Académie de Montpellier
Université Montpellier II
— Sciences et Techniques du Languedoc —

Mémoire de Stage de Master

Spécialité : **Recherche en Informatique**
Mention : **Informatique, Mathématiques, Statistiques**

effectué au laboratoire LIRMM/INFO
—
sous la direction de Abdelkader GOUAICH, Stefano A. CERRI

Grid Services & Software Agents Interoperation

par

Zeine AZMEH

Soutenue le June 19, 2007

Remerciements

Je voudrais remercier Mr Abdelkader GOUAICH qui a pris de son temps pour m'encadrer, me diriger et me conseiller pendant ces six mois de stage.

Je tiens à remercier Mr Stefano CERRI pour son soutient, sa direction et ses conseils tout au long de lannée.

Je remercie Mr Roland DUCOURNAU pour sa direction du master et ses conseils.

Je remercie tous les chercheurs du LIRMM pour les cours quils nous ont donnés et leur gentillesse.

Enfin, je remercie Fady, pour sa présence à mes cotés et son encouragement.

CONTENTS

CHAPTER 1

Grid Services & Software Agents Interoperation

1.1 Introduction

Service-oriented computing (SOC) is an emerging paradigm that is changing the way systems are designed, architected, deployed, and used. It utilizes services as fundamental elements for developing applications/solutions. Services are self-describing, cross-platform computational elements that support rapid, low-cost composition of distributed applications. Services perform functions, which can be anything from simple requests to complicated business processes. Services allow organizations to expose their core competencies programmatically over the Internet (or intra-net) using standard (XML-based) languages and protocols, and be implemented via a self-describing interface based on open standards [1].

The idea of providing services over the Web is quite old. The primary difference between older service offerings and contemporary Web services is that human intervention was previously required, and that Web services now are built upon principles and standards for connection, communication, description, and discovery.

There has been an increase in interest lately within the Grid community towards Service-Oriented Computing. Service-oriented Grid architectures promise effective means to share functions and processes among dynamic virtual organizations [2]. Its fundemental element is the Grid service, which is an extension of the Web service notion that is capable of addressing Grid requirements such as statefulness, lifetime management, inspection and monitoring. Grid services provide a set of well-defined interfaces and follow specific conventions to facilitate coordinating and managing collections of Web service providers.

Recently, the service-orientation approach can be noticed in the community of agents and multi-agent systems, in that, agents are producers, consumers and indeed brokers of services. A software agent is a self-contained entity, capable of problem solving, acts autonomously and flexibly in uncertain, dynamic environments, and it can adapt specific plans in pursuit of its aims and objectives [3]. Multiple software agents can work in collaboration, can negotiate and interact flexibly to solve expected and unexpected problems of the environment.

Thus, Grid and agents share the notion of a service, but each one of them uses different

technologies and offers various capabilities. Great benefits can be achieved if we can exploit synergies between Grid and agents. In fact, agents have much to offer to the grid, and the grid too has much to offer to agents. This issue is discussed in several papers like [4] for Foster et al. that described that agents and Grids need each other just like brain and brawn, and there are also other papers like [5], [6], [7], [8], [5].

The following points summarize some of the benefits that the grid and agents can get from each other:

- Grids can benefit from agents in becoming more flexible and agile
- Agents need to be built upon a robust infrastructure like the Grid, so that they become robust and secure
- In service-oriented Grid, it is necessary to organize sets of available services on demand in response to dynamic requirements and circumstances. The agents community has a wealth of expertise in this area. In the Grid, people are taking the first steps along this road, for example in WS-Agreement and WS-Negotiation.
- Automation is fundamental to the Grid but currently is not handled very flexibly. In contrast to Grid Services, autonomous agents work to achieve their individual objectives and they interact to meet their objectives in their common environment.
- The Grid needs to be self-healing, self-managing or self-organizing (exhibit autonomic behaviour), which can be achieved by the cooperation of the Grid community with the agents community.
- The composition of Grid services is possible by using the agent-based technology.
- Agents bring the dynamic decision-making, decentralization, coordination, and autonomous behaviour required to realize virtual organizations.
- Agent systems need to be built upon a robust platform, which can provide them with security and reliability, where they can discover, acquire, federate and manage their capabilities necessary to execute their decisions.
- The Grid offers agents real applications and real deployments.

But, since that Grid services and agents were originally developed separately with different standards and specifications. Software Agents and Multi Agent Systems specifications are governed by Foundation of Intelligent Physical Agents (FIPA) [9] and specifications of Web services are governed by W3C, hence there is a lot of difference among specifications of both technologies, so software agents and Web services cannot communicate with each other and their interoperation is not straightforward.

In order to achieve synergies between them, we must face the huge differences between the technologies that each of them use [10], and try to find the means that can map between them and enable the interaction and the cooperation between the Grid world and the Agents world.

Table 1.1 lists the technologies that the Grid service uses and those that the agent uses[1]:

[1]These technologies will be explained more precisely in next chapter.

Purpose	Grid Technology	Agent Technology
Service Description	WSDL	DFAgentDescription
Communication Protocol	SOAP	ACL
Publishing & Discovering	UDDI	DF
Messaging Type	Synchronous	Asynchronous

Table 1.1: Grid standards vs. Agent standards

- Agents and Web Services use different communication protocol; Agents use Agent Communication Language (ACL) to send asynchronous messages, whereas Web Services use Simple Object Access Protocol (SOAP) and send synchronous messages.
- Agents and Web Services use different service description languages; Agents use ontology named Directory Facilitator Agent Description (DF-Agent-Description) whereas Web Services use Web Services Description Languages (WSDL).
- Agents and Web Services use different service registries; Agents have Directory Facilitator (DF) based on FIPA specifications, whereas Web Services use Universal Description Discovery and Integration (UDDI) which is based on W3C specifications.

1.2 Problem Statement

The problem discussed in this research is that, having the two platforms:

- The Grid Platform
- The FIPA Agent Platform

That are developed separately with different specifications and standards, as shown in figure 1.1.

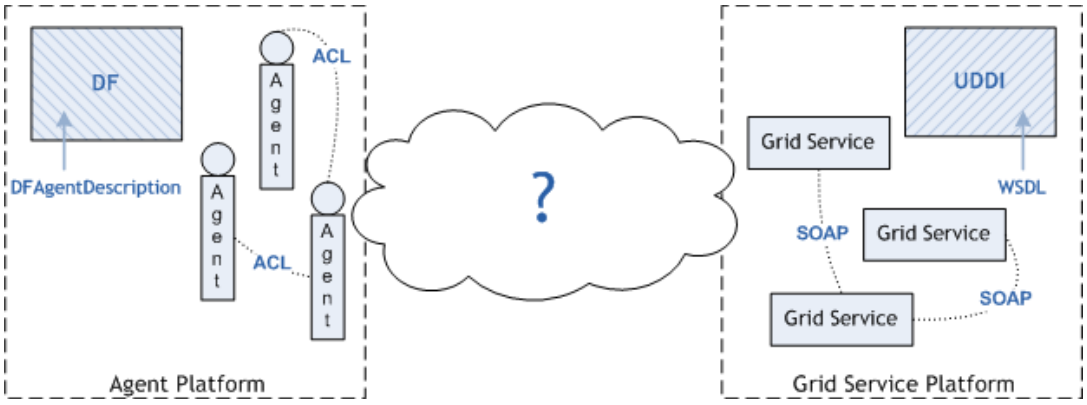

Figure 1.1: Mapping between Grid Service Platform and Agent Platform

We want to bridge the gap between these two platforms in order to enable them from:

i Discover each other

ii Communicate with each other

iii Cooperate and benefit from each others capabilities

In a flexible way and without changing any of their concepts or specifications.

1.3 Contributions

The main contributions of this research are:

1.3.1 The AgentGrid Gateway

The AgentGrid Gateway is a component that is composed of special agent services and special Grid services that work on providing the connection and communication between the Agent platform and the Grid platform; enabling agents from invoking Grid services and Grid services from invoking agents. This gateway is fully-specified and provides loose coupling integration (interoperation) between agents and Grid services in both ways, so that, it preserves the specifications and standardizations of both agents and Grid services worlds as they are, without the need of changing.

1.3.2 Flexible Invocation Protocols

In order to enable agents from invoking Grid services and vice versa, clear flexible protocols are defined, with the assumption that an agent knows that it will deal with a Grid service, and the Grid service also knows that it will deal with an agent. These protocols are of two types:

- The ServiceInvocationProtocol: A protocol for agents to enable them from invoking a Grid service
- The AgentInvocationProtocol: A protocol for Grid services to enable them from invoking an agent

1.3.3 Concepts Comprehension and Problems Identification

Many of the concepts and specifications of agent and Grid worlds are well studied and covered in this research, and the differences between the two platforms are clearly mentioned and identified. In theory, these differences may not stand in the way of implementing synergies between the two platforms. But practically, we can find that a lot of technical difficulties and limitations will exist, and which were not obvious in theoretical studies. So, in theory, a software agent can invoke a Grid service directly but practically this can't be done, because if an agent sends a synchronous SOAP call, it will lose its main concept of being an asynchronous active object that communicate using asynchronous messages only. And as for Grid services, they will never be able of sending an asynchronous ACL message to the agent world. On the other hand; Grid services are invoked using their automatically generated stubs that exist only in the Grid world, so how could any object from outside the Grid platform, have the ability of reaching these stubs to be able of invoking a Grid service? This is where difficulties and limits appear.

If we considered implementing an agent in the Grid world, so that it can uses the Grid service stubs to invoke the service, we can simply say that this can be done, but this is just in theory, because if we want to implement this, we will need both of the agent and Grid platforms to be mixed with all their standards and libraries, and this will not be an efficient scalable solution.

Another difficulty to be mentioned is the different registries that agents and Grid services use, so that, an agent will never be able of searching for a Grid service in the Grid service registry (UDDI), nor the Grid service will be able of searching for an agent in the agents registry (DF).

This is all in spite of the different identities of agents and Grid services, in that, each agent is identified by a unique agent ID, whereas a Grid service instance is identified by a unique URI address.

1.4 Outlines

The rest of this report is organized as follows:

- **Chapter 2:** This chapter presents essential concepts and definitions for Grid Computing and Grid Services and for Software Agents and Multi-Agent Systems. It also clarify their basic standards and technologies.
- **Chapter 3:** This chapter exhibits the existing solutions for the Grid services and agents interoperation and describes various other related works in this domain.
- **Chapter 4:** This chapter demonstrates the proposed solution for the problem stated in this research.
- **Chapter 5:** This chapter illustrates the experiments that are carried out on the proposed solution and explains the obtained results.

CHAPTER 2

Backgrounds

2.1 Service-Oriented Computing

Service-Oriented Computing (SOC) is a new computing paradigm that utilizes services as the basic constructs to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments [1].

SOC's vision is a world of cooperating services where application components are assembled with little effort into a network of services that can be loosely coupled to create flexible dynamic business processes and agile applications that may span organizations and computing platforms.

