AGIL specifications

Clement Jonquet, Pascal Dugenie and Stefano A. Cerri

May 2006

LIRMM, CNRS & University Montpellier II
161 Rue Ada, 34392 Montpellier Cedex 5, France
{cerri,dugenie,jonquet}@lirmm.fr

Abstract

The GRID and MAS (Multi-Agent Systems) communities believe in the potential of GRID and MAS to enhance each other because 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 to formalize integrated GRID-MAS 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 integrated GRID-MAS 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 Architecture, Web Service, Dynamic Service Generation.

Contents

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. In 2004, [FJK04] explained why MAS and GRID need each other as 'brain and brawn.' Even if using agents for GRID was very early suggested [MT99, RM00], Foster et al. [FJK04] propose the real first step in GRID-MAS integration as it examines 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 set-theory based formalization and a common graphical description language, inspired by both GRID and MAS key concepts.

Service as a unifying concept. The concepts of service and Service Oriented Architectures (SOA) seem good candidates for supporting this integration. 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 [FKNT02]. Quite recently, GRID acquired major importance in SOA 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 [CTT05]. On the other hand, agents are said to be autonomous, intelligent and interactive entities who may use and offers services (in the sense of particular problem-solving capabilities) [Fer99, Jen01]. MAS have also followed naturally the path towards SOA as interest turns to providing dynamic services, business process management, composed services, semantic services, etc. The Service Oriented Computing (SOC) community is also turning to MAS considering the important capacities agents have for providing and using services one another. 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 him (her/it). 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 [SH05] talk about *service engagement*, instead of simple method invocation. In [JC06] the authors extensively describe 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.) and may be met by integrating GRID and MAS complementarities.

GRID-MAS integration. In [JDC06] we present our vision of an integration of GRID and MAS models. We extracted for GRID and MAS related work some key concepts of these domains. These concepts are directly influenced by OGSA (Open Grid Service Architecture) and the STROBE agent communication and representation model[JC05b]. We proposed an integrated model by mapping these key concepts. The integrated model is step-by-step defined in section 3 and 4, however, we sum-up the two main ideas underlying the proposed integrated model:

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

Complementarily, in this paper, we first make a relatively detailed state of the art on related work about integration approaches in GRID, MAS and SOC. Afterwards, we proposes a set-theory based formalization and a common graphical language, called Agent-Grid Integration Language (AGIL), for describing GRID-MAS integrated systems.

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

- defines the respective domain ontologies;[2]
- uses the same terms and representations for an identical concept. For example, Virtual Organization (VO) and group; choreography of service and agent conversation, role and service, etc. The analogies in GRID and MAS concepts are precisely described in section 2.3.2;
- rigorously fixes the integration rule;
- 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 to both GRID and MAS;
- would promulgate the development of GRID-MAS integrated systems.

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

[1] Nowadays Web services (cf. section 2.1) are most of the time object oriented program (developed with J2EE or .NET) processed to produce a WSDL description and to become 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 passing, etc. We do not want Grid/Web services to be executed by simple object program but by intelligent, autonomous and interactive (the three fundamental properties distinguishing agents from objects) agents able to have conversations in order to realise DSG

[2] GRID needs a description language that summarizes and rigorously explains GRID concepts. Without considering the agent side, AGIL may play also this role.

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. Section 3 presents AGIL main concepts, relations and rules. Section 4 details the dynamics of such a model: agent interaction, service-capability instantiation, agent reproduction, etc. Section 5 sums up the integrated model using AGIL. Benefits for MAS, GRID and SOC is also discussed. Finally, Section 6 concludes the paper.

2 State of the art of integration approaches

Considering the reader is aware with basic concepts of SOC, GRID and MAS, we focus in this section just on these domains intersections.

2.1 GRID-SOC

The GRID aims to enable *flexible, secure, coordinated resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organization* [FKT01]. 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 that was 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`). Web services allow distributed functionalities on a network to be accessed in a standardized way to enable interoperability. 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, and WSCL (Web Services Conversation Language) on the conversation/choreography side [Pel03]. 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 characteristics in the so-called SOAs 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 SOC standards. The decision taken between the GGF and the 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) [FKNT02], on the top layer (architecture);
- the Web Services Resource Framework (WSRF) [FFG+04], 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 enables to ensure an efficient load-balancing of the distributed ressources. The GRID specifications main elements are:

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

[3] Also called service dynamicity.

- 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 allow authentification, delegation, privacy and integrity (cf. section 3.3.2).

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 acts upon stateful resources*. These services are modelled by an association between two entities: a stateless Web service, that do 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 5.2 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. For a recent precise overview of Grid service concepts and standardization, see for example [CTT05].

GRID evolution. Recently, the same semantic level add-on objective that took place in the Semantic Web community occurred also in Grid service. For an introduction to the concept of 'Semantic Grid,' see [RJS01, Gel04] or [CT03] for GRID-based distributed knowledge discovery. The GRID is also used as a 'Learning Grid' where GRID architecture and principles are used to change e-learning paradigms [ACR+05]. BPM and Grid service composition have become research topics in the GRID community; for example [KWvL02] 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 [HS98, Fer99, Woo02]. 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;
- 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 learn and reason to evolve;
- Deal with semantics associated to concepts by processing ontologies.

[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.

2.2.1 Agents and Web service

The research activity in MAS and SOC convergence is increasingly active.[6] As [SH05] 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." [GMM98] 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.

Today, most of the time, Web services are an interface for object-oriented programs, and one of the crucial evolutions toward DSG concerns the substitution by an agent-oriented kernel of the current object-oriented kernel for Web services. For Huhns et al. [HSB+05] 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, [Mor02]. [Huh02] 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 [LRCSN03, GC04]. 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) [Mor02, IYT04, Pet05].

- *MAS-based service oriented architectures.* MAS to support SOA. This approach is not directly interested in agent service-Web service interaction but rather in the use of MAS to enhance SOAs. For example, [MS03] discusses the use of agents for Web services selection according to the quality of matching criteria and ratings. Not detailed in this paper.

- *MAS-based Business Process Management.* Detailed in section 2.3.2.

Distinct view of agents and Web services. In this approach, [Mor02] discuss the difference between agents and Web services from an implementation perspective. It refers to the architecture of SOFAR (Southampton Framework for Agent Research), in which agents communicate via SOAP enabled *startpoint/endpoint* pairs that allow access and remote method invocation on distant objects. [LRCSN03] 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 [GC04], which proposes a Jade agent based on a Web Service Integration Gateway Service architecture that contains several components that operate on internal registries to maintain records of all registered services (both agent services and Web services). [GC04] takes some of the ideas generated by the Agentcities Web Services Working Group [DHKV03] 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 communications and synchronous behaviour of Web services interactions. For example, in [BV04] a Web service or an agent plays the role of a gateway that transforms a SOAP call into

[6] See for example the International Workshops on Web Services and Agent Based Engineering (WSABE) held in conjunction of the Autonomous Agents and Multi-Agent Systems International Conference in 2003 and 2004).

[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).

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. 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, [Pet05] 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. [Pet05, IYT04] 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 of mobility and mobile agents to benefit from the ability to compose standardised 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 [Fer99, Woo02]. 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 (ACL) such as KQML (Knowledge Query and Manipulation Language) [LF97] and FIPA-ACL (Foundation for Intelligent Physical Agents - ACL) [Fip02]. ACLs are interesting because they replace ad hoc 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 [Hug03, DG00]. 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 [DS83]. 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 [RPD99, MCd02, JC05a], 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.

Modelling dynamic agent conversation, sometime called dialogue, is still a research challenge in the MAS community. As section 2.3.2 explains, the SOC community has an equivalent challenge included in DSG: modelling dynamic business process and service composition.

