Virtual Organisations as Agent Societies

Jarred McGinnis, Kostas Stathis
Department of Computer Science, Royal Holloway, University of London, UK
Francesca Toni
Department of Computing, Imperial College London, UK

We propose a formal agent-based model for VOs in grid/service-oriented architectures, and an operational model for the creation of VOs. The models abstract away from any realisation choice and can be deployed within different agent systems. Within the proposed models, VOs are seen as emerging from societies of agents, where agents are abstractly characterised by goals and roles they can play within VOs. In turn, VOs are abstractly characterised by the agents participating in them with specific roles, as well as the workflow of services and corresponding contracts suitable for achieving the goals of the participating agents. We illustrate the proposed models in an earth observation scenario.

1 Introduction

The basic definition of Virtual Organisation (VO) is simple enough: organisations and individuals who bind themselves dynamically to one another in order to share resources within temporary alliances [9]. Several issues at various levels of abstraction arise when attempting this binding and sharing. We focus on an abstract, high-level description of VOs and their lifecycle and define a formal model for VOs and their formation that can guide their realisation. Like others (e.g. [4, 7, 10]) we focus on VOs that can be formed and managed automatically by intelligent agents. These agents "represent" organisations and individuals providing and requesting resources/services, by "wrapping" them or by connecting to their published interface. Agents are designed to incorporate the requirements of these organisations and individuals and exhibit some "human aspects" while supporting the decision-making processes automatically. Differently from others, we abstract away from concrete realisation choices so that our models can be deployed within different systems. Also, our models focus on what we believe are the essential elements of VOs, and ignore a number of lower-level aspects, e.g. as defined in the ARCON reference model for collaborative networks [6].

Our representation of the VO lifecycle reflects the orthodox managerial and technical views of VOs, as proposed by [2, 8, 13]. This lifecycle can be structured in three main phases: and the *selection of partners*, formation, operation, and dissolution. In this paper we focus on the formation phase with subphases: (i) *initiation*, whereby an initiating agent *identifies the goals* that it cannot achieve in isolation and *discovers the potential partners* who can assist it in achieving those goals; and (ii) *configuration*, involving some form of *negotiation*, trivial or otherwise, and the *selection of partners*, trimmed down from those discovered, who will constitute the members of the VO once it is started. We see a VO abstractly as a tuple consisting of *agents* participating in the VO, *roles* they play therein, *goals* the VO is set to achieve, the *workflow* of services being provided within the VO, and *contracts* associated with these services, to which agents are meant to conform. We then define the formation phase of a VO as a transition system between tuples providing partial approximations of the resulting VO. Within our model, for the purposes of VOs, agents are seen as existing within *agent societies*: we define these abstractly as our starting point. VOs then emerge within societies as a result of interactions amongst agents, as determined by the roles they play.

Submitted to:
FAVO 2009

Throughout, we illustrate the proposed models for VOs by showing how they cater for the following simple but realistic *scenario*. A government ministry requires the detection of an offshore oil spill. This *high-level goal* cannot be immediately satisfied by the ministry itself. Instead, the ministry contacts (some of) the companies that control satellites which may provide suitable images. Different satellites have different orbits, sensors, capabilities and costs and some satellites may be more appropriate than others in different circumstances. Satellites may include *Envisat*, *ERS-1*, *RADARSAT*. The raw images from any chosen satellites are not immediately suitable and need to be processed by appropriate software, for example for *format conversion* (into formats such as TIFF, JPEG, etc), *reprojection* (into different coordinate systems), *pattern recognition* (to detect ships, buildings, oil spills etc). In our scenario, the ministries would require a pattern recognition post-processing for oil spills. This process results into a VO consists of three parties: the *government ministry*, playing the role of *service requester*; the *satellite company* and the *post-processing company*, playing the role of *service providers*. The two requested services are orchestrated into a *workflow* by the ministry, where the post-processing of the image data requires that the image data is created first. To guarantee the properties and delivery of the services provided by the satellite company and the post-processing company and to ensure those companies are compensated for their efforts, all the parties involved sign a *contract*, which binds parties, in particular establishing their roles within the VO and defining a *Service Level Agreement* (SLA). In our scenario, a SLA may specify the resolution of images, quality threshold, and time of delivery.

The paper is organised as follows. The formation phase for VOs, defined as a state transition system, will be formalised in section 3. But first, in section 2, we formalise the (abstractions of) agents, their roles and social norms (specified as interaction protocols) in an agent society, services/resources, workflows combining them and contracts. These components will become the constituents of the VOs as they are produced. Finally, section 4 concludes.

2 Agent Societies

For the purposes of VOs, agents "representing" services can be seen as existing within a *society* (of agents). VOs emerge as a result of interactions amongst the agents in this society. In other words, the agent society can be seen as the breeding environment [5] for VOs. We will assume that an agent society exists prior to decisions and interactions leading to VOs. However, typically this society is intended to be "virtual", in that it is the implicit result of the existence of agents and services within an agent-enabled grid/service-oriented architecture.

