s)=0 and p(m-e-s|s)=0. Now again, three events (e or l or m) could have preceded l. The conditional probabilities p(e-l-e-s|l-e-s)=1.0 and p(l-l-e-s|l-e-s)=0 and p(m-l-e-s|l-e-s)=0. From these, we can infer that p(e-l-e-s|s)=0.3, p(l-l-e-s|s)=0 and p(m-l-e-s|s)=0.

At level 2, we are also interested to find out the occurrences of all episodes that are made up of two level events (indicated in figure 3 as levels 2a and 2b). Based on permutations with repetitions we know that for choosing two out of three events, there are 9 possible combinations (ee,el,em,le,ll,lm,me,ml,mm). The respective probabilities of each of these sub-episodes is 0.1,1,0,0.5,0,0.2,0.2 and 0,0.

The list given below shows the conditional probabilities of all sub-episodes that have a non-zero probability for all the three levels. We call these sub-episodes *suspected norms*. Note that for simplicity we assume that the representation of p(x|s) is p(x). Additionally, the hyphens will be omitted from the sub-episodes (e.g. *e-l-m* will be represented as *elm*).

1. p(e)= 0.3, p(l)=0.5, p(m)=0.3
2. p(ee)=0.1, p(el)=1, p(le)=0.5, p(lm)=0.2, p(me)=0.2
3. p(ele) = 0.3, p(eel)=0.1, p(lel)=0.2, p(mel)=0.2, p(elm)=0.2

Ranking sub-episodes and selecting candidate norms - The agent ranks sub-episodes based on these probabilities and creates a ranked list using the norm selection parameter (*ns*). An agent chooses only those sub-episodes that have conditional probabilities greater than *ns*. Elements in this subset of norms are referred to as *candidate norms*. For example, if *ns* is set to 50, the following are the candidate norms chosen from the set of suspected norms.

- el (100%)
- l (50%)
- le (50%)

Having compiled a set containing candidate norms, the agent passes this information to the norm verification and identification component.

4.3 Norm verification and identification

In order to find whether a candidate norm is a norm of the society, the agent asks another agent in its proximity. This happens in certain intervals of time (e.g. once in every 10 iterations) [2].

[2] We note that other techniques such as voting are possible.

When two agents A and B interact, A chooses its first candidate norm and asks B whether its current norm is A's candidate norm. If true, A stores this norm in its set of *identified norms*. Otherwise, it chooses a sub-episode of the norm and enquires whether that is the norm. It is possible that B might identify the sub-episode as the norm. If not, A moves on to the second candidate norm in its list [3].

In the case of the running example, the sub-episode *el* has the highest probability for selection and it is chosen to be communicated to the other agent. It asks another agent (e.g. an agent who is the closest) whether it thinks that the given candidate norm is a norm of the society. If it responds positively, the agent infers $prohibit(el)$ to be a norm. If the response is negative, this norm is stored in the bottom of the candidate norm list. It then asks whether the sub-episodes of *el*, which are *e* or *l* are the reasons for sanction. If yes, the appropriate action is considered to be prohibited. Otherwise, the next event in the candidate norm list is chosen. This process continues until a norm is found or no norm is found in which case, the process is re-iterated once a new signal indicating a sanction is generated. When one of the candidate norms has been identified as a norm of the society, the agent still iterates through the candidate norm list to find any co-existing norms.

It should be noted that an agent will have three sets of norms: suspected norms, candidate norms and identified norms. Figure 4 shows these three sets of norms. Once an agent identifies the norms of the system and finds that the norms identified have been stable for a certain period of time, it can forgo using the norm inference component for a certain amount of time. It only invokes the norm inference component in regular intervals of time to check if the norms of the society have changed, in which case it replaces the norms in the identified list with the new ones.

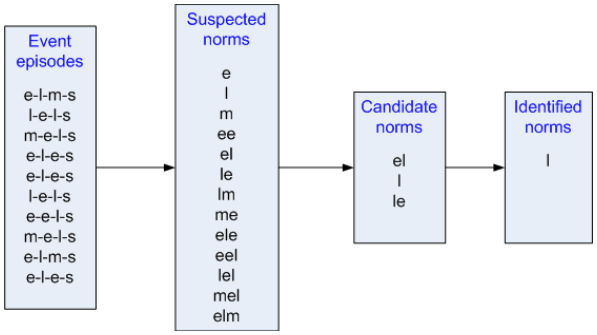


Fig. 4: Three sets of norms

[3] Other alternative mechanisms are also possible. For example, an agent could ask for all the candidate norms from another agent and can compare them locally.

4.4 Related event recommender

Even if the event immediately preceding a sanction was responsible for causing the sanction (e.g. event l), the agent would still be watchful of the event sequences that precede the sanctioned action 100% of the time (e.g. event e) for two reasons. One reason is that when it produces events e and then l, it could be sanctioned. Also when other agents produce events e-l, then if the observer were a sanctioning agent, it may have to sanction the litterer. The purpose of the related event recommender is to recommend event episodes that occur 100% of the time preceding a sanctioning action so that the agent can be warned about impending sanctions.

5 Experiments on norm identification

In an agent society, one or more norms can co-exist. In this section we demonstrate that the agents using our architecture are able to infer the norms of the society.

5.1 Scenario 1 : A society with one type of norm

We have experimented with an agent society comprising 100 agents. There are agents with three different personality types. They are learning litterers (*ll*), non-litterers (*nl*) and non-littering punishers (*nlp*). The learning litterers are litterers who learn to change their behaviour based on normative expectations inferred through the observation of interactions between agents. Non-litterers do not litter the park and non-littering punishers are the non-litterers who sanction littering because that action is against their personal norm.

There are 50 *ll* and 50 *nl* agents. Out of these 50 *nl* agents, 5 are *nlp* agents. In each iteration, an agent performs one of m,e,l or s. The agents are initialized with a uniform probability for choosing actions (p(m)=0.75, p(e)=0.25, p(l) having eaten in the previous interaction =0.5). The *nlp* agents punish other agents if they observe a littering action of an agent in the current iteration or the previous iteration with 6% probability (in both the cases). An agent stores the actions performed by other agents in its vicinity (in the current set up, a fully-connected network topology is assumed where an agent can see all other agents). We ran this experiment for 100 iterations and observed what kinds of suspected and candidate norms emerged in the mind of an agent.