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 performances/functionalities (e.g., directory services, scheduling, brokering services, task allocation, dynamic resource allocation and load balancing) is a very active topic in the MAS community.

[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

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

The CoABS (Control of Agent-Based Systems) project [MT99], proposed in 1999 by the 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 [MT99] the authors raise a lot of 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 et Moreau in [RM00]: "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." They specifically detail how agents can provide a useful abstraction at the Computational Grid layer and enhance resource and service discovery, negotiation, registries, etc. We may distinguish two major domains in which GRID-MAS integration occurs:

MAS-based GRID for resource management. For example, resource management is one of the central components of wide-area distributed computing systems like GRID. It is a core functionality of the GRID infrastructure. 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 agent allocating task, 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 aspect:

- The CoABS project mention before. The Major value added 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 [WOvSB02] provides a multi-agent infrastructure that can be employed to integrate and coordinate distributed resources in a computational GRID environment. AgentScape is specially concerned by 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 ARMS (Agent Based Resource Management System) project [CSJN05] 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 may also refer: [SLGW02] proposes to use an agent oriented negotiation method (an interaction protocol, an auction model or a game theory based model) to enhanced load balancing. This improves resource management by dynamically mapping user agents and resource agents. Other examples are [WBPB03, Tia05]. [GCL04] considers a system consisting of large number of heterogeneous reinforcement learning agents that share common resources for their computational needs. [LL04] provides a price-directed proportional resource allocation algorithm. [MBP05] presents an agent-based resource allocation model for GRID using three types of agents (job, resource brokering and resource monitoring agents).

MAS-based GRID for Virtual Organization management. Another domain where agent abilities are used is VO management, i.e., formation, operation and dissolution of VOs. The main work is this domain is the CONOISE-G project (Grid-enabled Constraint-Oriented Negotiation in an Open Information Services Environment). 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 [NPC+04, PTJ+05]. 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 works. We may also cite some other related work about agent and GRID: [GHCN99] 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. [TV01] presents an architecture for monitoring services in GRID based on mobile agents. [Mor02] apply and situate its reflection on the the comparison of agents and Web services to 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 goes beyond a simple use of one technology to enhance the other. We aim to adopt a common approach for the integration to be able to benefit from the most relevant aspects of both GRID and MAS. This common approach is centred on the concept of service. One of the crucial explorations concerns the substitution by an agent-oriented kernel of the current object-oriented kernel of services available in GRID.

2.3.2 GRID-MAS analogies

Through these states of the art, we have identified three GRID-MAS analogies that have strongly influenced our integrated model:

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 [Hew77].[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 [Pel03] is analogous to interaction protocol in agent communication [Hug03]. 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 [SH99]. [BVV03] and [BV04] explore 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. In particular, [BVV03] makes 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 value added by agents is more in choreography as agents dispose of good abilities to be engaged within conversation.

Conversation or service choreography is 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 scenarios was proposed in [MML05, AGP03, HNL02]. These papers explain that the engagement of Web services in long lived conversations is necessary to provide better adapted and smart services. Obviously, Web services have to turn to agent, and agent communication models to, as the best sophisticated approach for managing conversations:

- [AGP03] suggests using a dialogic agent communication 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) 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. [AGP03]'s ideas are very close to the ones adopted by the STROBE model [Cer99, JC05b] developed after.
- In [MML05, HNL02] 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.

[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 from distributed software component approaches.

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

- With the same idea, [MML05] proposes to use conversation schema (akin of interaction protocol) represented by state charts (akin of 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 Business Process Management at a semantic level [DCG05, BMMS03] but unfortunately without considering yet the great ability of agents to deal with this semantics during conversation. In particular, [DCG05] 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.

It is important to understand, from this little overview, that the question of modelling interaction in MAS is the same in SOC. Moreover, the challenge of having dynamic agent conversation (i.e., dialogues not described by prefixed interaction protocols), as presented in section 2.2.2 occurs also in SOC. To reach this challenge is one of the objective of DSG.

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, 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 our integrated model.

'Autonomous agents' means that they are able to manage their own resources. This is analogous to Grid service lifetime management which allows them 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's 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.) by people playing roles and sharing one or more goals. Both GRID and MAS choose an organizational perspective in their descriptions.

In OCMAS (Organization Centred MAS) [WJK00, FGM03], 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 structures the society. In particular, [FGM03] presents the main drawbacks of agent-centred MAS and proposes a very concise and minimal OCMAS model called AGR (for Agent-Group-Role). We will base our integration on this simple but very expressive model, summarized in Table 1.

3 AGIL: an integrated GRID-MAS systems description language

This section defines progressively each AGIL concept, relation between these concepts and rules. Each time a concept is presented, its representation in the graphical description language is presented, and related rules are described using a set theory based formalization. Notice that key GRID concepts presented in this section have been established by the OGSA or WSRF specifications. Similarly, key MAS concepts have been established by different MAS approaches, especially [JC05b, FGM03]. As we are focussing on concepts, we adopt the most convenient terminology from these sets of specifications.

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

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.

3.1 Grid resource representations

Grid resources are of two kinds. *Computing resources* and *storage resources*. Let Ω, the set of computing resources (ω) and Θ, the set of storage resources (θ). Grid resources are represented in AGIL by circles and squares as Figure 1 shows.

Figure 1: Computing resource and storage resource representations

GRID is a resource-sharing system. Grid resources are contributed by *hosts*. A host is either a physical association between a computing resource and a storage resource, called a *single host* or an association of several host coupled together, called a *host-coupling*. Let H, 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 2 shows.

Figure 2: Host representation

The *coupling* relation formalizes the relation between pairs of resources and hosts as well as relation between hosts. The *coupling* relation is represented in AGIL as Figure 3 shows. A pair (θ, ω) of resource forms one unique single host. A resource θ or ω is part of one unique single host. A host is part of a unique host-coupling. A host-coupling may be part of another host-coupling.

$$coupling : \Theta \times \Omega \rightarrow H \text{ (relation)}$$

$$coupling : H \rightarrow H \text{ (relation)}$$

Figure 3: *Coupling* relation representation (representation of single host and host-coupling)

3.1.1 Virtualization of resources

The sharing of these resources consist of a process in two steps: 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 sub-set of R are represented by a trapeze. The *virtualizing* relation formalizes the relation between hosts and virtualized resources. As there is no representation of virtualized resources, we graphically represent in AGIL the *virtualizing* relation between host and sub-set of virtualized resources (factorization), as Figure 4 shows. Hosts virtualize their resources in several virtualized resources. A virtualized resource is the result of virtualization of several hosts.[13]

$$virtualising : H \rightarrow R \text{ (relation)}$$

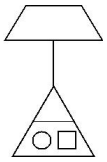

Figure 4: *Virtualizing* relation representation

Let V_h the value (in GBytes and/or GFlops) of the amount of resource of the host h. Let V_r the value of the amount of resource of a virtualized resource r.

Rule 1 *The amount of resource of a virtualized resource is a pondered sum of amount of resources of hosts that virtualize their resources.*

$$\forall i = 1..n, virtualizing(h_i) = r \Leftrightarrow \exists \alpha_i \in [0,1] \;\; V_r = \sum_{i=1}^{n} \alpha_i . V_{h_i}$$

The set of coefficients $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ is called the *virtualization vector*. For a given sub-set 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).

3.1.2 Reification of resources

The second step in the sharing of resource is the reification. Resources are reified in *service containers* (described in section 3.2.4). Let U, 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 5 shows. A part of virtualized resources may be reified in one or several service container. A service container is (partially) composed of a sub-set of virtualized resources.

$$reifying : R \rightarrow U \text{ (relation)}$$

[13] 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.