An agent society is characterised by roles that agents can adopt, services available to and controlled by agents in the society, possible combinations of these services within workflows, and possible contracts between agents. Formally, $AgentSociety = \langle Agents, Services, Roles, Workflows, Contracts \rangle$ where

- *Agents* is a (finite) set of agents, $\{A_1, \ldots, A_n\}$, with $n \geq 2$; each agent is equipped with a set of individual goals, an evaluation mechanism, and a set of roles it can cover (see section 2.3)
- *Services* is a (non-empty and finite) set of services represented by agents (see section 2.1)
- *Roles* is a (non-empty and finite) set of roles that agents can play within the society as well as the VOs, once they have been created; we require that there are roles for $requester(s)$ and $provider(s)$ in *Roles*, for all $s \in Services$; roles are associated with interaction protocols (see section 2.2)
- $Workflows$ is a (non-empty) set of possible combinations of services in *Services* (see section 2.4)
- $Contracts$ is a (non-empty) set of possible combinations of agents (in *Agents*), roles (in *Roles*), and workflows (in $Workflows$) as terms in a contract (see section 2.5)

Note that, in addition to roles for *requester* and *provider* of all available services in the society, *Roles* may also include roles for *consultant*, to provide information on how to obtain or use some services, *arbitrator* for some service, etc. Finally, note that there are no goals of the agent society itself, and goals exist within agents only. However, VOs are goal-oriented: we will see, in section 3, that the goals of VOs originate from those of individual agents.

The components of an agent society will be defined using several abstract underlying languages. Here we single out these languages. We adopt the following conventions: variables start with capital letters; constants start with lower-case letters or numbers; '_' stands for the anonymous variable.

We use a set AI_{as} of agent identifiers, that serve as unique "names" to address agents in the society, e.g. to support communication. An example is $satERS1Ag$, representing the satellite ERS-1.

We use a set RI_{as} of *role labels* for the definition of *Roles*. We require that $requester(s)$ and $provider(s)$ belong to RI_{as}, for all $s \in Services$.

We use *contracts identifiers*, CI_{as}, univocally identifying and distinguishing contracts in $Contracts$.

We will assume some given, shared *ontology* O_{as}, that, for the purposes of this paper, for simplicity we think of as a set of atomic and ground propositions.[1] O_{as} will include (i) (an abstract representation of) all services in *Services*, (ii) generic infrastructure knowledge, e.g. for querying registries holding information about agents and the services they provide. An example of the latter may be $provides(X, satImage(in,out))$ instantiating X to $satERS1Ag$, representing that the agent named $satERS1Ag$ represents a provider of service $satImage(in,out) \in Services$.

We will see that VOs emerge in an agent society by inter-agent communication-based interaction. As conventional, it is important that communicating agents share a *communication language*, that constitutes a "lingua franca" amongst the agents. We thus assume as given a language ACL_{as} of locutions. As conventional, locutions consist of a *performative* and a *content*. Examples of locutions in ACL_{as} may be $request(s)$ and $accept(s)$, where $s \in Services$ is the content,

Each individual agent is equipped with an *internal language* to express its knowledge/beliefs and goals. Since the goals of VOs are derived from the individual agents' goals, we need to assume that the agents share at least a fragment of their internal languages. This fragment can be also used to express, e.g., conditions in protocol clauses (see section 2.2). We will refer to this shared fragment of all agents' internal languages as L_{as}. We require that the sentence $true$ is contained in this language, as well as sentences built using the usual connectives $\wedge$ and $\neg$. We assume that this language is propositional. Examples of sentences in L_{as} may be $toBuy(satImage(in,out))$. Sentences are meant to be evaluated using the agents' internal evaluation mechanisms (see section 2.3).

Note that there are no eligibility conditions to choose which agents enter the society, as we assume an open setting where agents can freely circulate. In this context, VOs provide a mechanism for defining which agents can be suitably put together to help solving each others' goals.

2.1 Services

Concrete services, that can be executed by their providers, are described using sentences in O_{as}. Examples of concrete services are $satImage(in,out)$ with *in* and *out* representing the inputs and outputs for the service (e.g. *in* may be $[38.0, -9.4, 1000, 500, 5, optical, 3]$ and *out* may be $results.data$[2] and some

[1] In general, O_{as} may need to contain hierarchical concepts, for example a "generic service" may be defined as either a "satellite service" or a "processing service".

[2] Here, 38.0 and -9.4 are the latitude and longitute coordinates of the area to be scanned, 1000 is the resolution of the image in metres, 500 is the km^2 area to be scanned, 5 is the frequency in hours for the area to be scanned, *optical* is the type of sensor to be used, 3 is the wave frequency to be used in the scan, *results.data* is the name of the file produced by Envisat.

service for detecting oil-spills $detectOilSpills([a,5],b)$[3].

In order to accommodate negotiation for the provision of (concrete) services, during the formation of VOs, it is useful that agents are able to talk about *partially uninstantiated* and *abstract* services, before they commit to any concrete instantiation (actually, it may happen that this instantiation can only be provided at the time of execution of the services). For example, an agent may require, for some given a, $detectOilSpill([a,T],B)$, where the threshold T and the output processed data image B are as yet unspecified (T may be associated with constraints, e.g. $T \geq 5$).

In summary, we adopt the following description of services:

$serviceName(In,Out)$	:	uninstantiated service (*abstract service*)
$serviceName(in,Out)$	:	*partially uninstantiated service*
$serviceName(In,out)$	:	*partially uninstantiated service*
$serviceName(in,out)$	:	fully instantiated service (*concrete service*)
$serviceName$	=	predicate, with $serviceName(i,o) \in O_{as}$, for constants or lists of constants i,o
in,out	=	constants or lists of constants
In,Out	=	variables or lists of variables

The *serviceName* can be seen as representing the "type" of service being provided by $serviceName(in,out)$. We will often refer to an abstract service $serviceName(In,Out)$ simply as *serviceName*. Note that an abstract or partially instantiated service can be seen logically as representing a disjunction of concrete services (one for each possible instantiation). We could thus see the process of negotiating an instantiation of an abstract or partially instantiated service as the process of negotiating a concrete service in a set of alternatives (the disjuncts).