It should be noted that for an episode that is made up of 3 different events, allowing permutation with repetition, 39 sub-episodes can be created (3 in level 1, 9 in level 2 and 27 in level 3). It can be observed from figures 5 and 6 that out of 39 possible sub-episodes, only a subset of sub-episodes (13 of them) happen to appear (i.e. the suspected norms). Assuming that an agent's norm selection threshold is 0.45 to construct the list of candidate norms, there are two such norms, which are norms against *el* and *l*. The agent then moves on to the norm verification stage which identifies the norm against littering.

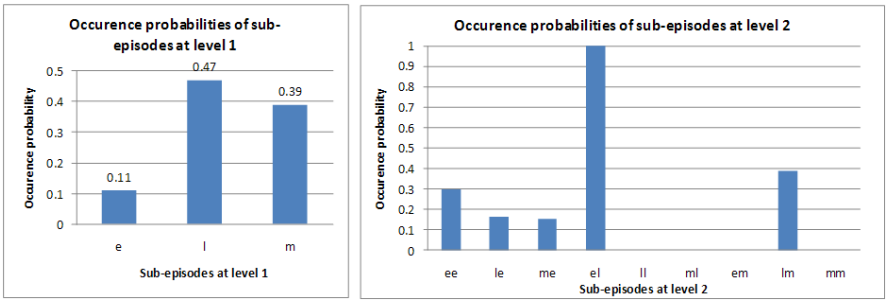


Fig. 5: Sub-episode occurrence probabilities (level 1 and 2)

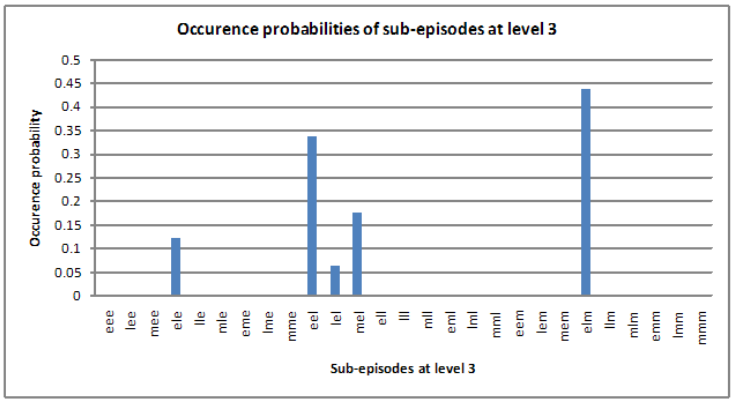


Fig. 6: Sub-episode occurrence probabilities (level 3)

5.2 Scenario 2 : Identification of co-existing norms in an agent society

Let us assume that there are two types of sanctioning agents, one that sanctions when an agent litters the park and the other sanctions if it sees anyone eating in the park. In these cases, our mechanism will be able to generate different sets of suspected norms. Retaining the experimental set-up used in the previous scenario, we have set the probability of a *nlp* punishing eating action to be 3% and the probability of punishing littering action to be 3%). The norm selection threshold has been set to 0.25. Occurrence probabilities of sub-episodes (i.e suspected norms) at level 2 is given in figure 7. It can be observed that there are more occurrences of events involving e that appear in these sub-episodes than event l. This is due to the set up of the system since p(e)=0.25 while p(l)=0.125. The important thing to note in this experiment is that our architecture allows for the identification of co-existing norms.

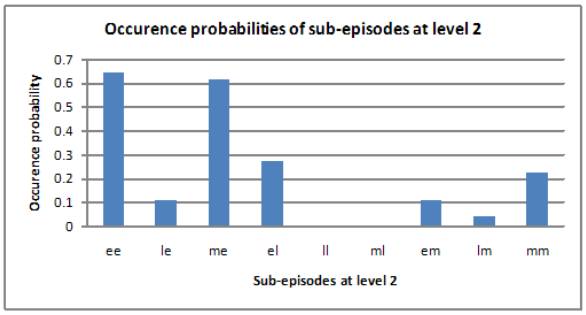


Fig. 7: Sub-episode occurrence probabilities (level 2) when two types of sanctioning agents were present

5.3 Scenario 3 : Identification of norms across different societies

Let us assume that there are three sections of a park. At any point of time, an agent might be present in one of these sections. Let us also assume that there are two types of sanctioning agents. One type of agents punish litterers while the other type of agents punish those who eat in the park. Assume that these types of agents are randomly placed in the three sections of a park. Our objective was to see what type of norms might emerge in these three sub-groups.

Figure 8 shows the candidate norms of three different agents that belong to three different sub-groups. The norm selection threshold was set to 0.3. It can be observed that different types of candidate norms are generated in the minds of these agents based on what they had observed in their respective agent society. Figure 9 shows what were the identified norms of each of these agents. It can be observed that the agent from sub-group one had identified the norm against littering and the one from the third group had identified the norm against eating while the agent from the second group had identified both these norms.

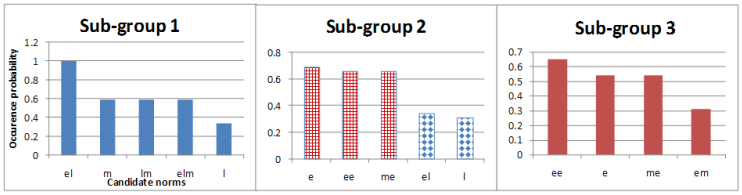


Fig. 8: Candidate norms of three agents that belong to three different sub-groups

An extension to this experiment is to allow an agent to move around in these three sections of the park and see how it accommodates the changes to its norms. Another extension will be to allow the sanctioning agents to move around which will enable dynamic change of norms in the society.

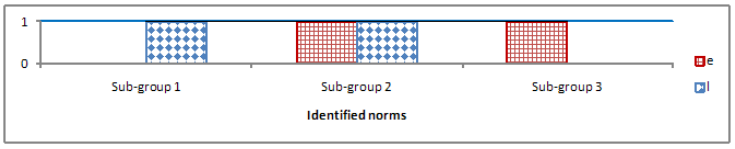


Fig. 9: Identified norms in three sub-groups

6 Discussion

We note that the experimental set up is simple. We have assumed that an agent considers three events (n=3) that precede a signal (a sanction or a reward). The value of n can change and an agent being a computation machine should be able to handle a large number of possible events. Most researchers agree that there will be some form of sanction or reward once a norm is established (e.g. [3, 4]). Hence, the notion of a reaction (positive or negative action) has been considered to be a top level entity in our work. We have assumed that even when a norm is being created, the notion of sanction is important for norm identification.