Services are autonomous, platform-independent computational entities that can be used in a platform independent way. Services can be described, published, discovered, and dynamically assembled for developing massively distributed, interoperable, evolvable systems. Services perform functions that can range from answering simple requests to executing sophisticated business processes requiring peer-to-peer relationships between possibly multiple layers of service consumers and providers. Any piece of code and any application component deployed on a system can be reused and transformed into a network-available service. Services reflect a "service-oriented" approach to programming, based on the idea of composing applications by discovering and invoking network-available services rather than building new applications or by invoking available applications to accomplish some task. Services are most often built in a way that is independent of the context in which they are used. This means that the service provider and the consumers are loosely coupled.

The Service-oriented architecture describes a software architecture that defines the use of loosely coupled software services to support quickly new business requirements. This architecture provides increased stability and availability to users, and reacts to component failure by finding and binding to equivalent components without requiring human intervention.

2.2 Web Services

Web Services are distributed software components that provide information to applications rather than to human [1]. They publish details of their functions and interfaces, but they

keep their implementation details private. They are applicable to distributed heterogeneous environments.

2.2.1 Architectural Model

The general architectural model for Web Services consists of three types of participants:

- Service providers: create Web services and advertise them to potential users by registering the Web services with service brokers.
- Service brokers: maintain a registry of advertised services and might introduce service providers to service requesters.
- Service requester: search the registries of service brokers for suitable service provider to use its services.

This model is founded on standards for: connection (XML), communication (SOAP), description (WSDL) and discovery (UDDI).

2.2.2 Web Services Standards

The success of Web services is due in large part to the development of standards by bodies such as the World Wide Web Consortium (W3C), the Organization for the Advancement of Structured Information Standards (OASIS) and the Distributed Management Task Force (DMTF).

The following standards are the key specifications used by Web services.

XML

The eXtensible Markup Language is the common language that is used for information exchange between providers and requesters of services. It conveys information about the meaning of data, it regularizes the syntax of HTML, so that it is easier to parse and process.

SOAP

The Simple Object Access Protocol is a simple and lightweight protocol for exchanging XML messages using HTTP from a sender to a receiver.
The SOAP specification defines the following fundamental components [11]:

- An envelope that defines a framework for describing message structure
- A set of encoding rules for expressing instances of application-defined data types
- A convention for representing remote procedure calls (RPC) and responses
- A set of rules for using SOAP with HTTP
- Message exchange patterns (MEP) such as request-response, one-way and peer-to-peer conversations

WSDL

The Web Service Description Language is an XML language for describing a programmatic interface to a Web service. The description includes definitions of data types, input and output message formats, the operations provided by the service, network addresses and protocol bindings.

UDDI

The Universal Description Discovery and Integration describes a mechanism for registering and locating Web services. It defines an online registry where service providers can describe their organization and register their Web services. The registry can then be interrogated by service requesters' SOAP messages to locate the services they need, UDDI will provide access to WSDL documents describing the protocol bindings and message formats required to interact with the listed Web services.

2.3 Grid Computing

Grid Computing refers to distributed computing where several resources are made available over a network, and are combined into large applications on demand [12]. Building complex applications over Grid architectures has been difficult, which has led to an interest in the more modular kinds of interfaces based on services. Accordingly, Grid services have been proposed in analogy with Web services.

We can define the Grid problem as achieving flexible, secure, coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [13]. The sharing that we are concerned with is the direct access to computers, software, data and other resources. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs.

2.3.1 Virtual Organizations

A virtual organization is a dynamic collection of individuals, institutions that have common goals and are defined around resource-sharing rules and conditions [13]. It should be seen as a group of people who share common interests or participate to a common enterprise, and who assemble, collaborate, and communicate in a loose, distant, virtual way, using network communication facilities, tools and resources. A VO may be defined by the set of services that members operate and share.

2.3.2 Grid Standards

The Grid is built by Web services based standards within the context of the Open Grid Services Architecture (OGSA).

The most important standards organizations are the Global Grid Forum (GGF), which is the primary standards setting organization for the Grid, and OASIS, a not-profit consortium that drives the development, convergence and adoption of e-business standards, which is having an increasing influence on Grid standards. In addition, the World Wide Web Consortium (W3C) is also active in setting Web services standards, particularly those that relate to XML.

There are many standards involved in building a service-oriented architecture, which form the basic building blocks that allow applications execute service requests. The Web services-based standards and specifications include [14]:

- Program-to-program interaction (SOAP, WSDL and UDDI);
- Data sharing (eXtensible Markup Language - XML);
- Messaging (SOAP and WS-Addressing);
- Reliable messaging (WS-ReliableMessaging);
- Managing workload (WS-Management);
- Transaction-handling (WS-Coordination and WS-AtomicTransaction);
- Managing resources (WSRF or Web Services Resource Framework);
- Establishing security (WS-Security, WS-SecureConversation, WS-Trust and WS-Federation);
- Handling metadata (WSDL, UDDI and WS-Policy);
- Building and integrating Web Services architecture over a Grid (OGSA);
- Overlaying business process flow (Business Process Execution Language for Web Services - BPEL4WS);
- Triggering process flow events (WS-Notification);

OGSA

The Open Grid Services Architecture (OGSA) is perhaps the most important standard that has emerged recently. OGSA was developed by the GGF. OGSA is an Informational specification that aims to define a common, standard and open architecture for Grid-based applications. The goal of OGSA is to standardize almost all the services that a Grid application may use. OGSA specifies a Service-Oriented Architecture (SOA) for the Grid computing environment for business and scientific use [15], OGSA is based on several other Web service technologies, notably WSDL and SOAP. Briefly, OGSA is a distributed interaction and computing architecture based around services, assuring interoperability on heterogeneous systems so that different types of resources can communicate and share information. OGSA has been described as a refinement of the emerging Web Services architecture, specifically designed to support Grid requirements.

WSRF

The WS-Resource Framework is concerned mainly with the creation, addressing, inspection and lifetime management of stateful resources. The framework provides the means to express state as stateful resources and codifies the relationship between Web Services and stateful resources in terms of the "implied resource pattern", which is a set of conventions on Web Services technologies, in particular XML, WSDL and WS-Addressing [16].

2.3.3 Grid Services

OGSA represents every resource as a Grid service which is, a Web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification and manageability; the conventions address naming and upgradeability[15].

Grid services can maintain an internal state; the existence of state distinguishes one instance of a service from another that provides the same interface [16]. The term Grid service instance is used to refer to a particular instantiation of a Grid service. Grid service instances can be created and destroyed dynamically.

2.3.4 GT4

The Globus Toolkit is a community-based, open-architecture, open source set of services and software libraries that support Grids and Grid applications [15]. The toolkit addresses issues of security, information discovery, resource management, data management, communication, fault detection and portability.

The toolkit components that are most relevant to OGSA are the Grid Resource Allocation and Management (GRAM) protocol and its "gatekeeper" service, which provides for secure, reliable, service creation and management; the Meta Directory Service (MDS-2), which provides for information discovery through soft state registration, data modeling and a local registry; and the Grid Security Infrastructure (GSI), which supports single sign on, delegation and credential mapping. These components provide the essential elements of a service-oriented architecture, but with less generality than is achieved in OGSA.

2.4 Software Agents

A software agent is a self-contained program which has a persistent state and is capable of controlling its own decision making and acting, based on its perception of its environment, in pursuit of one or more of its objectives [17].

Three distinct classes of agent can be identified:

- "gopher" agents, which execute straightforward tasks based on pre-specified rules and assumptions.
- "service performing" agents, which execute a well defined task at the request of a user.
- "predictive" agents, which volunteer information or services to a user, without being explicitly asked, whenever it is deemed appropriate.

Key hallmarks of agenthood:

- Autonomy: the ability to perform the majority of their problem solving tasks without the direct intervention of humans or other agents, and having a degree of control over their own actions and their own internal state.
- Social ability: the ability to interact with other software agents and humans in order to complete their own problem solving and to help others with their activities when appropriate.

- Responsiveness (reactiveness): the ability to perceive their environment and respond in timely fashion to changes which occur in it.
- Proactiveness: the ability to exhibit opportunistic, goal-directed behavior and take the initiative where appropriate.

A multi-agent system (MAS) is a system composed of several agents, collectively capable of reaching goals that are difficult to achieve by an individual agent or monolithic system.

2.4.1 The BDI Architecture

The Belief-Desire-Intention (BDI) architecture is an agent mental states architecture, addresses how beliefs (i.e., the information an agent has about its surroundings), desires (i.e., the things that an agent would like to see achieved) and intentions (i.e., things that an agent is committed to do) are represented, updated and processed [18].

2.4.2 FIPA Agents

FIPA, the Foundation for Intelligent Physical Agents is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.
A FIPA agent is a computational process that implements the autonomous, communicating functionality of an application. Typically, Agents communicate by exchanging messages which represent speech acts, and which are encoded in an agent communication-language. [9].

FIPA ACL Message

A FIPA ACL message contains a set of one or more message parameters, the mandatory parameter in all ACL messages is the performative, which represents the communicative act that the agent realize [19].
Communicative acts are the basic blocks of a dialogue between two agents. A communicative act has a meaning that is independent of the content and is performed just sending a message from one agent to another. The meanings of the communicative acts make reference to the mental attitudes belief, uncertainty and intention. the ACL message structure is composed of three levels, as shown in figure 2.1:

- Content level which is often described with a content language (PROLOG, Scheme, FIPA-SL, ...)
- Message level which is defined by the performative, the ontology, the language, etc. used in the message
- Communication level which allows to identify the metadata on the transmission of the message itself: the sender, the receiver, the protocol used, the conversation identifier, etc.

The Directory Facilitator

The DF (Directory Facilitator) is a centralized registry of entries which associate service descriptions to agent IDs. So, it provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.

COMMUNICATION LEVEL	MESSAGE LEVEL	CONTENT LEVEL
:sender	:performative	:content
:receiver	:language	
:reply-to	:encoding	
:protocol	:ontology	
:conversation-id		
:reply-with		
:in-reply-to		
:reply-by		

Figure 2.1: FIPA-ACL Message Parameters [19]

The DFAgentDescription

the DFAgentDescription (DFD), is used to describe an agent and enable him of registering in the DF, so that other agents can search for him.

2.4.3 Agent Platforms

There has recently been an increasing trend towards making agent technology a serious basis for building complex, distributed systems. Several agent development environments support specific agent architectures and provide libraries of interaction protocols, like Madkit, JADE, ...

Madkit

MadKit (Multi-Agent Development Kit) is a modular and scalable multiagent platform written in Java and built upon the AGR (Agent/Group/Role) organizational model: agents are situated in groups and play roles. MadKit allows high heterogeneity in agent architectures and communication languages, and various customizations [20].
MadKit communication is based on a peer to peer mechanism, and allows developpers to quickly develop distributed applications using multiagent principles.

JADE

JADE (Java Agent DEvelopment Framework) is a software Framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middleware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases [21].

JADE VS. MADKIT

The following figure 2.2, shows some of JADE platform properties compared to Madkit platform properties. This comparison enables us from clearly distinguish between the two platforms, and from using the one that best corresponds to our given requirements.

