Agent-Grid Integration Language

Clement Jonquet, Pascal Dugénie, Stefano A. Cerri

Agent-Grid Integration Language

Clement Jonquet, Pascal Dugenie and Stefano A. Cerri

April 2007

LIRMM, University Montpellier 2 & CNRS
161 Rue Ada, 34392 Montpellier, France
{jonquet,dugenie,cerri}@lirmm.fr

Abstract

The GRID and MAS (Multi-Agent Systems) communities believe in the potential of GRID and MAS to enhance each other as these models have developed significant complementarities. Thus, both communities agree on the 'what' to do: promote an integration of GRID and MAS models. However, while the 'why' to do it has been stated and assessed, the 'how' to do it remains a research problem. This paper addresses this problem by means of a service-oriented approach. Services are exchanged (i.e., provided and used) by agents through GRID mechanisms and infrastructure. The paper first consists of a set of states of the art about integration approaches in GRID, MAS and Service-Oriented Computing (SOC). It secondly proposes a model for GRID-MAS integrated systems. Concepts, relations between them and rules are semantically described by a set-theory formalization and a common graphical description language, called Agent-Grid Integration Language (AGIL). This language may be used to describe future GRID-MAS integrated systems. AGIL's concepts are directly influenced by OGSA (Open Grid Service Architecture) and the STROBE agent communication and representation model.

Keywords: Distributed systems, Agent-Grid Integration, Multi-Agent Systems, Agent, STROBE model, Grid, Grid service, OGSA, Service-Oriented Computing, Web Service, Dynamic Service Generation.

1 Introduction

Even though GRID and MAS (Multi-Agent Systems) are both kinds of distributed systems, their underlying motivations are different. GRID focuses on a reliable and secure resource-sharing infrastructure, whereas MAS focuses on flexible and autonomous behaviour in uncertain and dynamic open environments. However their integration has been suggested. Despite different original motivations, we explain in this paper why these two complementary domains join with the concept of service. In 2004, Foster et al. [20] explained why MAS and GRID need each other as 'brain' and 'brawn'. Even if using agents for GRID was very early suggested [44, 53, 56], Foster et al. [20] propose the real first step in GRID-MAS integration as they examine work in these two domains, first to communicate to each community what has been done by the other, and second to identify opportunities for cross fertilization as they explained how GRID and MAS developed significant complementarities. This paper suggests a second step by proposing a GRID-MAS integrated model described by a set-theory formalization and a common graphical description language.

Service as a unifying concept. Service-Oriented Computing (SOC) is the domain of Informatics which deals with the concept of service: Web service, Grid service, semantics, business processes, etc. GRID is said to be the first distributed architecture (and infrastructure) really developed in a service-oriented perspective: Grid services are compliant Web services, based on the dynamic allocation of virtualized resources to an instantiated service [21]. Quite recently, GRID acquired major importance in Service-Oriented Architectures (SOAs) by augmenting the basic notion of Web Service with two significant features: service state and service lifetime management. This is the concept of Grid service [21]. On the other hand, agents are said to be autonomous, intelligent and interactive entities who may use and offer services (in the sense of particular problem-solving capabilities) [17, 34]. MAS have also followed naturally the path towards SOA as interest turns to providing what we may call dynamic high level processes of services: business process management, service composition, semantic services, etc. The SOC community is also turning to MAS considering the important capacities agents have for using and providing services to one another [32]. The concept of service is clearly at the intersection of the GRID and MAS concerns.

Dynamic Service Generation. Web services are currently the main framework chosen by the SOC community to implement SOA. However, Web services are often criticized because they are no more than Remote Procedure Calls (RPC) which have no user adaptation, no memory, no lifetime management, no conversation handling capabilities (simple request/answer interaction). They are passive, they lack semantics and they do not take into account the autonomy of components or do not use a high level, history aware, communication language. Actually, to provide a service means to identify and offer a solution (among many possible ones) to the problem of another. The next generation of services will consist of dynamically generated services, i.e., services constructed on the fly by the service provider according to the conversation it has with the service user. In *Dynamic Service Generation* (DSG), the user is an agent (human or artificial) that is not assumed to know exactly what the provider (also human or artificial) can offer. He finds out and constructs step by step what he wants based on the service provider's reactions. The central idea of DSG is that a service may be based on a conversation. It is not a simple product delivery. Actually, DSG highlights the idea of process and the creation of something 'new' instead of merely delivering something that already exists. In everyday life, when somebody needs new clothes, *buying ready-to-wear clothes* is analogous to asking for a product, whereas *having clothes made by a tailor* is analogous to requiring a service to be generated. Singh and Huhns [61] talk about *service engagement*, instead of simple method invocation. In [36] the author extensively describes the DSG concept by enumerating a set of characteristics that are related to DSG. The two main inspiring sources of requirements for DSG are MAS and GRID. Therefore, new needs in service exchange scenarios are clearly highlighted (dynamicity, composition, conversation based, user-centred behaviour, business processes, semantics, etc.). An integration of GRID and MAS may address these needs.

GRID-MAS integration. In order to propose a GRID-MAS integrated model, we extracted from GRID and MAS related work some key concepts. These concepts are directly influenced by OGSA (Open Grid Service Architecture) and the STROBE agent communication and representation model [10, 38, 36]. These concepts are step-by-step defined in section 3 and 4, however, we sum-up here the two main underlying ideas:

- The representation of agent's capabilities as Grid services in a service container, i.e., viewing a Grid service as an 'allocated interface' of an agent's capability by substituting the object-oriented kernel of Web/Grid services with and agent oriented one;[1]

---

[1]Nowadays Web services (cf. section 2.1) are most of the time object oriented programs (developed with J2EE or .NET) processed in order to produce a WSDL description and able to communicate with SOAP messages. Thus, services benefit of the powerful abstraction characteristic available with the object oriented paradigm, such as encapsulation, inheritance, message

- The assimilation of the service instantiation mechanism – fundamental in GRID as it allows Grid services to be stateful and dynamic – with the dedicated cognitive environment instantiation mechanism – fundamental in STROBE as it allows an agent to dedicate to another one a conversation context.

This paper presents firstly a state of the art on related work about integration approaches in SOC, GRID and MAS (GRID-SOC, MAS-SOC, GRID-MAS). This survey shows that there is no GRID-MAS integrated model in the literature. Afterwards, we propose a set-theory formalization and a common graphical language, called Agent-Grid Integration Language (AGIL), for describing GRID-MAS integrated systems. AGIL raises the challenge to be both a model for GRID-MAS integrated systems and a description language for those systems. The graphic symbols and the set-theory formalization address together this double objective.

Agent-Grid integration language.   Describing simply and clearly key GRID and MAS concepts and their integration is a real need for both communities. GRID and MAS should appreciate a common description language which:

- defines the respective domain ontologies. For instance, GRID needs a description language that summarizes and rigorously explains GRID concepts. Without considering the agent side, AGIL may play also this role;

- uses the same terms and representations for an identical concept. For example, virtual organization and group; choreography of service and agent conversation, role and service, etc. The analogies in GRID and MAS concepts will be described in section 2.3.2;

- rigorously fixes the integration rules e.g., a service instance is instantiated by a unique service factory or any agent member of a VO holds a X509 certificate;

- may help researchers of GRID, MAS or SOC communities to specify and model their GRID-MAS integrated applications and systems. A kind of UML, applicable for GRID-MAS integrated systems;

- would promulgate the development of GRID-MAS integrated systems by proposing a uniform way of describing GRID and MAS together.

AGIL is such a language. It takes a specific graphical representation for each concept and their relations. Moreover, the rules of the integration are all expressed by a set-theory formalization which rigourously defines constraints on these concepts and relations.

Paper overview.   The rest of the paper is organized as follows: Section 2 makes a state of the art on related work about GRID-SOC, MAS-SOC, and GRID-MAS integrations. Considering the reader is aware with basic concepts of SOC, GRID and MAS, we focus in this section just on these domains intersections. Through this state of the art, we have identified three GRID-MAS analogies, that have strongly influenced our integrated model, presented in section 2.3.2. Section 3 presents AGIL's main concepts, relations and rules. Each time a concept is introduced, the corresponding graphical representation is illustrated and the relation(s) between this concept and the other ones is formalized. Related rules are also specified. Whereas this section rather presents the 'static' concepts and relations of AGIL, section 4 details the dynamics of such a model: agent interaction, service-capability instantiation, agent

---

passing, etc. We do not want Grid/Web services to be executed by simple object programs but by intelligent, autonomous and interactive (the three fundamental properties distinguishing agents from objects) agents able to have conversations in order to realize DSG.

reproduction, etc. Section 5 discusses the proposition and shows a simple example. Finally, Section 6 concludes the paper and gives some perspectives.

2 State of the art of integration approaches

2.1 GRID-SOC

The GRID aims to enable *flexible, secure, coordinated resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organization* [22]. Actually, it was originally designed to be an environment with a large number of networked computer systems where computing (Grid computing) and storage (data Grid) resources could be shared as needed and on demand. GRID later extended to any kind of services and it is now a SOA based on the dynamic allocation of virtualized resources to a service instance.

Heritage of Web services.   Nowadays, the main way of implementing a SOA (framework) is by means of Web services (`www.w3.org/2002/ws`) [6]. Web services allow distributed functionalities on a network to be accessed in a standardized way to enable interoperability.[2] They are describable, discoverable and message-based software components that perform specific functions. The two main objectives of Web services are standardization and interoperation. It is clear that the main advantage of Web services is that we can compose them to create new services. So, from service invocations, which are single-shot two-party interactions, Web services started evolving into business processes that are typically long-lived multiparty interactions. This is called Business Process Management (BPM). For an example, see BPEL4WS (Business Process Execution Language for Web Services) on the workflow/orchestration side [8], and WSCL (Web Services Conversation Language) on the conversation/choreography side [51, 62]. BPM is also detailed in section 2.3.2.

Web services have some important drawbacks as mentioned in introduction. This section explains how integrating GRID and Web services may be a solution for some these drawbacks (e.g., memory, life-time); the next section explains how integrating MAS and Web services may be a solution for other ones (e.g., semantics, conversation).

Grid services.   Recently GRID introduces two major aspects in the so-called SOA by distinguishing service factory from service instance. In other words, services are instantiated with their own dedicated resources and for a certain amount of time. These characteristics enable (i) service state management: Grid services can be either stateful or stateless; (ii) service lifetime management:[3] Grid services can be either transient or persistent. In order to be widely accepted, the GRID standardization activity faced the question of convergence of GRID and Web service standards. The decision taken between the Global Grid Forum (GGF) and the World Wide Web Consortium (W3C) was to introduce the resource management capabilities of GRID into the Web services framework. This resulted to a GRID standardization split into two sets of layered specifications:

- the Open Grid Service Architecture (OGSA) [21], on the top layer (architecture);

- the Web Services Resource Framework (WSRF) [19], on the bottom layer (infrastructure).

OSGA.   OGSA (`www.globus.org/ogsa`) specifies how Grid services are created, and how service state and lifetime is managed. The capability to manage service state and service lifetime in OGSA

---

[2]Web services inherit from work and experiences from distributed software component approaches (CORBA, RMI, etc.)
[3]Also called service dynamicity.

enables to ensure an efficient load-balancing of the distributed ressources. The GRID specification main elements are:

- the *Grid Resource Allocation and Management* (GRAM) function is responsible to manage the available resources;

- the *inheritance of the the Web Services standards* to provide open service interface description (WSDL) and an open service communication protocol (SOAP);

- the *Grid Security Infrastructure* (GSI) which allows authentication, delegation, privacy and integrity.

WSRF. More recently, WSRF (`www.globus.org/wsrf`) defines uniform mechanisms for defining, inspecting, and managing stateful resources in Web/Grid services. The motivation of WSRF comes from the fact that even if today's Web services successfully implement applications with state, they need to be standardized to enhance interoperability. Therefore, WSRF describes how to implement OGSA functionalities using Web services.[4] WSRF considers *stateless services that act upon stateful resources*. These services are modelled by an association between two entities: a stateless Web service, that does not have state, and stateful resources that do have state.[5] WSRF calls the resulting association a WS-Resource. Both stateful resources and stateless services can be members of several WS-Resources. However, a WS-Resource is unique and produced by a WS-Resource factory. Introducing state in services is very important. Without state, modelling two service performances will always be the same. Since there is no state transitions, a stateless service does not enable interaction, adaptation, negotiation, engagement etc. A fortiori, a stateless service cannot implement DSG. We will see in section 5.3 how the GRID-MAS integrated model goes further toward DSG than WSRF, enabling a couple service-resource (akin to a WS-Resource) to adapt user specific needs both at the resource and service levels.

GRID evolution. Recently, the same semantic level add-on objective that took place in the Semantic Web community occurred also in GRID. For an introduction to the concept of 'Semantic Grid', see [56]. The GRID is also used as a 'Learning Grid' where GRID architecture and principles are used to change e-learning paradigms [2]. BPM and Grid service composition have become research topics in the GRID community; note [39] which addresses the challenge of modelling workflow of Grid services.

2.2 MAS-SOC

Agents and MAS have been extensively studied in literature, for example [31, 17, 67]. Historically, the agent paradigm originates from the object paradigm, but differs in three major aspects: *autonomy*, i.e., for example, the fact of being able to refuse to answer a message; *intelligence*, i.e., the ability to change its own state alone, without message passing; *interaction*, i.e., the faculty to communicate directly and asynchronously with its environment and other agents by means of direct or indirect message passing with a communication language that is independent of the content of the communication and of the internals of agents. Agents are said to use and provide services (in the sense of particular problem solving capabilities). Actually they have many characteristics interesting for service exchanges:

- Reactive and proactive;

- Efficient and adaptive;

---

[4]WSRF is a re-factoring of the initial specification OGSI (Open Grid Services Infrastructure).

[5]Stateful resources are elements with state, including physical entities (e.g., databases, file systems, servers) and logical constructs (e.g., business agreements, contracts) that are persistent and evolve because of service interactions.

- Know about themselves (self-consciousness);

- Memory and state persistence;

- Able to have conversation (but not yet dialog, cf. section 2.2.2) and work collaboratively;

- Able to negotiate;

- Able to learn and reason to evolve;

- Deal with semantics associated to concepts by processing ontologies.

2.2.1 Agents and Web services

The research activity about the convergence of MAS and SOC is increasingly active.[6] As [61] states: 'Researchers in MAS confronted the challenges of open systems early on when they attempted to develop autonomous agents that would solve problems cooperatively, or compete intelligently. Thus, ideas similar to SOAs were developed in the MAS literature'. [26] prefigured the use of agents in e-commerce scenario (i.e., for service delivery scenarios). The authors made an overview of e-commerce application in 1998 and identify issues in which agent abilities may enhance these scenarios: recommender systems (product/merchant brokering), user interface approaches, and negotiation mechanisms.