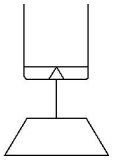

Figure 5: *Reifying* relation representation

Let V_u the value of the amount of virtualized resource of the service container u.

Rule 2 *The amount of virtualized resource of a service container is a pondered sum of amount of resources of reified virtualized resources.*

$$\forall i = 1..n, reifying(r_i) = u \Leftrightarrow \exists \beta_i \in [0,1] \;\; V_u = \sum_{i=1}^{n} \beta_i . V_{r_i}$$

The set of coefficients $beta = (\beta_1, \beta_2, \ldots, \beta_n)$ is called the *reification vector*. For a given sub-set of virtualized resources, the set of reification vectors form the *reification matrix*. This matrix models at a given time the all the affectation of resources (who consume and how much).

Rule 3 *At a given time the amount of virtualized resource reified for of a service container can be expressed in function of the amounts of resource of hosts (not demonstrated here).*

$$\forall i, j = 1..n, virtualizing(h_j) = r_i \wedge reifying(r_i) = u \Leftrightarrow \exists \alpha_{j_i} \beta_i \in [0,1] \;\; V_u = \sum_{j=1}^{n} \left(\sum_{i=1}^{n} \beta_i \alpha_{j_i} \right) . V_{h_j}$$

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

3.2 Service instance representations

A *service* is a compliant with SOA standards interface of a functionality. Let Σ, be the set of services (σ). As we will see later, Community Authorization Service (CAS) and service container are themselves service instances. In order to distinguish them, we call *normal services*, services that are not CAS or service container. Let S, the set of normal services (s), C, the set of CAS (c), and U, the set of service containers (u). 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 Web service can be restarted following failure without concern for its history of prior interactions, and new copies (instances) of a stateless Web service can be created (and subsequently destroyed). Stateless services 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, much like a function.

Stateful services require additional consideration: They have their own running context where is kept the contextual state memory. This state evolve not simply with external messages (as simple objects) but also according to autonomous rules.their own state. 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 created by out-of-band mechanisms such as the initialization of a new service container. Stateful services 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 6 shows. If the service is stateful, a lozenge represents its context; if it is transient, a short line represent its lifetime.

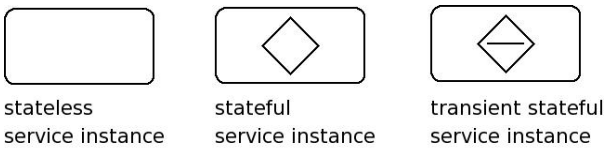

Figure 6: Normal service representations

3.2.2 Community Authorization Service representation

The relation between a VO (let O, the set of virtual organizations (o), sub-set of agents, 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 [PWF+02]. 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, and the matrix nodes are deontic rules. These rules permit the accurate specification of the right levels 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 7 shows.

Figure 7: CAS representation

3.2.3 Handling relation

Each service is identified by a *handle*. A running service can be retrieved with its handle. A Grid Service Handle (GSH) (as specified in OGSA) or endPoint reference (as specified in WSRF) provides a unique pointer that is a URI, to a given service. The *handling* relation formalizes both the relation between VOs and CASs and between services and CASs. The *handling* relation is represented in AGIL by a thin arrow as Figure 8 shows. A VO handles a unique CAS (each agent of the VO have an entry as a column in a CAS matrix) and vice and versa. Moreover, each CAS or normal service is handled by a unique CAS (each service as a row entry in a CAS matrix).

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

$$handling : S \cup C \rightarrow C \text{ (application)}$$

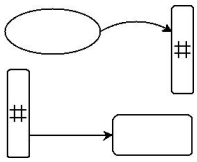

Figure 8: *Handling* relation representation

Rule 4 *A CAS is handled by itself.*
$\forall c \in C, handling(c) = c$

3.2.4 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 resource available in a secure and reliable manner. Let U, the set of service containers (u). 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: $U = \{(r, c, s), r \in \mathcal{P}(R) - \{\emptyset\}, c \in C, s \in \mathcal{P}(S)\}$. 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 mechanism. Service containers are represented in AGIL as Figure 9 shows.

Figure 9: 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 10 shows. All normal services and CAS are included in a unique service container. A service container includes several services.

$$including : S \cup C \rightarrow U \text{ (surjection)}$$

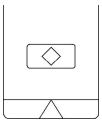

Figure 10: *Including* relation representation

Rule 5 *The set of virtualized resources that partially compose a service container are reified for it.*
$\forall u = (r, c, s) \in U \Rightarrow \forall r_i \in r, reifying(r_i) = u$

Rule 6 *All the services (CAS and normal services) that partially compose a service container are included in this service container.*
$\forall u = (r, c, s) \in U \Rightarrow \forall s_i \in s, including(c) = u \wedge including(s_i) = u$

Rule 7 *A service is handled by a CAS if and only if they are both included in the same service container.*
$\forall \sigma \in S \cup C, c \in C, handling(\sigma) = c \Leftrightarrow including(\sigma) = including(c)$

3.3 Agent representation

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 give to the agent learning and reasoning skills. It also contains the agent knowledge, objectives, and mental states (e.g., BDI). The body contains the agent's capabilities; these capabilities are said to be executed in specific conversation context called Cognitive Environment (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: $A = \{(b, e), b \in B, e \in \mathcal{P}(E) - \{\emptyset\}\}$. Agents are represented in AGIL by a skittle (Figure 11). 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.

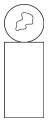

Figure 11: Agent representation

Remark — Agents are currently the best metaphors for humans in computing. HA are, of course, autonomous, intelligent and interactive; we can consider that they have a set of capabilities as well as a dedicated context for conversations. Of course, this is a simple (and restrictive) HA representation.

3.3.1 Virtual organization

Agents are members of *Virtual Organizations* (VO). A virtual organization is a dynamic collection of individuals, institutions and resources sharing common goals, bundled together in order to share resources and services. A service container is allocated to (and created for) one and only one VO. 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.' An agent may be *member* of several VOs, and a VO contains at least one agent. Let O, the set of virtual organizations (o). A VO is a sub-set (non empty) of agents: $O \in \mathcal{P}(A) - \{\emptyset\}$. VOs are represented in AGIL by large ellipses as Figure 12 shows.

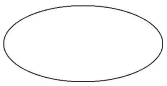

Figure 12: VO representation

The relation *membershiping* formalizes the relation between agents and VOs. Each agent may be a member of several VOs. The *membershiping* relation is represented in AGIL by a full dot, and a simple line binding skittle and full dot(s) as Figure 13 shows.[14]

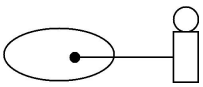

Figure 13: *Membershiping* relation representation

3.3.2 X509 certificate

The *X509 certificate* is the second important element of the GRID security infrastructure represented in AGIL. It allows mutual authentication and delegation. A X509 certificate is valid if it has been signed by a trusted Certification Authority (CA). Let X, the set of X509 certificate (x). There are four kinds of X509 certificates: Grid user certificate, Grid host certificate, Grid proxy certificate, CA certificate. X509 certificates are represented in AGIL as Figure 14 shows.