Jade	Madkit
Open source	Open source
Agent-oriented	More modularity because of the usage of AGR (Agents, Groups, Roles)
FIPA compliant Specific organizing agents, ACL, compatibility with other FIPA platforms	Supports ACL messages and It's more efficient because we're not obliged to use the complete ACL
Ontology Management, using SL0 language	Doesn't manage SL (describe the content)
More difficult	Easier development & test environment
When using several complex & fat agents (individual, their planning, action & execution) capable of complex behaviors & interactions; agents that interact with other systems & speak ACL & use ontologies	Easier & more preferable when managing a big number of simple agents with simple, reactive tasks; global behavior of MASs (MA simulations)

Figure 2.2: JADE vs. Madkit

CHAPTER 3

State of the Art

3.1 Introduction

Web services and software agent technologies are two areas that have attracted substantial research and industry interests in recent years. On one hand, the Web services technology is gaining popularity because of its well-defined infrastructure aiming at enabling interoperability among heterogeneous applications. On the other hand, the agent technology aims at providing intelligent autonomous capabilities for distributed components. A combination of these two technologies could create an environment where Web services and agents can employ and compliment each other's strengths.

Recently, Grid Computing and Web Services have started to merge. The goal of the Web Services effort is to provide infrastructure for services to be advertised, found, and accessed over the Internet using Web-style protocols. Web Services and Grid Computing face many similar challenges: description and advertisement of services, description and matchmaking (lookup) of service requests, invocation of services (possibly with attached contracts regarding quality of service or other constraints), and the accounting mechanisms to charge for resource usage.

Several works has been proposed in the domain of Grids and Agents interoperation, each one of them has a particular purpose of using Grids with Agents.

3.2 Related Works

The integration of software agents and Web services in general has been proposed by Luck et al. [22] as one of the major tasks for the agent community. The main obstacles in integrating Web services and agents are the mismatches in description and communication used by these two technologies.

Significant research has been carried out in the field of Grid/Web services and agents integration. As an example of such effort that aimed at addressing the issue of Web services and agents integration, we mention the proposition of the Agentcities [23], which describes a theoretical high-level architectural recommendation. In this architecture, as shown in figure 3.1, a FIPA agent service environment is assumed to exist in parallel with a Web Service environment. The "FIPA Agent Service to Web service Gateway" on the border between

the two environments is to allow FIPA agents to access Web Services by translating ACL messages to Web Service invocations. In the reverse order, the "Web Service to FIPA Agent Gateway" exposes and registers agent services in UDDI Registry Server so that any Web Service client can use them.

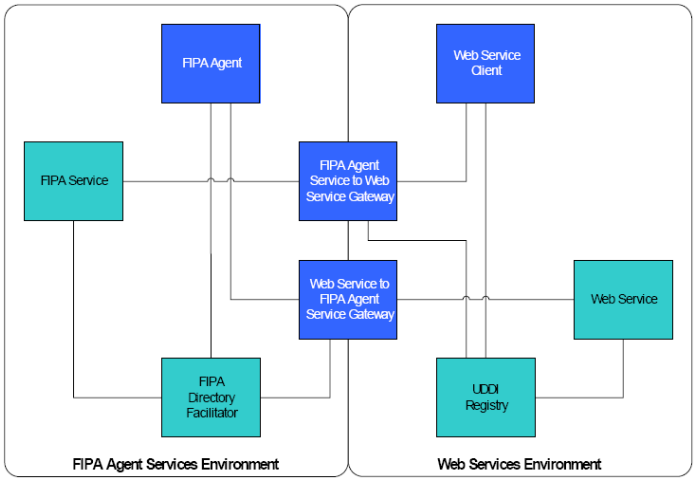

Figure 3.1: Agentcities Recommendations, Integration Architecture [23]

Following the Agentcities recommendations, several separate implementations have been proposed:

- Exposure of Web Services to Jade agents, by Sztaki (WSDL2Agent), [24].
- Exposure of Jade agent services to Web Services, by Whitestein Technology (WSAI, WSIGS), [25][26][27].
- Exposure of Web services as Jade agents' own services at runtime, by Swinburne University of Technology (WS2JADE), [28].

3.2.1 WSDL2Agent

The WSDL2Agent tool is used to help the integration of existing web services into agent based systems, thus to bridge the gap between existing non-agent systems and agent systems [24][29].
The WSDL2Agent tool has two parts: WSDL2Jade, and WSDL2Protégé.

The input to the WSDL2Agent tool is the WSDL file of an arbitrary web service, which after being analyzed, two types of output will be generated as shown in figure 3.2:

- the proxy agent ontology that is closely related to the web service call signature, and which is generated by the WSDL2Protégé part.
- the code of the proxy agent that can be deployed in any Jade agent platform, and is generated by WSDL2Jade part.

The generated ontology can be published for client agents that can then send requests to and accept responses from the proxy agent.

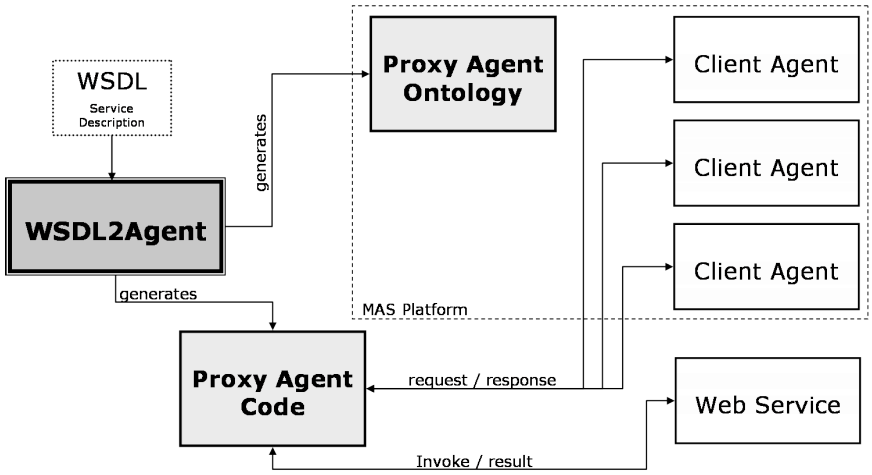

Figure 3.2: The WSDL2Agent Tool

The proxy agent in turn is able to accept agent requests, invoke the web service, and send the results back to the client agent. By performing all the necessary translations between ACL messages (agent requests, responses) and SOAP messages (web service calls, results) before and after it calls the web service, respectively.

3.2.2 WSAI

WSAI (Web Services and Agents Integration), is an open source project proposed by Whitestein Technology. WSAI's objective is the easy integration of Web Services in the world of FIPA software agent services, and vice versa.
It allows Web service clients to use Jade agents' services, by generating WSDL files that correspond to these agent services[25].

Technically, at this stage WSDL files are created manually from these agents' behaviors. It also requires manual programming of "interface agents" to communicate with the target agent. These "interface agents" are created and destroyed per Web service client invocation of the service. However it appears that the applicability of WSAI in practical situations is limited because of two major obstacles:
Firstly, Jade agent service specification is specified loosely and not based on message passing levels. This makes it difficult to automatically generate WSDL interfaces.
Secondly, the default single-threaded mode of Jade agent, and the asynchronous and stateful nature of agent communication do not fit well in the Web Service communication model.

3.2.3 Web Service Agent Integration Project

This project was initiated and lead by Whitestein in the Agentcities context. It came as a result of the WSAI open-source project (mentioned above).
The project consists of two main components:

- The Agent Gateway (WSAG) that does the actual transition from agents to web services
- The Agent Generator, which is a supporting tool for generating Gateway Agents - entities which provide a concrete web service interface for a particular agent.

The Agent Gateway (WSAG) and the Agent Generator are both implemented using JADE.

3.2.4 WSIGS

The WSIGS (Web Service Integration Gateway Service)was proposed by Whitestein Technology, it is a JADE add-on that provides support for bidirectional invocation of Web services from JADE agents, and JADE agent services from Web service clients [26][27].

A clearly useful feature of JADE would be a means to search for and invoke Web services directly from application agents. The WSIGS meets these needs by providing the means to register Web services in the JADE DF mapped onto FIPA DF agent descriptions. Registered Web services can then be invoked from agents by directing the invocation to the Gateway agent. In addition, the WSIGS is engineered to be bidirectional in operation so that Web service clients can also search for and invoke agent services hosted within JADE containers.

The WSIGS supports the standard Web services stack, consisting of WSDL for service descriptions, SOAP message transport and a UDDI repository for registering Web services using tModels.

The architecture of WSIGS is shown in figure 3.3, it consists of a JADE agent, called the Gateway agent, which controls the gateway from within a JADE container. This agent uses a set of xerces based codecs to bidirectionally translate ACL/SL0 into WSDL, tModels and SOAP according to the specific context. It also has a socket connection to an Axis Web server from where agent services can be exposed as callable stubs as if they were Web services. The Axis server also receives calls for invocation from external Web service clients onto agent services.

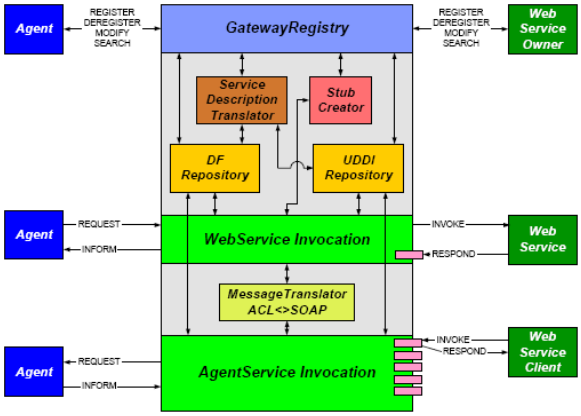

Figure 3.3: The WSIGS Architecture [26]

JADE agents can register their services with the normal platform DF and if tagged with type=”wsigs” these registrations will be automatically detected by the Gateway agent, converted into WSDL/tModel form and registered with the WSIGS UDDI directory. From there they can be discovered by external Web service clients and invoked via stubs exposed through the Axis Web server. Web services can register directly with the WSIGS UDDI repository. These registrations are also automatically detected by the Gateway agent, converted into ACL/SL0 service description and registered with the JADE platform DF. From there they can be discovered by agents and invoked by sending a FIPA Request message to the Gateway agent with the service invocation identity and parameters.
The operations of the WSIGS

- Agent Service published as a Web service endpoint The WSIGS allows an agent to publish a service description as a Web service endpoint. All the agent need to do is set the Type parameter of the agent service description to "web-service". The DF will use this type specification as a trigger to inform the Gateway Agent to translate the ACL/SL0 service description into WSDL/tModel, register it with the WSIGS UDDI and create an endpoint for external Web service clients to invoke the agent service.

- Web Service published as a JADE agent service The WSIGS allows an external Web service to publish a service description as a JADE agent service description with the JADE platform DF. The Web service is registered with the WSIGS UDDI as it would be with any other UDDI - nothing special is required for WSIGS operation. When a new registration is detected the Gateway agent translates it into an equivalent FIPA Directory Entry and sends this as a registration message to the Platform DF.