For Huhns et al. [32] the key MAS concepts are reflected directly in those of SOC: ontologies, process models, choreography, directories and service level agreements. Some work has already been proposed for using agents to enhance Web services or integrating the two approaches. For a detailed comparison between these two concepts see, for example, [49]. [30] points out some drawbacks of Web services which significantly distinguishes them from agents: they know only about themselves, and they do not possess any meta-level awareness; they are not designed to utilize or understand ontologies; and they are not capable of autonomous action, intentional communication, or deliberatively cooperative behaviour. According to us, different kind of approaches may be distinguished in agent-Web service integration:

- *Distinct view of agents and Web services.* Agents are able both to describe their services as Web services and to search/use Web services by using mappings between MAS standards and SOA standards [49, 42, 25, 57]. This approach is based on a gateway or wrapper which transforms one standard into another. As the main approach in agent standardization is FIPA's, this work often only considers FIPA agents and resolves relations between SOA and FIPA standards;

- *Uniform view of agents and Web services.* Agents and Web services are the same entities. All services are Web services and they are all provided by agents (that means that the underpinning program application is an agent-based system) [33, 52];

- *MAS-based service-oriented architecture mechanisms.* MAS to support SOA mechanisms. This approach is not directly interested in agent service-Web service interaction but rather in the use of MAS to enhance SOA mechanisms. For example, [47] discusses the use of agents for Web services selection according to the quality of matching criteria and ratings. This aspect is not detailed;

- *MAS-based high level process of services.* Detailed in section 2.3.2.

---

[6]See for example from 2003 to 2007, the Web Services and Agent Based Engineering (WSABE), Service-Oriented Computing and Agent-Based Engineering (SOCABE), and Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE) international workshops held in conjunction of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).

Distinct view of agents and Web services. In this approach, [49] discusses the difference between agents and Web services from an implementation perspective. Agents communicate via SOAP enabled `startpoint/endpoint` pairs that allow access and remote method invocation on distant objects. [42] identifies two key ideas: (i) agents should be able to publish their services as Web services for the potential use of non-agent clients; (ii) agents should advertise other entities using both a SOA standard registry (e.g., UDDI registry with WSDL entries) and an agent standard registry (e.g., FIPA DF with FIPA SD entries). This is also the case of [25], which proposes a Jade agent architecture, based on a Web Service Integration Gateway Service (WSIGS), that contains several components that operate on internal registries to maintain records of all registered services (both agent services and Web services). [25] takes some of the ideas generated by the Agentcities Web Services Working Group [11] with Web Service Agent Gateway (WSAG). A drawback of this approach lies in the fact it is limited to FIPA compliant agents.[7]

A particularly difficult factor in this approach is communication. The challenge consists of bridging the gap between asynchronous behaviour of agent communication and synchronous behaviour of Web service interactions. For example, in [8] a Web service or an agent plays the role of a gateway that transforms a SOAP call into an ACL message. With WSAG, when a Web service invokes a SOAP call on the WSAG, the gateway transforms this synchronous call into an asynchronous FIPA-ACL message it sends to the agent that provides the service. In the same sense, the WSAG transforms a FIPA-ACL message into a SOAP call but ignores the semantics, which cannot map in SOAP. Actually, this aspect is very important as the biggest risk in ACL - SOAP integration is to reduce high level and semantically rich agent communication to simple request/response semantically poor Web service interaction. As another example, the [25] approach reduces WSIGS-agents interactions to FIPA `request` and `inform` message in order to map with the basic request/answer behaviour of Web services. We further think that one condition to avoid this risk, is that the SOAP - ACL transformation should be done by the agent itself and not by another entity (Web service or agent): the agent and the Web service should be the same as it is the case in the next approach.

Uniform view of agents and Web services. In this approach, [52] claims that in order to integrate agent and Web service technology, components have to be designed, which map between the different mechanisms for service description, service invocation, and service discovery, in both worlds. [52, 33] are specifically interested in mobile agents (agents which have the ability to move from one host to another with their state). They propose respectively a 'Web service engine' architecture which aims to provide bidirectional integration of both technologies and 'mobile Web services' as an integration of Web services and mobile agents in order for Web services to benefit from mobility and mobile agents to benefit from the ability to compose standardized loosely coupled distributed applications.

2.2.2 Agent communication

Simply grouping together several agents is not enough to form a MAS. It is communication between these agents that makes it. Communication allows cooperation and coordination between agents [17, 67]. Communication between cognitive agents is most of the time direct mode asynchronous communication by message passing.[8] Communication in MAS is done by means of Agent Communication Languages (ACLs) such as KQML (Knowledge Query and Manipulation Language) [40] and FIPA-ACL (Foundation for Intelligent Physical Agents - ACL) [1]. ACLs are interesting because they replace ad hoc

---

[7]One of the force of the agent paradigm is the diversity and heterogeneity of agents. Standardization is a strong aspect for Web service interoperation but non-standardization is one of the reason for which MAS are today so different and able to address a large scope of problems. Agents and Web services need each other exactly because they come from different universes (i.e., industry for Web services and academy for MAS).

[8]Direct means that messages are transmitted directly from an agent to the other. Asynchronous means that a message can be buffered and that an agent is not blocked when it sends/answers to a message

communication languages that previously were used in MAS. ACLs allow managing semantics, ontology, rules, protocols etc. of a conversation. Indeed, in agent communication the biggest challenge is conversation modelling. Traditionally, agent conversations are modelled by interaction protocols or conversation policies [29, 14]. These protocols represent the interaction structure and specify rules that must be respected during the conversation. A famous example of interaction protocol is the Contract Net protocol proposed by Smith [12]. However, this approach has weaknesses, especially interoperability, composition and verification of protocols. Agents are forced to follow a policy restricting their autonomy and the dynamic interpretation of messages. The only way for an agent to consider the entire conversation is to look at the protocol, which was previously determined (before the conversation) and which cannot change dynamically. By contrast to interaction protocols, other approaches, for example [55, 46, 37], try to model dynamic conversations by explaining that the next answer message of a conversation can not be foreseen, and should be determined only after the interpretation of the previous incoming message. Instead of modelling mental states and their dynamics in conversation, these approaches deal directly with speech acts rather than on hidden intentions.

2.3 GRID-MAS

2.3.1 Status of current integration activities

There is an increasing amount of research activity in GRID and MAS convergence taking place.[9] Using MAS principles to improve core GRID functionalities (e.g., directory services, scheduling, brokering services, task allocation, dynamic resource allocation and load balancing) is a very active topic.

The Control of Agent-Based Systems (CoABS) project [44], proposed in 1999 by DARPA, is the first research initiative in GRID-MAS integration. In this project, priority was given to GRID development, but the participants already envisage a combination of GRID and MAS domains. In [44] the authors raise several questions to characterize the 'Agent Grid' as a construct to support the rapid and dynamic configuration and creation of new functionality from existing software components and systems. The use of agents for GRID was very early suggested also by Rana and Moreau in [53]: 'By their ability to adapt to the prevailing circumstances, agents will provide services that are very dynamic and robust, and therefore suitable for a Grid environment.' The authors specifically detail how agents can provide a useful abstraction at the Computational Grid layer and enhance resource and service discovery, negotiation, registries, etc. MAS has also been established in 2001 as a key element of the Semantic Grid [56]. In this research agenda, De Roure et al. consider a service-oriented view of the Grid where service users and providers may be considered as agents. In particular, they insist on the important role that agents may play for implementing e-Science marketplaces i.e., the environment in which service users and providers interact one another. In 2004, Foster et al. wrote a paper explaining why MAS and GRID need each other as brain and brawn [20]. From our analysis, we may distinguish two major domains in which GRID-MAS integration occurs:

MAS-based GRID for resource management. Resource management is one of the central components of wide-area distributed computing systems like GRID. It is a core functionality of GRID infrastructure (e.g., GRAM). It is not a trivial task, since the nature of the GRID environment is hardly predictable. Agents may be used for an effective management of the vast amount of resources that are made available within a GRID environment as they have, for example, excellent trading and negotiation abilities (negotiation between resource agents and allocator agent). Another example could be the use of machine learning algorithms to allow agents to allocate tasks, and to learn each time a previous allocation turns out to be a good/bad one. We may cite some important projects that have investigated this

---

[9]See, for example, 2001-2007 Agent-Based Cluster and Grid Computing workshops, and 2005-2006 Smart Grid Technologies workshops, the Multi-Agent and Grid System journal.

aspect:

- The CoABS project mentioned before. The major added value of CoABS Agent Grid is coordination and seamless integration of the available distributed resources (including heterogeneous MAS, object based systems, legacy systems, etc.);

- The AgentScape project [65] provides a multi-agent infrastructure that can be employed to integrate and coordinate distributed resources in a computational Grid environment. AgentScape is specially concerned with scalability i.e., wide-scale or Internet scale (based on the fact that peer-to-peer interaction strategies, as embraced by MAS, seems to be the promising approach for scalability);

- The Agent based Resource Management System (ARMS) project [9] had started an implementation of an agent-based resource management for GRID in which an agent system bridges the gap between Grid users and resources in order to efficiently schedule applications that require Grid resources. The idea of ARMS was to consider that each agent acts as a representative for a local Grid resource.

In using agent techniques for GRID resource management we can also refer: [58] proposes to use an agent oriented negotiation method (an interaction protocol, an auction model or a game theory based model) to enhance load balancing. This improves resource management by dynamically mapping user agents and resource agents. Other examples are [66, 63]. [23] considers a system consisting of large number of heterogeneous reinforcement learning agents that share common resources for their computational needs. [41] provides a price-directed proportional resource allocation algorithm. [45] presents an agent-based resource allocation model for GRID using three types of agents (job, resource brokering and resource monitoring agents). More recently, [5] presents an adaptive holonic multi-agent system for resource discovery.

MAS-based GRID for VO management. Another domain where agent abilities are used to enhance core GRID functionalities is VO management i.e., formation, operation and dissolution of VOs. The main work in this domain is the Grid-enabled Constraint-Oriented Negotiation in an Open Information Services Environment (CONOISE-G) project. It seeks to support robust and resilient VO formation and operation, and aims to provide mechanisms to assure effective operation of agent-based VOs in the face of disruptive and potentially malicious entities in dynamic, open and competitive environments [50]. This project allows a VO manager to find service providers thanks to yellow pages, check service quality thanks to a QoS consultant, add/remove service provider in the VO, etc. Each of these roles are played by agents interacting one-another.

Other related work. [24] describes an approach, called the Mobile Agents Team System (MATS), to dynamic distributed parallel processing using a mobile agent-based infrastructure. In MATS, large computations are initiated under control of a coordinating agent that distributes the computation over the available resources by sending mobile agents to these resources. [64] presents an architecture based on mobile agents for monitoring services in GRID. [49] applies and situates its reflection on the comparison of agents and Web services in the context of bioinformatics Grid.

Our vision of integration. However, none of this work proposes a real integration of MAS and GRID. Rather, they focus on how MAS and AI techniques may enhance core GRID functionalities. Our vision of a GRID-MAS integration is not a simple interoperation of the technologies. It goes beyond a simple use of one technology to enhance the other. Integration refers to the idea of putting diverse concepts

together to create an integrated whole. This contrasts with interoperation, which refers to making services work together by sharing the appropriate messages and using narrow, agreed-upon, interfaces, but without any single conceptual integration [61]. There is currently no GRID-MAS integrated model in the literature. In order to be able to benefit from the most relevant aspects of both GRID and MAS, we propose one centred on the concept of service.

2.3.2 GRID-MAS analogies

a. Agent communication vs. BPM and service interaction.

Message passing based communication. GRID and MAS use the same communication principles: direct message passing-based communication, as it was originally suggested by Hewitt [28].[10] MAS communication is also based on, direct or indirect, message passing.[11] In message passing based communication, an interface (more or less standardised) hides the specific implementation detail, and private representations of the Web/Grid service or agent.

Orchestration and choreography of services vs. interaction protocol and agent conversation. Workflow or service orchestration [51] is analogous to interaction protocol in agent communication [29]. Both terms describe a common interaction structure that specifies a set of intermediate states in the communication process as well as the transitions between these states. The applicability of MAS to workflow enactment has been noted by [60]. [8] explores the relation between Web services, MAS, and workflows. The authors note that, traditionally in workflow approaches, strict adherence to prescribed workflows implies that systems are largely unable to adapt effectively to unforeseen circumstances. They explain why workflow have some weaknesses and how decentralized, multi-agent workflow-enactment techniques can bridge the gap to dynamic service composition. Their vision is to create adaptive workflow capability through decentralized workflow enactment mechanisms that combine Web service and agent technologies. The authors make a strict comparison between workflow (in BPEL4WS) and interaction protocol (as FIPA has defined them).

Workflows prescribe exactly what can be done by each of the participants at any moment in time. The participants, therefore, do not need to understand the whole workflow. They can be implemented as simple reactive agents. Therefore, orchestrating services is very important, but the real added value by agents is more in choreography as agents dispose of good abilities to be engaged within conversation.

Conversation or service choreography is also analogous to agent conversation. Conversations are long-lived high-level interactions which need a peer-to-peer, proactive, dynamic and loosely coupled mode of interaction. Using agent conversations to enhance service exchange is an active research topic [43, 3, 27]. These papers explain that the engagement of Web services in long lived conversations is necessary to provide better adapted and smart services. Conversations allow leveraging Web services from passive components to active ones. Obviously, Web services have to turn to agent, and agent communication models, as the best sophisticated approach for managing conversations:

- [3] suggests using a dialogue agent conversation modelling approach. The authors do not explain how to represent Web services by agents, but propose a conversational model (speech act inspired) that is not based on a diagram of messages to be sent (i.e., interaction protocol)

---

[10] In SOA, methods of the services are not directly invoked by users, but when messages are received, a service provider decides how to proceed by interpreting the user's message and invoking itself the appropriate method(s). This is an important difference with distributed software component approaches.

[11] Always asynchronous for MAS and both synchronous and asynchronous for Grid services.

but rather on local operation calls: the interaction is modelled as a sequence of turns where one of the peers requires that the other peer performs an operation. The service provider has to maintain a set of context-explicit interaction contexts, corresponding to each of its users. [3] ideas are very close to the ones adopted by the STROBE model hereafter described.

- In [43, 27] the formalisms used to concretely express conversations are some kind of Finite State Machines and/or Petri Nets. Exactly the same formalisms that agent communication uses in the interaction protocols based conversation approach.

- With the same idea, [43] proposes to use conversation schema (akin to interaction protocol) represented by state charts (akin to Finite State Machine) to model conversation in order to compose Web services. They suggest a 'context aware' approach representing all the possible states of a Web service, a conversation, a composite Web service.

- BPEL4WS, WSCL, WSCI, etc. only support orchestration or choreography at a syntactic level. They are too low level (implementation focused) to support reasoning at a conceptual and semantic level. Therefore, the Semantic Web service community recently started to address high level process of services at a semantic level [15, 7] but unfortunately without considering yet the great ability of agents to deal with this semantics during conversation. In particular, [15] proposes a formal definition of choreography based on Finite State Machine in the Internet Reasoning Service (IRS)[12] however the relation with MAS is not established.