Our experimental set-up can be improved in many ways. Firstly, we do not assume that there is a cost associated with sanctions. The cost for sanctions can be included in the model. Secondly, our model identifies co-existing norms. If the cost of sanctions is considered there could also be competing or conflicting norms. For example, some agents might punish other agents when they litter while some others may punish one when the littering agent is within 20 meters from the rubbish bin. When there are competing norms the society might be divided into groups. This type of dynamics will be interesting to study. Thirdly, the experiments have only made use of observational information ignoring the personal experience. We believe that the inclusion of personal experience will speed up the rate at which norms are identified. Fourthly, we have assumed that when an agent identifies a norm, it will follow the norm. Agents owing to their autonomy do not always follow the norm. An agent might have its own personal agenda and it can be an opportunistic norm follower. Autonomy of an agent needs to be addressed in the future. In our architecture, this has been encapsulated as a part of the norm assessment component which will be elaborated in a future work. Fifthly, it might not always be possible to associate sanctions or rewards with the events that immediately precede them. For example, speeding might result in a fine that is sent to an agent after a couple of days. An observer might not be able to recognize this sanction. In this work, we have only considered those norms where the sanctions can be recognized by an observer and the events that caused the sanction occurred within an immediate window of time before the sanction. Lastly, the problem of false negatives and positives for norm identification needs to be dealt with in the future.

However, we believe that our work reports some advancements. Firstly, the question of "how an agent comes to find out what the norm of society is" is being dealt with by at least one other research group [14]. We have made some progress in that regard by proposing an internal agent architecture and demonstrating

how an agent will identify the norms of a society. Secondly, other prominent works identify one norm that exists in the society [6, 10, 17]. In our architecture an agent is able to identify several norms that might exist in the society. Thirdly, most works have not addressed how an agent might be able to identify whether a norm is changing in a society and how it might react to this situation. In our model, the agents will be able to identify the norm change and dynamically add, remove and modify norms. Fourthly, our architecture can be used to study norm emergence. We believe through norm identification at the agent level, we are also in the realm of addressing how norms emerge using a bottom-up approach.

7 Conclusions

In this paper we have explained the internal agent architecture for norm identification. Through simulations we have shown how an agent infers norms in an agent society. We have also discussed the related work and have identified issues that should be addressed in the future.

8 Acknowledgments

We thank the three anonymous reviewers for their insightful comments.

References

1. Rymaszewski, M., Au, W.J., Wallace, M., Winters, C., Ondrejka, C., Batstone-Cunningham, B., Rosedale, P.: Second Life: The Official Guide. SYBEX Inc., Alameda, CA, USA (2006)
2. Ullmann-Margalit, E.: The Emergence of Norms. Clarendon Press (1977)
3. Coleman, J.: Foundations of Social Theory. Belknap Press (August 1990)
4. Elster, J.: Social norms and economic theory. The Journal of Economic Perspectives **3**(4) (1989) 99–117
5. Verhagen, H.: Simulation of the Learning of Norms. Social Science Computer Review **19**(3) (2001) 296–306
6. Hoffmann, M.: Entrepreneurs and Norm Dynamics: An Agent-Based Model of the Norm Life Cycle. Technical report, Department of Political Science and International Relations, University of Delaware, USA (2003)
7. Shoham, Y., Tennenholtz, M.: Emergent conventions in multi-agent systems: Initial experimental results and observations (preliminary report). In: KR. (1992) 225–231
8. Walker, A., Wooldridge, M.: Understanding the emergence of conventions in multi-agent systems. In Lesser, V., ed.: Proceedings of the First International Conference on Multi–Agent Systems, San Francisco, CA, MIT Press (1995) 384–389
9. Sen, S., Airiau, S.: Emergence of norms through social learning. In: Proceedings of Twentieth International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, MIT Press (2006) 1507–1512
10. Axelrod, R.: An evolutionary approach to norms. The American Political Science Review **80**(4) (1986) 1095–1111

11. Castelfranchi, C., Conte, R., Paolucci, M.: Normative reputation and the costs of compliance. Journal of Artificial Societies and Social Simulation **vol. 1, no. 3** (1998)
12. Hales, D.: Group reputation supports beneficent norms. Journal of Artificial Societies and Social Simulation **5** (2002)
13. Neumann, M.: A classification of normative architectures. In: Proceedings of World Congress on Social Simulation. (2008)
14. Andrighetto, G., Conte, R., Turrini, P., Paolucci, M.: Emergence in the loop: Simulating the two way dynamics of norm innovation. In Boella, G., van der Torre, L., Verhagen, H., eds.: Normative Multi-agent Systems. Number 07122 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
15. Andrighetto, G., Campenni, M., Cecconi, F., Conte, R.: How agents find out norms: A simulation based model of norm innovation. In: 3rd International Workshop on Normative Multiagent Systems (NorMAS 2008), 15-16 July, 2008, Luxembourg. (2008)
16. Campenni, M., Andrighetto, G., Cecconi, F., Conte, R.: Normal = normative? the role of intelligent agents in norm innovation. In: The Fifth Conference of the European Social Simulation Association (ESSA), University of Brescia, September 1-5, 2008. (2008)
17. Epstein, J.M.: Learning to be thoughtless: Social norms and individual computation. Comput. Econ. **18**(1) (2001) 9–24

Playing with agent coordination patterns in MAGE

Visara Urovi and Kostas Stathis

Department of Computer Science,
Royal Holloway, University of London, UK
{visara,kostas}@cs.rhul.ac.uk

Abstract. MAGE (Multi-Agent Game Environment) is a logic-based framework that uses games as a metaphor for representing complex agent activities within an artificial society. More specifically, MAGE seeks to (a) reuse existing computational techniques for norm-based interactions and (b) complement these techniques with a coordination component to support complex interactions. The reuse part of MAGE relates physical actions that happen in an agent environment to count as valid moves of a game representing the social environment of an application. The coordination part of MAGE supports the construction of composite games built from component sub-games and corresponds to coordination patterns that support complex activities built from sub-activities. To illustrate the MAGE approach, we discuss how to use the framework to specify the coordination patterns required to form a virtual organisation in the context of a service-oriented scenario.

1 Introduction

Early work in multi-agent system has focused on the representation of agent interaction construed in terms of communication protocols that agents can use to interact with each other. As these protocols standartise the way in which agents partake in social activities, more recent work has put the emphasis on normative concepts such as obligation, permission, and prohibition, amongst other, to specify the social rules that represent agent protocols (see [3, 15]). However, despite the plethora of frameworks that support agent interactions about social concepts, there is relatively less work on how to represent systematically more complex activities that require agents to coordinate their actions when playing many protocols at the same time. There is, in other words, the need for computational frameworks that compose complex interactions and allow for their coordination.