- Web Service Invocation by an Agent A JADE agent would first seek the DF service description of the Web service in the platform DF, as it would if discovering an agent service. An invocation request must then be sent to the WSIGS Gateway agent as a FIPA Request message. No performatives than FIPA Request and FIPA Inform are treated by the Gateway. The content of the FIPA Request message must contain the identity of the service to be invoked and any necessary parameters. When received by the Gateway agent, the FIPA Request is parsed and the presence of an entry in the UDDI repository corresponding to the requested service is verified. If present a SOAP message is created to invoke the Web service, populated with parameters from the FIPA Request and sent to the Web service endpoint. If a response is expected a temporary endpoint is set up on the Axis server. This endpoint is removed once a response is received and has been passed to the Gateway parsers for return as a FIPA Inform to the original sender agent.

- Agent Service Invocation by a Web Service It is assumed that the agent service has previously been registered with the WSIGS Gateway. A Web service, or Web service client, would first seek the tModel service description of the Agent service in the WSIGS UDDI repository, as it would if discovering a Web service. An invocation request is then issued as a SOAP message onto the service endpoint exposed by WSIGS Axis server. The content of the SOAP message must contain the identity of the service to be invoked and any necessary parameters.

When received by the Gateway agent, the SOAP request is parsed and a FIPA Request message created to invoke the Agent service, populated with parameters from the received SOAP message. The FIPA Request is sent to the appropriate agent and if a response is expected, the Gateway agent waits to receive a FIPA Inform from the target agent. Once a response is received it is parsed into a SOAP message and returned to the invoking Web service client.

3.2.5 WS2JADE

The WS2JADE (Web Service to Jade) tool was proposed by the Swinburne University of Technology, for the purpose of integrating Web services and the Jade agent platform. It allows JADE agents to offer Web services as their own services at runtime [28].

This tool has two distinct layers:

- The interconnecting layer which contains interconnecting entities that glue Web services and agents together.
- The management layer which is creates and manages those interconnecting entities. These entities consist of special agents called WSAG, ontology and protocol specifications.

The WSAG (Web Service Agents) are the agents capable of communicating with Web services and of offering them as their own services.

In figure 3.4, we can see the WS2JADE system components and their link to Jade.

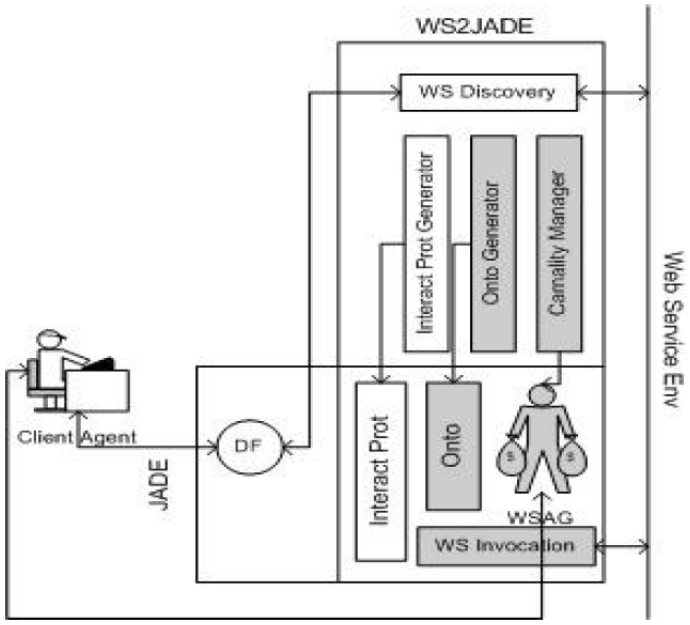

Figure 3.4: The WS2JADE System [28]

The WS2JADE system has the following components:

- **Ontology Generator:** responsible for ontology generation and ontology management. It translates data and its structure from Web service WSDL interfaces into meaningful information for Agents.
- **Interaction Protocol Generator:** translates SOAP messages into ACL messages.
- **Cardinality Manager**: responsible for cardinal mapping and service deployment management, a WSAG can offer more than one Web Service and a Web Service can be offered by more than one WSAG.
- **WS Discovery**: a piece of software that can use Web service discovery protocols and translate the received information into agent service description for DF.

When a client agent searches the DF for some service. The DF triggers WS2JADE to look up for available services in the Web service environment. If some Web services are found, their corresponding ontology and interaction models will be generated. Also, a WSAG that is capable of accessing the Web service will be generated. This WSAG registers the Web service as its own service on the DF and communication between the client agent and this WSAG can start if the client agent wants the service.

3.2.6 Autonomous Semantic Grid Project

The Autonomous Semantic Grid Project is aimed to provide an infrastructure for emerging distributed applications by improvements and using already available technologies.It provides a framework for open distributed systems and is based on synergy of Web Services, Grid Computing and Multi Agent Systems. It tries to solve the problem of agents and Web services interoperability, taking into account the different technologies that each one of them uses:

- Different directory services
- Different transport services
- Different languages (syntax and semantics)

Hence, it has a layered architecture, each layer represents a component that is dedicated for solving the mappings of the different technologies that are mentioned above. These components will be described throughly in the following sections, and they are:

- The AgentWeb Gateway [30]
- The Ontology Gateway [31]
- The Comtec Semantic Grid Framework [31][32]

AgentWeb Gateway

The AgentWeb Gateway is a middleware that solves the problem of different technologies that the Agents and the Grid services use, by providing appropriate transformation mechanisms [30].

Figure 3.5 shows us the AgentWeb Gateway architecture:

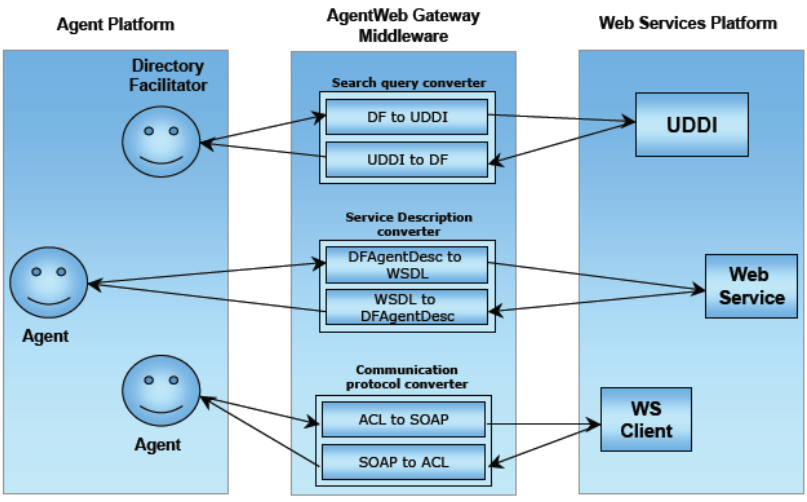

Figure 3.5: The AgentWeb Gateway [30]

- **Service Description transformation**, so that, Web services can be published in the DF, the agents registry, by using the WSDL to DFAgentDescription converter, and agents can publish their services in the UDDI, the Web services registry, by using the DFAgentDescription to WSDL converter.

- **Service Discovery transformation**, so that, agents can discover services in the UDDI, and Web services clients can discover services in the DF.
- **Communication Protocol transformation**, so that, services in vocation can be enabled among agents and Web services; agents can invoke Web services by using the ACL to SOAP converter, and Web services clients can invoke agents using the SOAP to ACL converter.

Ontology Gateway

The Ontology Gateway enables a bidirectional communication between agents and web services by translating FIPA ACL messages into equivalent OWL messages and vice versa [31], as can be seen in figure 3.6. It provides the semantic interoperability in distributed environments where technologies like agent applications and grid systems are combined and reused to achieve the autonomous semantic grid and thus providing a service-oriented framework.

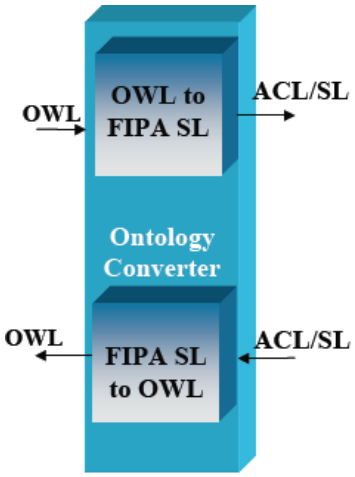

Figure 3.6: The Ontology Gateway [31]

Comtec Semantic Grid Framework

This Comtec Semantic Grid Framework is an agent-based framework for implementing Semantic Grid Services. It aims at more flexible and applicable Grid computing that processes ontology-based data and provides services over habitual Grid environment [31]. Agents in the framework will analyze application puposes and semantics of input and output in order to make plans and execute necessary procedures to achieve them.

The framework has the following architecture shown in figure 3.7:

Each agent in the figure provides a service to other agents by combining executable operation according to each purpose.

- The Application Agent is the core of this framework that represents Semantic Grid application as a service of the agent, it performs a specific task for each Semantic Grid application.

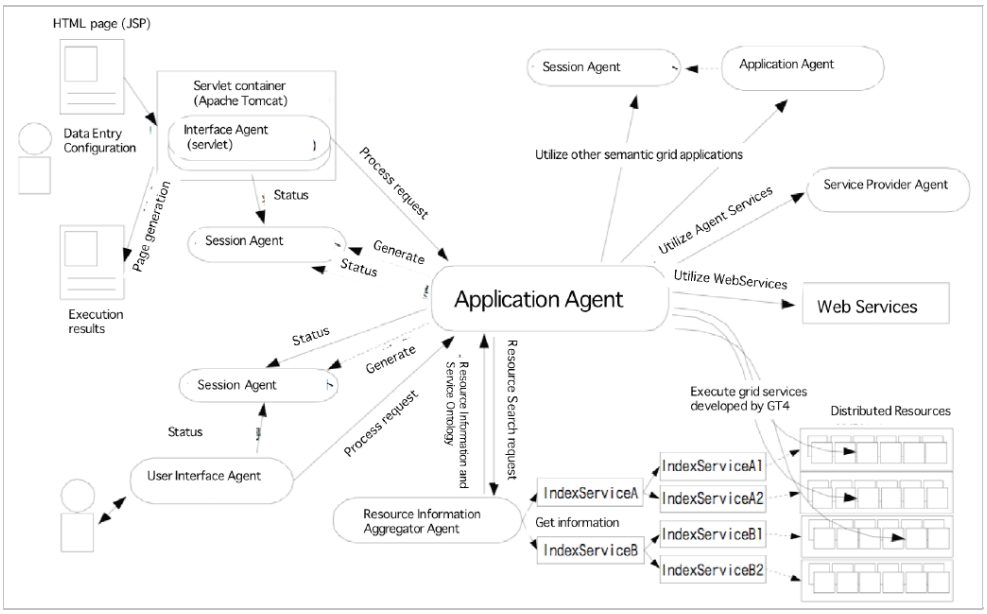

Figure 3.7: Comtec Semantic Grid Framework [31]

- The session Agent manages a series of tasks performed by application agents as a session and maintains the result of the application until the end-user has acquired it. It is generated on each invocation of services provided by application agent.
- Resource Information Aggregator Agent is an agent that manages information on distributed resources and allocates such resources according to requests from application agents.
- The Interface Agent interacts with human end users.