- More recently, we may also cite [48], which proposed an approach based on agents to control the executions of business processes via agent behaviour rules.

It is important to understand, from this little overview, that the question of modelling interaction in MAS exists also in SOC. Moreover, the challenge of having dynamic agent conversation (i.e., dialogues not described by prefixed interaction protocols) occurs also in SOC with the question of dynamic high level process of services. One of the objectives of DSG is to solve this challenging problem.

b. Agent autonomy and intelligence vs. stateful and dynamic Grid service. 'Intelligent agents' means that they are able to keep their own internal states and make them evolve in a way not necessarily dependent on the messages they receive, but for example, with machine-learning algorithms. In an analogous manner, Grid services are stateful, i.e., they own their running context, where the contextual state memory is stored. An analogy can also be made between an agent having a conversation dedicating a context (i.e., a part of its state) to the conversation and a Grid service factory which instantiates a new service instance with its own state. This idea, more detailed in section 4.2, is fundamental in the GRID-MAS integrated model.

'Autonomous agents' means that they are able to manage these states and their own resources alone, without the intervention of another entity (agent, process, etc.). This is analogous to lifetime management which allows Grid services to be dynamic and to manage by themselves the resources allocated to them.

c. Organizational structure. As the number of entities in the system increases, as the number of problems to be tackled grows, and as the system becomes more and more distributed, different perspectives appear on how to define the system behaviour. Instead of being centred on how each of these entities should behave, one may alternatively adopt the perspective of organizations. An organization is a rule-based partitioned structure in which actions can be taken (problem solving, interaction, etc.)

---

[12]The IRS is a framework and platform for developing Semantic Web Services which utilizes the WSMO ontology (www.wsmo.org)

by people playing roles and sharing one or more goals. Both GRID and MAS choose an organizational perspective in their descriptions.

In Organization Centred MAS (OCMAS) [68, 18], the concepts of organizations, groups, roles, or communities play an important role. They describe the social relation of agents without imposing some mental state representation or convention, but simply by expressing external rules that structure the society. In particular, [18] presents the main drawbacks of agent-centred MAS and proposes a very concise and minimal OCMAS model called Agent-Group-Role (AGR). We will base our GRID-MAS integrated model on this simple but very expressive model, summarized in table 1.

Table 1: Organizational-structure analogies between GRID and MAS

MAS	GRID
Agent	Grid user
An agent is an active, communicating entity playing roles and delegating tasks within groups. An agent may be a member of several groups, and may hold multiple roles (in different groups).	A Grid user is an active, communicating entity providing and using services within a VO. A Grid user may be a member of multiple VOs, and may provide or use several services (in different VOs).
Group	VO
A group is a set of (one or several) agents sharing some common characteristics and/or goals. A group is used as a context for a pattern of activities and for partitioning organizations. Two agents may communicate only if they are members of the same group. An agent transforms some of its capabilities into roles (abstract representation of functional positions) when it integrates into a group.	A VO is a set of (one or several) Grid users sharing some common objectives. A VO and the associated service container is used as a context for executing services and for partitioning the entire community of Grid users. Two Grid users may exchange (provide/use) services only if they are members of the same VO. A Grid user publishes some of its capabilities into services when it integrates into a VO.
Role	Service
The role is the abstract representation of a functional position of an agent in a group. A role is defined within a group structure. An agent may play several roles in several groups. Roles are local to groups, and a role must be requested by an agent. A role may be played by several agents.	The service is the abstract representation of a functional position of a Grid user in a VO. A service is accessible via the CAS service. A Grid user may provide or use several services in several VOs. Services are local to VOs (situated in the associate container), and a Grid user must be allowed to provide or use services in a VO. A service may be provided by several Grid users.

3   AGIL: a GRID-MAS integrated systems description language

This section defines progressively each AGIL's concept, relations between these concepts and rules. Appendix B summarizes both AGIL's concepts and relations. Notice that key GRID concepts presented in this section have been established according to the OGSA or WSRF specifications. Similarly, key MAS concepts have been established by different approaches in the MAS literature, especially the STROBE model [10, 38, 36]. As we are focussing on concepts, we adopt the most convenient terminology from these sets of specifications.

3.1 Grid resource representations

The Grid is a resource-sharing system. The two elementary physical resource types shared in the Grid are *computing resources* and *storage resources*. These are classes of resources. Example of instances of these classes of resources are clusters, simple PCs, processors farms, network storage, data bases, servers, etc.[13] Let Ω be the set of computing resources (ω) and let Θ be the set of storage resources (θ). Grid resources are represented in AGIL by circles and squares as figures 1 and 2 show.

Figure 1: Computing resource representation

Figure 2: Storage resource representation

However, the Grid does not directly deals with these elementary elements but with association of different types of elementary resources called *hosts*. A host is an abstraction used by GRID to see a Grid resource in a unique and standard way. To simplify this abstraction we can consider a host as either a physical association (resource-coupling) between a computing resource and a storage resource, called a *single host* or an association of several hosts (host-coupling) coupled together, called a *coupled host* (notice a coupled host can be part of another coupled host.). Let H be the set of hosts (h); a single host is a pair (θ, ω), $\theta \in \Theta$, $\omega \in \Omega$. Hosts are represented in AGIL by triangles as figure 3 shows.

Figure 3: Host representation

The *resource-coupling* relation formalizes the relation between pairs of resources and hosts. The *resource-coupling* relation is represented in AGIL as figure 4 shows. Any[14] pair of storage and computing resource is coupled by zero or exactly one single host. Any host couples at most one pair of resource.

$$resource - coupling : \Theta \times \Omega \to H \text{ (function)}$$

Figure 4: *Resource-coupling* relation representation (single host)

---

[13]Grid users are sometime considered as resources. We address that later in the paper.

[14]In order to precise AGIL's relation cardinalities, we will always use the article 'any' to identify all the elements of a set, and we will specified if an element is in relation with zero, (at least, at most, exactly) one or several elements of the other set. We add a rule each time the type of relation (function, application, surjection, etc.) doest not includes the restriction.

Rule 1 *A pair of storage and computing resource is coupled by a unique host.*
$\forall \theta \in \Theta, \forall \omega \in \Omega, \forall h_1, h_2 \in H, resource - coupling(\theta, \omega) = h_1 \wedge resource - coupling(\theta, \omega) = h_2 \Rightarrow h_1 = h_2$

Rule 2 *A host couples one unique pair of storage and computing resource.*
$\forall \theta_1, \theta_2 \in \Theta, \forall \omega_1, \omega_2 \in \Omega, \forall h \in H, resource - coupling(\theta_1, \omega_1) = h \wedge resource - coupling(\theta_2, \omega_2) = h \Rightarrow \theta_1 = \theta_2 \wedge \omega_1 = \omega_2$

The *host-coupling* relation formalizes the relation between hosts. The *host-coupling* relation is represented in AGIL as figure 5 shows. Any host is coupled by zero or exactly one coupled host. Any host couples either zero or at least two hosts.

$$host - coupling : H \rightarrow H \text{ (function)}$$

Figure 5: *Host-coupling* relation representation (coupled host)

Rule 3 *A host is coupled by a unique coupled host.*
$\forall h_1, h_2, h_3 \in H, host - coupling(h_1) = h_2 \wedge host - coupling(h_1) = h_3 \Rightarrow h_2 = h_3$

Rule 4 *A coupled host couples at least two hosts.*
$\forall h_1, h_2 \in H, host - coupling(h_1) = h_2 \Rightarrow \exists h_3 \in H, host - coupling(h_3) = h_2 \wedge h_1 \neq h_3$

Rule 5 *The host-coupling relation is irreflexive i.e., a coupled host does not couples itself.*
$\forall h_1, h_2 \in H, host - coupling(h_1) = h_2 \Rightarrow h_1 \neq h_2$

Rule 6 *The host-coupling relation is asymmetric i.e., a coupled host that couples a host, cannot be coupled by this host.*
$\forall h_1, h_2, h_3 \in H, host - coupling(h_1) = h_2 \wedge host - coupling(h_2) = h_3 \Rightarrow h_1 \neq h_3$

3.1.1 Virtualization of resources

The sharing of these resources consists of a two steps process: virtualization and reification. The virtualization consists to accumulate the available resources provided by hosts. The result of the virtualization process is a set, R, of virtualized resources (r). Virtualized resources are not directly represented in AGIL, but a subset of R is represented by a trapeze as figure 6 shows.

Figure 6: Set of virtualized resources representation

The *virtualizing* relation formalizes the relation between hosts and virtualized resources. The *virtualizing* relation is represented in AGIL as figure 7 shows. Any host virtualizes its resources in exactly one subset of virtualized resources (maybe empty). Any subset of virtualized resources is virtualized by zero, one or several hosts.[15]

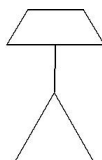$$virtualizing : H \rightarrow \mathcal{P}(R) \text{ (application)}^{16}$$

Figure 7: *Virtualizing* relation representation

3.1.2 Reification of resources

The second step in the sharing of resource is the reification. Virtualized resources are reified in (i.e., re-allocated to) *service containers* (described in section 3.2.3). Let U be the set of service containers (u). The *reifying* relation formalizes the relation between virtualized resources and service containers. The *reifying* relation is represented in AGIL as figure 8 shows. Any service container reifies exactly one subset of virtualized resources (non empty). Any subset of virtualized resources is reified in zero, one or several service containers.

$$reifying : U \rightarrow \mathcal{P}(R) - \{\emptyset\} \text{ (application)}$$
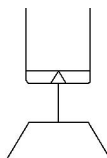
Figure 8: *Reifying* relation representation

Notice that resource virtualization and reification is done at the core GRID level (middleware). The rest of core GRID functionalities (e.g., container, CAS, etc.) described in AGIL is itself described by a single unit: the Grid service.

Formalization of mechanisms. Rigorously formalizing GRID and MAS concepts and their relations, allow us to rigourously express mechanisms occurring in GRID-MAS integrated systems. For example:

Let V_h be the value (e.g., in GBytes and/or GFlops) of the amount of resource of the host h. Let V_r be the value of the amount of resource of a virtualized resource r. Therefore, the amount of resource of a virtualized resource may be expressed as a weighted sum of amount of resources of hosts that virtualize their resources:

---

[15]A good metaphor to understand virtualization is currency conversion: each host uses a different currency to measure its amount of resources. To virtualize means to transform (following a rate of conversion) a part of its amount in a common currency.

[16]In set theory, $\mathcal{P}(S)$ represents the power set of S i.e., the set of all subsets of S.

$$\forall h_i \in H, \forall r \in R, r \in virtualizing(h_i) \Leftrightarrow \exists \alpha_i \in [0,1] \ V_r = \sum_i \alpha_i.V_{h_i}$$

The set of coefficients $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ may be called the *virtualization vector*. For a given subset of virtualized resources, the set of virtualisation vectors form the *virtualization matrix*. This matrix models at a given time all the sharing of resources (who share and how much).

Let V_u be the value of the amount of virtualized resource of the service container u. The amount of virtualized resource of a service container may be expressed as a weighted sum of amount of resources of reified virtualized resources:

$$\forall r_i \in R, \forall u \in U, r_i \in reifying(u) \Leftrightarrow \exists \beta_i \in [0,1] \ V_u = \sum_i \beta_i.V_{r_i}$$

The set of coefficients $beta = (\beta_1, \beta_2, \ldots, \beta_n)$ may be called the *reification vector*. For a given subset of virtualized resources, the set of reification vectors form the *reification matrix*. This matrix models at a given time all the allocation of resources (who consumes and how much). At a given time the amount of virtualized resource reified for a service container can be expressed as a function of the amounts of resource of hosts (not demonstrated here):

$$\forall h_j \in H, \forall r_i \in R, \forall u \in U, r_i \in virtualizing(h_j) \wedge r_i \in reifying(u) \Leftrightarrow$$

$$\exists \alpha_{j_i}, \beta_i \in [0,1] \ V_u = \sum_j \left(\sum_i \beta_i \alpha_{j_i} \right).V_{h_j}$$

3.2 Service instance representations

For SOC an GRID communities, a *service* is an interface of a functionality (or capability) compliant with SOA standards. Let Σ be the set of services (σ). As we will see, community authorization services and service containers are themselves service instances. In order to distinguish them, we call *normal services*, services that are not CAS nor service container. Let S be the set of normal services (s), and let C be the set of CAS (c). Then, $\Sigma = U \cup C \cup S$.

3.2.1 Normal service representations

We may distinguish three kinds of normal services:

- stateless services;

- transient stateful services;

- persistent stateful services.

Services are said to be stateless if they delegate responsibility for the management of the state to another component. Statelessness is desirable because it can enhance reliability and scalability. A stateless service can be restarted following failure without concern for its history of prior interactions, and new copies (instances) of a stateless service can be created (and subsequently destroyed). Stateless service instances are quite restrictive: they are synchronous (i.e., messages can not be buffered and do block the sender or receiver), point-to-point (i.e., used by only one user) and interact via simple one-shot interaction (i.e., request/answer). A stateless service does not establish a conversation. Instead, it returns a result from an invocation, akin to a function.

Stateful services require additional consideration: they have their own running context where is kept the contextual state memory. This state may evolve not simply with external messages (as simple

objects) but also according to autonomous rules. A stateful service has an internal state that persists over multiple interactions. They can be persistent or transient. Transient services are instantiated by a service factory, whereas persistent services are generally created by out-of-band mechanisms such as the initialization of a new service container. Stateful service instances may be multipoint (i.e., used by several users) and may interact by simple one-shot interaction or long-lived conversation. Stateful services may be synchronous or asynchronous.

In AGIL, normal services are represented with a rectangle with round angles as figure 9 shows. If the service is stateful, a lozenge represents its context; if it is transient, a short line represents its lifetime.

stateless
service instance

stateful
service instance

transient stateful
service instance
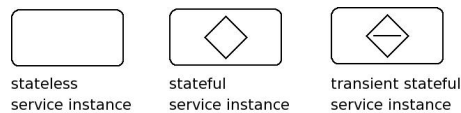
Figure 9: Normal service representations

3.2.2 Community authorization service representation

The relation between a VO (described in section 3.3.1) and a service container is realized by a *Community Authorization Service* (CAS) which formalizes the VO-dedicated policies of service by members. The CAS is the first important element of the Grid security infrastructure represented in AGIL. A CAS is itself a service. The CAS may be viewed as a MxS matrix, where M corresponds to the number of members of the VO, S to the number of currently active services available for the VO, and the matrix nodes are deontic rules. These rules permit the accurate specification of the right level for a member on a service (e.g., permissions, interdictions, restrictions, etc.). This matrix contains at least one column (a VO exists only with at least one user) and one row (the CAS has itself an entry in the CAS). CASs are represented in AGIL as figure 10 shows.
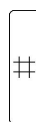

Figure 10: CAS representation

Let O be the set of virtual organizations (o). The *authorizing* relation formalizes the relation between VOs and CASs. The *authorizing* relation is represented in AGIL by a curve thin arrow as figure 11 shows. Any VO is authorized by exactly one CAS (each agent of the VO has an entry as a column in this CAS matrix). Any CAS authorizes exactly one VO.