For our scenario, we need four types of services, namely *satImage* and three processing services (with *serviceName* one of $formatConversion$, $reprojection$ and $detectOilSpill$). We have already seen examples of concrete services of type *satImage* and $detectOilSpill$.

2.2 Roles and Protocols

A role is defined as a tuple $\langle rid, PC \rangle$ where $rid \in RI_{as}$ is the label of the role, and *PC* is a *Protocol Clause*, understood in this paper as a (non-empty and finite) set of *Operations* defined as follows:

$Operation$	=	$\psi[send(m,i,rid')]\phi$	(send operation)
	|	$\psi[receive(m,i,rid')]\phi$	(receive operation)
ψ	$\in$	$L_{as} \cup O_{as}$	(precondition)
ϕ	$\in$	L_{as}	(postcondition)
m	$\in$	ACL_{as}	(locution)
i	$\in$	AI_{as}	(unique identifier of agent)
rid'	$\in$	RI_{as}	(role label)

Intuitively, each operation has has three parts: a locution m in ACL_{as}, an identifier i in AI_{as} of the communicative partner (i.e. the intended recipient or the actual sender of message m, respectively for *send* and *receive*), and the identifier rid' of the role that the agent i is intended to be playing when receiving or sending the message (respectively for *send* and *receive*). An agent can send or receive the

[3] Here, a represents the input raw satellite data, 5 is the acceptable threshold for false positives and b is the output processed data image, as computed by the provider of $detectOilSpill$. Note that algorithms for detecting oil spills may occasionally give false positives, namely indicate that there is an oil spill at some location where in reality no oil spill is present there. The lower the acceptable false positive threshold requested from a service, the more expensive the service.

locution (according to what the operation specifies) if and only if the evaluation mechanism of the agent evaluates the precondition ψ to true. Once the message is sent or received, the postcondition ϕ *will automatically hold* (namely the evaluation mechanism of the agent will evaluate this condition positively after the message is sent or received, until further changes). Moreover, when an agent i' playing some role $\langle rid, PC\rangle$ sends a locution $send(m,i,rid')$ to some other agent i, i receives the message from i' indicating that i' sent it while playing role rid: $receive(m,i',rid)$. This message will be "processed" by i using the role with identifier rid'.

Note that our models could adopt other formalisms for communication, e.g. non-determinisitc finite-state automata. We have chosen here the simple formalism of protocol clauses in order to ground our models fully, and as this formalism is an abstraction of several existing formalisms for modelling inter-agent communication, e.g. LCC [12].

To illustrate roles, consider a simple example where an agent playing the role of *requester* (of some service S) sends a *request* to an agent it believes to be a provider of S, and requiring it to be playing the role of *provider* of S. The *provider* agent replies with *accept* or *refuse* depending on whether it is indeed a provider of that service S (and it wants to sell that service) or not (respectively).

$\langle$*requester*(S), { `toBuy(S)`$\wedge$`provides(Ag,S)`[*send(request(S),Ag,provider(S))*]`r(S,Ag)`,
`r(S,Ag)`[*receive(accept,Ag,provider(S))*]`bought(S)`,
`r(S,Ag)`[*receive(refuse,Ag,provider(S))*]`true` }$\rangle$

$\langle$*provider*(S), { `true`[*receive(request(S),Ag,requester(S))*]`reqBy(Ag,S)`,
`reqBy(Ag,S)`$\wedge$ `toSell(S)`[*send(accept,Ag,requester(S))*]`sold(S)`,
`reqBy(Ag,S)`$\wedge$ $\neg$ `toSell(S)`[*send(refuse,Ag,requester(S))*]`true` }$\rangle$

Note that we use here two *protocol clause schemata*, where variables `S` and `Ag` are used instead of concrete values. These variables are implicitly universally quantified over the appropriate languages.

A protocol clause for a role defines the communicative actions for any agent adopting the role. However, protocol clauses typically relate to other protocol clauses and give a global picture of the interaction amongst agents and roles. For the earlier example, the two roles, *requester*(S) and *provider*(S), are related to one another to form a simple *negotiation protocol*. Intuitively, a protocol is a (non-empty and finite) set of protocol clauses for roles in *Roles* that are "related" to one another, possibly indirectly.

With an abuse of notation, we will often refer to a role simply by its identifier and will use the identifier to stand for the corresponding protocol clauses. Moreover, when an agent needs to play the role of *requester* for any service, we use *requester* to stand for *requester*(s) for any service $s \in Services$. Finally, we use *provider*$(serviceName)$ to indicate that a service provider can provide all instances of an abstract service $serviceName(In,Out)$ or when we are interested in the provision of some instance of this service without specifying which one.

2.3 Agents

For the purposes of VOs, an agent can be seen abstractly as a tuple $\langle i,R,G\rangle$ where $i \in AI_{as}$ is the unique identifier for the agent; $R \subseteq Roles$ is a (non-empty) set of roles that the agent can play within the society; $G \subseteq L_{as}$ is a (non-empty) set of goals of the agent.