Figure 14: X509 certificate representation

The *holding* relation formalizes the relation between agents, Grid resources and X509 certificates. The *holding* relation is represented in AGIL as Figure 15 shows. All X509 certificates are held by agents or hosts.[15] Not all

[14] Notice that as a VO is a sub-set of agents, the *membershiping* relation is intrinsically represented in set-theory by the inclusion ($\in$) relation.
[15] We do not consider CA certificates in the holding relation because CAs are not represented in AGIL.

agents or hosts hold X509. A host holds a unique Grid resource certificate whereas an agent may hold several Grid user certificates or proxy certificates.

$$holding : X \rightarrow A \cup H - \{\text{CA certificates}\} \text{ (application)}$$

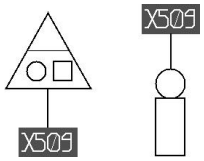

Figure 15: *Holding* relation representation

Rule 8 *All agents members of a VO hold a X509 certificate.*
$\forall a \in A, o \in O, a \in o \Rightarrow \exists x \in X, holding(x) = a$

Rule 9 *All hosts that virtualize their resources in GRID hold a X509 certificate.*
$\forall h \in H, r \in R, virtualising(h) = r \Rightarrow \exists x \in X, holding(x) = h$

Entities with a X509 certificate are sometimes called a Grid *node*. These set of nodes and authorities form Grid *trusts*. Grid node and Grid trust are not represented in AGIL.

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. It is a first-class black-box abstraction that can be stored, named, called with arguments and that return a result. Let Λ, the set of capacities (λ). Capabilities are represented in AGIL by rectangles with a fold back upper right corner as Figure 16 shows.

Figure 16: Capability representation

A strong element of the GRID-MAS integration consist in viewing services as allocated interfaces of agents' capabilities. A service is viewed 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 services available in the different VOs it is a member of. The process of 'transforming' a capability into a service is called the *servicization process*.[16] 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 service container by assigning it a handle and by allocating it private resources;
- the requesting of the VO 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 registry, if it exists;
- the notification to the VO members of the VO that a new service is available;
- etc., according to VO local rules.

[16] 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.

This servicization process is not discrete but continuous. For example, if the capability of the agent changes then the service changes at the same time. Consider also that the service's right levels in the VO CAS or service allocated resources may evolve through time. With this viewpoint, an agent can provide different services in different VOs. Notice also that a service is agent-specific, that means that only one agent can execute the service in a container. However, this 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 AA and HA transform their capabilities in the VO service container modulo a graphical interface. For example, an AA may publish its capability to compute square roots (i.e., a function that receives a number as a parameter and returns a float as result), and a HA may publish 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 CE lifetime is not necessarily the same. Even if a service is transient the corresponding capability maybe persistent according the the owner's agent initiative.

Remark: In order to avoid the problem of mapping between SOAP and ACLs (cf. section 2.2.1) we do not consider that there is a transformation of a message (or of any interaction) between the service and the agent 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.

The *interfacing* relation formalizes the relation between capabilities and services. The *interfacing* relation is represented in AGIL by a dotted-line as Figure 17 shows. Each service is an (allocated) interface of a unique agent capability. All agent capabilities are not systematically interfaced as services.

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

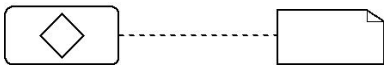

Figure 17: *Interfacing* relation representation

3.4 Conversation context representation

3.4.1 Cognitive Environment

Agents' capabilities may be executed in a particular context called a *Cognitive Environment* (CE). An agent may have several Cognitive Environments[17] which correspond to the different languages it develops by interaction with other agents. CEs are represented in AGIL by folders as Figure 18 shows.

Figure 18: CE representation

The *executing* relation formalizes the relation between capabilities and CEs. The *executing* relation is represented in AGIL as Figure 19 shows. Each agent capability is executed in an unique CE. A CE is a context of execution of several capacities.

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

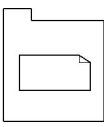

Figure 19: *Executing* relation representation

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

3.4.2 The STROBE model

We can explore further the concept of *Cognitive Environment* (CE), which is a relatively new, but very important, concept related to the STROBE agent and communication model [Cer99, JC05b]. Conversations and their states are represented in the STROBE model by CEs. STROBE agents are able to interpret communication messages in a given CE that includes an interpreter, dedicated to the current conversation. We showed in [JC05b] how communication enables dynamic changes in values in a CE and especially how these interpreters can dynamically adapt their way of interpreting messages (meta-level learning by communication). By representing a CE as a context composed of a pair [environment, interpreter], we can say that they correspond to different languages[18] that an agent develops by interaction with other agents. The same concept of putting the communication contexts at the centre of the agent architecture in which it interprets messages appears also in [AGP03], which assumes that each agent in service exchanges may separately maintain its own internal context of the conversation state.

Actually, a STROBE agent has two types of CEs: i) A *global* one, unique and private, which belongs to the agent and represents its own beliefs and generic capabilities; ii) Some *local* ones, dedicated to a specific interlocutor or group of interlocutors. 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 *owing* and *dedicating* relations formalize the relation between agents and CEs. The *owning* relation is represented in AGIL as Figure 20 shows. The *dedicating* relation cannot be graphically easily represented. A CE is owned by an unique agent. An agent is owner of several CEs (at least one, the global one). A CE is dedicated to a subset (maybe singleton but non empty) of agents. Not all the subset of agents have a dedicated CE.

$$owning : E \rightarrow A \text{ (surjection)}$$

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

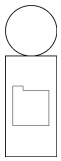

Figure 20: *Owning* relation representation

A CE is said to be 'totally' dedicated if the set of agents it is dedicated to is a singleton. If not a CE is said to be 'partially' dedicated. A convention is adopted for CE denotation:

$E_A^A \Leftrightarrow owning(E) = A \wedge dedicating(E) = \{A\} \Leftrightarrow$ A global E
$E_B^A \Leftrightarrow owning(E) = A \wedge dedicating(E) = \{B\} \Leftrightarrow$ A local E totally dedicated to B
$E_{B,C,D}^A \Leftrightarrow owning(E) = A \wedge dedicating(E) = \{B, C, D\} \Leftrightarrow$ A local E partially dedicated to B, C and D

Figure 21 shows three different STROBE agents A, B and C, with different kinds of CEs. Each agent has a global CE, noted $E_X^X \forall X \in \{A, B, C\}$. A and C have a local CE dedicated to each interlocutor, noted respectively E_B^A, E_C^A and E_A^C,E_B^C. Whereas B has one only local CE dedicated to both A and C, noted $E_{A,C}^B$.

Rule 10 *An agent is owner of all CEs that form it.*
$\forall a = (b, e) \in A, e_i \in e, owning(e_i) = a$

Rule 11 *The global CE is unique.*
$\forall a \in A, \exists! e \in E, owning(e) = a \wedge dedicating(e) = \{a\}$

Rule 12 *The global CE is totally dedicated to the agent itself.*

[18] 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.

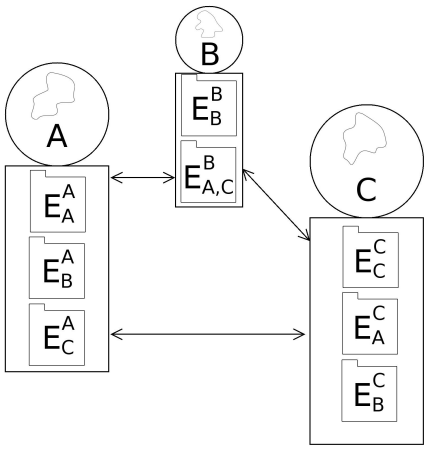

Figure 21: Three STROBE agents communicating one another

$\forall a_1, a_2 \in A, e \in E, owning(e) = a_1 \wedge a_2 \in dedicating(e) \Rightarrow a_1 \notin dedicating(e)$

The STROBE model presupposes an agent to have 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 have one and only one conversation context for an interlocutor. This is expressed by rule 13.

Rule 13 *The intersection of the set of agents for which two CE, owning to the same agent, are dedicated to, is empty.*
$\forall e_1, e_2 \in E, owning(e_1) = owning(e_2) \Rightarrow dedicating(e_1) \cap dedicating(e_2) = \{\emptyset\}$

Advantage of the STROBE model. In one hand, using OGSA specifications 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 different approaches and models. This section justifies why STROBE is convenient. It consists of three main reasons:

- CEs represent dedicated conversation contexts;
- Capabilities (and thus services) are dedicated;
- Meta-level learning by communicating.