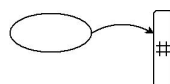$$authorizing : O \rightarrow C \text{ (bijection)}$$

Figure 11: *Authorizing* relation representation

Each service is identified by a *handle*. A running service can be retrieved with its handle. A Grid service handle, as specified in OGSA, or a endPoint reference, as specified in WSRF, provides a unique

pointer that is a URI, to a given service. The *handling* relation formalizes the relation between services and CASs. The *handling* relation is represented in AGIL by a straight thin arrow as figure 12 shows. Any CAS and any normal service are handled by exactly one CAS (each service as a row entry in this CAS matrix). Any CAS handles at least one service (itself).
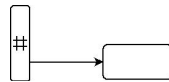
$$handling : S \cup C \to C \text{ (surjection)}$$

Figure 12: *Handling* relation representation

Rule 7 *The handling relation restricted to C is reflexive i.e., a CAS handles itself.*
$\forall c \in C, handling(c) = c$

3.2.3 Service container representation

Services need a hosting environment to exist and to evolve with their own private contexts (i.e., set of resources). This is the role of the *service container* which implements the reification of a portion of the virtualized resources available in a secure and reliable manner. A service container is defined as a triplet composed of a non empty set of virtualized resources, a CAS and a set (maybe empty) of normal services: $U = \{(R_u, c, S_u), R_u \in \mathcal{P}(R) - \{\emptyset\}, c \in C, S_u \in \mathcal{P}(S)\}$. Since a container is a particular kind of service, it is created either through the use of a service factory or by a direct core GRID functionality. Service containers are represented in AGIL as figure 13 shows.

Figure 13: Service container representation

A service container includes several types of services. The *including* relation formalizes the relation between services and service containers. The *including* relation is represented in AGIL as figure 14 shows. Any normal service and any CAS are included in exactly one service container. Any service container includes at least one services (one CAS and zero, one or several normal services).
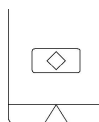
$$including : S \cup C \to U \text{ (surjection)}$$

Figure 14: *Including* relation representation

Rule 8 *The subset of virtualized resources that partially composes a service container is reified for it.*

18

$$\forall u = (R_u, c, S_u) \in U, \forall r \in R_u, r \in reifying(u)$$

Rule 9 *All the services (CAS and normal services) that partially compose a service container are included in this service container.*

$$\forall u = (R_u, c, S_u) \in U, \forall s \in S_u, including(c) = u \wedge including(s) = u$$

Rule 10 *A service is handled by a CAS if and only if they are both included in the same service container.*

$$\forall \sigma \in S \cup C, \forall c \in C, handling(\sigma) = c \Leftrightarrow including(\sigma) = including(c)$$

3.3 Agent representation

In AGIL, the term *agent* is used to uniformly denote artificial agent, human agent and Grid user. In particular, by viewing Grid users as agents, we may consider them as potential artificial entities. An *agent* possesses both intelligent and functional abilities. These are represented respectively by the agent *head* and *body*. The head contains a *brain*, composed of a set of rules and algorithms (e.g., machine learning) that gives to the agent learning and reasoning skills. It also contains the rest of the agent's knowledge, it objectives, and eventually mental states (e.g., BDI). The body contains the agent's capabilities; these capabilities are said to be executed in specific conversation contexts called here Cognitive Environments (CEs) – see section 3.3.3 and 3.4.1. Let A be the set of agents (a), B, the set of brains (b), and E, the set of cognitive environments (e). Then, an agent is composed of a brain and a non empty set of cognitive environments:

$$A = \{(b, E_a), b \in B, E_a \in \mathcal{P}(E) - \{\emptyset\}\}$$

Agents are represented in AGIL by a skittle (figure 15) as it is sometime done in the MAS community [18]. The cylinder represents the body of the agent while the sphere represents the head. As brains of agents are never the same, we graphically represent them by a shapeless form (figure 16).

Figure 15: Agent representation

Figure 16: Brain representation

AGIL is limited to the MAS concepts described in this paper. For example, there is no representation of the environment (i.e., the world surrounding the agents) or objects present in this environment [17].

3.3.1 Virtual organization

Agents are members of *Virtual Organizations* (VO). A virtual organization is a dynamic collection of individuals, institutions and resources bundled together in order to share resources and services as they share common goals. It should be seen as a group of people who share common interests or participate to a common enterprise, and who assemble, collaborate, and communicate in a loose, distant, virtual way, using network communication facilities, tools and resources. A VO may be defined by the set of services that members operate and share. Examples of VOs are: members of the same company, a consortium, the organic chemistry community, etc. The term VO unifies the concept of organization or community in GRID and the concept of group in MAS. Thus we can now talk about 'VO of agents'. A service container is allocated to (and created for) one and only one VO as it is formalized by the *authorizing* relation. A VO is a subset (non empty) of agents: $O \in \mathcal{P}(A) - \{\emptyset\}$. VOs are represented in AGIL by large ellipses as figure 17 shows.

Figure 17: VO representation

The *membershipping* relation formalizes the relation between agents and VOs. The *membershipping* relation is represented in AGIL by a full dot, and a simple line binding skittle and full dot as figure 18 shows. Any agent is a member of zero, one or several VOs. Any VO contains at least one agent.
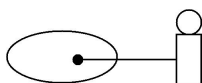
$$membershipping : A \to O \text{ (relation)}$$

Figure 18: *Membershipping* relation representation

$\forall a \in A$, we denote $membershipping(a)$ the set of VOs a is a member of. Notice that as a VO is a subset of agents, the *membershipping* relation is represented in the set-theory by the inclusion ($\in$) relation i.e., $\forall o \in O, \forall a \in A, a \in o \Leftrightarrow o \in membershipping(a)$. In the following, we will prefer the inclusion symbol.

Rule 11 *A VO contains of at least one member.*
$\forall o \in O, \exists a \in A, a \in o$

3.3.2 X509 certificate

The *X509 certificate* is the second important element of the Grid security infrastructure represented in AGIL with the CAS. It allows mutual authentication and delegation. A X509 certificate is valid if it has been signed by a trusted Certification Authority (CA). There are four kinds of X509 certificates: Grid user certificate, Grid host certificate, Grid proxy certificate, CA certificate. Let X be the set of X509 certificates (x). X includes all the types of certificates, except CA certificates (not formalized yet). X509 certificates are represented in AGIL as figure 19 shows.

Figure 19: X509 certificate representation

The *holding* relation formalizes the relation between agents, hosts and X509 certificates. The *holding* relation is represented in AGIL as figure 20 shows. Any X509 certificate is held by either exactly one agent or exactly one host. Any agent holds zero, one or several X509 certificates (i.e., type Grid user and proxy certificate). Any hosts holds at most one X509 certificate (i.e., type Grid host certificate).

$$holding : X \rightarrow A \cup H \text{ (application)}$$
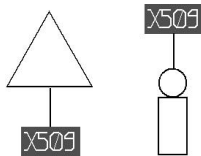
Figure 20: *Holding* relation representation

Rule 12 *A host can hold at most one X509 certificate (type Grid host certificate)*
$$\forall h \in H, \forall x_1, x_2 \in X, holding(x_1) = h \wedge holding(x_2) = h \Rightarrow x_1 = x_2$$

Rule 13 *All agents members of a VO hold a X509 certificate.*
$$\forall a \in A, \forall o \in O, a \in o \Rightarrow \exists x \in X, holding(x) = a$$

Rule 14 *All hosts that virtualize their resources in the Grid hold a X509 certificate.*
$$\forall h \in H, \forall r \in R, r \in virtualizing(h) \Rightarrow \exists x \in X, holding(x) = h$$

3.3.3   Capability representation

The body of an agent is composed of a set of *capabilities* which correspond to the agent's capacity or ability to do something, i.e., to perform some task. They will represent the ability of an agent to provide a service. It is a black-box abstraction that can be stored, named, called with arguments and that returns a result. Capabilities are represented in AGIL by rectangles with a fold back upper right corner as figure 21 shows. Let Λ be the set of capabilities (λ).

Figure 21: Capability representation

A strong element of the GRID-MAS integration consists in viewing services as allocated interfaces of agents' capabilities. A Grid service is seen as the interface of a capability published in a service

container and with allocated resources. An agent has a set of capabilities it may transform into Grid services available in the different VOs it is a member of. The process of 'transforming' or 'publishing' a capability into a service is called the *servicization process*.[17] When a capability is servicized, it means:

- the interfacing of this capability with SOA standards i.e., mainly WSDL;

- the addition (possibly by using an add-service service) of this service to the VO's service container by assigning it a handle and by allocating it private resources;

- the requesting of the VO's CAS service to add an entry for this service (the agent has to decide the users' right levels);

- the publishing of the service description in the VO's registry, if it exists;

- the notification to the VO's members of the VO that a new service is available;

- etc., according to VO or service container local rules.

When an agent servicizes one of its capability into a service available for a VO it uses a set of services of this VO. Each of the previous step of the servicization process is achieved using a specific VO local service (e.g., interfacing, adding, notifications services). This servicization process is not discrete but continuous. Service and capability keep aligned one another. For example, if the capability of the agent changes, then the service changes at the same time. With this viewpoint, an agent can provide different VOs with different services. Notice also that a service is agent-specific, that means that only one agent executes (i.e., provides (section 4.1)) the service in a container. However, it does not prevent another agent of the VO from providing the same type of service. What is important in this servicization process is that it remains the same irrespective of the kind of agent involved. Both AAs and HAs transform their capabilities in the VO's service container modulo different (graphical) interfaces. For example, an AA may servicize its capability to compute square roots (i.e., a function that receives an integer as a parameter and returns a float as result), and a HA may servicize its pattern-recognition capability (i.e., a function that receives an image as a parameter and returns a concept as result (described in an ontology)). Notice that the service and the capability lifetimes are not necessarily the same. Even if a service is transient in a service container the corresponding capability maybe persistent in the agent's body according to its initiative.

Remark – In order to avoid the problem of mapping between SOAP and ACLs (section 2.2.1) we do not consider that there is a message transformation (or any interaction) between the service and the agent's capability; they are the same thing represented differently. Agents are supposed to be able to interpret directly SOAP messages corresponding to the capabilities they servicized in WSDL. For example, the question of semantics is not resolved by the servicization process.

The *interfacing* relation formalizes the relation between capabilities and services. The *interfacing* relation is represented in AGIL by a dotted-line as figure 22 shows.[18] Any service interfaces exactly one capability. Any capability is interfaced by at most one service.

$$interfacing : \Sigma \rightarrow \Lambda \text{ (injection)}$$

---

[17]We can say that as GRID virtualizes resources and reifies them in a service container, an agent virtualizes capabilities and reifies them in a service container.

[18]Notice that figure 22 illustrates the *interfacing* relation with a normal service. The same representation is used with CAS and service container.
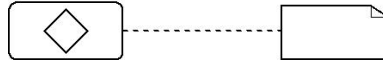
Figure 22: *Interfacing* relation representation

Remark – Grid resources are available for services (i.e., servicized capabilities) execution. The agent itself is still executed autonomously with its own resources and process (e.g., on an agent platform (JADE, MadKit) or somewhere else for mobile agents).[19]

3.4 Conversation context representation

3.4.1 Cognitive Environment

Agents' capabilities may be executed in a particular context called here a *Cognitive Environment* (CE).[20] Basically, these contexts contain the bindings allowing an agent both to execute a function and to interpret a message. CEs are represented in AGIL by folders as figure 23 shows.

Figure 23: CE representation

The *executing* relation formalizes the relation between capabilities and CEs. The *executing* relation is represented in AGIL as figure 24 shows. Any capability is executed in exactly one CE. Any CE executes at least one capability (explain hereafter).

$$executing : \Lambda \rightarrow E \text{ (surjection)}$$

Figure 24: *Executing* relation representation

3.4.2 The STROBE model

We can explore further the concept of cognitive environment, which is a relatively new, but very important, concept related to the STROBE model [10, 38, 36]. STROBE agents are able to interpret communication messages in a given CE that includes an interpreter, dedicated to the current conversation. The key idea consists in enabling an agent to develop a dedicated language for each of its interlocutor or group of interlocutors. The STROBE model, is an agent communication and representation model developed in a DSG perspective. STROBE shows how generic agent conversations may be described and implemented by means of *STReams* of messages to be exchanged by agents represented as *OBjects* exerting control by means of procedures (and continuations) and interpreting messages in multiple *Environments*. The

---

[19]Except if the agent platform, or the agent use an 'allocating resources Grid service' for its private execution! The agent platform may also be deployed on the Grid.

[20]The term environment is here used with its programming language meaning, that is to say, a structure that binds variables and values.

model is highly influenced by applicative/functionnal constructs. STReams, OBjects and Environments are the three programming constructs that support the model. These are Scheme first-class primitives. Actually, Scheme is used in STROBE as a description language (such as lambda calculus or denotational semantics) as well as an implementation language.

Dedicated conversation contexts.   The STROBE model was initially thought to address the question of representing an agent's multi-points of views. How to address the fact that a concept has been defined in different ways by one and another interlocutor? The model proposes a solution based on the reconstruction of the interlocutor model using a dedicated structure for conversations: the cognitive environment. Each STROBE agent has a set of CEs that represents its knowledge and capabilities. Each CE is dedicated to an interlocutor (or group of interlocutors). Actually, these CEs play the role of interlocutor models because they are the conversation contexts in which STROBE agents evaluate messages. The term 'conversation context' is inspired from the term 'execution context' that exists in traditional programming languages and means the context in which an expression of a language (i.e., a program) is evaluated to produce some results. An environment is basically a stateful context in which several functions or procedures are executed. Thus a CE is a set of bindings and as in the pure programming language approach, a binding is an association between a variable and a value. In particular, we call *capabilities* procedure type bindings. Actually, a STROBE agent has two types of CEs:

- One *global CE*, unique and private, which represents the agent own beliefs and contains and executes generic capabilities;

- Several *local CEs*, dedicated to a specific interlocutor or group of interlocutors. They represent both the model of the interlocutor (i.e., others beliefs and dedicated capabilities) and conversation contexts (i.e., a part of the agent state dedicated to others) as explained above. Each time an agent receives a message, it selects the unique corresponding CE dedicated to the message sender in order to interpret the message.