Our specific motivation results from our participation in ARGUGRID [1], a research project that aims at providing a new model for programming a service Grid at a semantic, knowledge-based level of abstraction through the use of argumentative agent technology. Agents act on behalf of (a) users who specify abstract service requests or (b) providers who offer electronic services on the Grid. User requests result in agents interacting with other agents by forming dynamic Virtual Organisations (VOs) in

order to enable the transformation of abstract user requests to concrete services that the Grid can support. To guarantee that interactions in VOs are of a certain standard, agent-oriented provision of services must conform to service level agreements, while agent interaction more generally must be governed by electronic contracts. One of the requirements of ARGUGRID is that agreements and contracts need to be negotiated on the fly by agents, so there is the need to support protocols and workflows that enable the activities of VO creation, operation, and dissolution. One of the issues then becomes how to represent these complex activities at a knowledge-based level, suitable for argumentation-based agents to use as a framework to coordinate their interactions.

To manage agent coordination for VOs we present a logic-based framework that we call MAGE (Multi-Agent Game Environment). The idea behind MAGE is that the rules of a communication protocol between agents are viewed as the rules of an atomic game played amongst players, the speech acts uttered by agents represent the legal moves in the game, and the roles of agents in the interaction represent the roles of the players in the game. The contribution of MAGE is that given the representation of atomic games it provides a computational framework in which atomic games can be composed into composite ones and provides a systematic framework for their coordination. To illustrate how the resulting framework can be applied to a practical application, we show how to apply it in an ARGUGRID scenario that specifies workflows in terms of agent protocols to support the creation of a VO and its relevant electronic contracts.

The rest of the paper is organized as follows. Section 2 presents the context of the problem that we try to formulate and relates it to two kinds of games: atomic and compound. Atomic games and their specification are discussed in Section 3, while compound games and their specification are discussed in Section 4. Section 5 places our research in the context of existing literature and compares it to related work. We conclude with Section 6 where we also discuss our plans for future work.

2 ARGUGRID Games

We present a scenario that has motivated our work together with negotiation protocol used to negotiate services. We also discuss the link between the envisaged agent interactions and their representation as games. Once we have established this relation, we use it as the base of the MAGE computational framework developed in the next section.

2.1 The Earth Observation Scenario

The ARGUGRID scenario considers a government ministry official requiring data about the detection of an offshore *oil spill* [18]. This abstract and high-level goal cannot be immediately satisfied by data within the ministry itself and requires the help of satellite companies that observe parts of the earth at different days. These companies publicise their services on a service Grid that is managed by agents. In this scenario a

software agent takes the abstract request of the official and tries to instantiate it in a detailed set of services that can be invoked in sequence to provide the requested information. The scenario further assumes that satellite companies provide different services, each with different capabilities and costs, and one satellite may be more appropriate than another given certain conditions that the ministry sets. The official's software agent based on a set of preferences over the service requested, selects the suitable satelite companies and engages in a contract negotiation process with provider agents to create a VO that will instantiate the lower level services required to meet the official's request.

2.2 The Minimal Concession Protocol

Negotiation of contract terms in ARGUGRID uses a minimal concession protocol, with or without rewards, described in Dung et al [7], see Fig.1.

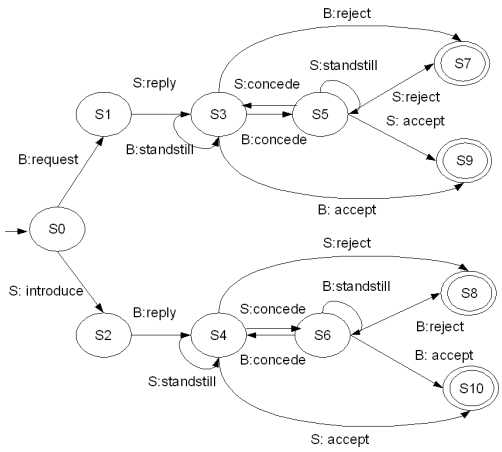


Fig. 1. The *Minimal Concession* protocol with Rewards [7]

The protocol provides the following set of locutions available to agents: *request, introduce, reply, concede, standstill, accept, reject.* The protocol assumes two agent roles, a *buyer* (B) and a *seller* (S). The protocol can start with an *introduce* move made by the seller or with a *request* move made by the buyer. These moves are used to respectively request or introduce an offer e.g. an oil spill detection service with some properties. Afterwards, a *reply* move can be made from the buyer to reply to an *introduce* move, or from a seller to reply to a *request* move. After this move, *standstill, reject* or *concede* an offer are all moves that can be made by any role. The *accept* move terminates successfully the protocol

and the accepted offer is considered the value of the result of the game. Three consecutive *standstill* moves are considered as a *reject* move, which terminates the protocol with no agreement.
An important property of this protocol is that if two agents use the protocol in conjunction with a minimal concession strategy, then every negotiation terminates successfully and the minimal concession strategy is in symmetric Nash equilibrium [7]. A minimal concession strategy is used if the offered service/product does not match what is requested. An agent can concede on a property of the service/product when it is possible to do so. Afterwards the agent will expect the other agent to concede as well allowing the offer to get closer to match the request and vice versa. If the agent decides not to concede, it can standstill. The other agent will reply to a standstill with a concede locution if standstill is not a consecutive locution, otherwise it will standstill as well.

2.3 The VO Life-Cycle in ARGUGRID

The minimal concession protocol is only a component of the more complex activities that in ARGUGRID allow agents to form and participate in VOs, as shown in Fig. 2.

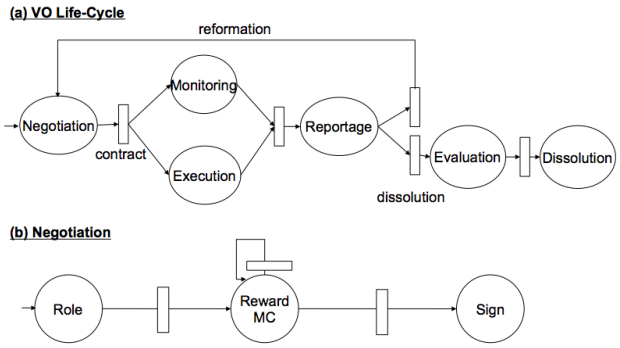


Fig. 2. Negotiation in the *VO Life-cycle* of ARGUGRID

Fig. 2(a) shows how by negotiating a successful contract starts the execution and monitoring activities of a VO. Issues raised by the monitoring or execution activities are reported and the VO must in this case be reformed via re-negotiation. If, however, reformation does not apply to these issues or if the execution has fulfilled the goals of the VO creation, the VO is dissolved by having its result being evaluated first. Activities in VOs may require further control-flows for sub-activities as shown in Fig. 2(b); this illustrates how the activity of negotiation is in fact a more complex activity that requires first to determine the roles of the agents in

the VO, then negotiate the terms of the VO contract using the minimal concession protocol, and finally the complete contract must be signed by all relevant parties. The details for the remaining activities of monitoring, execution, reportage, evaluation, and dissolution, are beyond the scope of this work. In the remainder of this paper we focus on how to model the control flows of activities as a complex game exemplified by the negotiation activity.