An agent is also equipped with an *evaluation mechanism* for determining whether (i) preconditions in roles are satisfied, (ii) goals are fulfilled by the agent in isolation. This mechanism is affected by the execution of protocols in that postconditions of protocol clauses are taken into account by this evaluation mechanism (they are satisfied after the protocol clause is executed, until overwritten by further postconditions). We do not include this evaluation mechanism within the representation of an agent in a

society as this mechanism is private to agents and different agents may adopt different such mechanisms in general.

Roles and goals of an agent $\langle i, R, G\rangle$ are inter-related as follows:

(a) $\forall r \in R$, $\exists g \in G$ which "enables" i to adopt r, namely the need to fulfil g is a precondition for one of the protocol clauses in r;

(b) $\forall g \in G$, $\exists r \in R$ such that playing the role r gives i a "chance of fulfilling" g namely one of the protocol clauses in r admits the fulfilment of g as one of its postconditions.

For example, in the earlier protocol clause for the role $requester(S)$, `toBuy(S)` corresponds to a goal, and `bought(S)` corresponds to the goal being fulfilled. Example agents for our scenario are

$\langle clientAg, \{requester\}, \{toBuy(someI), toBuy(someD)\}\rangle$
$\langle satERS1Ag, \{requester(detectOilSpill), provider(satImage)\}, \{toSell(someI)\}\rangle$
$\langle detectAg, \{provider(detectOilSpill)\}, \{toSell(someD)\}\rangle$

where $someI$ is of the form $satImage(in, Out)$ and $someD$ is of the form $detectOilSpill([Out, t], Out')$ for some in and t (as discussed earlier). Here, the agent identified by $clientAg$ represents the government ministry and can only play the $requester$ role (for any service), the agent identified by $satERS1Ag$ represents the ERS-1 satellite and can play both the $requester$ role for $detectOilSpill$ services and the $provider$ role for $satImage$ services, and the agent identified by $detectAg$ can play only the $provider$ role for $detectOilSpill$ services. The agents' goals allow them to engage in interactions following the protocols for the roles they are equipped with (see the simple protocol clauses of section 2.2).

2.4 Workflows

We see workflows simply as (non-empty) sets of services[4] possibly annotated with "constraints", that are sentences in L_{as}. Services may be abstract, partially instantiated or concrete, as in section 2.1. As an example, consider the annotated workflow (consisting of a single partially instantiated service)

$$\{satImage([38.0, -9.4, Res, 500, 5, SensorType], Output)\} \bigcup \{(Res \in [900, 1100], (SensorType \in \{radar, optical\}\}$$

where the resolution Res and $SensorType$ arguments are constrained within the annotation.

We require that the constraints annotating workflows are satisfiable in L_{as}. Annotations only make sense for workflows with at least one partially instantiated or abstract service. They are intended to restrict the possible instantiations of the services in the workflow. Typically, as in the example above, they pose restrictions on the variables occurring in the services in the workflow.

We will adopt the following terminology: an *abstract workflow* is a workflow with at least one abstract or partially instantiated service (with or without annotations); a *concrete workflow* is a workflow consisting solely of concrete services (without annotations).

Note that a concrete workflow may or may not be executable, and that, prior to execution of a workflow, may need to be appropriately set up. In this paper, we focus on the *formation* phase of VOs and ignore execution issues that may occur in the *operation* phase.

2.5 Contracts

Inspired by web service contract standards [1], we define a contract as $\langle Cid, Context, SDT, GT\rangle$ where

[4] More generally, workflows can be compositions, e.g. by sequencing or parallel execution, of services.

- $Cid \in CI_{as}$ is a unique identifier for the contract
- *Context* indicates all agents bound by the contract (the "contracted parties"), and their role in the contract; formally, it is a set of pairs of the form $(AgentId, AgentRole)$ such that $\langle AgentId, R, \_ \rangle \in Agents$ and $AgentRole \subseteq R$
- *SDT*, the *Service Description Terms*, is a workflow, consisting of services being contracted,
- *GT*, the *Guarantee Terms*, is a (possibly empty) set of sentences in L_{as} that define the assurances with regards to the services described in *SDT*.

The *GT* component of a contract may also include rewards/penalties for the contracted parties and refer to roles (and protocols) to be used by agents in the case of exceptional system behaviour. For example, if blame for failure is disputed, there may be a clause in *GT* defining a protocol for arbitration.

By definition of *Context*, we require that the contracted parties play, within the contract, some of the roles they are equipped with. We require that there are at least two different agents involved in any contract, and that there is at least one agent playing the role of $requester(s)$ and at least one agent playing the role of $provider(s)$ for some service s, namely:

- $\exists (id1, role1), (id2, role2) \in Context$ such that $id1 \neq id2$, and $requester(s) \in role1$, $provider(s) \in role2$.

We exclude the possibility that the same agent may be a provider and a requester for the same service:

- $\nexists (id, role) \in Context$ such that, for some $s \in Services$, $\{requester(s), provider(s)\} \subseteq role$.

We require that for all the services in *SDT* there exists an agent in *Context* providing that service:

- $\forall s \in SDT$, $\exists (id, role) \in Context$ such that $role = provider(s)$.