The *owning* relation formalizes the relation between agents, CEs and brains. The *owning* relation is represented in AGIL as figure 25 shows. The *dedicating* relation is not graphically represented. Any CE and any brain are owned by exactly one agent. Any agent owns exactly one brain and at least one CE (the global CE).
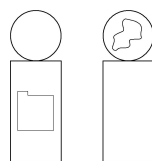
$$owning : E \cup B \to A \text{ (surjection)}$$

Figure 25: *Owning* relation representation

The *dedicating* relation formalizes also the relation between agents and CEs. Any CE is dedicated to exactly one subset (maybe singleton but non empty) of agents. Any subset of agents dedicates zero, one or several CEs.

$$dedicating : E \to \mathcal{P}(A) - \{\emptyset\} \text{ (application)}$$

Rule 15 *An agent owns all the CEs and the brain that form it.*

24

$$\forall a = (b, E_a) \in A, \forall e \in E_a, owning(e) = a \land owning(b) = a$$

Rule 16 *An agent owns a unique brain.*
$$\forall a = (b, E_a) \in A, \forall b' \in B, owning(b') = a \Rightarrow b = b'$$

Rule 17 *An agent owns at least one CE, the global one. This global CE is unique and totally dedicated to this agent.*
$$\forall a = (b, E_a) \in A, \exists! e \in E_a, owning(e) = a \land dedicating(e) = \{a\}$$

The STROBE model presupposes an agent owns one unique local CE for a given interlocutor, i.e., interlocutor messages are always interpreted in the same CE (partially or totally dedicated). It is preferable for DSG to own one and only one conversation context for an interlocutor. This is expressed by rule 18.

Rule 18 *The intersection of the set of agents for which two CEs, owning to the same agent, are dedicated to, is empty.*
$$\forall e_1, e_2 \in E, owning(e_1) = owning(e_2) \Rightarrow dedicating(e_1) \cap dedicating(e_2) = \{\emptyset\}$$

A local CE is said to be *totally* dedicated if the set of agents it is dedicated to is a singleton. If not, a local CE is said to be *partially* dedicated. Therefore, A's global CE is noted E_A^A, A's local CE totally dedicated to B is noted E_B^A and A's local CE partially dedicated to B, C and D is noted $E_{B,C,D}^A$. Figure 26 shows three different STROBE agents A, B and C, with different kinds of CEs. Each agent has a global CE. A and C have a local CE dedicated to each interlocutor, whereas B has only one local CE dedicated to both A and C.
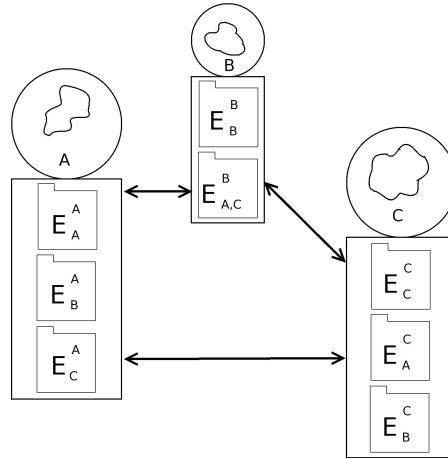

Figure 26: Three STROBE agents communicating one another

4 AGIL dynamics

4.1 Agent interaction

A very important aspect of MAS is interaction. When two agents send and receive messages one another, we say that they communicate or interact. In AGIL, agent-agent interactions include all other kinds of

interactions that may occur in GRID or MAS (Grid user-Grid service, Grid service-Grid service, agent-agent, etc.). These interactions are realized by means of asynchronous message passing between agents. The *interacting* relation formalizes the relation of interaction between agents. The *interacting* relation is represented in AGIL by a double thin arrow as figure 27 shows. Any agent interacts with zero, one or several agents.

$$interacting : A \rightarrow A \text{ (relation)}$$

In the following, $\forall a \in A$, we denote $interacting(a)$ the set of agents which interact with a.
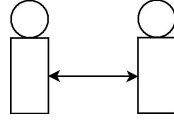
Figure 27: *Interacting* relation representation

Rule 19 *The interacting relation is symmetric.*
$\forall a_1, a_2 \in A, a_1 \in interacting(a_2) \Leftrightarrow a_2 \in interacting(a_1)$

Rule 20 *The interacting relation is irreflexive i.e., an agent does not interact with itself.*
$\forall a \in A, a \notin interacting(a)$

There is two kind of interactions:

- *Direct agent-agent interaction.* Messages are exchanged directly from agent to agent. These are interactions in a general sense, i.e., any interaction, standardized or ad hoc, protocol guided or not, semantically described or not, long-lived or one-shot, etc. These interactions may occur within a VO, but also outside it;

- *Through-service agent-agent interaction.* They occur during service exchange. Messages are exchanged from agent to agent through a service. These are interactions that an agent may have with another agent, without directly communicating with the other agent but instead via the service interface this second agent offers in the VO's service container. These 'through-service interactions' occur only within a VO.

4.1.1 Direct agent-agent interaction representation

It is not the aim of AGIL to model or detail the first type of interaction (that can be totally detached from GRID). This work is a current research domain, called agent communication, addressed by MAS community (section 2.2.2). The only assumption about this relation with STROBE agents is expressed by rule 21.

Rule 21 *Two interacting agents have each one a unique CE dedicated (partially or totally) to the other.*
$\forall a_1 = (b_1, E_1), a_2 = (b_2, E_2) \in A, a_2 \in interacting(a_1) \Leftrightarrow \exists! e_1 \in E_1, \exists! e_2 \in E_2, a_2 \in dedicating(e_1) \wedge a_1 \in dedicating(e_2)$

4.1.2 Through service agent-agent interaction representation

The second type of interaction is on the other hand the main element of the service-based integration of GRID and MAS. Actually, from a pure MAS point of view, we should say that AGIL describes service exchange interactions in MAS; it does it using advantages of GRID mechanisms and principles. We say that agents may all together exchange several services. The *exchanging* relation formalizes the through-service agent-agent interaction. It can be decomposed in two sub-relations, *using* and *providing*, that formalize the relation between agents and services. The *exchanging* relation is not directly represented in AGIL, however, it may be graphically simplified as figure 28 shows. The *using* and *providing* relations may be also simplified in AGIL as figures 29 and 30 show. Any agent exchanges with zero, one or several agents. Any service is used by zero, one or several agents. Any agent uses zero, one, or several services. Any agent provides zero, one or several services. Any service is provided by exactly one agent.

$$exchanging : A \rightarrow A \text{ (relation)}$$

$$using : \Sigma \rightarrow A \text{ (relation)}$$

$$providing : \Sigma \rightarrow A \text{ (application)}$$

In the following, $\forall a \in A$, we denote $exhanging(a)$ the set of agents interacting with a; $\forall \sigma \in \Sigma$, we denote $using(\sigma)$ the set of agents which use the service σ.
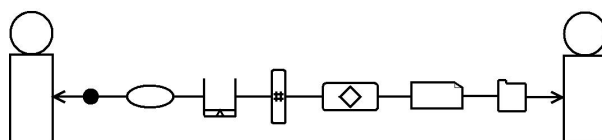

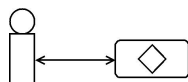Figure 28: *Exchanging* simplified representation
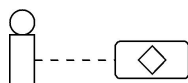

Figure 29: *Using* relation simplified representation

Figure 30: *Providing* relation simplified representation

Rule 22 *The exchanging relation is symmetric.*
$$\forall a_1, a_2 \in A, a_1 \in exchanging(a_2) \Leftrightarrow a_2 \in exchanging(a_1)$$

Rule 23 *The exchanging relation is irreflexive.*
$$\forall a \in A, a \notin exchanging(a)$$

Rule 24 *Exchanging a service is a sub kind of interaction.*
$\forall a_1, a_2 \in A, a_2 \in exchanging(a_1) \Rightarrow a_2 \in interacting(a_1) \wedge a_1 \in interacting(a_2)$

Rule 25 *An agent uses a service only if it is member of a VO for which the service is available.*
$\forall a \in A, \forall \sigma \in S \cup C, a \in using(\sigma) \Rightarrow \exists o \in O, a \in o \wedge authorizing(o) = handling(\sigma)$

The previous rule comes directly from OGSA specification. In OGSA a Grid user (called an agent in AGIL) has some right levels on the services available in his/her VO(s). Using a service occurs only within a VO. In AGIL, we should also detail the rules for service providing, because in the GRID-MAS integrated model, a service available within a VO is provided by an agent. Must this agent be member of the VO for which it provides the service? To answer to this question we should look back on the servicization process presented in section 3.3.3. When an agent servicizes one of its capability into a service available for a VO, it uses a set of services proposed by the VO e.g., add service, interfacing service, notification service, etc. In order to use these services, according to rule 25, it must be a member of the VO. This is expressed by rule 26.

Rule 26 *An agent provides a service within a VO only if it is a member of the VO and it has a capability that interfaces the service in the corresponding service container.*
$\forall a = (b, E_a) \in A, \forall \sigma \in S \cup C, providing(\sigma) = a \Rightarrow \exists \lambda \in \Lambda, \exists o \in O, \exists e \in E_a, a \in o \wedge authorizing(o) = handling(\sigma) \wedge interfacing(\lambda) = \sigma \wedge executing(\lambda) = e$

Therefore, within a VO, service exchanges occur when an agent uses a service another one provides. This is what append for all service exchanges in AGIL. This general case is expressed by rule 27.

Rule 27 *Two agents exchange a service only if they are members of the same VO and if one of them provides, in this VO, the service the other one uses.*
$\forall a_1, a_2 \in A, a_2 \in exchanging(a_1) \Rightarrow \exists o \in O, \exists \sigma \in S \cup C, a_1, a_2 \in o \wedge authorizing(o) = handling(\sigma) \wedge a_1 \in using(\sigma) \wedge providing(\sigma) = a_2$

Rules 25, 26 and 27 specify using, providing and exchanging of a normal service or a CAS. The case of service containers is special because containers are not included in containers.

Rule 28 *An agent uses a service container service only if it is a member of the corresponding VO.*
$\forall a \in A, \forall u \in U, a \in using(u) \Rightarrow \exists c \in C, \exists o \in O, a \in o \wedge authorizing(o) = c \wedge including(c) = u$

4.2 Service-capability instantiation

4.2.1 Mapping of Grid service instantiation and CE instantiation mechanisms

GRID introduces the notions of service factory and service instance. Basically, the service factory is a generator, like a class in object-oriented programming. The service instance is one of the many instances that can be generated by the service factory. Each service instance is running its own dedicated context with its own resource. For this reason this factory-instance model is well appropriated for managing service state. The distinction between a service factory and a service instance is not necessary in AGIL, because even a service factory is a service instance of another factory. What is important to address is then the bootstrap of this system: not all services are created by instantiation mechanism, some of them can have been created by core GRID functionality (out-of-band and bootstrap mechanisms).[21]

---

[21]Notice that even if a service is created by an out-of-band mechanism, it can nevertheless be associated to an agent's capability.

In MAS, when an agent has a conversation, it dedicates a part of its state to this conversation. It is called the conversation context. In the STROBE model, it is done by CEs. When a STROBE agent receives a message from another agent for the first time, it instantiates a new local CE for this agent following three policies: (i) copying the global CE; (ii) copying a local CE; (iii) sharing a local CE.

In the GRID-MAS integration the key GRID idea of service instantiation is mapped with the STROBE's instantiation of CE mechanism. It means that instantiating a new service in GRID is equivalent to instantiating a new CE in MAS; the processes are the same thing viewed differently. The new CE contains the new capability and the service provider applies the servicization process on it in order to make available the new service instance for the service user(s). In order to map exactly the STROBE model with the WSRF specification, we should say that a new CE can be viewed as a new WS-Resource i.e., a CE is a dedicated association between capabilities and stateful resources. Integrating these two instantiation mechanisms make capabilities to benefit from standardization, interoperation and allocated resources from GRID, and Grid services to benefit from a dedicated context of execution and local conversation representation from MAS. The *instantiating* relation formalizes the relation between service-CE couples. The *instantiating* relation is represented in AGIL by a large arrow as figure 31 shows. Any service-CE couples is instantiated by at most one other couple. Any service-CE couple instantiates zero, one or several other couples.

$$instantiating : \Sigma \times E \rightarrow S \times E \text{ (function)}$$
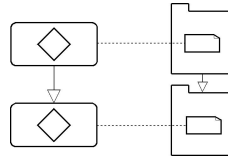

Figure 31: *Instantiating* relation representation

Rule 29 *Only normal services can be service factories.*
$$\forall (\sigma_f, e_1), (\sigma_i, e_2) \in \Sigma \times E, instantiating(\sigma_i, e_2) = (\sigma_f, e_1) \Rightarrow \sigma_f \in S$$

Rule 30 *A service-CE couple can be instantiated only by a unique couple.*
$$\forall (s_1, e_1), (s_2, e_2) \in S \times E, \forall (\sigma_3, e_3) \in \Sigma \times E, instantiating(\sigma_3, e_3) = (s_1, e_1) \wedge instantiating(\sigma_3, e_3) = (s_2, e_2) \Rightarrow s_1 = s_2 \wedge e_1 = e_2$$

Rule 31 *The instantiating relation (restricted to $S \times E$) is irreflexive i.e., a service-CE couple is not instantiated by itself.*
$$\forall (s, e) \in S \times E, instantiating(s, e) \neq (s, e)$$

Rule 32 *The instantiating relation (restricted to $S \times E$) is asymmetric i.e., a service-CE couple that instantiates another couple, cannot be instantiated by this couple.*
$$\forall (s_1, e_1), (s_2, e_2) \in S \times E, instantiating(s_2, e_2) = (s_1, e_1) \Rightarrow instantiating(s_1, e_1) \neq (s_2, e_2)$$

Rule 33 *A service-CE couple instantiates another couple only if the two CEs execute the two capabilities interfaced by the two services.*

$$\forall(s_1, e_1) \in S \times E, \forall(\sigma_2, e_2) \in \Sigma \times E, instantiating(\sigma_2, e_2) = (s_1, e_1) \Rightarrow \exists\lambda_1, \lambda_2 \in \Lambda, executing(\lambda_1) = e_1 \wedge executing(\lambda_2) = e_2 \wedge interfacing(s_1) = \lambda_1 \wedge interfacing(\sigma_2) = \lambda_2$$

Remark – The service factory and the service instance are not necessary provided by the same agent because of the reproducing situations (detailed hereafter).

4.2.2 Different cases of instantiation

Figure 32 shows three kinds of instantiation that GRID enables. These three cases of service instantiation may be commented at the light of the integration:

1. A service factory instantiates a new service instance in the same service container. The classical case.

2. A service factory instantiates a new service container. This is a core GRID situation. It supposes the agent which provides the service container factory is able to access or request GRID middleware to create a new service container, as well as the associated VO (B on figure 32), and to affect the VO's virtualized resources. Similarly, another core GRID situation (not represented in figure 32) is the instantiation of a CAS service instance. Allowing these core GRID functionalities to be realized by agents is a powerful aspect of the AGIL's model. It realizes a part of the MAS-based GRID approaches presented in section 2.3.1.