In the STROBE model, generic communication may be described 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*. STROBE agent capabilities are located in CE i.e., an agent has specific capabilities for each agent it communicates with. Capabilities are said to by dedicated. The fact of having dedicated capabilities is the basic feature of the STROBE model to implement DSG. The aim of the model is to put at the centre of the agent architecture the conversation contexts in which it interprets messages from its interlocutors. The main idea is to develop a communication language for each agent interlocutor or group of interlocutors.

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 capability is executed in the agent body. We choose here to use the notion of CE because we previously proposed STROBE as a model designed and constructed for DSG (meta-level learning, dynamic specification, stream-based conversation, etc.) [JC05b]. Moreover, section 4.2 explains why the idea of instantiation of CE is very adapted for service instantiation.

4 AGIL dynamics

Section 3 presented AGIL static concepts. This section deals more with AGIL dynamics i.e., the specification of the dynamic behaviour of the systems AGIL aims to describe. In particular, dynamics of service exchanges.

4.1 Agent interaction

A very important aspect of MAS is interaction: agents may interact together. In AGIL, agent-agent interactions include all other kinds of interactions (Grid user-Grid service, Grid service-Grid service, agent-agent, etc.). The *interacting* relation formalizes this relation between agents.

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

The *interacting* relation is represented in AGIL by a double thin arrow as Figure 22 shows. 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 interactions, 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. 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 via the service interface this second agent offers in the VO service container. These 'service through interactions' occur only within a VO.

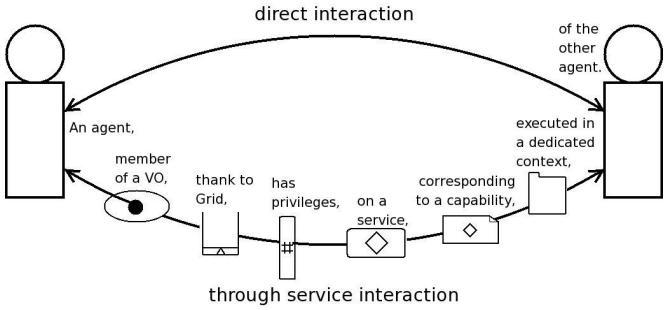

Figure 22: *Interacting* relation representation

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 (cf. section 2.2.2). However, we just mention a rule related to the STROBE model concerning Cognitive Environments.

Rule 14 *Two agents interacting have each one a unique CE dedicated (partially or totally) to the other.*
$\forall a_1 = (b_1, e_1), a_2 = (b_2, e_2) \in A, interacting(a_1) = a_2 \Leftrightarrow \exists! e_{1_i} \in e_1, e_{2_i} \in e_2, a_2 \in dedicating(e_{1_i}) \wedge a_1 \in dedicating(e_{2_i})$

4.1.2 Through service agent-agent interaction representation

The second type of interaction is on the other hand the main element of the GRID-MAS integration. Actually, from a pure MAS point of view, we should say that AGIL describes service exchange interactions in MAS; it does it using advantages and power 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 decompose in two sub-relations, *using* and *providing*, that formalize the relation between agents and services. The *using* and *providing* relations may be simplified in AGIL as Figure 23 shows. An agent may use several (or none) services. A service may be used by several (or none) agents. An agent may provide several (or none) services. A service is provided by one and only one agent.

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

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

$providing : \Sigma \rightarrow A$ (application)

Figure 23: *Using* and *providing* relation simplified representation

Rule 15 *Exchanging a service is a sub kind of interaction.*
$\forall a_1, a_2 \in A, exchanging(a_1) = a_2 \Rightarrow interacting(a_1) = a_2$

Rule 16 *An agent may use a service if it is member of a VO for which the service is available.*
$\forall a \in A, \sigma \in \Sigma, using(\sigma) = a \Rightarrow \exists o \in O, a \in o \wedge handling(o) = handling(\sigma)$

Rule 17 *An agent may provide a service within a VO if it is a member of the VO and if it has a capability that interfaces the service in the corresponding service container.*
$\forall a = (b, e) \in A, \sigma \in \Sigma, providing(\sigma) = a \Rightarrow \exists \lambda \in \Lambda, o \in O, e_i \in e, a \in o \wedge interfacing(\lambda) = \sigma \wedge executing(\lambda) = e_i$

Rule 18 *Two agents may exchange a service only if they are member of the same VO and if one of them provide the service the other one use.*
$\forall a_1, a_2 \in A, exchanging(a_1) = a_2 \Rightarrow \exists o \in O, \sigma \in \Sigma, a_1, a_2 \in o \wedge using(\sigma) = a_1 \wedge providing(\sigma) = a_2$

4.2 Service-capability instantiation

4.2.1 Mapping of Grid service instantiation and CE instantiation

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 in 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 may have been created by core GRID mechanism (out-of-band and bootstrap mechanisms).[19]

In MAS, when an agent has a conversation, it can dedicate a part of its state to this conversation. It is called the conversation context. In the STROBE model, when an agent receives a message from another agent for the first time, it instantiates a new local Cognitive Environment 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 to the STROBE CE instantiation 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[20] and the service provider applies the servicization process on it in order to make available the new service instance for the service user(s). Integrating theses two instantiation mechanisms make capabilities to benefit from standardization, interoperation and allocated resources from GRID, and services to benefit from a dedicated context of execution and local conversation representation from MAS. The *instantiating* relation formalizes the relation between service-capability couples. The *instantiating* relation is represented in AGIL by a large arrow as Figure 24 shows. It is a special kind of relation as it correspond to the creation of new service-capability couple.

$instantiating : S \times \Lambda \rightarrow \Sigma \times \Lambda$ (relation)

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

[20] In order to map exactly the STROBE mechanisms to OGSA and WSRF specifications, we should say that a new CE may be viewed as a new WS-Resource, i.e., a CE is a dedicated association between capabilities and stateful resources.

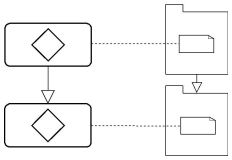

Figure 24: *Instantiating* relation representation

$\forall(s_f, \lambda_1) \in S \times \Lambda, (\sigma_i, \lambda_2) \in \Sigma \times \Lambda$ if $instantiating(s_f, \lambda_1) = (\sigma_i, \lambda_2)$ then s_f is called the *service factory* and σ_i the *service instance*.

Rule 19 *An instantiated service is different from its service factory and interfaces a different capability than the one the service factory interfaces.*
$\forall(s_f, \lambda_1) \in S \times \Lambda, (\sigma_i, \lambda_2) \in \Sigma \times \Lambda, instantiating(s_f, \lambda_1) = (\sigma_i, \lambda_2) \Rightarrow s_f \neq \sigma_i \wedge \lambda_1 \neq \lambda_2$

Rule 20 *If the service factory and instance may be in included in the same service container, the two corresponding capabilities are not executed in the same Cognitive Environment (even not included in the same agent).*
$\forall(s_f, \lambda_1) \in S \times \Lambda, (\sigma_i, \lambda_2) \in \Sigma \times \Lambda, instantiating(s_f, \lambda_1) = (\sigma_i, \lambda_2) \Rightarrow executing(\lambda_1) \neq executing(\lambda_2)$

4.2.2 Different cases of instantiation

Figure 25 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. The service provider creates a new CE including the new capability and servicizes this capability in the same service container. The new service becomes available for the service user.

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 25), and affect it GRID virtualized resources. Similarly, another core GRID situation (not represented in Figure 25) is the instantiation of a CAS service instance. Allowing these core GRID functionalities to be realized by agent is a powerful means of the integrated model propose. 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 is necessary 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 capability.