3.3 Other Related Works

There are many various works that discussed the issue of exploiting synergies between Grid computing and Multi-Agent Systems, that can yield more robust autonomous open-systems that encompass capabilities from both Grid systems and agent systems.
These works described different perspectives, and had different intentions. Most of them are noticeable in the domain of computational Grids, where multi-agent systems are used for several purposes. In [33] and [34], agent-based architectures are defined to manage Grid services and resources, one of these architectures is called ARMS, which is implemented using a hierarchy of homogeneous agents coupled with a performance prediction toolkit [35]. Another integration framework to be mentioned is called CRUTCH for agents and Web services [36]. There is also a work that uses Grid computing with multi-agent systems in order to create the Wisdom Grid [37], in which knowledge can be maintained, shared and discovered.

3.4 Analysis Criteria

The related works that are described in previous sections will be analyzed and compared according to table 4.1, the comparison attributes will be:

	GS	2-ways	Scalable	Specification	Interoperation
WSIGS	No	Yes	No	Yes	No
WS2JADE	No	No	No	No	Yes
WSDL2Agent	No	No	No	No	Yes
AS Grid	Yes	Yes	No	No	Yes

Table 3.1: Analysis Criteria

- GS: This attributes indicates whether the architecture takes into account Grid services or Web services
- 2-ways: To indicate if the model is from agent to Grid service and from Grid service to agent
- Scalable: To specify if the architecture can handle multiple requests with out suffering from the bottleneck problem
- Specification: To indicate whether the architecture has clear prototype and specifications or not
- Interoperation: To indicate if the architecture is based on interoperation or integration (later in this section, there will be a comparison between the two terms)

Discussion

- WSIGS: This architecture uses Web services not Grid services. It realizes two-ways communications. It is not scalable because it is a one component gateway that has to serve multiple requests, from agents and Web services, which may create a bottleneck problem. It has a specification that is not so clear. It uses integration not interoperation.
- WS2JADE: This architecture uses Web services not Grid services. It is a one-way solution, from Jade agents to Web services. It is not scalable because for each agent request, it will trigger the component and search in the UDDI, and for every matching Web service, the corresponding ontology and interaction models will be generated, and also the agent that will register these services in the DF, and all of this is done, even if no one wants the Web service. It is not specified clearly. It uses interoperation.
- WSDL2Agent: It uses Web services not Grid services. It is one-way only, from agent to Web service. It is not scalable because each time a WSDL may change, the proxy agent must be generated again (code and ontology). It is not specified clearly. It uses interoperation.
- Autonomous Semantic Grid: It uses Grid services. It is a two-way architecture. It is not scalable because it uses only one component to handle all the requests of agents and Grid services. It is not specified clearly. It uses interoperation.

Interoperation VS. Integration

Integration and interoperation are two terms that are used interchangeably; The term "integration" indicates that the given components are gathered together into a single schema, whereas the term "interoperation" means that the given components work together.

Usually, integration is not favorable because it would violate the autonomy of the participating components, besides, reasons of maintainability make integration undesirable and interoperation of autonomous entities more reasonable.

CHAPTER 4

AgentGrid Gateway

4.1 Requirements

The solution presented in this chapter, the AgentGrid Gateway, solves the problem of Agents, Grid services interoperation by opening sockets (end-points) [38] for the Grid Platform and sockets for the Agent platform. Using these sockets, messages will be exchanged between the two platforms and they will be transmitted over a transport protocol like the TCP/IP on prespecified port numbers.

The AgentGrid Gateway answers the following requirements:

- Software agents can discover Grid services, and can invoke them flexibly
- Grid services can discover software agents, and can invoke them flexibly
- Bridging the gap between asynchronous behaviour of agent communication and synchronous behaviour of Web service interactions
- Agents will send ACL messages for communicating with Grid services and these messages will be transformed into SOAP calls that Grid services can understand
- Grid services will send SOAP calls, and these calls will be transformed into appropriate ACL messages that agents can understand
- Grid services will have automatically generated stubs, so in order to invoke Grid services, these stubs must be instantiated dynamically
- Various exceptions may be raised in this context, and they must be caught and handled, so that applications built upon this Gateway can become more reliable and dependable

4.2 The Architecture

The solution architecture is composed of two main components and two protocols that enable the use of these components, as shown in figure 4.1:

- In the Agent platform, there are:
  - The Agent Component, which will enable agents of invoking Grid services
  - The ServiceInvocationProtocol, which gives the agent client a flexible way to specify a Grid service to be invoked
- In the Grid platform, there are:
  - The Grid Component, which will enable Grid services of invoking agents
  - The AgentInvocationProtocol, which gives the Grid service client a flexible way to specify an agent to be invoked

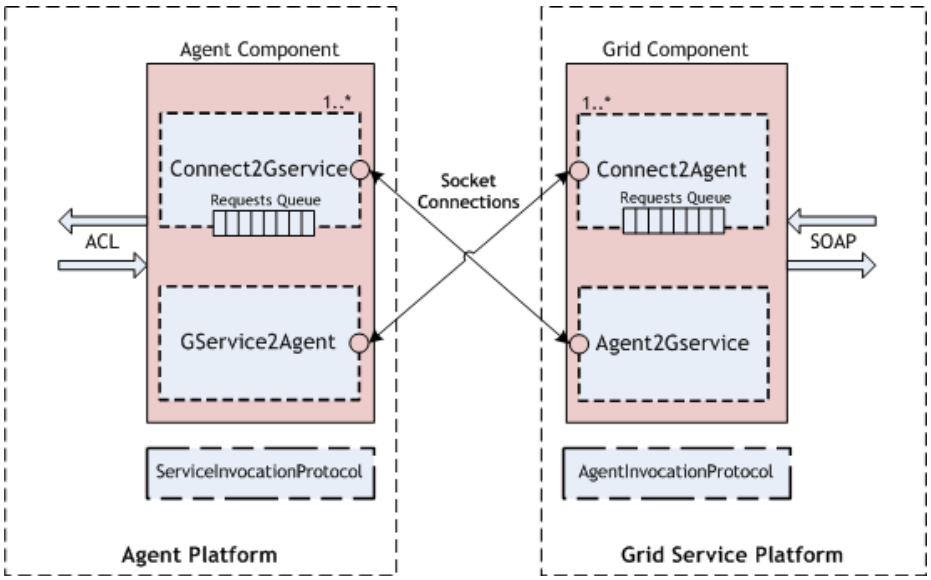

Figure 4.1: Agent Component & Grid Component

The Agent Component and the Grid Component will talk to each other by opening communication sockets using prespecified port numbers and using the TCP/IP transmission protocol. They will work on connecting agents with Grid services, and Grid services with agents.

We supposed that an agent already knows that he will contact and invoke a Grid service, and also a Grid service already knows that it will contact an agent to ask for certain actions. So, we provided agents with a special protocol that they can use to talk to Grid services and invoke them (the ServiceInvocationProtocol). For Grid service clients, we also provided them with a special protocol that enables them from contacting agents and get services from them (AgentInvocationProtocol).

Inside the Agent component we can find two special agent services:

- GridGate: It will act as a server for agents to invoke Grid services.
- AgentCaller: It will invoke agent services in response to AgentGate's requests.

Inside the Grid component we can find two special Grid services:

- AgentGate: It will act as a server for Grid services to invoke Agent services.
- GServiceCaller: It will invoke Grid services in response to GridGate's requests.

GridGate

The GridGate (agent) will have a common port number with the GServiceCaller (Grid service).
It is an agent service that is registered in the DF registry, so that other agents can search for it and communicate with it using usual ACL messages in the purpose of invoking a Grid service. It has a queue for agent requests, so that each request (ACL message) it receives will be put in the queue if there is a request in processing, to be processed later in its turn.

Processing an ACL request comprises parsing the request to extract the information about the service to be invoked, then reformulate these information into another message and send it through the socket connection to the GServiceCaller service in the Grid platform. When the GServiceCaller service receives a message from the GridGate agent, it will:

- Extract the service name, requested methods names and their arguments
- Send this service name to the UDDI registry to get the service URI
- Create an endpoint reference to the service using the URI
- Use the service stubs by the Java reflection mechanism
- Invoke the requested method with the supplied arguments by sending a SOAP request to the Grid service
- Get the SOAP response from the Grid service
- Form a new message that contain the result and send it through the socket to the GridGate agent

When the GridGate get the answer, it will form an ACL response and send it to the requesting agent (see figure 4.2).

AgentGate

The AgentGate (Grid service) will work in a similar way to the GridGate agent. It will have a common port number with the AgentCaller (Agent).
It is a Grid service that is registered in the UDDI registry, so that Grid service clients can search for it, invoke it and communicate with it using usual SOAP messages in the purpose of invoking an agent service. It has a queue for Grid service clients requests, so that each request (SOAP message) it receives will be put in the queue if there is a request in processing, to be processed later in its turn.

Processing a SOAP request comprises parsing the request to extract the information about the agent service to be invoked, then reformulate these information into another message and send it through the socket connection to the AgentCaller agent in the Agent platform. When the AgentCaller agent receives a message from the AgentGate service, it will:

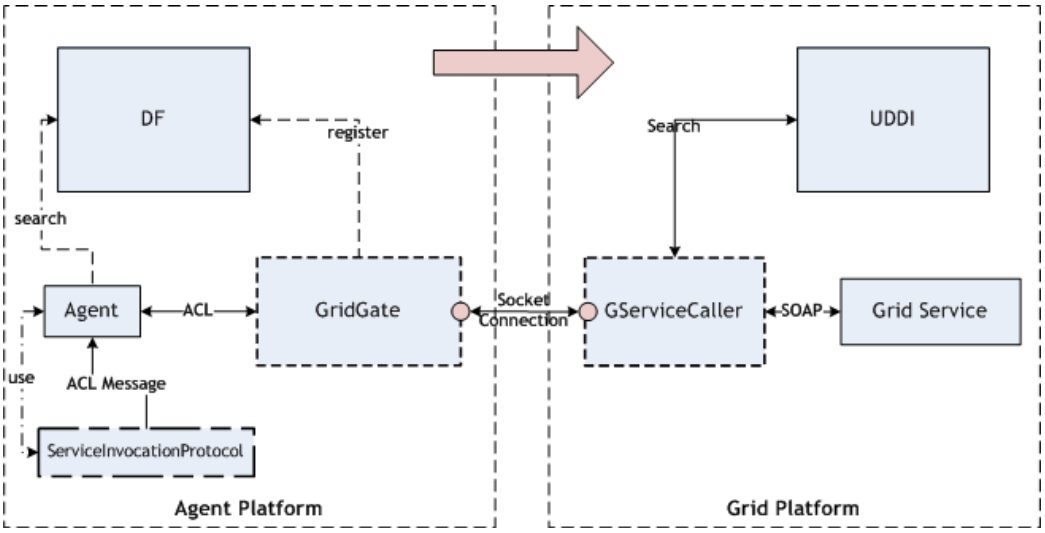

Figure 4.2: Grid service invocation from the Agent Platform

- Extract the agent name, requested action names and the other information related to the request
- Search for this agent in the DF registry
- Formulate the requested action and the request information into an ACL message and send it to the specified agent

According to the requested action, the AgentCaller may or may not send a reply to the AgentGate service. See figure 4.3.