3. A service factory instantiates a new service instance in another service container, via some authorization provided by the CAS. In this case the service factory provider agent is necessarily member of A and B.

In AGIL, whatever is the situation, the service factory is always a normal service. And the service instance is always an allocated interface of an agent's capability.
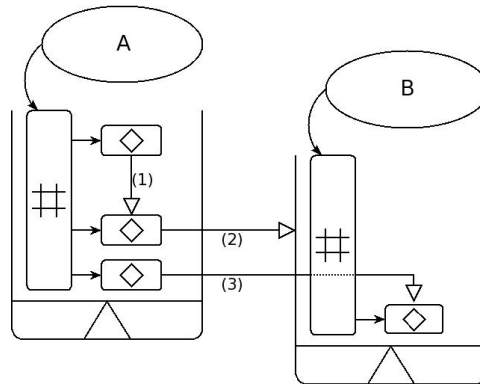

Figure 32: Examples of different kind of instantiations in a Grid service container

Remark – GRID enables also to instantiate services in sequence: a service factory instantiates a new service factory and so on. Sequences of instantiation are still possible in AGIL, because the STROBE's instantiation mechanism enables any kind of CE to instantiate a new one according to the agent policy.[22]

---

[22]Notice however these sequence of instantiations are not ordinary in GRID as well as in MAS.

4.3 Agent reproduction

One of the hypothesis of the STROBE model, in order to enable customized and adapted services, is to consider that an agent has one unique local CE totally or partially dedicated to another agent it is interacting with.[23] This is expressed by rule 18. To respect this hypothesis agents must reproduce themselves if they want to dedicate more than one local CE to an interlocutor. For example, if a service user agent wants two instances of the same service, the service provider agent must reproduce itself. The *reproducing* relation formalizes this relation between agents. The *reproducing* relation is represented in AGIL by a dotted large arrow as figure 33 shows. Any agent is reproduced by at most one parent agent (i.e., not all agents are result of reproduction). Any agent reproduces in zero, one or several children agents.
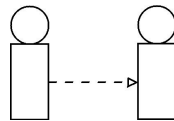
$$reproducing : A \rightarrow A \ \text{(function)}$$

Figure 33: *Reproducing* relation representation

Rule 34 *An agent can have a unique parent.*
$$\forall a_1, a_2, a_3 \in A, reproducing(a_1) = a_2 \land reproducing(a_1) = a_3 \Rightarrow a_2 = a_3$$

With this rule an agent can have zero or one parent. Indeed, some agents are not the result of the *reproducing* relation. They are created directly by the designer of a GRID-MAS integrated system.

Rule 35 *The reproducing relation is irreflexive i.e., an agent is not reproduced by itself.*
$$\forall a \in A, reproducing(a) \neq a$$

Rule 36 *The reproducing relation is asymmetric i.e., an agent that reproduces another one, cannot be reproduced by this one.*
$$\forall a_1, a_2 \in A, reproducing(a_1) = a_2 \Rightarrow reproducing(a_2) \neq a_1$$

When an agent reproduces, it produces a new agent that is exactly the same than itself except considering the set of local CEs (same brain and same global CE). The child agent is created with a local CE dedicated to the interlocutor of the father agent which provokes the reproduction; none of the father local CEs are inherited. Local CEs represent the model of an interlocutor (or group of interlocutor) an agent has. They are the property of agents and cannot be shared. By opposition, the global CE represent the generic knowledge of an agent. Therefore, it seems logic than the global CE should be copied in a child agent, but not the local CEs. The new agent will develop its own local CEs, thanks to his global CE, according to the interaction it has in the future. Indeed, since the child agent is created, it starts evolving alone autonomously: changes its brain, changes its global CE, exchange some services, etc. The father agent cannot instantiate CE in the child body anymore. When created, the child agent is a member of the same VOs than its father. This reproduction situation is illustrated in figure 34.

---

[23]Notice this hypothesis concern only local CE. The situation for the global CE is a little bit specific (for a bootstrapping reason). We will not detail this aspect here, but simply notice services corresponding to capabilities of a global CE are not dedicated.
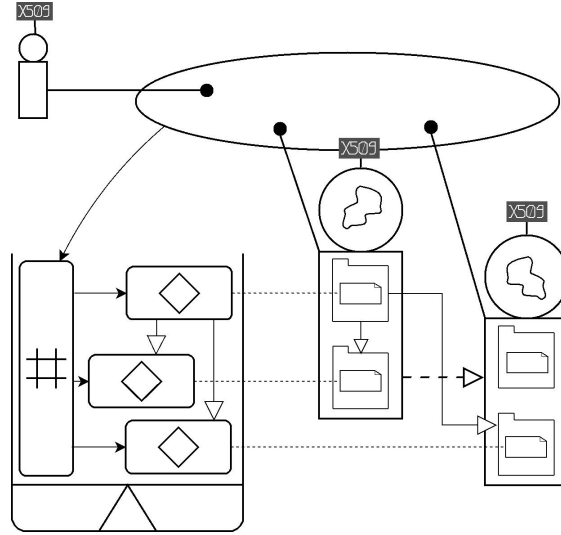
Figure 34: Example of agent reproduction

Rule 37 *A child agent is member of the same VOs than its father.*

$\forall a_1, a_2 \in A, \forall o \in O, reproducing(a_1) = a_2 \wedge a_2 \in o \Rightarrow a_1 \in o$

Rule 38 *An agent reproduces itself only in an interactive situation with another agent. It does it because it already has a local CE dedicated to this interlocutor.*

$\forall a_1 = (b_1, E_1), a_2 = (b_2, E_2) \in A, reproducing(a_2) = a_1 \Rightarrow \exists a_3 \in A, a_3 \in interacting(a_1) \wedge a_3 \in interacting(a_2) \wedge \exists (s_f, e_{1_i}) \in S \times E_1, \exists (\sigma_i, e_{1_j}) \in \Sigma \times E_1, \exists (\sigma_2, e_2) \in \Sigma \times E_2, instantiating(\sigma_{i_1}, e_{1_j}) = (s_f, e_{1_i}) \wedge instantiating(\sigma_2, e_2) = (s_f, e_{1_i}) \wedge a_3 \in dedicating(e_{1_j}) \wedge a_3 \in dedicating(e_2) \wedge providing(s_f) = a_1 \wedge providing(\sigma_2) = a_2$

Remark – We do not detail here the exact mechanism that allows an agent to reproduce. It may be an ad hoc mechanism for different GRID-MAS integrated systems (e.g., how an agent global CE is created, what are is brain modules, etc.) We just formalize here the set of relations that exists in such a situation.

4.4 Summary of the integration

In our integrated model, we consider agents exchanging services through VOs they are members of: both service users and service providers are considered to be agents. They may decide to make available some of their capabilities in a certain VO but not in another. The VO's service container is then used as a service publication/retrieval platform (the semantics may also be situated there). A service is executed by an agent but with resources allocated by the service container. Figure 35 shows the complete integrated model described in AGIL.

5 Discussion, example and comparison

5.1 Discussion about the STROBE model

On one hand, using OGSA specification in a GRID-MAS integration is natural: OGSA is the unique specification of GRID and Grid services. On the other hand, the MAS community proposes many
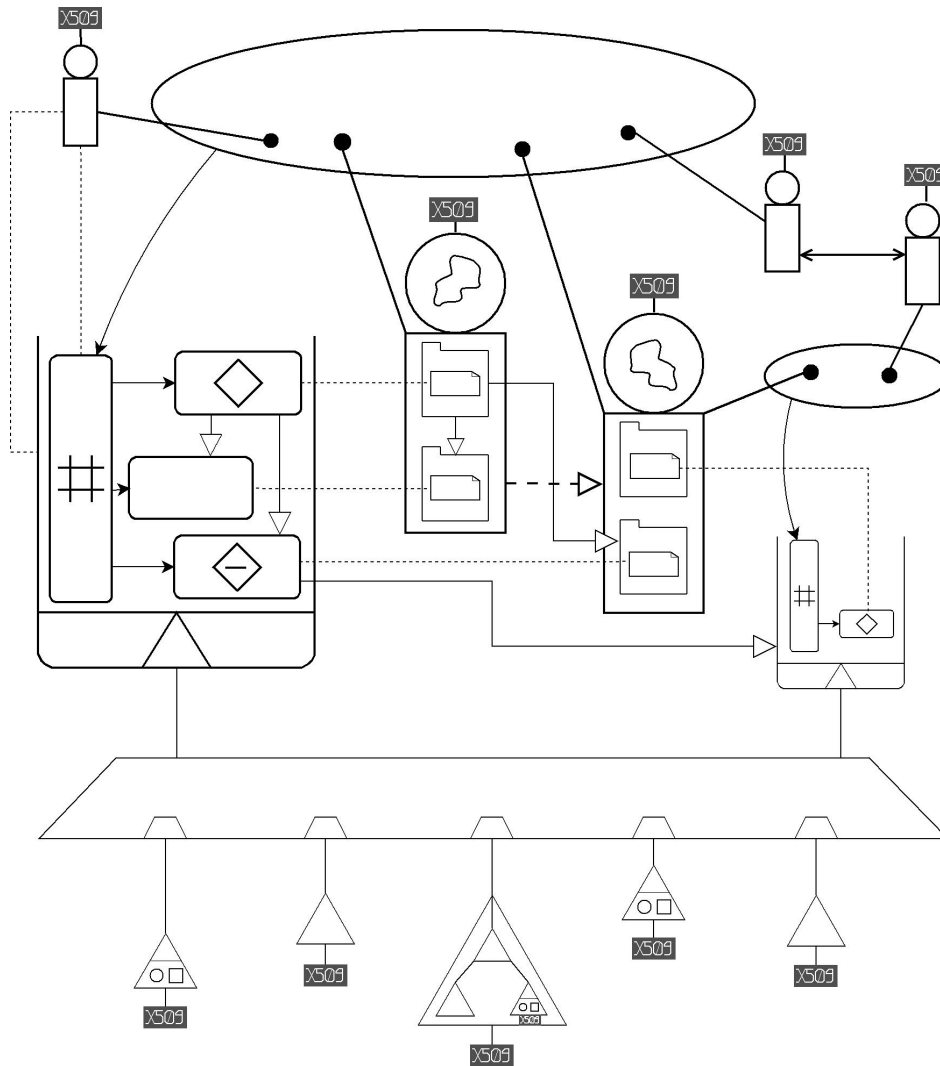
Figure 35: The AGIL's GRID-MAS integrated model

different approaches and models. We may extract two requirements that an agent communication model should provide, with respect to which the STROBE model is compliant:

- To consider the communication effects on interlocutors we must consider that agents can change their goal or beliefs while communicating. Thus, they must be autonomous and should adapt during communication. This adaptation is made through language learning. Indeed, agents need a language to communicate. Thus, assuming as we do, that agents share a minimal common language, what is important is to allow language enrichment.

By representing a CE as a context composed of a pair [environment, interpreter], we can say that they correspond to different languages[24] that an agent develops by interaction with other agents. In the STROBE model, agents are able to interpret messages in a given environment, with a given interpreter both dedicated to the sender of these messages. Communication between agents enables to dynamically

---

[24]Strongly influenced by the functional and applicative programming languages community (e.g., Scheme, LISP), we here assume that a language is basically a pair consisting of a language expression evaluation mechanism and a memory to store this mechanism and abstractions constructed with the language.

change these environments and these interpreters in order to change their way of interpreting messages. We are used to talk about data, control and interpreter levels learning. STROBE agents learn-by-being-told by other agents, and change their representations according to their interlocutors' information given by messages. Some simple concrete experimentations that illustrate the potential of the model are presented in [36]. In particular: (i) meta-level learning by communicating i.e., how an agent can change at run time the way it interprets an interlocutor's messages by re-interpreting its dedicated interpreter; (ii) dynamic specification i.e., how an agent can express one by one its constraints on the way a service is provided by another agent thanks to non-deterministic interpreters. By allowing learning at the three levels STROBE allows agents to dynamically realize the language enrichment of their dedicated languages.

- To tend towards dialogue modelling, agents should only deal with messages from interlocutors without being knowledgeable of their internal beliefs. They should be able to handle the entire conversation dynamically and not simply interpret messages one-by-one following a previously determined structure.

Basically, CEs may be viewed as contexts where messages of conversations are interpreted and capabilities executed to produce results and answer messages. The integration of such programming language constructs in the setting of agent communication is proved correct because they leverage the way agents communicate and adapt during conversations. One of the key ideas of the STROBE model, to address the question of DSG, is to have these CEs dedicated. It means that a STROBE agent can have different representations or capabilities for each agent it communicates with. With the concept of CE multiple environments are simultaneously available within the same agent. These CEs represent the agent's knowledge – for example, the value of a variable, the definition of a procedure – they embody the agent's KB, as they represent a part of the different languages known by an agent (data and control levels). Then CEs address the question of representation of each interlocutor or group of interlocutors. An interlocutor model enables an agent to reconstruct as much as possible the interlocutor's internal state. Interlocutor models allow to free an agent communication model of the assumption of mental state ACLs that mutual beliefs of agents are correct. Moreover, one of the original idea of the model is to represent messages by streams. Agents generate the next message to send only after having received the last partner's answer. This allows to remove the necessity of global conversation planning (i.e. communication protocol) substituting it by history memorization and one-step, opportunistic planning.

In a sake of openness to other approaches in agent modelling, AGIL is not restricted to the STROBE model. In other agent architectures, CE may simply be viewed as a conversation context. Without doing any supposition on agent architecture, we should simply say that an agent's capability is executed in the agent body. We choose here to use the notion of CE because we have proposed STROBE as a model designed and constructed for DSG [36]. The fact of having dedicated capabilities is the basic feature of the STROBE model to implement DSG. To summarize, the choice of the STROBE model for AGIL is justified for three main reasons:

- CEs represent dedicated conversation contexts (i.e., stateful resources) and execute dedicated capabilities-services;

- the STROBE's instantiation of CE mechanism map perfectly Grid service instantiation mechanism (section 4.2);

- STROBE's advantages such as meta-level learning or dynamic-specification are available.

Remark – The simple representation of agent in the STROBE model unifies the artificial agent and human agent representations. Human agents are, of course, autonomous, intelligent and interactive; we

can easily consider that they have a set of capabilities as well as dedicated contexts for conversations. Each human agent has a set of interlocutors and different representations of these ones. He/she develops different languages for each of them. Other human agents do not know what he/she really thinks, they can just deduce facts from messages they received. Each time a human agent interacts with another one he/she changes his/her representations and mental states. Human agents are able to infer, or deduce knowledge from what they know or learn. Of course, this is a simple (and restrictive) human agents representation.