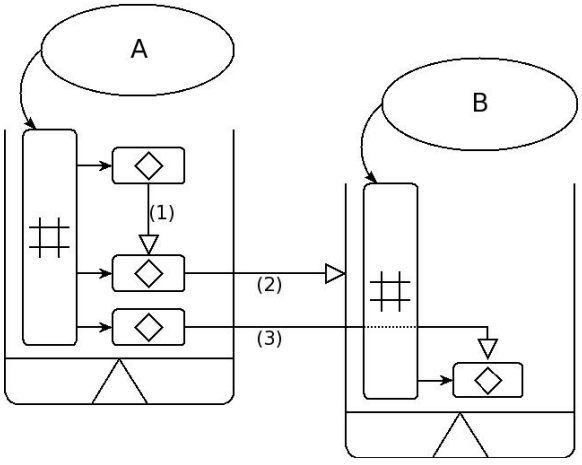

Figure 25: Examples of different kind of instantiations in Grid service container

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 maximum local CE totally or partially dedicated to another agent it is interacting with.[21] This is expressed by rule 13. 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 26 shows. An agent can have several children. An agent has zero or one parent (i.e., not all agents are result of reproduction).

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

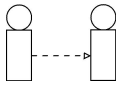

Figure 26: *Reproducing* relation representation

Rule 21 *A service provider agent reproduces itself only in a service exchange situation. It does it because it already has a local CE dedicated to a specific service user agent.*
$\forall a_1 = (b_1, e_1), a_2 = (b_2, e_2) \in A, reproducing(a_1) = a_2 \Leftrightarrow exchanging(a_1) = (a_2) \wedge \exists e_{1_i} \in e_1, e_{2_i} \in e_2, instantiating(e_1) = e_2$

When an agent reproduces, it produces a new agent exactly the same than itself except considering the set of local CEs. The child agent only has a local CE dedicated to the service user which provokes the reproduction; none of the father local CEs are inherited. Since the child agent is created, it starts evolving alone autonomously: changes its brain, changes its global CE, publishes some services, etc. The father agent cannot instantiate CE in the child body anymore. When created, the new agent is only a member of the VO of the service user that provokes its creation. This situation is illustrated in Figure 27.

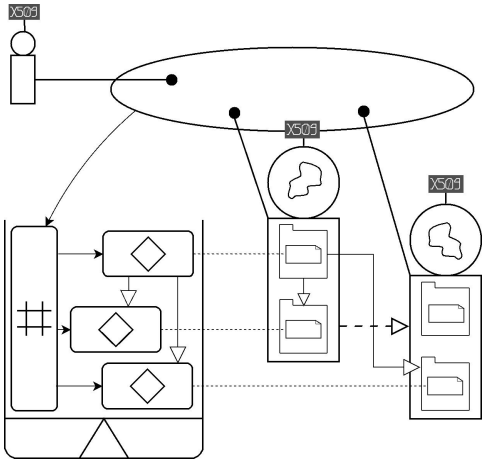

Figure 27: Examples of agent reproduction

4.4 Service adaptation

What is important in this integrated model is to consider how a service may be adapted to serve user agent(s) in order to implement DSG. We identify four ways:

[21] 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.

1. The agent service provider adapts the service according to interactions of the agent service user with it. The first experimentation of [JC05b] illustrates this way by a teacher-student scenario, with a meta-level learning where an agent dynamically learns a new performative (communicative act) by simply executing a capability;

2. The agent service provider may offer another service to change or adapt the original service. The second experimentation of [JC05b] illustrates this way by an e-commerce scenario, with a constraint based mechanism, where an agent dynamically adds constraints to a train ticket research capability.

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

4. Direct agent-agent interactions may occur between the agent service user and the agent service provider and within these interactions (1) and (2) may occur in a pure ad hoc form (not via service).

5 Towards an OGSA-STROBE model

5.1 Sum-up 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 one of their capabilities in a certain VO but not in another. The VO service container is then used as a service publication/retrieval platform (the semantics may also be situated there). A service is executed or engaged, by the agent who proposes it but it (the service) uses resources allocated by the service container. Figure 28 shows the complete integrated model described in AGIL.

5.2 Simple example

Figure 28 shows the generic elements of the GRID-MAS integrated model. Figure 29 shows the same elements but illustrated on a concrete simple example. Let us consider a VO of four users A,B,C and D. D published in the VO 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 29 shows both the VO service container and the D agent body. Some services and CEs have been instantiated. S1 and S2 correspond to D 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[22]). 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 capacity was created in E_C^D even if D did not publish it as a service in the VO service container. It may be made later if C asks for a decrementing service.

- S3 and S4, which are instances of the same service factory, 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 4.4.