A simple example of a contract is:

$$\begin{aligned} \langle \quad & contractX, \{\langle clientAg, \{requester(formatConversion)\}\rangle, \langle procF, \{provider(formatConversion)\}\rangle\}, \\ & \{formatConversion([image.jpeg, jpegTOgif], imageGIF.gif)\}, \\ & \{dueBy(ImageGIF.gif, 1400hrs, 12.4.09), \\ & \ priceReduced(ImageGIF, 1400hrs, 12.4.09, reduction(0.5))\}\rangle \end{aligned}$$

The contract, identified as *contractX*, is between *clientAg* and *procF* for the delivery of (an instance of) the service $formatConversion$, for converting the file *image.jpeg* using the operation called $jpegTOgif$. The service has a due delivery date specified using the *dueBy* predicate. The clause on *priceReduced* specifies that the price charged will be halved if the provider fails to deliver on time.

2.6 From Agents and Services to the Agent Society

Given *Agents* as in section 2.3 and *Services* as in section 2.1 an agent society "emerges" with:

- $Roles = \bigcup_{\langle i,R,G \rangle \in Agents} R$ (the possible roles are all roles that agents within the society can play)
- the concrete workflows in $Workflows$ are all possible (non-empty) sets of services, namely $(2^{Services} - \emptyset) \subset Workflows$; the remaining elements of $Workflows$ are abstract, possibly annotated "versions" of these concrete workflows
- *Contracts* is built solely from elements of $Workflows$, *Roles* and *Agents*.

Moreover, we require that each service is "represented" by an agent within the society, in other words the possible services in the society correspond to all services the agents can provide:

- $\forall s \in Services, \exists \langle provider(s), \_ \rangle \in Roles$

However, it may be the case that several alternative protocols exist in the society for the same role, namely: $\langle rid, PC \rangle$ and $\langle rid, PC' \rangle$ both belong to $Roles$ for $PC \neq PC'$. The creation of a VO will need to address the choice of protocols being used for negotiation of workflows and contracts.

3 The VO Formation Phase

VOs can be defined as tuples $< A_{vo}, G_{vo}, R_{vo}, Wf_{vo}, Con_{vo} >$ where:

- $A_{vo} \subseteq Agents^*$ with $Agents^* = \{\langle i, R', G' \rangle | \langle i, R, G \rangle \in Agents$ and $R' \subseteq R$, $G' \subseteq G\}$; A_{vo} contains at least two agents and exactly one agent in A_{vo} is referred to as the *initiating agent*, denoted ag_0
- G_{vo} is a set of goals for the agents in A_{vo}, which contains at least a goal of the initiating agent: $G_{vo} \subseteq \bigcup_{\langle i,R,G \rangle \in Agents} G$ and $G_0 \cap G_{vo} \neq \emptyset$, where $\langle ag_0, R_0, G_0 \rangle \in Agents$
- R_{vo} is a set of roles to be played by the participating agents, where $R_{vo} \subseteq Roles$
- $Wf_{vo} \subseteq Workflows$ is the workflow of services currently agreed amongst the agents in A_{vo}
- $Con_{vo} \subseteq Contracts$ is a set of contracts between the agents in A_{vo}

$Agents^*$ represents the set of all possible "full" or "partial" specifications of agents, corresponding to concrete choices of roles agents may play and goals they may focus on within a specific VO. A_{vo} describes the (partial specifications of) agents involved in the VO, as providers or requestors of services, or in whichever other roles, as indicated by R_{vo}. A_{vo} only describes the agents insofar as their involvement in the VO is concerned (and thus possibly omitting some of their goals and roles, not relevant to the VO). The Wf_{vo} and Con_{vo} components determine the behaviour of the VO and its members during the execution and dissolution phases of VOs. The G_{vo} and V_{vo} components are related to the the A_{vo} component in that $G_{vo} = \bigcup_{\langle i,R,G \rangle \in A_{vo}} G$ and $R_{vo} = \bigcup_{\langle ag_0,R_0,G_0 \rangle \in A_{vo}} R$.

In our model, a VO is produced, during the formation phase, through interactions amongst the agents composing the VO. These interactions can be understood in terms of several operations progressively refining "partial" representations of VOs. These operations are defined as transitions, as outlined below. In the remainder, $Ids(A)$ refers to all identifiers of (partial specifications of) agents in the set A.

3.1 Goal Identification

The *identify goals* transition results in the additions of the initiating agent ag_0 and (some of) its goals into the partial VO tuple. These are goals that the agent cannot achieve on its own. Given $< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$ $\xrightarrow{identify\ goal}$ $< A_{init}, G_{init}, \emptyset, \emptyset, \emptyset >$, then

- $A_{init} = \{\langle ag_0, \emptyset, G_{init} \rangle\}$ for some $\langle ag_0, R_0, G_0 \rangle \in Agents$ such that some goals $G'_0 \subseteq G_0$ cannot be fulfilled by ag_0 in isolation;
- $G_{init} = G'_0$

Here, goal fulfilment is determined using the evaluation mechanism of agent ag_0 (see section 2.3). Note that no role is yet identified for ag_0: this will be done in transition *establish roles* (see section 3.4).

Assume, for example, that *clientAg* is informed by its user that a possible oil spill has been reported by a passing vessel. The user provides the latitude and longitude, acceptable false positive threshold and scan area. Armed with its user's parameters the agent initiates the VO formation process by first identifying its goals. The parameters correspond to high-level goals, that will later be decomposed into specific workflows. In the example, the high-level goal $detectPossibleOilspill(38.0, -9.4, 500, 5)$ given by the user is to detect an oceanic oil spill off the Portuguese coast at a latitude and longitude of 38.0 and -9.4 with an acceptable false positive threshold of 5% for the surrounding 500 square kilometres. The agent may decide that to satisfy this high-level goal it needs two services:

$g_1 = toBuy(satImage([38.0, -9.4, \_, 500, \_, radar, \_], \_))$, and
$g_2 = toBuy(oilSpillDetect([\_, \_, 5], \_))$

where the sensor-type for the satellite providing the image must be radar, because of the required resolution and weather conditions, and once this image data is obtained a service is needed to provide the actual identification of the oil spills on the images. In summary, *identify goals* will compute $A_{init} = \{\langle clientAg, \emptyset, G_{init}\rangle\}$ and $G_{init} = \{g_1, g_2\}$.