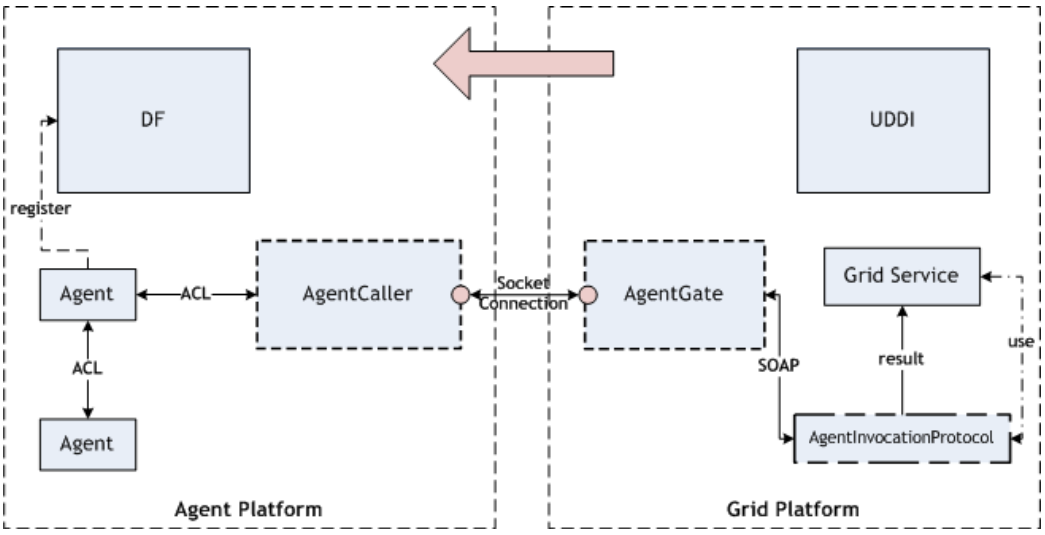

Figure 4.3: An agent invocation from the Grid Platform

As mentioned previously, there are two types of protocols that enable an agent from contacting a Grid service and enable a Grid service from contacting an agent: The ServiceInvocationProtocol and the AgentInvocationProtocol.

ServiceInvocationProtocol

The automatic invocation of a Grid service by an agent is a complex task because of the great differences between agents and Grid services that are mentioned previously in this research. Thus, we implemented the ServiceInvocationProtocol, which hides the complexities of technologies transformations and enables an agent from invoking a Grid service in a clear simple way, by providing special methods that the agent can use to give the service name, the methods, and for each methods specify the list of arguments that the method takes.
Then the protocol will gather the information supplied by the agent and will return to him these information but formed into an ACL message that the agent will send to the GridGate agent.

AgentInvocationProtocol

This protocol enables a Grid service client from invoking an agent service in a clear simple way, by providing special methods that the Grid service client can use to give the agent name, the required action to be performed, and any further possible information that can be sent to the agent (content).
Then the protocol will gather the information supplied by the Grid service client and will return to him these information but formed into a SOAP message that the Grid Service client will send to the AgentGate service.

The GServiceACL The AgentGrid Gateway works on bridging the gap between asynchronous agent communications and synchronous Grid service communications, so when a Grid service invokes a SOAP call on this gateway, this synchronous call will be transformed into asynchronous ACL message. In the other way, when an agent wants to send an ACL message to invoke a Grid service, the gateway define a new ACL message called the GServiceACL. This GServiceACL message is an extension of an ACL message that will be used especially for invoking Grid services.

4.3 Design and Specification

4.3.1 The Use Case Diagram

The Agent Gateway

As we can see in figure 4.4, the Agent Gateway has the following actors:

- Agent Client, which represents any FIPA-compliant agent that wants to invoke a Grid service, and it can perform the following use cases:
  - Search for an agent in the DF
  - Create an ACL request for invoking a Grid service using the ServiceInvocationProtocol
  - Send ACL requests and receive ACL responses
- GridGate, which is the agent that is capable of connecting to the Grid platform to invoke a Grid service. It has the following use cases:

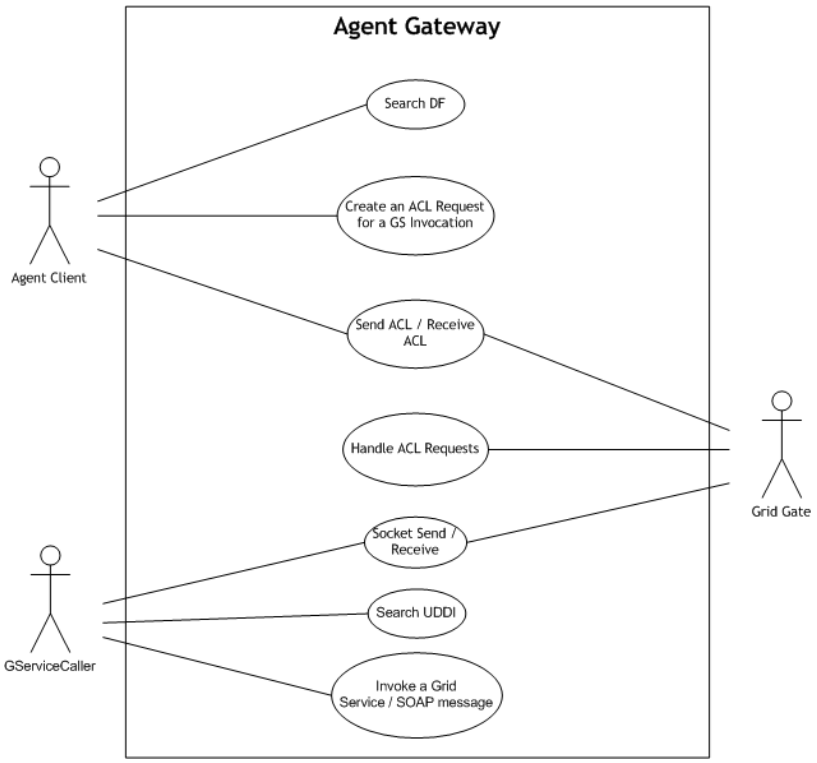

Figure 4.4: Use Case - Agent Gateway

  - Receive ACL requests for invoking Grid services, and sending ACL responses
  - Handle ACL requests, by putting them in a request queue, and processing them one after another
  - Send processed ACL request through the socket connection to the Grid platform so that to invoke the corresponding Grid service, and then receive the result of invoking the service which is returned also through the socket connection

- GServiceCaller, which is a Grid service in Grid platform, responsible of:
  - Receiving the socket requests, and sending the responses back through the socket
  - Searching the UDDI for the Grid service URI and name space information
  - Invoking the corresponding Grid service

The Grid Gateway

In figure 4.5, we can see that the Grid Gateway has the following actors:

- Grid Service Client, which represents any Grid service client that wants to invoke an agent, and it can perform the following use cases:
  - Search for a Grid service in the UDDI

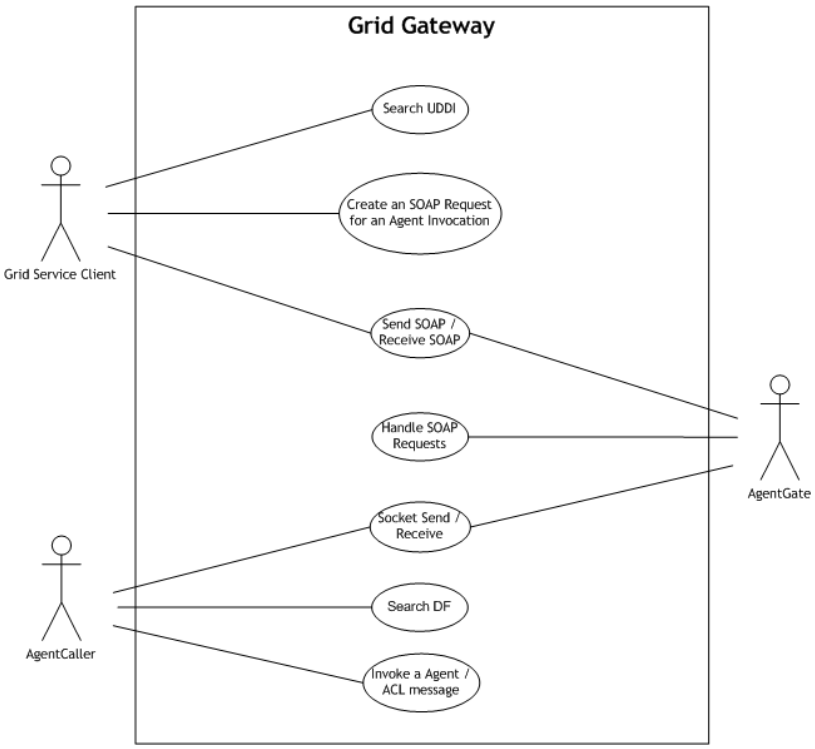

Figure 4.5: Use Case - Grid Gateway

  - Create an invocation request for invoking an agent using the AgentInvocation-Protocol

- AgentGate, which is the Grid service that is capable of connecting to the Agent platform to invoke an agent. It has the following use cases:

  - Receive SOAP requests for invoking agents, and returning the results back
  - Handle the requests, by putting them in a request queue, and processing them one after another
  - Send processed SOAP request through the socket connection to the Agent platform so that to invoke the corresponding agent, and then receive the result of invoking the agent which is returned also through the socket connection

- AgentCaller, which is an agent in Agent platform, responsible of:

  - Receiving the socket requests, and sending the responses back through the socket
  - Searching the DF for the agent ID
  - Invoking the corresponding agent