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

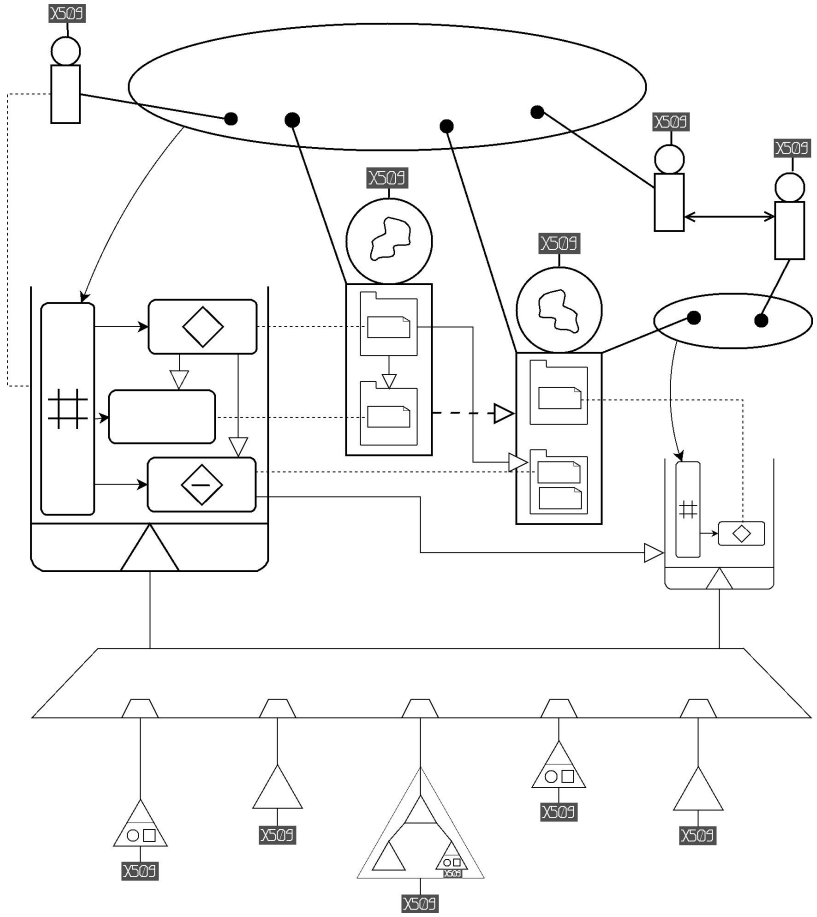

Figure 28: The GRID-MAS integrated model described in AGIL

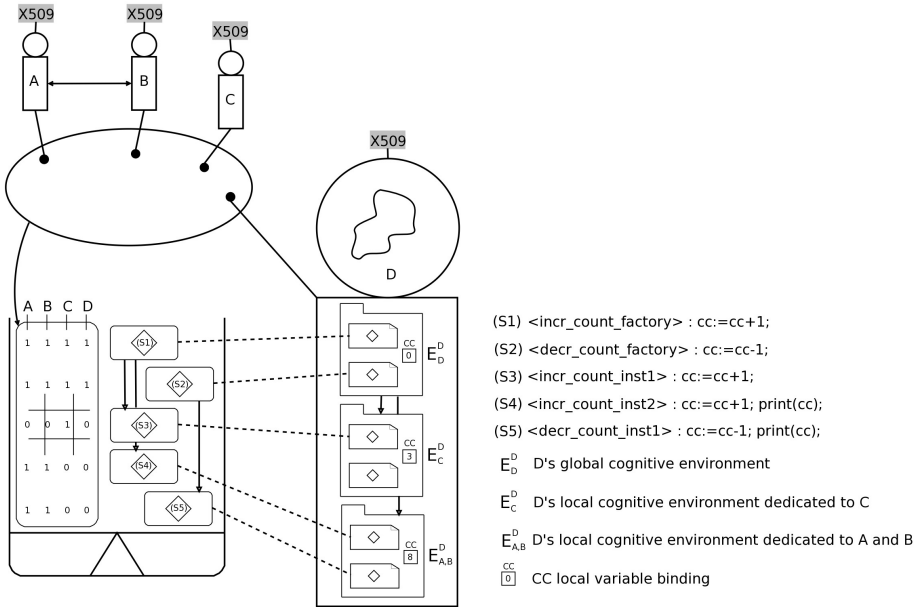

Figure 29: Example of the GRID-MAS integration

Comparison with WSRF. Figure 30 shows the same example but according to WSRF specification. The S1 WS-Factory instantiated two different WS-Resources (S3 and S4) which share the same stateless service but associated with different stateful resources. The S2 WS-Factory instantiated only one WS-Resource (S5) which shares the same stateful resource as S4 but with another stateless service. Notice that the incrementing service is unavoidably the same for the two WS-Resource S3 and S4. A stateless service may be part of several WS-Resource, but doing that it cannot be really dedicated to the WS-Resource user(s). Integrating WSRF with agents allows to have the stateless service executed by an intelligent agent, but furthermore, using STROBE agents allow to have this service dedicated. 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 evolve differently to fit better service user needs or wants. This is not possible with WSRF.

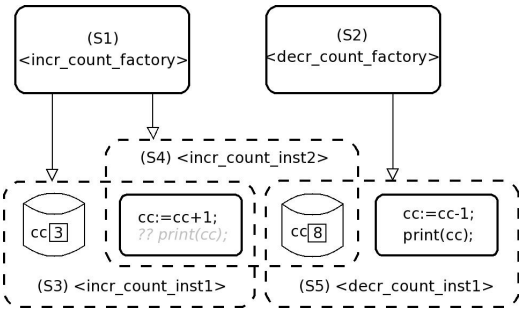

Figure 30: WSRF elements corresponding to the example

5.3 Discussions and benefits for GRID, MAS and SOC

- There is no real standard in the MAS community to describe agent capabilities. The integration will help MAS developers in presenting and interfacing agent capabilities, and therefore augment MAS interoperation. It is what we called previously the servicization of the capability.
- 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 with 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; those 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 exchanges in this integrated model benefit 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 that dynamically composing and choreographing services in business processes in SOC.
- Moreover, our integrated model 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 services as executed also by agents.

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 are now sufficiently mature to enable justifying exploring the path towards an integration of the two domains. 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. We put this enhanced concept of service into the perspective of Dynamic Service Generation (DSG).

Besides describing why GRID and MAS need each other, we explained how they can become synergic. Through an analysis of concrete models (mainly OGSA and STROBE), we extracted a few key concepts and presented a set of GRID and MAS analogies. Then, inspired by the analogies and related work, we proposed an integrated model that respects all the constraints and foundations of both GRID and MAS. We proposed a new set-theory based formalization and a common graphical description language, called AGIL, to describe part of the DSG mechanisms: i.e., through-service agent-agent interaction.

Indeed, the concepts of the integrated GRID-MAS model can be summarized by analyzing the interactions in this model (Figure 22). *The integration may be viewed as a formalization of service exchange interactions in MAS using GRID mechanisms.* The most significant contribution in this paper, which links together all the elements of Figure 22, is the relation introduced between i) a Grid service instance and its state and ii) an agent capability and its context. OGSA fits well for the first part of the interaction and the STROBE model fits well for the second part.

The integration proposed in this paper is feasible considering today's state of GRID and MAS technologies. However, future developments in GRID and MAS may rely on 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.

Support for non-synchronous protocols and semantics. GRID and SOA standards must evolve to fit better agent communications 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.

Finally, 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.

Appendix

The AGIL language is currently under specification under the form of an ontology 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. The MadKit platform is itself being deployed on a Grid node within a research project lead at LIUPPA (`liuppa.univ-pau.fr`).

Acknowledgement

Work partially supported by the European Community under the Information Society Technologies (IST) programme of the 6^{th} Framework Programme for RTD - project ELeGI, contract IST-002205. This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of data appearing therein.

References

[ACR+05] Colin Allison, Stefano A. Cerri, Pierluigi Ritrovato, Angelo Gaeta, and Matteo Gaeta. Services, Semantics and Standards: Elements of a Learning Grid Infrastructure. *Applied Artificial Intelligence Journal, Special issue on Learning Grid Services*, 19(9-10):861–879, October-November 2005.

[AGP03] Liliana Ardissono, Anna Goy, and Giovanna 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.

[BMMS03] Joanna J. Bryson, David Martin, Sheila A. McIlraith, and Lynn 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.

[BV04] Paul A. Buhler and José M. Vidal. Integrating Agent Services into BPEL4WS Defined Workflows. In *4th International Workshop on Web-Oriented Software Technologies*, 2004.

[BVV03] Paul A. Buhler, José M. Vidal, and Harko Verhagen. Adaptive Workflow = Web Services + Agents. In *International Conference on Web Services, ICWS'03*, pages 131–137, Las Vegas, NV, USA, July 2003. CSREA Press.

[Cer99] Stefano A. Cerri. Shifting the Focus from Control to Communication: the STReam OBjects Environments Model of Communicating Agents. In J.A. Padget, editor, *Collaboration between Human and Artificial Societies, Coordination and Agent-Based Distributed Computing*, volume 1624 of *Lecture Note in Artificial Intelligence*, pages 74–101. Springer-Verlag, Berlin, Germany, 1999.

[CSJN05] Junwei Cao, Daniel P. Spooner, Stephen A. Jarvis, and Graham R. Nudd. Grid load balancing using intelligent agents. *Future Generation Computer Systems*, 21(1):135–149, January 2005.

[CT03] Mario Cannataro and Domenico Talia. The Knowledge Grid. *Communications of the ACM*, 46(1):89–93, January 2003.

[CTT05] Carmela Comito, Domenico Talia, and Paolo Trunfio. Grid Services: Principles, Implementations and Use. *Web and Grid Services*, 1(1):48–68, 2005.

[DCG05] John Domingue, Liliana Cabral, and Stefania 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.

[DG00] Frank Dignum and Mark Greaves, editors. *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Germany, 2000.

[DHKV03] Jonathan Dale, Akos Hajnal, Martin Kernland, and Laszlo Zsolt Varga. Integrating Web Services into Agentcities Recommendation. Agentcities Technical Recommendation actf-rec-00006,, Agentcities Web Services Working Group, November 2003.

[DS83] Randall Davis and Reid G. Smith. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence*, 20(1):63–109, January 1983.

[Fer99] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison Wesley Longman, Harlow, UK, 1999.

[FFG+04] Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Donald F. Ferguson, Frank Leymann, Martin Nally, Igor Sedukhin, David Snelling, Tony Storey, William Vambenepe, and Sanjiva Weerawarana. Modeling Stateful Resources with Web Services. Whitepaper Ver. 1.1, The Globus Alliance, May 2004.