2.4 VO Activities as Complex Games

The games metaphor was originally proposed to model human-computer interaction by Stathis and Sergot in [17] and was subsequently applied to formulate agent interaction protocols in [16]. We extend this model to support agent coordination patterns in ARGUGRID as games.

The basic unit of the games metaphor is the notion of an atomic game, which describes a set of rules about an initial state, a set of player roles, a set of game moves, the effects the moves have on the state, a specification of when a move is legal, a set of terminating states, and a set of results [16]. The minimal concession protocol described earlier seen as a game implies that the initial state of the protocol is the initial state of the game, the roles of the participating agents are the roles of the players, the protocol locutions are the game moves, the effects of locutions on the protocol state are the effects of moves on the state of the game, the preconditions of locutions are the valid moves, the final protocol states are the terminating states of the games, the set of protocol outcomes are the possible game results. The result of a game does not necessarily need to be zero-sum [12], by requiring a winner and a loser, but it can also give rise to a win/win or loose/loose situations.

To obtain complex interactions we combine atomic games to build more complex, composite games. An example of a complex interaction is the control flow of the Fig. 2(b), where we need to combine three different games: first the agents can play a *role negotiation* game to determine their roles, after establishing their roles, they play a *minimal concession with reward* game to agree on the terms of the contract, they can reiterate this game for as long they find an agreement and finally the *sign* game becomes active for the agents to sign the contract.

In a composite game we want to be able to parallelise, choose or synchronize atomic games. To capture these control-flow aspects of complex games we produce a coordination framework that allows us to coordinate complex interactions build from simpler ones. The resulting framework is then applied to support workflow activities. In general, the term *workflow* refers to the specification of a work procedure or a business process in a set of *atomic activities* and relations between them in order to coordinate the *participants* and the activities they need to perform [2]. The link here is that the *participants* are the agents and the *atomic activities* are the atomic games. By relating atomic games as atomic activities we then use basic coordination patterns to enable agents to play more complex activities as complex games. We will see later how our example of the negotiation composite game (illustrated in Fig. 2(b)) will be defined

as an aggregation of three patterns: a sequence, a conditional, and an iteration pattern.

3 Atomic Games in MAGE

Following earlier work on the games metaphor [17], we view communicative interactions within an agent society abstractly as game interactions [16]. As the rules of a game represent all valid evolutions of the game's state, we use the following logic program to describe the rules of a game:

```
game(State, Result)←
      terminating(State, Result).
game(State, Result)←
      not terminating(State, Result),
      valid(State, Move),
      effects(State, Move, NewState),
      game(NewState, Result).
```

To formulate a particular game we need to decide how to represent a game state, its initiating and terminating states, how players make valid moves, and how the effects of these moves change the current state to the next one until the terminating state is reached.

3.1 The State of Atomic Games

To represent the State of a game we use a term of the form Id@T, where Id is a unique identifier of a complex term describing the attributes of the state's configuration, and T is the system's time that uniquely identifies the actual evolutions of the complex term as a result of the interaction. The rationale behind this kind of representation is that in MAGE we acknowledge the fact that the interaction within a multi-agent system application can become quite complex. To cater for the complexities of practical applications we assume that complex terms have an underlying object-based data-model. To represent complex terms we use the syntax of C-Logic [5]. A term of the form:

```
min_concession:mc1 [
    parties⇒ {agent:a1 [role ⇒ seller], agent:a2 [role⇒buyer]},
    buyer_position ⇒ offer:o1 [price ⇒80, resolution⇒20, delivery ⇒2],
    seller_position ⇒ offer:o2 [price ⇒100, resolution⇒20, delivery ⇒2],
    standstill_count ⇒ 1,
    result ⇒ nil
]
```

is identified by mc1 denoting an instance of an object whose class is the minimal concession protocol with two participating agents a1 and

a2, complex terms whose role attribute is seller and buyer respectively, where the buyer in the previous round has made an offer o1 (a complex term), while the seller has made another offer o2 (another complex term), there is one standstill move that has been encountered, and the result of the interaction is still incomplete as the value is still nil. Such a complex term has a first-order logic translation, see [5] for details.

3.2 State Evolution

The moves of the game are represented by complex terms too. The complex term below

speech_act:m1[actor ⇒ a1, act⇒introduce, offer ⇒ o1, role⇒ seller],

describes that the seller agent a1 utters introduce about an offer o1. Such moves are used as the contents of events that happen at a specific time. An assertion of the form happens(m1, 12), states that move m1 has happened at time 12. Such an event changes the state of a game.

```
holds_at(Id, Class, Attr, Val, T)←
  happens(E, Ti), Ti ≤ T,
  initiates(E, Id, Class, Attr, Val),
  not broken(Id, Class, Attr, Val, Ti, T).

broken(Id, Class, Attr, Val, Ti, Tn)←
  happens(E, Tj), Ti < Tj ≤Tn,
  terminates(E, Id, Class, Attr, Val).

holds_at(Id, Class, Attr, Val, T)←
  method(Class, Id, Attr, Val, Body),
  solve_at(Body, T).

attribute_of(Class, X, Type)←
  attribute(Class, X, Type).
attribute_of(Sub, X, Type)←
  is_a(Sub, Class),
  attribute_of(Class, X, Type).

instance_of(Id, Class, T)←
  happens(E, Ti), Ti ≤ T,
  assigns(E, Id, Class),
  not removed(Id, Class, Ti, T).
removed(Id, Class, Ti, Tn)←
  happens(E, Tj), Ti < Tj ≤ Tn,
  destroys(E, Id).

assigns(E, Id, Class)←
  is_a(Sub, Class),
  assigns(E, Id, Sub).

terminates(E, Id, Class, Attr, _)←
  attribute_of(Class, Attr, single),
  initiates(E, Id, Class, Attr, _).

terminates(E, Id, _, Attr, _)←
  destroys(E, Id).
terminates(E, Id, _, Attr, IdVal)←
  destroys(E, IdVal).
```