4.3.2 The Sequence Diagram

Grid Service Invocation Sequence

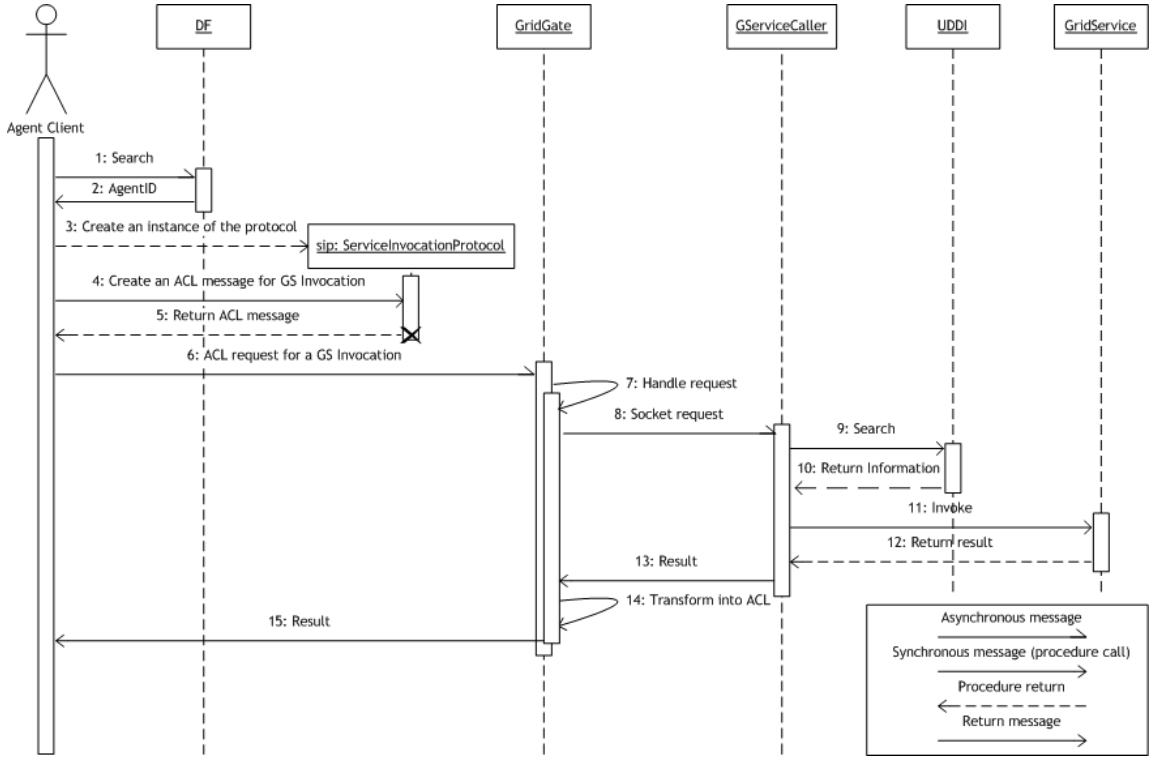

Figure 4.6: Sequence Diagram - Grid Service Invocation

As shown in figure 4.6, the interactions between the different architecture components can be represented by this sequence diagram. This diagram demonstrates the exchanged messages and calls during the process of a Grid service invocation by some FIPA agent. These messages and calls are numbered by the order of occurrence, and the process can be explained as follows:

1. An agent client will search the DF for an agent service that can communicate with the Grid platform
2. The DF will return the agent ID of the requested agent
3. The agent client will create an instance of the protocol ServiceInvocationProtocol
4. The agent will use the protocol to specify the information concerning the Grid service that he wants to invoke
5. The protocol will return an ACL message that contains the invocation information to the agent client
6. The agent client will send this ACL message to the agent ID obtained from (2), which is called, the GridGate agent

7. The GridGate agent will receive the request and handle it, by either putting it in the request queue or by scanning it and extracting the invocation information
8. The GridGate will send the extracted invocation information through the socket connection to the Grid platform
9. The GServiceCaller (Grid service) will receive the socket request, will extract the service name, and will search for its information like the URI and name space by querying the UDDI registry
10. The UDDI will return the service information
11. The GServiceCaller will invoke the service (synchronous procedure call)
12. The Grid service will return a result
13. The GServiceCaller will send the result over the socket back to the GridGate agent
14. The GridGate agent will receive the result from the socket, and he will transform it into an ACL message
15. The GridGate agent will send the ACL message that contains the result back to the agent client

Agent Invocation Sequence

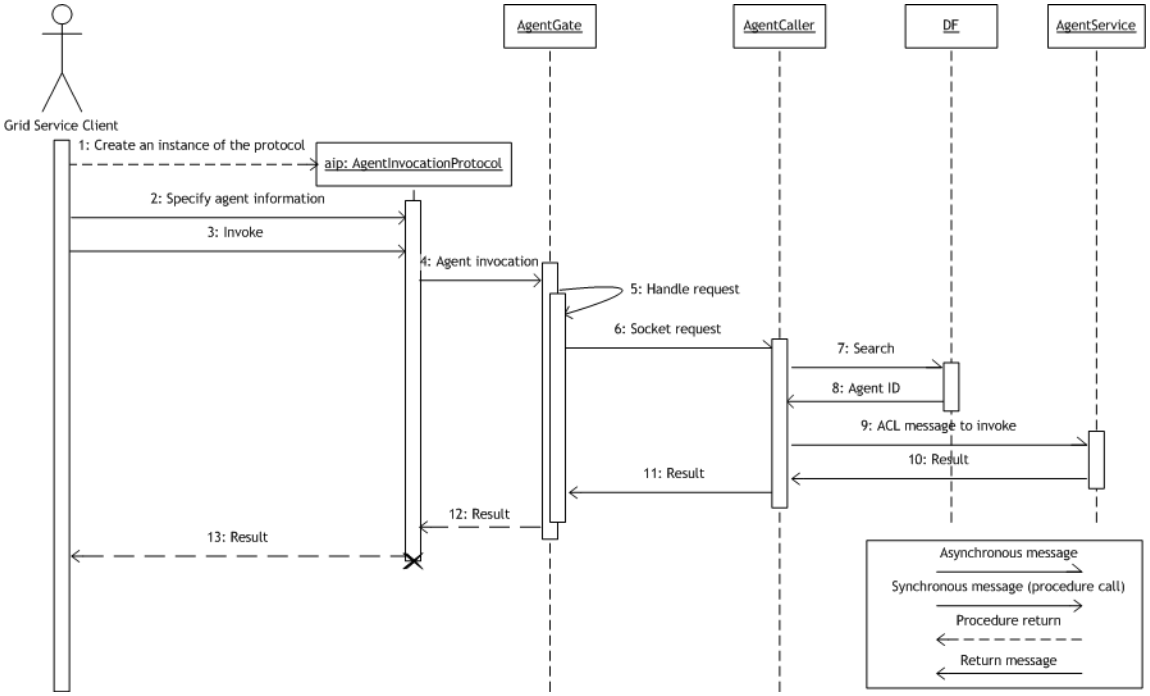

Figure 4.7: Sequence Diagram - Agent Invocation

When a Grid service client wants to invoke an agent, this invocation will have a sequence of interactions, exchanged messages and calls. This is demonstrated by figure 4.7:

1. A Grid service client will create a new instance of the AgentInvocationProtocol
2. He will specify to the protocol all the information related to the agent he wants to invoke
3. He will send a synchronous invoke SOAP message
4. The protocol instance will send a SOAP message to the AgentGate Grid service
5. The AgentGate service will receive the request and handle it, by either putting it in the request queue or by scanning it and extracting the invocation information
6. The AgentGate service will send the extracted invocation information through the socket connection to the agent platform
7. The AgentCaller (agent) will receive the socket request, and extract the agent information, then it will search for the agent in the DF registry
8. The DF will return the agent ID
9. The AgentCaller will send an ACL message to the agent to invoke it
10. The agent will return a result
11. The AgentCaller will send the result over the socket back to the AgentGate service
12. The AgentGate service will receive the result and will send it back to the protocol as a SOAP message
13. The protocol will send the result SOAP message back to the requesting client

4.3.3 The Class Diagram

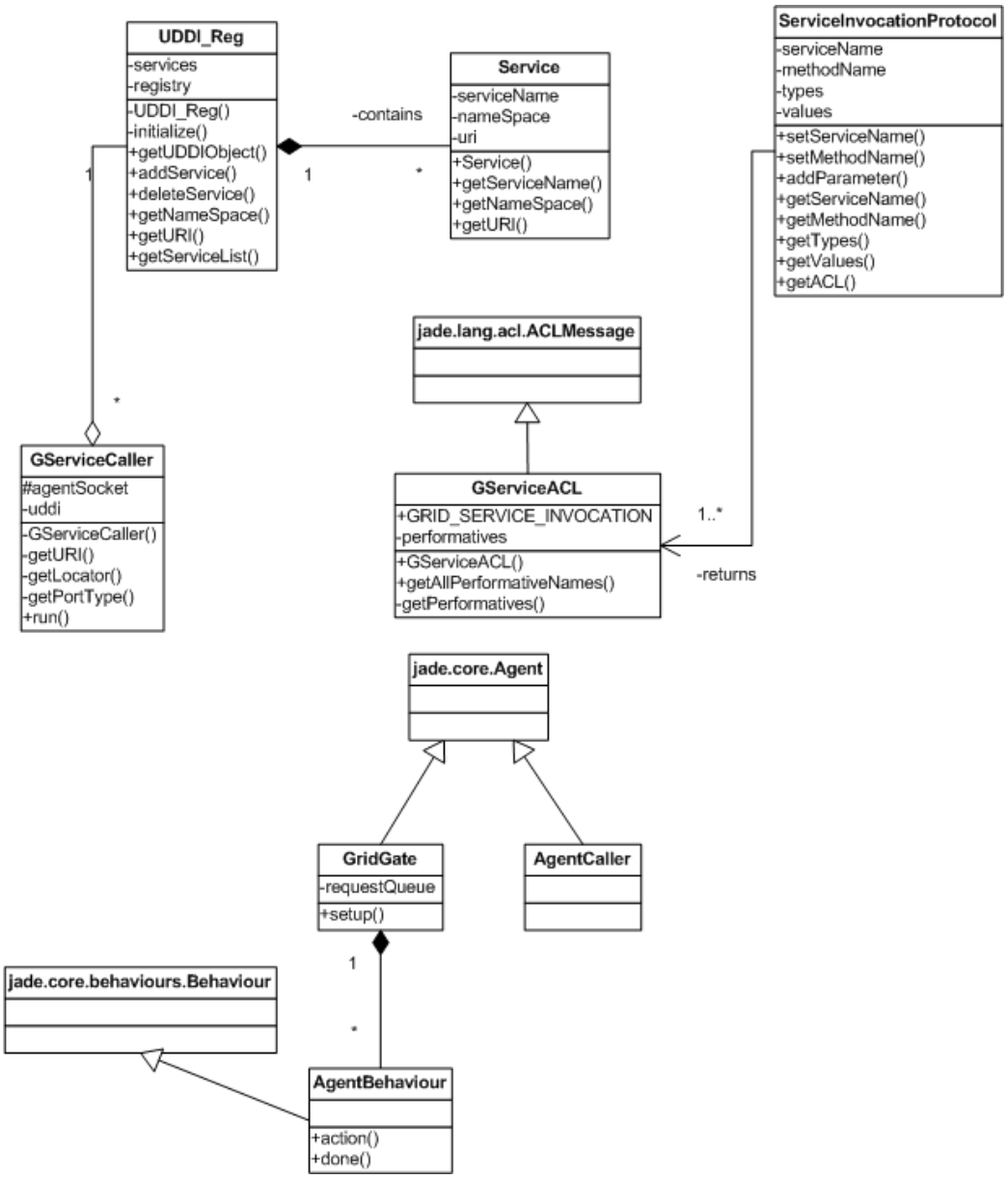

Figure 4.8: Class Diagram

The class diagram is shown in figure 4.8, it contains the following classes:

- UDDI_Reg: This class is a singleton, it represents the UDDI registry, in which all Grid services will be registered. It has a vector of services and a registry object, and has methods for setting and getting information related to services.
- Service: This class contains the Grid service information (name, namespace, uri) with methods to set and get them. It represents a part of the UDDI_Reg class.
- GServiceCaller: This is the service that is responsible of answering socket requests

and invoking specified Grid services. It has a socket instance for the agent, and it deals with the UDDI_Reg object.