Related work. There is a large number of agent models and architectures in MAS literature. They mainly distinguish by type of knowledge representation and type of control. Most of the time an agent architecture is decomposed in modules (e.g., interaction module, organization module, service execution module, etc.). The main approach in agent architecture is mental states architectures and the well known is the Belief-Desire-Intention (BDI) architecture [54]. We may cite also for example: (i) the Vowels approach [13], according to which an agent is defined following four modules: Agent (A), Environment (E), Interaction (I), Organization (O); (ii) the Advanced Decision Environment for Process Tasks (ADEPT) architecture [35] is one of the first agent architectures that really assesses the abilities of agents for providing service and for being involved in business processes. One of the new aspects is to have a set of modules dedicated to service exchange; (iii) the SMART and *act*SMART approach [4].

Most of the agent architectures are more or less agent centred and they neglect interaction; [59] explained why mentalist architectures are limited compared to social ones. In particular, none of these architectures propose a complete and completely dedicated conversation context. This is the reason why we propose the STROBE model. Actually, the STROBE model is not a specification of a complete architecture, but a specification of some modules, in particular the interaction or communication module and the service execution module viewed as the same thing, a CE. Integrating interaction and service execution in the same (dedicated) module is one of the innovative aspects of the STROBE model in order to implement DSG. Another strong aspect which differs with other architectures is the dedicated aspect of CE. The same concept of putting the conversation contexts at the centre of the agent architecture in which it interprets messages appears also in [3].

5.2 Service adaptation

What is important in this integrated model is to consider how a service may be adapted by a service provider agent for a service user agent, in order to implement DSG. We identify four ways:

1. The service provider agent adapts the service according to its interactions with service user agent;

2. The service provider agent may offer another service to change or adapt the original service;

3. The service provider agent may use dynamic intelligent reflection rules to change the service it is currently providing;

4. Direct agent-agent interactions may occur between the service user agent and the service provider agent and within these interactions (1) and (3) may occur in a pure ad hoc form (not via service).

5.3 Simple example

Figure 36 shows some of the elements of figure 35 but illustrated on a concrete simple example. Let us consider a VO of four users A, B, C and D. D servicized in the VO's service container the `<incr_count_factory>` (S1) and `<decr_count_factory>` (S2) services, corresponding to its capabilities of incrementing and decrementing a given counter. C wants to use alone D's incrementing service; A and B want to use together D's incrementing service and decrementing service. Figure 36

shows both the VO's service container and the D agent's body. Some services and CEs have been instantiated. S1 and S2 correspond to D's capabilities (located in E_D^D), these service factories are accessible by all agents of the VO (`1 1 1 1` line in the CAS[25]) S3 is a service instance accessible only by agent C and thus corresponding to a capability stored in a local CE dedicated totally to C, noted E_C^D. Notice that the local `cc` variable has for value 3, which means the `<incr_count_inst2>` service was used three times by agent C. S4 and S5 are both service instances accessible by A and B and thus corresponding to two capabilities stored in a local CE dedicated to A and B, noted $E_{A,B}^D$. The local `cc` variable has for value 8. Some important remarks may be done on this simple example:

- A and B share the same counter services. It means that if A changes the state (i.e., the counter value) by using the incrementing or decrementing service, it will have an influence on the next use of the services by B;

- A CE instantiation copies all the included capabilities. Therefore, the decrementing capability was created in E_C^D even if D not servicized it as a service in the VO's service container. It may be made later if C asks for a decrementing service;

- S3 and S4, are instances of the same service factory but do not exactly do the same thing. S4 was adapted by D for A and B in order to print the new counter value after incrementing it. S3 stay the same as the service factory. This adaptation may have been done by the four methods cited in section 5.2.
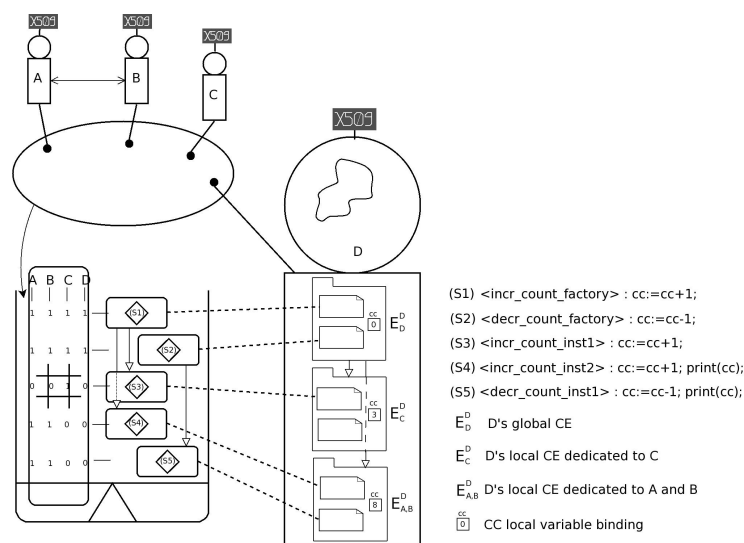

Figure 36: Example of the GRID-MAS integration

Comparison with WSRF. Figure 37 shows the same example but according to WSRF specification. A WS-Resource is an association between a stateful resource and a stateless service instantiated by a WS-Factory. In the example, the functionalities `cc:=cc+1` and `cc:=cc-1` represent the stateless services and the variable `cc` represent the stateful resource. The S1 WS-Factory instantiates two different WS-Resources (S3 and S4) which share the same stateless service but associated with different stateful

---

[25]In a sake of simplicity, on this simple example, only two right levels (levels of permission) are considered: accessible (`1`) or not (`0`).

resources (represented by a cylinder). The S2 WS-Factory instantiates only one WS-Resource (S5) which shares the same stateful resource as S4 but with another stateless service.

Notice that the incrementing service shared by S3 and S4 is unavoidably the same i.e., with WSRF a service may be involved in different WS-Resources, but it has to remain the same service for all the WS-Resources. A WS-Resource has thus only one part that can be really dedicated to its user(s). Integrating WSRF with agents allows to have the stateless service executed by an intelligent agent, which is already a good thing. However, using STROBE agents is actually more interesting because, thanks to the local CEs, a service, that interfaced a capability of a local CE, can be dedicated to the user(s). With the STROBE model, we always have an association between a capability (stateless) and a CE (stateful), several capabilities can share the same CE, but the capability is always dedicated as well as the CE. It is the fundamental difference coming from using STROBE in our GRID-MAS integrated model: a STROBE agent can make its dedicated capabilities evolve differently (following experience, learning algorithm, etc.). Then in our approach S3 and S4 can evolve differently to fit better service user's needs or wants, such as for example print the counter value. This kind of dedication of the stateless service are not possible with WSRF.
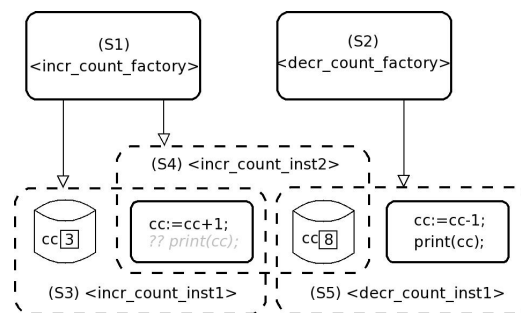

Figure 37: WSRF elements corresponding to the example

5.4 Discussions and benefits for GRID, MAS and SOC

- According to the STROBE agent's policy, when a new local CE is instantiated, all capabilities of the father CE are copied in the child CE; only some of them are servicized. Even if these capabilities are not used yet, they may help the STROBE agent in combining, or composing, its capabilities one another. Later, they may be servicized.

- There is no real standard in the MAS community to describe agents' capabilities between different agents or MAS. The integration with GRID will help MAS developers in presenting and interfacing agents' capabilities using SOA standards (WSDL, etc.), and therefore augment MAS interoperation.

- This integrated model does not restrict MAS or GRID in any way. In particular, it does not prevent direct agent-agent interactions and thus, for example, it does not prevent agents to perform tasks to one another in a purely ad hoc manner. This is important if we want the integration to be followed by other MAS approaches and models (different from STROBE); these models can keep their internal formalisms for their internal operations.

- In this integration, VO management benefits from both GRID and MAS organizational structure formalism, e.g., AGR, CAS service, X509 certificate, etc.

- Service exchange in this integrated model benefits from the important agent communications abilities, e.g., dealing with semantics, ability to hold a conversation, etc. The challenge of modelling conversation not by a fixed structure (interaction protocol) but by a dynamic dialogue in MAS becomes the same one that dynamically create high level process of services in SOC/GRID.

- AGIL subsumes a significant number of the MAS-based GRID approaches cited in section 2.3.1. Indeed, thanks to the reflexivity of GRID, which defines some core GRID functionalities as (meta-)Grid services (e.g., service container, CAS), we may consider these core GRID functionalities as executed also by agents. This establishes an important part of the MAS-based GRID approaches which use MAS techniques to enhance core GRID functionalities.

- The GRID community may appreciate the key GRID concepts formalization done by AGIL. Indeed, without considering the MAS side, AGIL clearly defines and simplifies them even if it is not complete – some GRID concepts are not represented and relations are quite simplified.

AGIL is a set of concepts, relations between them and rules. This rigorous formalization may be useful in order to specify some of the mechanisms occurring in the model (as it is illustrated in section 3.1.2, on the evaluation of the amount of virtualized resource reified for a service container). Moreover, it can also be used to formalize instances of the model and determine if those GRID-MAS integrated systems are consistent with the model. These three elements directly express the semantics of GRID-MAS integrated systems e.g., agent A provides service S will be expressed *providing(S)=A*.

The graphical description language gives to each AGIL concept and relation a graphic symbol that can be used in different diagrams. This language is less powerful than the set-theory formalization, because the rules cannot be expressed graphically. However, it has some important benefits. In particular, it formalizes the model and specifies a reusable and pertinent graphical language i.e., both the model and the instances of this model can be represented by AGIL (this is illustrated respectively by figures 35 and 36). This formalization may have been done with a description language such as Unified Modeling Language(UML), however AGIL is different from these languages because it specifies a set of re-usable graphic symbol and not simply classes and instances. These graphical symbols are an easy way to illustrate and define a GRID-MAS integrated system. It is also very didactic. Besides, using UML would not avoid to express rules independently, because UML class diagram can hardly represent rules such as rules 25, 26 and 27. Nevertheless, UML collaboration or sequence diagrams could be very appropriated for modelling processes and interactions that occur in such GRID-MAS integrated systems.

From a simplified point of view, the concepts of the GRID-MAS integrated model can be summarized by analyzing the interactions in this model. **The AGIL's integrated model may be seen as a formalization of agent interactions for service exchange on the Grid.** The most significant contribution of the model links together all the elements of figure 28: it is the relation introduced between a Grid service instance and its state and an agent's capability and its context (the *interfacing* relation). OGSA fits well for the first part (left hand side) of the interaction and the STROBE model fits well for the second part (right hand side).

Open questions. AGIL does not answer of course all questions yet, for example: (i) should we consider the host-coupling relation a transitive one? (ii) does a coupled host that couples n hosts should hold 1, n or n+1 X509 certificates? (iii) does a service container has an entry in the matrix of the CAS it includes? (iv) an agent providing a service container service should be a member of the corresponding VO or not? (v) does rule 37 have to be limited to the VO in which the agent reproduction occurs? We do not answer yet these question, however answers could be later added as AGIL rules.

6   Conclusion and perspectives

Identifying key factors to demonstrate the convergence of MAS and GRID models is not an easy task. We pointed out that the current state of GRID and MAS research activities is sufficiently mature to enable justifying the exploration of the path towards an integration of the two domains. Besides describing why GRID and MAS need each other, we explained how they can become synergic. At the core of this integration is the concept of service. The bottom-up vision of service in GRID combined with the top-down vision of service in MAS bring forth a richer concept of service, integrating both GRID and MAS properties. In this paper, we put this enhanced concept of service into the perspective of DSG. The main contributions of the paper are:

- Through an analysis of concrete models (mainly OGSA and STROBE) and a survey of integration approaches, we extracted a few key concepts of SOC, GRID and MAS;

- Inspired by the analogies and related work, we proposed an integrated model that respects all the constraints and foundations of GRID and MAS and significantly enhance these domains;

- We proposed a set-theory formalization and a graphical description language, called AGIL, to describe the GRID-MAS integrated model and future GRID-MAS integrated systems;

The integration proposed in this paper is feasible considering today's state of SOC, MAS and GRID technologies. Future developments in GRID and MAS may leverage from this integration in order to progress. For example, here are aspects of the STROBE model and some aspects of the OGSA model that need to evolve:

Better support for more than one interlocutor dedicated for a CE. Most of the scenarios using the STROBE model take as a rule that STROBE agents must have a local CE dedicated to a group of only one interlocutor. However, this viewpoint is not adequate for a GRID-MAS integration because it would imply that each service instance in a service container can be used by one and only one agent. Therefore, the STROBE model needs to evolve to fit better the cases where a local CE is used for a group of interlocutors. This is part of a larger perspective proposing within AGIL a real mapping between organizational structures of GRID and MAS.

Support for non-synchronous protocols and semantics. GRID and SOC must evolve to fit better agent communication properties. In particular, the question of synchronicity (HTTP and thus SOAP are still synchronous communication protocols) and the question of semantics (which starts with the work of the Semantic Web community) need to be addressed.

Service containers as Semantic Web/Grid services platform. In GRID, (as well as in the AGIL's integrated model), the service container is mainly considered as a hosting environment for services. Priority was given to the mechanisms enabling the allocation of specific resources (for a certain time) to services. However, service containers may play a more important role. It could be interesting to consider service containers as semantic platforms enabling the development of Semantic Grid services. This kind of platform already exists for Semantic Web services, such as the Internet Reasoning Service (IRS) [15]. Integrating the work of the Semantic Web community for the development of Semantic Grid service containers is an interesting research perspective.

Following the evolutions of SOC, GRID and MAS, AGIL will have to evolve as well. Even today, a number of perspectives may be highlighted. For example:

- To introduce into AGIL more concepts coming from (i) GRID specification such as certification authority, trust, etc; (ii) MAS approaches when they are considered as standard e.g., environment (surrounding world of an agent) or objects in that environment, etc;

- To add into AGIL new rules, corresponding to non yet answered questions, or to new aspects that GRID-MAS integrated systems will develop;

- To propose AGIL extensions such as the representation of an agent's capability by several services in different service containers e.g., an agent may want to use a given service in two different VOs it is a member of;

- To detail the processes occurring in the AGIL's model e.g., the capability-service instantiation. This could be done by using a Process Description Language (PDL) e.g., BPEL4WS, WSCL, WSCI, or UML collaboration or sequence diagrams;

- To give a better specification of how core GRID functionalities are executed by agents (using work presented in section 2.3.1);

Finally, the GRID and MAS communities, mainly industrial for the former and mainly academic for the latter, have addressed the question of services in distributed systems from completely different angles and have thus developed different complementary aspects. Integrating these aspects according to the guidelines given in this paper seems to us a good way to capitalize past, present and future work in order to simplify the scenarios and use fruitfully the power of distributed services, exchanged among communities of humans and machines.