Fig. 3. A subset of the *Object-based Event Calculus* from [9]

We use the object-based event calculus (OEC) of Kesim and Sergot [9] to capture state changes of complex terms. A subset of the OEC is given in Fig. 3. The first two clauses derive the value of an attribute for a complex term holds at a specific time. The third clause describes how to

represent derived attributes of object as method calls computed by means of a solve_at/2 meta-interpreter as specified in [10]. The fourth and fifth clauses support a monotonic inheritance of attributes for a class limited to the subset relation. The sixth and seventh clauses determine how to derive the instance of a class at a specific time. The effects of an event on a class is given by assignment assertions; the eighth clause states how any new instance of a class becomes a new instance of the super-classes. Finally, the ninth clause deletes single valued attributes that have been updated, while the tenth and eleventh clauses delete objects and dangling references.

3.3 Valid Moves and their Effects

Before the event of a move being made in the state of the game, we must have a way to check that the move is valid. One simple definition is to make valid moves equivalent to the legal moves of the game:

```
valid(State, Move) ↔ legal(State, Move).
```

To specify valid moves, we specify when moves are legal. For example, to specify when a request move is legal in the minimal concession protocol we write:

```
legal_at(Id@T, Move) ←
   instance_of(Id, min_concession, T),
   speech_act:Move[actor ⇒ A, act⇒request, offer ⇒ Product, role⇒ buyer],
   holds_at(S, agent_of, A, T),
   holds_at(A, role, buyer, T).
```

Other definitions of valid moves are possible, for instance, Artikis et al [3] provide a more detailed account of valid moves in terms of social concepts such as obligations, permission and power. The important point here is that our framework can accommodate these for an application by providing a different definition of valid/2.
Once a move has been determined as valid, a new state of the game must be brought about due to the effects of the move. As by making moves players cause events to happen, if we assume that the happening of such moves take only one unit of time, we can specify their effects as:

```
effects(Id@T, Move, Id@NewT) ←
      add(happens(Move T)),
      NewT is T + 1.
```

In our representation of state, once an event has happened, its effects are added to the state implicitly, via inititiates/4 definitions that initiate

new values for attributes of a state term, terminates/4 clauses that remove attribute values from a state term, and assigns/3 definitions for assigning to ids new instances of terms. An example, of how new values are initiated for attributes for the minimal concession protocol is given below:

```
initiates(Ev, Id, seller_position, Offer)←
    happens(Ev, T),
    instance_of(Id, min_concession, T),
    Ev[act ⇒ Act, actor ⇒ Aid, role ⇒ seller, offer ⇒ Offer],
    changes_seller_position(Act).

changes_seller_position(introduce).
changes_seller_position(concede).
changes_seller_position(reply).
```

The above definition initiates the current position made by a seller to be stored in the state of the game as a result of a request, reply or concede move. The old offer is terminated and substituted by a new request because of the way the object event calculus is specified (see the ninth clause in Fig. 3).
It is important to note that other specifications of effects/3 are possible depending on what assumptions we make about the duration of moves captured in events. In addition, the state could be represented explicitly as a set of assertions as in [16] rather that implicitly, with rules that define what holds in it, as in MAGE. Both of these issues, however, are beyond the scope of this paper. It suffices to say here that once a choice of state representation has been made, the framework can accommodate it by suitably adjusting the effects/3 definition.

3.4 Initial and final states of a game

For the state of an atomic game to be created, the framework discussed so far requires the assertion of an event that will first create the term via an assigns/3 assertion. The assertion:

```
assigns(Ev, Id, min_concession)←
    Ev[act ⇒ construct, protocol ⇒ min_concession, id ⇒ Id].
```

will allow the creation of an instance for the minimal concession protocol, which can then be queried using the sixth clause of Fig. 3. To complete the instantiation process we also need to specify the initial values for the attributes of the complex term representing the minimal concession protocol. For this we need to define separately the initiates/4 rules as the one below:

```
initiates(Ev, Id, party_of, Val)←
      Ev[act ⇒ construct, protocol ⇒ min_concession, parties ⇒ agent: Val].
```

Additional initiates/4 clauses are needed to define the whole of the initial state, one for each attribute value.
The initial state of the game will evolve as a result of moves been made in the state of a game. This state will eventually reach the final state from which we can extract the game's result. We specify this via terminating/2 predicates. For example, the definition:

```
terminating(Id@T, Result)←
      instance_of(Id, min_concession, T),
      holds_at(Id, result, Result, T),
      not Result==nil.
```

specifies the conditions under which the minimal concession protocol terminates and at the same time returns the result.

4 Compound Games in MAGE

Compound games are complex games composed from simpler, possibly atomic, sub-games. Based on our previous work in applying compound games to develop multi-agent systems [16], in this section we show how to develop compound games in the MAGE framework, with aim to support the coordination of complex agent activities such as ARGUGRID workflows.

4.1 A Compound Game

To give an example of how sub-games will appear in the main game, consider as an example the state of the VO negotiation in ARGUGRID, as specified in Fig. 2.

```
vo_negotiation: Id [
 parties ⇒ {agent:a1, agent:a2, agent:a3},
 process ⇒ Workflow
]
```

The sub-games of VO negotiation are specified in the Workflow value of the process attribute, instantiated to terms of the form:

```
seq([
      roles:r1,
      if(r1[result⇒success], repeat(mcwr:r1, m1[result⇒exit])),
      if(m1[agreement⇒achieved], sign:s1)
])
```

The above term states that the process of the negotiation is a sequence (seq) of sub-games involving first a sub-game of roles game with identifier r1. This game must be played, and if the result of the roles game is success, it means that the roles of the agents in the VO have been agreed, and the workflow must continue with repeatedly creating a minimal concession protocol mcwr with identifier m1 and playing it until the result of this game is exit (meaning that either an agreement has been achieved during the negotiation or the game has been played more than a certain maximum and no agreement was achieved). Only if the agreement attribute of m1 is set to achieved, the sign game with identifier s1 is started and played to complete the negotiation process.