Note that in our model goals of VOs emerge from goals of agents. Once the goals of VOs have been identified, they will dictate the behaviour of agents during the operation of VOs.

3.2 Partner Discovery

The *discover partners* transition results in the addition of a number of agents to the set of agents in the current partial VO (after *identify goals*). Whether by traditional means such as a yellow page registry or through 'friend of a friend' discovery utilising the multiagent system, the VO tuple is transformed to include potential partners that the initiating agent believes will help it reach its goals, notably by providing suitable services. Given $< A_{init}, G_{init}, \emptyset, \emptyset, \emptyset > \xrightarrow{discover\ partners} < A_{pot}, G_{init}, \emptyset, \emptyset, \emptyset >$, $A_{pot} = A_{init} \bigcup A_{queryresult}$ with $A_{queryresult}$ including these potential partners is such that

- $A_{queryresult} \subseteq Agents^* - A_{init}$ and each element in $A_{queryresult}$ is of the form $\langle i, \emptyset, \emptyset\rangle$
- each agent in $A_{queryresult}$ is a provider of one of the services in G_{init} namely, for each $i \in Ids(A_{queryresult})$, if $\langle i, R, G\rangle \in Agents$ then $\exists s$ such that $toBuy(s) \in G_{init}$ such that $\langle provider(s), PC\rangle \in R$.

In our example, *clientAg* finds that two satellite image providers advertise the services it is interested in. Both agents *satERS*1*ag* and *radSatAg* represent a radar-capable polar satellite with orbits amenable to the area of interest. Moreover, there is one agent, *procOSAg*, who can provide the type of image processing in which *clientAg* is interested. After this transition is completed,

$A_{queryresult} = \{\langle satERS1ag, \emptyset, \emptyset\rangle, \langle radSatAg, \emptyset, \emptyset\rangle, \langle procOSAg, \emptyset, \emptyset\rangle\}$

3.3 Partner Selection

The set of potential partners discovered by ag_0 may include unreliable or untrustworthy ones. The *select partners* transition allows the agent to prune the results of the *discover partners* transition. We do not provide a detailed specification of the pruning mechanism needed to support this stage as this is largely dependent on mechanisms for assessing trustworthiness and reliability. Several such mechanisms exist in the literature. Any could be used here.

Generally, given $< A_{pot}, G_{init}, \emptyset, \emptyset, \emptyset > \xrightarrow{select\ partners} < A_{pre}, G_{init}, \emptyset, \emptyset, \emptyset >$, then

- $A_{pre} \subseteq A_{pot}$

- $ag_0 \in Ids(A_{pre})$
- for each s such that $toBuy(s) \in G_{init}$ there exists $i \in Ids(A_{pre})$ such that
  if $\langle i, R, G\rangle \in Agents$ then
  $\exists \langle provider(s), PC\rangle \in R$.

In the example, after the *select partners* transition is completed:
$A_{pre} = \{\langle clientAg, \emptyset, G_{init}\rangle, \langle satERS1ag, \emptyset, \emptyset\rangle, \langle procOSag, \emptyset, \emptyset\rangle\}$.
Note that in general several providers of the same service may still be in A_{pre} after the pruning.

3.4 Establish Roles

Before the agents are able to negotiate workflows and contracts, the roles they will be playing in the negotiation need to be established. These roles (with their protocols) are the social norms used for forming the VO. Establishing these roles also amounts to establishing protocols for them (as our roles include protocols). Roles include requester and provider roles, but may also include other roles (e.g. that of arbitrator, or contract-negotiator if agents other than provider and requester agents may be needed to support the contract negotiation stage of VO formation). For simplicity, we assume that these roles are decided by the initiating agent, and that, given $< A_{pre}, G_{init}, \emptyset, \emptyset, \emptyset > \xrightarrow{establish\ roles} < A_{roles}, G_{init}, R_{vo}, \emptyset, \emptyset >$, then

- $Ids(A_{pre}) \subseteq Ids(A_{roles})$
- $A_{roles} = A^*_{pre} \cup A_{rest}$, where
  - $A^*_{pre} = \bigcup_{\langle i,\emptyset,G\rangle \in A_{pre}} \{\langle i, R_i, G_i\rangle\}$ for some R_is such that $R_i \subseteq R^*_i$ and $G_i \subseteq G^*_i$ where $\langle i, R^*_i, G^*_i\rangle \in Agents$
  - $A_{rest} \subseteq Agents^*$ (where A_{rest} may be empty)
  - $A_{rest} \cap A^*_{pre} = \emptyset$
- $R_{vo} = \bigcup_{\langle i,R_i,\_\rangle \in A_{roles}} \{R_i\}$
- for each s such that $toBuy(s) \in G_{init}$, there exists exactly one $\langle provider(s), PC_{provider(s)}\rangle$ and exactly one $\langle requester(s), PC_{requester(s)}\rangle$ in R_{vo}
- for every $\langle r_1, PC_{r_1}\rangle$ and $\langle r_2, PC_{r_2}\rangle$ in R_{vo}, if $r_1 = r_2$ then $PC_{r_1} = PC_{r_2}$, namely there is exactly one role for each role identifier in R_{vo}.