Acknowledgement

A   Implementations

The AGIL language is currently under specification under the form of an ontology [16] developed with the Protégé Ontology editor (`protege.stanford.edu`). This integrated model is also under current development and testing at LIRMM (`www.lirmm.fr`) within an implementation of STROBE agents within the MadKit multi-agent platform (`www.madkit.org`) developed by LIRMM.

B   AGIL's concepts and relations

References

[1] FIPA-ACL message structure specification. FIPA specifications SC00061G, Foundation for Intelligent Physical Agents, December 2002. www.fipa.org/specs/fipa00061/.

[2] C. Allison, S. A. Cerri, P. Ritrovato, A. Gaeta, and M. Gaeta. Services, semantics and standards: elements of a learning Grid infrastructure. *Applied Artificial Intelligence, Special issue on Learning Grid Services*, 19(9-10):861–879, October-November 2005.

Table 2: AGIL's concepts sum up

AGIL'S CONCEPT	SET	ELEMENT	GRAPHICS
computing resource	Ω	ω	
storage resource	Θ	θ	
host	H	h	
virtualized resource	R	r	nothing
service container	U	u	
normal service	S	s	
CAS	C	c	
service	Σ	σ	nothing
agent	A	a	
virtual organization	O	o	
brain	B	b	
cognitive environment	E	e	
capability	Λ	λ	
X509 certificate	X	x	

[3] L. Ardissono, A. Goy, and G. Petrone. Enabling conversations with Web services. In *2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS'03*, pages 819–826, Melbourne, Australia, July 2003. ACM Press.

[4] R. Ashri, M. Luck, and M. d'Inverno. From SMART to agent systems development. *Engineering Applications of Artificial Intelligence*, 18(2):129–140, March 2005.

[5] M. Bakhouya, J. Gaber, and A. Koukam. Distributed holonic multi-agent system for resource discovery in grids. *Multiagent and Grid Systems*, 2(1):1–9, 2006.

[6] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture. W3C working group note NOTE-ws-arch-20040211, World Wide Web Consortium, February 2004. www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[7] J. J. Bryson, D. Martin, S. A. McIlraith, and L. A. Stein. Agent-based composite services in daml-s: the behavior-oriented design of an intelligent semantic web. In N. Zhong, J. Liu, and Y. Yao, editors, *Web Intelligence*, pages 37–58. Springer, 2003.

[8] P. A. Buhler and J. M. Vidal. Integrating agent services into BPEL4WS defined workflows. In *4th International Workshop on Web-Oriented Software Technologies, IWWOST'04*, Munich, Germany, July 2004.

[9] J. Cao. *Agent-based resource management for Grid computing*. PhD thesis, University of Warwick, Coventry, UK, October 2001.

[10] S. A. Cerri. Shifting the focus from control to communication: the STReam OBjects Environments model of communicating agents. In J. Padget, editor, *Collaboration between Human and Artificial*

Table 3: AGIL's relations sum up

AGIL'S RELATION	DOMAIN	RANGE	TYPE	GRAPHICS
$resource - coupling$	$\Theta \times \Omega$	H	function	
$host - coupling$	H	H	function	
$virtualizing$	H	$\mathcal{P}(R)$	application	
$reifying$	U	$\mathcal{P}(R) - \{\emptyset\}$	application	
$authorizing$	O	C	bijection	
$handling$	$S \cup C$	C	surjection	
$including$	$S \cup C$	U	surjection	
$membershipping$	A	O	relation	
$holding$	X	$A \cup H$	application	
$interfacing$	Σ	Λ	injection	
$executing$	Λ	E	surjection	
$owning$	$E \cup B$	A	surjection	
$dedicating$	E	$\mathcal{P}(A) - \{\emptyset\}$	application	nothing
$interacting$	A	A	relation	
$exchanging$	A	A	relation	
$using$	Σ	A	relation	
$providing$	Σ	A	application	
$instantiating$	$\Sigma \times E$	$S \times E$	function	
$reproducing$	A	A	function	

Societies, Coordination and Agent-Based Distributed Computing, volume 1624 of *Lecture Note in Artificial Intelligence*, pages 74–101. Springer-Verlag, Berlin, Germany, 1999.

[11] J. Dale, A. Hajnal, M. Kernland, and L. Z. Varga. Integrating Web services into agentcities recommendation. Agentcities technical recommendation actf-rec-00006, Agentcities Web Services Working Group, November 2003.

[12] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, January 1983.

[13] Y. Demazeau. From interactions to collective behaviour in agent-based systems. In *1st European Conference on Cognitive Science*, pages 117–132, Saint-Malo, France, April 1995.

[14] F. Dignum and M. Greaves, editors. *Issues in agent communication*, volume 1916 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Germany, 2000.

[15] J. Domingue, L. Cabral, and S. Galizia. Choreography in IRS-III - Coping with heterogeneous interaction patterns in Web services. In *4th International Semantic Web Conference, ISWC'05*, Galway, Ireland, November 2005.

[16] F. Duvert, C. Jonquet, P. Dugénie, and S. A. Cerri. Agent-Grid Integration Ontology. In R. Meersman, Z. Tari, and P. Herrero, editors, *International Workshop on Agents, Web Services and Ontologies Merging, AWeSOMe'06*, volume 4277 of *Lecture Notes in Computer Science*, pages 136–146, Montpellier, France, November 2006.

[17] J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison Wesley Longman, Harlow, UK, 1999.

[18] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: an organizational view of multi-agent systems. In P. Giorgini, J. P. Müller, and J. Odell, editors, *4th International Workshop on Agent-Oriented Software Engineering, AOSE'03*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230, Melbourne, Australia, July 2003. Springer-Verlag.

[19] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. F. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. Modeling stateful resources with Web services. Whitepaper Ver. 1.1, The Globus Alliance, May 2004.

[20] I. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: why Grid and agents need each other. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'04*, volume 1, pages 8–15, New York, NY, USA, July 2004.

[21] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the Grid: an Open Grid Services Architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*. The Globus Alliance, June 2002.

[22] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: enabling scalable virtual organizations. *Supercomputer Applications*, 15(3):200–222, 2001.

[23] A. Galstyan, K. Czajkowski, and K. Lerman. Resource allocation in the Grid using reinforcement learning. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'04*, volume 3, pages 1314–1315, New York, NY, USA, August 2004. IEEE Computer Society.

[24] R. Ghanea-Hercock, J. C. Collis, and D. T. Ndumu. Co-operating mobile agents for distributed parallel processing. In *3rd International Conference on Autonomous Agents*, pages 398–399, Seattle, WA, USA, May 1999. ACM Press.

[25] D. Greenwood and M. Calisti. Engineering Web service - agent integration. In *IEEE Systems, Cybernetics and Man Conference, SMC'04*, The Hague, Netherlands, October 2004. IEEE Computer Society.

[26] R. H. Guttman, A. G. Moukas, and P. Maes. Agent-mediated electronic commerce: a survey. *The Knowledge Engineering Review*, 13(2):147–159, July 1998.

[27] J. E. Hanson, P. Nandi, and D. W. Levine. Conversation-enabled Web services for agents and e-business. In *3rd International Conference on Internet Computing, IC'02*, pages 791–796, Las Vegas, NV, USA, June 2002.

[28] C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323–364, June 1977.

[29] M.-P. Huget, editor. *Communication in multiagent systems, agent communication languages and conversation policies*, volume 2650 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2003.

[30] M. N. Huhns. Agents as Web services. *Internet Computing*, 6(4):93–95, July-August 2002.

[31] M. N. Huhns and M. P. Singh, editors. *Readings in agents*. Morgan Kaufmann, San Francisco, CA, USA, 1998.

[32] M. N. Huhns, M. P. Singh, M. Burstein, K. Decker, E. Durfee, T. Finin, L. Gasser, H. Goradia, N. R. Jennings, K. Lakkaraju, H. Nakashima, V. Parunak, J. S. Rosenschein, A. Ruvinsky, G. Sukthankar, S. Swarup, K. Sycara, M. Tambe, T. Wagner, and L. Zavala. Research directions for service-oriented multiagent systems. *Internet Computing*, 9(6):65–70, November-December 2005.

[33] F. Ishikawa, N. Yoshioka, and Y. Tahara. Toward synthesis of Web services and mobile agents. In *2nd International Workshop on Web Services and Agent Based Engineering, WSABE'04*, pages 48–55, New York, NY, USA, July 2004.

[34] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, April 2001.

[35] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *Applied Artificial Intelligence*, 14(2):145–189, February 2000.

[36] C. Jonquet. *Dynamic Service Generation: Agent interactions for service exchange on the Grid*. PhD thesis, University Montpellier 2, Montpellier, France, November 2006.

[37] C. Jonquet and S. A. Cerri. i-dialogue: modeling agent conversation by streams and lazy evaluation. In *International Lisp Conference, ILC'05*, pages 219–228, Stanford University, CA, USA, June 2005.

[38] C. Jonquet and S. A. Cerri. The STROBE model: Dynamic Service Generation on the Grid. *Applied Artificial Intelligence, Special issue on Learning Grid Services*, 19(9-10):967–1013, October-November 2005.

[39] S. Krishnan, P. Wagstrom, and G. von Laszewski. GSFL: a workflow framework for Grid services. Draft paper, Globus Alliance, July 2002.

[40] Y. Labrou and T. Finin. A proposal for a new KQML specification. Technical report TR-CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland, Baltimore, MD, USA, February 1997. www.cs.umbc.edu/kqml/.

[41] C. Li and L. Li. Competitive proportional resource allocation policy for computational Grid. *Future Generation Computer Systems*, 20(6):1041–1054, August 2004.

[42] M. Lyell, L. Rosen, M. Casagni-Simkins, and D. Norris. On software agents and Web services: usage and design concepts and issues. In *1st International Workshop on Web Services and Agent Based Engineering, WSABE'03*, Melbourne, Australia, July 2003.

[43] Z. Maamar, S. K. Mostéfaoui, and M. Lahkim. Web services composition using software agents and conversations. In D. Benslimane, editor, *Les services Web*, volume 10 of *RSTI-ISI*. Lavoisier, 2005.

[44] F. Manola and C. Thompson. Characterizing the agent Grid. Technical report 990623, Object Services and Consulting, Inc., June 1999.

[45] S. S. Manvi, M. N. Birje, and B. Prasad. An agent-based resource allocation model for computational Grids. *Multiagent and Grid Systems*, 1(1):17–27, 2005.

[46] N. Maudet and B. Chaib-draa. Commitment-based and dialogue-game based protocols - News trends in agent communication language. *The Knowledge Engineering Review*, 17(2):157–179, June 2002.

[47] E. M. Maximilien and M. P. Singh. Agent-based architecture for autonomic Web service selection. In *1st International Workshop on Web Services and Agent Based Engineering, WSABE'03*, Sydney, Australia, July 2003.

[48] H. Mei, X. Sun, X. Liu, W. Jiao, and G. Huang. An agent-based approach to composing web services to support adaptable business processes. *Multiagent and Grid Systems*, 2(4):383–399, 2006.

[49] L. Moreau. Agents for the Grid: a comparison with Web services (part 1: the transport layer). In H. E. Bal, K.-P. Lohr, and A. Reinefeld, editors, *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID'02*, pages 220–228, Berlin, Germany, May 2002. IEEE Computer Society.

[50] 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, and S. Thompson. Agent-based virtual organisations for the Grid. *Multiagent and Grid Systems*, 1(4):237–249, 2005.

[51] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, October 2003.

[52] J. Peters. Integration of mobile agents and Web services. In *1st European Young Researchers Workshop on Service-Oriented Computing, YR-SOC'05*, pages 53–58, Leicester, UK, April 2005. Software Technology Research Laboratory, De Montfort University.

[53] O. F. Rana and L. Moreau. Issues in building agent based computational Grids. In *3rd Workshop of the UK Special Interest Group on Multi-Agent Systems, UKMAS'00*, Oxford, UK, December 2000.

[54] A. S. Rao and M. P. Georgeff. Modelling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484, San Mateo, CA, USA, April 1991. Morgan Kaufmann.

[55] P.-M. Ricordel, S. Pesty, and Y. Demazeau. About conversations between multiple agents. In *1st International Workshop of Central and Eastern Europe on Multi-agent Systems, CEEMAS'99*, pages 203–210, St. Petersburg, Russia, June 1999.

[56] D. D. Roure, N. R. Jennings, and N. Shadbolt. Research agenda for the Semantic Grid: a future e-science infrastructure. Technical report, University of Southampton, UK, June 2001. Report commissioned for EPSRC/DTI Core e-Science Programme.

[57] A. E. F. Seghrouchni, S. Haddad, T. Melitti, and A. Suna. Interopérabilité des systèmes multi-agents à l'aide des services Web. In O. Boissier and Z. Guessoum, editors, *12èmes Journées Francophones sur les Systèmes Multi-Agents, JFSMA'04*, pages 91–104, Paris, France, November 2004. Hermès.

[58] W. Shen, Y. Li, H. H. Ghenniwa, and C. Wang. Adaptive negotiation for agent-based Grid computing. In *1st International Agentcities Workshop on Challenges in Open Agent Environments*, pages 32–36, Bologna, Italy, July 2002.

[59] M. P. Singh. Agent communication languages: rethinking the principles. *Computer*, 31(12):40–47, December 1998.

[60] M. P. Singh and M. N. Huhns. Multiagent systems for workflow. *Intelligent Systems in Accounting, Finance and Management*, 8(2):105–117, June 1999.

[61] M. P. Singh and M. N. Huhns. *Service-Oriented Computing, Semantics, Processes, Agents*. John Wiley & Sons, 2005.

[62] B. Srivastava and J. Koehler. Web service composition - Current solutions and open problems. In *Workshop on Planning for Web Services*, pages 28–35, Trento, Italy, June 2003.

[63] H. Tianfield. Towards agent based Grid resource management. In *5th IEEE International Symposium on Cluster Computing and the Grid, CCGRID'05*, volume 1, pages 590–597, Cardiff, UK, May 2005. IEEE Computer Society.

[64] O. Tomarchio and L. Vita. On the use of mobile code technology for monitoring Grid systems. In *1st International Workshop on Agent-based Cluster and Grid computing*, pages 450–455, Brisbane, Australia, May 2001. IEEE Computer Society.

[65] N. J. Wijngaards, B. J. Overeinder, M. van Steen, and F. M. Brazier. Supporting internet-scale multi-agent systems. *Data & Knowledge Engineering*, 41(2-3):229–245, June 2002.

[66] R. Wolski, J. Brevik, J. S. Plank, and T. Bryan. Grid resource allocation and control using computational economies. In F. Berman, G. Fox, and A. Hey, editors, *Grid Computing: Making The Global Infrastructure a Reality*, pages 747–772. John Wiley & Sons, 2003.

[67] M. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, Chichester, UK, February 2002.

[68] 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, September 2000.