4.2 Coordination of active sub-games

The main issue to be considered in compound games is the coordination of moves in active sub-games. We define coordination specifying the predicate active_at/3. Using active sub-games, we can define valid moves in a complex game to include all the valid moves in the active sub-games:

```
valid(Id@T, Move) ←active_at(Id, SubId, T), valid(SubId@T, Move).
```

For VO negotiation we define active subgames as follows:

```
active_at(Id, SubId, T)←
      instance_of(Id, neg, T),
      Id [process⇒Workflow],
      pattern(Workflow),
      runs(Id, Workflow, SubId, T).
```

Patterns in our framework are interpreted by a runs/4 predicate that parses the coordination structure and checks which sub-games are running. For the VO negotiation process three patterns are required: a sequence, an if-conditional, and a repeat loop, as specified below.

```
runs(G, seq([A|_]), A, T)←
    not pattern(A),
    not terminating(A@T,_).
runs(G, seq([A|B]), C, T)←
    not pattern(A),
    terminating(A@T,_),
    runs(G, seq(B), C, T).

runs(G, seq([A|B]), C, T)←
    pattern(A),
    (runs(G, A, C, T);
    runs(G, seq(B), C, T)).
```

```
runs(G, if(Id[Prop⇒Val], P), C, T)←
    holds_at(Id, Prop, Val, T),
    (pattern(P) →
    runs(G, P, C, T); C=P).
runs(G, repeat(P, Id[Prop⇒Val]), A, T)←
    not holds_at(Id, Prop, Val, T),
    runs(G, P, A, T).

pattern(P)← sequence(P).
pattern(P)← if_conditional(P).
pattern(P)← repeat_loop(P).

sequence(seq(_)).
if_conditional(if(_,_)).
repeat_loop(repeat(_,_)).
```

Note that the top-level game G is required as a parameter in the definition of runs/4 as a reference to the global variables of the interaction. Note also that the definition of the above patterns can be combined to form arbitrary complex structures, which is indicative of the expressive power of the framework.
More workflow primitives [19] can be specified in a similar manner. We show next an and_split pattern to illustrate how to support parallel composition. This pattern is specified as

and_split(A, Condition, Activities)

and states that after activity A is completed, if the Condition is true, then the set of Activities must be carried out in parallel. To support the parallel composition required for this coordination pattern, we define runs/3 as follows:

```
runs(G, and_split(A,_,_), A, T) ←
    not pattern(A),
    not terminating(A@T, _).
runs(G, and_split(A, Id[Prop ⇒Val], Activities), C, T)←
    terminating(A@T,_),
    holds_at(Id,Prop,Val,T),
    member(Activity,Activities),
    not terminating(Activity@T,_),
    (pattern(Activity) → runs(G,Activity,C,T); C=Activity).
```

We have formulated similarly the patterns for and_join, xor_split, and xor_join, but we cannot discuss them here due to lack of space. We plan to present these in future work.

4.3 Status of the work

Implementation We have built a prototype of MAGE that allows the deployment of a set of distributed objects in the GOLEM platform [4]. We call these objects *Game Calculators*. They are used by GOLEM agents to interact with each other and to coordinate their interactions, see Fig. 4. More specifically, GOLEM agents can call methods of a calculator object by means of actions performed in the environment. The content of such actions represents a move in the compound game. We believe this to be advantageous in two ways: (a) space and time decoupling, i.e. because game calculators are a mediation service, agents do not need to be in the same place at the same time in order to interact; and (b) we do not have to treat everything as an agent to develop an application.
To implement games, we link the internal part of the Game Calculator object with a TuCSoN tuple centre [13], a Linda-like extension of the concept of tuple space as a reactive logic based blackboard. The reason why we chose TuCSoN to implement Game Calculators is that it allows

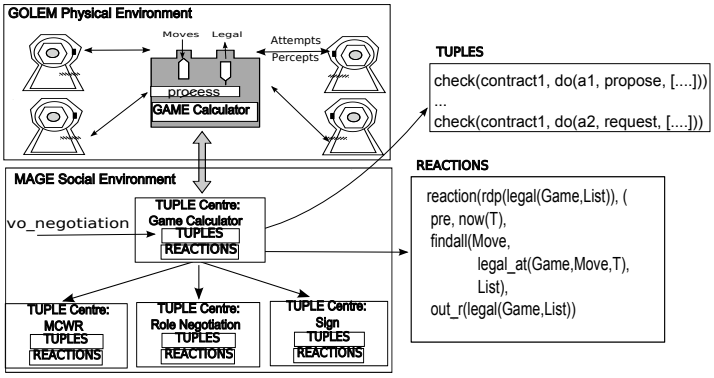


Fig. 4. Implementing MAGE using TuCSoN and GOLEM

us to use a main tuple centre and distribute the state of a compound game in other tuple centres, each tuple centre could in principle map to atomic or compound sub-games. To support this we use a combination of the ReSPeCT language [13] and the OEC discussed here. A Game Calculator can be configured from an agent (either a coordinator agent or the agent who is interested to start the negotiation) to work as a specific compound game (such as VO negotiation). Further details of the implementation are beyond the scope of this work; we plan to present the implementation separately in future work.

Evaluation MAGE is a mediation framework acting as a social environment that supports interactions between heterogeneous self-interested agents. We have developed MAGE so that it can work as a component-based social infrastructure for the GOLEM agent environment [4] to support practical applications. To do this we have tried to be flexible with the way norms are incorporated in the system using the notion of valid moves and we have focused on coordination. One of our contributions is that we have extended the games metaphor, presented in previous work, with the treatment of coordination patterns that this framework did not support before. From our experimentation with the minimal concession protocol in ARGUGRID VOs, a feature that we have found interesting is that we can specify the interaction with workflows at run-time, by keeping the same Game Calculator but changing the protocol and the workflow activities in a plug-and-play style. Moreover, using object-based indexing of events already available in the Event Calculus, we have experimented with interactions that give rise to approximately 1,000 events within a protocol, with acceptable performance. Again, we plan to discuss these details separately, as future work.

5 Related Work

The Electronic Institution (EI) approach and the AMELI framework [8] uses organisational concepts to model the interaction of agents. Our framework is similar to EIs in the sense that their *scenes* are our atomic games and their *norms* as the rules that capture the valid moves for the agent as the game progresses. EIs also support a *performative structure* that enables a developer to define dependencies such as choice points, synchronization and parallelism mechanisms between scenes based on role flow policies among scenes specifying which paths can be followed by which agent's role. In our framework the EI performative structures are defined as compound games that structure atomic games, which can be coordinated by activity patterns. One of the differences of MAGE with AMELI is that we expect agents to interact via Game Calculators and we do not use mediating agents such as AMELI governors. An explicit feature of our approach is that the state of the interaction in a Game Calculator is easily inspectable, while in EIs agent playing specific roles need to communicate to build a coherent state. In addition we naturally support complex games that consist of complex sub-games, while EIs would require hierarchical performative structures and thus increase the complexity of the overall EI approach.