[FGM03] Jacques Ferber, Olivier Gutknecht, and Fabien 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. Springer-Verlag, Berlin, Germany, 2003.

[Fip02] FIPA-ACL Message Structure Specification. FIPA Specifications SC00061G, Foundation for Intelligent Physical Agents, December 2002. www.fipa.org/specs/fipa00061/.

[FJK04] Ian Foster, Nicholas R. Jennings, and Carl 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.

[FKNT02] Ian Foster, Carl Kesselman, Jeff Nick, and Steve 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.

[FKT01] Ian Foster, Carl Kesselman, and Steve Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Supercomputer Applications*, 15(3):200–222, 2001.

[GC04] Dominic Greenwood and Monique Calisti. Engineering Web Service - Agent Integration. In *IEEE Systems, Cybernetics and Man Conference*, The Hague, Netherlands, October 2004. IEEE Computer Society.

[GCL04] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource Allocation in the Grid Using Reinforcement Learning. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'04*, New York, NY, USA, July 2004.

[Gel04] Marije Geldof. The Semantic Grid: will Semantic Web and Grid go hand in hand? Technical report, European Commission DG Information Society Unit 'Grid technologies', June 2004.

[GHCN99] Robert Ghanea-Hercock, Jaron C. Collis, and Divine T. Ndumu. Co-operating mobile agents for distributed parallel processing. In *3rd International Conference on Autonomous Agents*, pages 398–399, New York, NY, USA, 1999. ACM Press.

[GMM98] Robert H. Guttman, Alexandros G. Moukas, and Pattie Maes. Agent-mediated Electronic Commerce: A Survey. *The Knowledge Engineering Review*, 13(2):147–159, July 1998.

[Hew77] Carl Hewitt. Viewing Control Structures as Patterns of Passing Messages. *Artificial Intelligence*, 8(3):323–364, June 1977.

[HNL02] James E. Hanson, Prabir Nandi, and David 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.

[HS98] Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, San Francisco, CA, USA, 1998.

[HSB+05] Michael N. Huhns, Munindar P. Singh, Mark Burstein, Keith Decker, Ed Durfee, Tim Finin, Les Gasser, Hrishikesh Goradia, Nick Jennings, Kiran Lakkaraju, Hideyuki Nakashima, Van Parunak, Jeffrey S. Rosenschein, Alicia Ruvinsky, Gita Sukthankar, Samarth Swarup, Katia Sycara, Milind Tambe, Tom Wagner, and Laura Zavala. Research Directions for Service-Oriented Multiagent Systems. *Internet Computing*, 9(6):65–70, November-December 2005.

[Hug03] Marc-Philippe 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.

[Huh02] Michael N. Huhns. Agents as Web Services. *Internet Computing*, 6(4):93–95, July-August 2002.

[IYT04] Fuyuki Ishikawa, Nobukazu Yoshioka, and Yasuyuki 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.

[JC05a] Clement Jonquet and Stefano 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.

[JC05b] Clement Jonquet and Stefano A. Cerri. The STROBE model: Dynamic Service Generation on the Grid. *Applied Artificial Intelligence Journal, Special issue on Learning Grid Services*, 19(9-10):967–1013, October-November 2005.

[JC06] Clement Jonquet and Stefano A. Cerri. Characterization of the Dynamic Service Generation Concept. Research Report 06007, LIRMM, CNRS & University Montpellier II, France, February 2006. www.lirmm.fr/~ jonquet/Publications/.

[JDC06] Clement Jonquet, Pascal Dugenie, and Stefano A. Cerri. Service-based integration of Grid and Multi-Agent Systems models. Research Report 06012, LIRMM, CNRS & University Montpellier II, France, February 2006. www.lirmm.fr/˜ jonquet/Publications/.

[Jen01] Nicolas R. Jennings. An Agent-based Approach for Building Complex Software Systems. *Communications of the ACM*, 44(4):35–41, April 2001.

[KWvL02] Sriram Krishnan, Patrick Wagstrom, and Gregor von Laszewski. GSFL: A Workflow Framework for Grid Services. Draft paper, Globus Alliance, July 2002.

[LF97] Yannis Labrou and Tim 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/.

[LL04] Chunlin Li and Layuan Li. Competitive proportional resource allocation policy for computational grid. *Future Generation Computer Systems*, 20(6):1041–1054, August 2004.

[LRCSN03] Margaret Lyell, Lowell Rosen, Michelle Casagni-Simkins, and David 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.

[MBP05] Sunilkumar S. Manvi, M. N. Birje, and Bhanu Prasad. An Agent-based Resource Allocation Model for computational grids. *Multiagent and Grid Systems*, 1(1):17–27, 2005.

[MCd02] Nicolas Maudet and Brahim 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.

[MML05] Zakaria Maamar, Soraya K. Mostéfaoui, and Mohammed Lahkim. Web services composition using software agents and conversations. In D. Benslimane, editor, *Les services Web*, volume 10 of *RSTI-ISI*. Lavoisier, 2005.

[Mor02] Luc 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, *Second IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID'02*, pages 220–228, Berlin, Germany, May 2002. IEEE Computer Society.

[MS03] E. Michael Maximilien and Munindar 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.

[MT99] Frank Manola and Craig Thompson. Characterizing the Agent Grid. Technical report 990623, Object Services and Consulting, Inc., June 1999.

[NPC+04] Timothy J. Norman, Alun Preece, Stuart Chalmers, Nicolas R. Jennings, Michael Luck, Viet D. Dang, Thuc D. Nguyen, Vikas Deora, Jianhua Shao, Alex Gray, and Nick J. Fiddian. Agent-based formation of virtual organisations. *Knowledge Based Systems*, 17(2-4):103–111, May 2004.

[Pel03] Chris Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10):46–52, October 2003.

[Pet05] Jan 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.

[PTJ+05] 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.

[PWF+02] Laura Pearlman, Von Welch, Ian Foster, Carl Kesselman, and Steven Tuecke. A Community Authorization Service for Group Collaboration. In *3rd International Workshop on Policies for Distributed Systems and Networks, POLICY'02*, pages 50–59, Monterey, CA, USA, June 2002. IEEE Computer Society.

[RJS01] David De Roure, Nicholas Jennings, and Nigel 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.

[RM00] Omer F. Rana and Luc 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.

[RPD99] Pierre-Michel Ricordel, Sylvie Pesty, and Yves 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.

[SH99] Munindar P. Singh and Michael N Huhns. Multiagent Systems for Workflow. *Intelligent Systems in Accounting, Finance and Management*, 8(2):105–117, June 1999.

[SH05] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing Semantics, Processes, Agents*. John Wiley & Sons, 2005.

[SLGW02] Weiming Shen, Yangsheng Li, Hamada H. Ghenniwa, and Chun Wang. Adaptive Negotiation for Agent-Based Grid Computing. In *AAMAS'02 Workshop on Agentcities: Challenges in Open Agent Environments*, pages 32–36, Bologna, Italy, 2002.

[Tia05] Huaglory 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.

[TV01] Orazio Tomarchio and Lorenzo Vita. On the use of mobile code technology for monitoring Grid systems. In *First International Workshop on Agent based Cluster and Grid computing*, pages 450–455, Brisbane, Australia, May 2001. IEEE Computer Society.

[WBPB03] Rich Wolski, John Brevik, James S. Plank, and Todd 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.

[WJK00] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, September 2000.

[Woo02] Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, UK, February 2002.

[WOvSB02] Niek J.E. Wijngaards, Benno J. Overeinder, Marteen van Steen, and Frances M.T. Brazier. Supporting Internet-Scale Multi-Agent Systems. *Data & Knowledge Engineering*, 41(2-3):229–245, June 2002.