Note that we do not impose that ag_0 plays the role of requester of all the services in the workflow it wants to instantiate: indeed, in general it may be possible that ag_0 delegates the task of requesting some or all services to some other agent. In particular, it may be the case that $\langle ag_0, \emptyset, G_{init}\rangle$ belongs to A_{roles}. Also, we allow for the same agent to play several roles in a VO (namely, $\langle i, R_i, G_i\rangle$ may belong to A_{roles} with R_i containing more than one role).

In our running example, once the *establish roles* stage is completed:

$A_{roles} = \{\ \langle clientAg, \{\langle requester(satImage([38.0, -9.4, \_, 500, \_, radar, \_], \_)), PC_1\rangle,$
$\langle requester(oilSpillDetect([\_, \_, 5], \_)), PC_2\rangle\}, G_{init}\rangle,$
$\langle satERS1ag,$
$\{\langle provider(satImage([38.0, -9.4, \_, 500, \_, radar, \_], \_)), PC_3\rangle\}, \emptyset\rangle,$
$\langle procOSag, \{\langle provider(oilSpillDetect([\_, \_, 5], \_)), PC_4\rangle\}, \emptyset\rangle\}$
$R_{vo} = \{\ \langle requester(toBuy(satImage([38.0, -9.4, \_, 500, \_, radar, \_], \_))), PC_1\rangle,$
$\langle requester(toBuy(oilSpillDetect([\_, \_, 5], \_))), PC_2\rangle,$
$\langle provider(toBuy(satImage([38.0, -9.4, \_, 500, \_, radar, \_], \_))), PC_3\rangle,$
$\langle provider(toBuy(oilSpillDetect([\_, \_, 5], \_))), PC_4\rangle\}$

Here, the PC_i are protocol clauses that the agents commit to follow during the negotiation of workflows. In this specific example, no role/protocol is specified for the *agree contract* transition. Note that other agents may be brought into A_{roles} at this stage to play these new roles.

3.5 Negotiation

The negotiation activities in the VO formation amount to 1) agreeing a concrete workflow (*agree Wf*) and 2) agreeing a set of contracts amongst agents contributing to the workflow, by providing services in it, and the initiating agents (stage *agree contract*). Both transitions make use of roles (and protocols) identified at the *establish roles* transition: communicating by following these protocols, agents agree on the provision of services and contracts. Negotiation may result in additional goals to be added, as goals of agents providing services. The *agree contract* transition may cause no changes in the partial VO tuple, if no suitable roles have been computed by the *establish roles* transition. For lack of space we will only describe the *agree Wf* transition.

In order for the computed VO to be meaningful, it needs to compute a workflow that is concrete or partially instantiated, but can be fully instantiated when the workflow is executed. This workflow instantiates the abstract workflow corresponding to the *toBuy* goals in G_{init}. This instantiation may be obtained after several negotiations, each following the protocols of the roles identified after the *establish roles* transition, each resulting in a service becoming instantiated. After each instantiation, the initiating agent puts those instantiated services into the workflow component of the VO tuple.

Generally, given $< A_{roles}, G_{init}, R_{vo}, \emptyset, \emptyset > \xrightarrow{agree\ Wf} < A_{vo}, G_{vo}, R_{vo}, Wf_{vo}, \emptyset >$, then

- $Ids(A_{vo}) \subseteq Ids(A_{roles})$
- $ag_0 \in Ids(A_{vo})$
- for each s such that $toBuy(s) \in G_{init}$ there exists exactly one agent $i \in Ids(A_{vo})$ such that $\langle i, R_i, G_i \rangle \in A_{vo}$ and $provider(s) \in R_i$, and a successful dialogue between ag_0 and this agent i with ag_0 playing the role of $requester(s)$ and i playing the role of $provider(s)$
- for each $\langle i, R_i, G_i \rangle \in A_{vo}$, if $\langle i, R_i^*, G_i^* \rangle \in A_{roles}$ then
  $R_i^* = R_i$ (namely roles cannot be changed at this stage)
  $G_i \supseteq G_i^*$ (namely goals can only be added at this stage)
  if $\langle i, R_i^{**}, G_i^{**} \rangle \in Agents$ then $G_i \subseteq G_i^{**}$ (namely all goals are chosen from the pool of goals of the agent)
- $G_{init} \subseteq G_{vo}$
- $G_{vo} = \bigcup_{\langle i, R_i, G_i \rangle \in A_{vo}} G_i$

Intuitively, agents may decide to add goals at this stage to avoid agreements to provide a service which could prevent the fulfillment of some of their goals. We impose that the initiating agent is not allowed to change the workflow. However, it can add constraints or services to it, as soon as no new role is required by this addition. For example, this would be needed and useful to support shimming[5] of services. Goals of provider agents may render this shimming necessary (e.g. because a service provider does not want to interface to another service provider).

[5]Informally, shimming is the introduction of a service into a workflow to ensure that the output of a preceding service matches the type required by the input of the subsequent service.

Note that one single provider per service is required. These providers will need to be selected amongst all agents that have successfully completed a dialogue with ag_0. We do not impose any constraints on how this choice is performed: the given protocols may typically dictate this.