Artikis et al [3] propose a model for norm-governed multi-agent systems as executable specification of open agent societies. This work represents social constraints by making a clear distinction between physical capabilities, institutional power, permissions and sanctions to enforce policies. Social constrains are a sophisticated version for defining our valid moves of a game that captures the social state of the interaction. We too distinguish between possible actions happening in the environment supported by GOLEM, from social actions happening in MAGE, and we link them via physical objects that support agent coordination. As our focus is on coordination and as their emphasis is on normative concepts, the two approaches can be seen as complementary to each other, especially as they both use the Event Calculus as the underlying computational mechanism, even if we assume an object-based data-model. However, in our model we do not prove properties of interactions, which can be an extension of our work.

McBurney and Parson [11] present an abstract framework to represent complex dialogues as sequences of moves in a combination of dialogue games. Agents agree the game they need to play at a control layer, in our terms a compound game, and then play the protocol at an execution layer, in a our case a sub-game. The framework admits combinations of different dialogue types that in our framework corresponds to the coordination of compound games. However, McBurney and Paron's dialogical games abstract away from the game state and they do not define the valid moves as a way of analysing the different kinds of pre-conditions and post-conditions on the state of the interactions. Instead their formalism is based on agents selecting and agreeing to play these dialogues. On the contrary our framework seeks to provide a computational mechanism for coordination in complex interactions that are construed as compound games.

Kesim et al [6] propose a framework to specify and execute workflows based on Event Calculus. In this work, EC is used to describe the specification and execution of activities in a workflow. The activities are assigned to agents using a coordinator agent that knows which agents can perform which activities. Like Kesim's work we use the EC to define workflows but we use games to dynamically define compositions of workflows. Similarly, Omicini et al [14] propose a model to distribute a workflow among different tuple centres (conceived as the entities that coordinate agent's activities) by linking tuple centres with linkability operators. In our approach we use the linkability of tuple centres as the coordination mechanism that the Game Calculator uses to start and terminate new games. We also provide a representational framework that can be used systematically to represent patterns of interactions, like workflows.

6 Conclusions and Future Works

We have presented MAGE, a logic-based framework that uses games as a metaphor for representing complex agent activities within an artificial society. We have illustrated how MAGE can reuse existing computational techniques for norm-based interactions and support their coordination. Using examples from the ARGUGRID projects, we have illustrated how the reuse part of MAGE relates physical actions that happen in an agent environment to count as valid moves of a game representing the social environment of an application. Coordination in MAGE supports the construction of complex games built from component sub-games and corresponds to coordination patterns that support complex activities built from sub-activities. We have discussed how to use the framework to specify the coordination patterns required to form a virtual organisation in ARGUGRID.
Future work involves formulating the VO lifecycle of ARGUGRID in MAGE to build a library of reusable coordination patterns for similar applications.

References

1. ARGUmentantion as a foundation for the semantic GRID (ARGUGRID). http://www.argugrid.eu/, 2009.
2. Workflow Management Coalition. http://www.wfmc.org/, 2009.
3. Alexander Artikis, Marek J. Sergot, and Jeremy V. Pitt. Specifying Norm-Governed Computational Societies. *ACM Trans. Comput. Log.*, 10(1), 2009.
4. Stefano Bromuri and Kostas Stathis. Situating Cognitive Agents in GOLEM. In *Engineering Environment-Mediated Multi-Agent Systems, EEMMAS 2007*, volume 5049/2008 of *Lecture Notes in Computer Science*, pages 115–134. Springer, 2007.
5. W. Chen and D. S. Warren. C-logic of Complex Objects. In *PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 369–378, New York, NY, USA, 1989. ACM Press.

6. Nihan Kesim Cicekli and Yakup Yildirim. Formalizing Workflows Using the Event Calculus. In *DEXA'00: Proceedings of the 11th International Conference on Database and Expert Systems Applications*, pages 222–231, London, UK, 2000. Springer-Verlag.
7. Phan M. Dung, Phan M. Thang, and Francesca Toni. Argument-based Decision Making and Negotiation in E-business: Contracting a Land Lease for a Computer Assembly Plant. In *9th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA)*, Dresden, 2008.
8. Marc Esteva, Bruno Rosell, Juan A. Rodriguez-Aguilar, and Josep Ll. Arcos. Ameli: An agent-based middleware for electronic institutions. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 236–243, Washington, DC, USA, 2004. IEEE Computer Society.
9. F. Nihan Kesim and Marek Sergot. A Logic Programming Framework for Modeling Temporal Objects. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):724–741, 1996.
10. Nihan Kesim. *Temporal Objects in Deductive Databases*. PhD thesis, Imperial College, 1993.
11. Peter McBurney and Simon Parsons. Games that agents play: A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information*, 11(3):315–334, 2002.
12. Roger B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, September 1997.
13. Andrea Omicini and Enrico Denti. From Tuple Spaces to Tuple Centres. *Science of Computer Programming*, 41(3):277–294, nov 2001.
14. Andrea Omicini, Alessandro Ricci, and Nicola Zaghini. Distributed workflow upon linkable coordination artifacts. In Paolo Ciancarini and Herbert Wiklicky, editors, *Coordination Models and Languages*, volume 4038 of *Lecture Notes in Computer Science*, pages 228–246. Springer, 2006.
15. Adrian Paschke and Martin Bichler. SLA Representation, Management and Enforcement. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pages 158–163, Washington, DC, USA, 2005. IEEE Computer Society.
16. Kostas Stathis. A Game-based Architecture for Developing Interactive Components in Computational Logic. *Journal of Functional and Logic Programming*, 2000(5), 2000.
17. Kostas Stathis and Marek J. Sergot. Games as a Metaphor for Interactive Systems. In *In HCI'96, People and Computers XI.*, pages 19–33. Springer-Verlag, 1996.
18. Francesca Toni. E-business in ArguGRID. In Jörn Altmann and Daniel Veit, editors, *Grid Economics and Business Models, 4th International Workshop, GECON 2007*, volume 4685 of *Lecture Notes in Computer Science*, pages 164–169. Springer, 2007.
19. Wil M. P. van der Aalst, Arthur ter Hofstede, Bartosz Kiepuszewski, and Ana Barros. Workflow patterns home page. http://www.workflowpatterns.com/, 2009.