- Wf_{vo} is the result of instantiating Wf by the given sequence of successful dialogues, as dictated by G_{vo}; the providers of the services are given by A_{vo}.

4 Conclusions

We have described a formalisation for VOs in grid and service-oriented architectures, formed from agent societies, using a realistic scenario for illustration throughout. This formalisation is abstract and independent of any realisation choices (in terms of agent architectures, communication platform etc). It can guide the development of (agent-based) VOs, in that it identifies essential components (such as several underlying languages for services, identifiers, communication, as well as protocol-based roles for negotiation of services and contracts). We have experimented with some of the interactions presented here for the earth observation scenario [3] with emphasis on the coordination patterns agents should follow when creating a VO [15].

Our emphasis on the use of protocols to support VOs is also advocated by [9]. The CONOISE-G [10] project presented an agent-based model for VOs on the grid, but focused on the challenging task of engineering a working system and thus making concrete realisation choices (e.g. agents use a constraint satisfaction algorithm for decision-making). We have taken a more abstract view of agents, agent society and VOs, to ensure that the definitions can be ported to any other agent-enabled grid systems to support VOs in general. Papers such as [4] speculate on the consequences of introducing software agents as a means to alleviate the burden on human decision-making. We have a similar focus in that we see an opportunity, by utilising the multiagent paradigm. There are a few papers that have formalised aspects of agent-enabled VOs, for example [11] look at voting protocols for VOs while [14] focuses on the representation of contracts in VOs based on a specific commitment-based approach for them. We have taken a more exhaustive view by considering all components of agent-enabled VOs but more abstractly.

As future work, it will be important to further validate the proposed model with further examples, e.g. in e-business and pharma settings, as well as formally verifying that the VO formation model provided results in "coherent" VOs, namely VOs where all agents involved can fulfil their relevant goals as a result of the participation in the VO, given that the VO is executed as agreed.

Acknowledgements

We would like to thank the anonymous referees for their comments on a previous version of this paper. The work reported here was partially funded by the Sixth Framework IST programme of the EC, under the 035200 ARGUGRID project.

References

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke & M. Xu (2005). *Web Services Agreement Specification (WS-Agreement)*.

[2] W. Appel & R. Behr (1998): *Towards the Theory of Virtual Organisations: A Description of Their Formation and Figure*. *VoNet - Newsletter* 2(2), pp. 15–35. Http://www.virtual-organizations.net.

[3] Stefano Bromuri, Visara Urovi, Maxime Morge, Kostas Stathis & Francesca Toni (2009): *A multi-agent system for service discovery, selection and negotiation*. In: Carles Sierra, Cristiano Castelfranchi, Keith S. Decker & Jaime Simão Sichman, editors: *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*. IFAAMAS, Budapest, Hungary, pp. 1395–1396.

[4] L. M. Camarinha-Matos & H. Afsarmanesh (2001): *Virtual enterprise modeling and support infrastructures: applying multi-agent system approaches*. In: *ACAI 2001*, *LNAI* 2086. Springer-Verlag New York, Inc., pp. 335–364.

[5] L. M. Camarinha-Matos & H. Afsarmanesh (2007): *A framework for virtual organization creation in a breeding environment*. *Annual Reviews in Control* 31(1), pp. 119–135.

[6] (2007). *European Collaborative Networked Organisations Leadership Initiative*. http://ecolead.vtt.fi/. Last visited:28/09/2009.

[7] I. Foster, N. R. Jennings & C. Kesselman (2004): *Brain meets brawn: Why Grid and agents need each other*. In: *Proc. 3rd Int. Conf. on Autonomous Agents and Multi-Agent Systems*. New York, USA, pp. 8–15.

[8] I. Foster & C. Kesselman, editors (2004): *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers. Second eddition.

[9] I. Foster, C. Kesselman & S. Tuecke (2001): *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. *Intl. J. Supercomputer Applications* 15(3).

[10] J. Patel, W. T. L. Teacy, N. R. Jennings, M. Luck, S. Chalmers, N. Oren, T. J. Norman, A. Preece, P. M. D. Gray, G. Shercliff, P. J. Stockreisser, J. Shao, W. A. Gray, N. J. Fiddian & S. Thompson (2005): *CONOISE-G: Agent-based virtual organisations for the Grid*. In: *Proc. of the 4th international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS2005)*. pp. 1459–1460.

[11] Jeremy V. Pitt, Lloyd Kamara, Marek J. Sergot & Alexander Artikis (2005): *Formalization of a voting protocol for virtual organizations*. In: Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh & Michael Wooldridge, editors: *4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*. ACM, pp. 373–380.

[12] D. Robertson (2004): *Multi-agent Coordination as Distributed Logic Programming*. In: *Proceedings for International Conference on Logic Programming*. Springer, pp. 416–430.

[13] G. Sung (2004). *Conceptual Framework for Virtual Project Management*. Available at `http://www.ve-forum.org/Projects/285/P2/CF_VPM.doc`.

[14] Yathiraj B. Udupi & Munindar P. Singh (2006): *Contract Enactment in Virtual Organizations: A Commitment-Based Approach*. In: *AAAI*.

[15] Visara Urovi & Kostas Stathis (2009): *Playing with Agent Coordination Patterns in MAGE*. In: *Workshop on Coordination, Organization, Institutions and Norms in Agent Systems (COIN@AAMAS09)*. Budapest, Hungary.