- GService: It is the new type of ACL message that is used to invoke Grid services.
- GridGate & AgentCaller: They are both agents, they extend the jade.core.Agent class.

4.4 Analysis Criteria

The same analysis criteria that has been used in the previous chapter will be used, in order to compare our proposed gateway with the related works, and clarify its added values.
By noticing table 4.1, we can see that, the AgentGrid Gateway has verified all of the attributes of the analysis criteria, in that:

	GS	2-ways	Scalable	Specification	Interoperation
WSIGS	No	Yes	No	Yes	No
WS2JADE	No	No	No	No	Yes
WSDL2Agent	No	No	No	No	Yes
AS Grid	Yes	Yes	No	No	Yes
AgentGrid Gateway	Yes	Yes	Yes	Yes	Yes

Table 4.1: Analysis Criteria

- It uses Grid services
- It is a two-ways architecture, from agents to Grid services and vice versa
- It is scalable because, it implements the multi-threading property, so that, it creates a new thread for each request it receives (whether from an agent or a Grid service)
- It is well specified and designed, with clear diagrams that clarify the functionality, and it is also implemented
- It verifies the interoperation

After noticing the analysis criteria, we can see that the AgetGrid gateway has extended the state of the art with new features, like scalability and the clear specifications and Grid services using.

CHAPTER 5

Experimentations

5.1 Introduction

The AgentGrid gateway presented in this research can be used to realize a number of daily real applications. This gateway will provide such applications with agent capabilities like: autonomy, decision making, negotiation, and intelligence combined with the robustness, security, numerous resources, and high computations and efficiency that are obtained from the Grid.

There are several examples of such applications, which may require the capabilities of agents to exhibit intelligent behaviours, and the capabilities of the Grid to optimize computations and performance. One of these examples that worth being mentioned is the applications that are based on knowledge discovery and data mining techniques. This kind of applications is known to be extremely costing and time consuming, so they may make use of the Grid to accelerate the whole process and improve performance, and also use agents to help with the selection of the optimized deterministic knowledge model.

5.2 The Scenario

5.2.1 Problem

The process of information searching in big sets of data (KDD - Knowledge Discovery in Databases) consists of different steps [39]. In this scenario, we are interested in the data mining step, after the creation of the description language (relevant attributes that represent extracted information from database).

Let's consider that a user - starting from a big predefined description language - wants to build a classification procedure that has the best performance. To do this, the user will have to choose the most adapted classification algorithm to the description language. But, because of the huge volume of the description language, the applying of any data mining algorithm will take too much time. In addition, for each algorithm there are many parameters, and for each parameter there are many values. And they must be all experimented to decide which one will yield the best performance.

In the following, we will propose a solution to this problem based on our architecture.

5.2.2 Solution

The user must send the request to an agent (first-level agent) who must analyze the description language to determine a set of classification algorithms that is capable of dealing with the language. Once this set is created, this agent will distribute each algorithm to an agent (second-level agent). These agents of the second-level, must associate the algorithm that has different parameters values with an agent (third-level agent). Then, each agent of the third-level must send the request to a Grid service, the request contains the classification algorithm with specific values.
The request sending process is explained in figure 5.1.

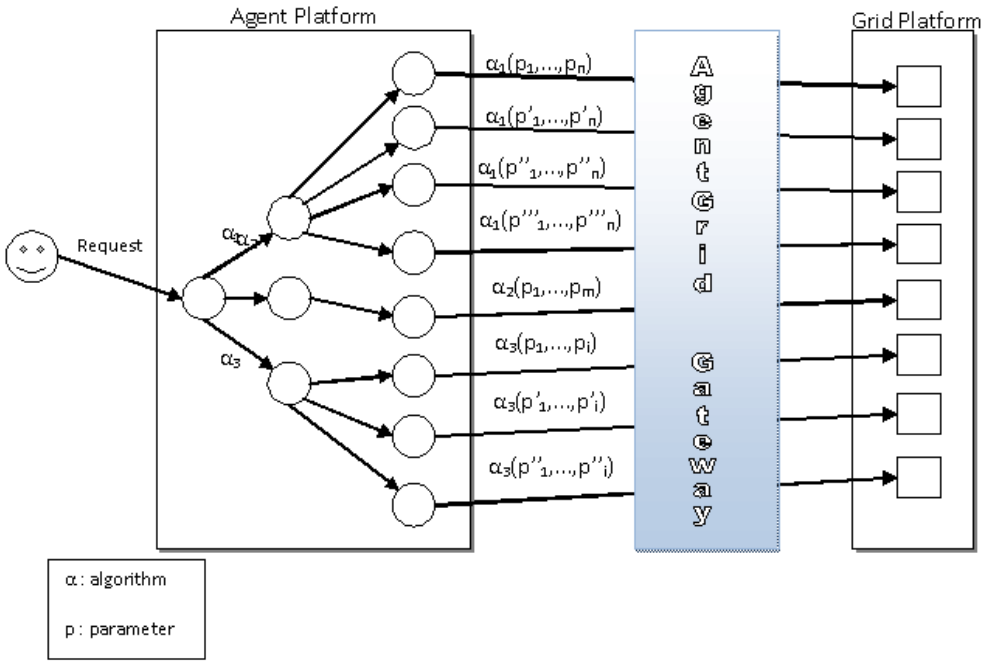

Figure 5.1: Request (algorithm and parameters)

Once a Grid service has handled a request and generated a classification procedure, it will send it back to the requesting agent (third-level). When all the procedures are created, the third-level agents that received the same classification algorithm, must negotiate to specify the best parameters values. Then, every group of agents will return the chosen values and the procedure to the algorithm requester.
In the same way, the second-level agents will also negotiate to determine the best returned procedure. Then at last, they will return to the first-level agent, the parameters, the optimal classification procedure and the applied algorithm.
These returned data will be send back by the first-level agent to the requesting user. The process of returning the response is explained in figure 5.2.

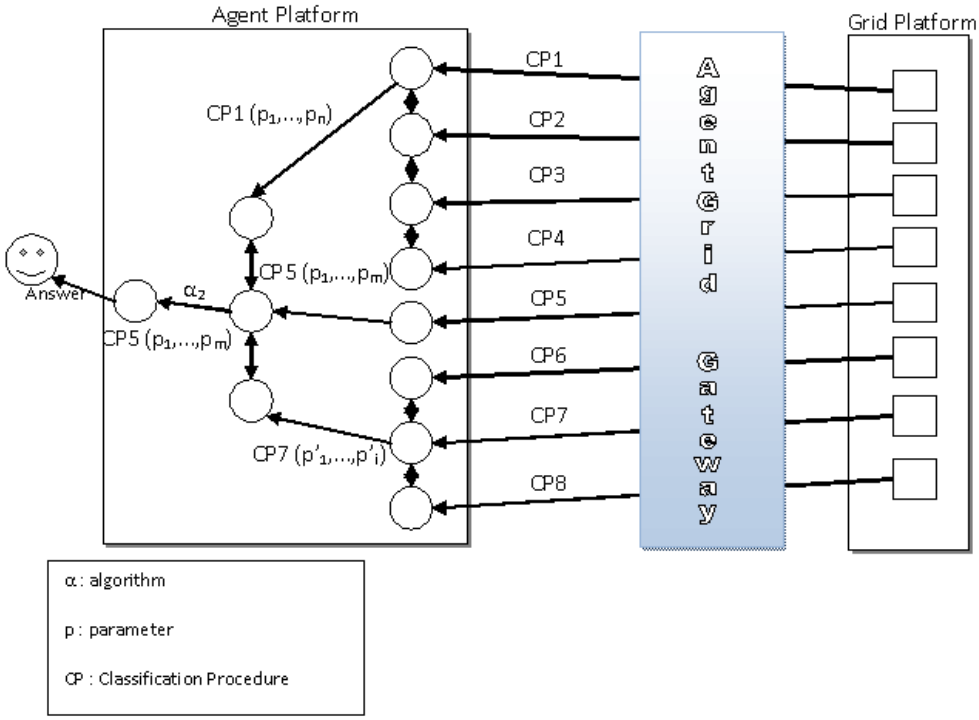

Figure 5.2: Response (algorithm, parameters and procedure)

Conclusion

Grid and multi-agent systems are two kinds of distributed systems, but with different perspectives. Grid focuses on a reliable and secure resource sharing infrastructure, while multi-agent systems focus on flexible and autonomous behaviour in uncertain and dynamic open environments.

The service-oriented architecture (SOA) describes a software architecture that defines the use of loosely coupled software services to support quickly new business requirements. The SOAs have been applied to the actual Grid computing in the form of OGSA, that defines the notion of a Grid service as an extension of a Web service, but with additional significant features like: service state and service lifetime management.

Multi-agent systems have also followed the path towards SOA, in that, agents are producers, consumers and brokers of services.

Although Grid and MAS are totally two independent domains, they meet with the concept of service, and there are good opportunities of exploiting synergies between them, by bridging the gap between them, so that to be able of building more robust autonomous open-systems.

This research has proposed the AgentGrid gateway for achieving the interoperation between software agents and Grid services in the domain of service-orientation. This gateway provides the bridge between Grid world and agent world, enabling them from discovering each other, communicate, cooperate, and complement the strength and capabilities of each other. It depends on opening socket connections between the two platforms, and translate messages and invocation calls, so that, an agent would be able of invoking a Grid service and vice versa.

There are several existing related works that are explained in chapter 3. They are analyzed by a specific criteria and are compared to the AgentGrid gateway.

The AgentGrid gateway presents a scalable mechanism, and it has the advantage of preserving Grid and agents specifications and standards without changing. This gateway has faced the various technical challenges that are described in chapter 1. It was implemented and provided with a clear specification and design diagrams that are demonstrated in chapter 4. These diagrams illustrate the different gateway's components and their functionalities.

We have also proposed in this research two protocols that provide a simple flexible means for agents to invoke Grid services and for Grid services to invoke agents. Then, as to

show how this gateway can be used, an example scenario was demonstrated to clarify the role that the gateway can play, in order to improve the performance and efficiency of some applications.

As a future work, we may consider studying the idea of defining new specifications of a notion of an intelligent service. This service will gather properties from both Grid services and software agents, and will give the best optimized structure that has the capabilities of the two.

BIBLIOGRAPHY

[1] Munindar P. Singh and Michael N. Huhns.
Service-Oriented Computing: Semantics, Processes, Agents.
WILEY, 2005.

[2] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke.
Grid services for distributed system integration.
Computer, 35(6):37–46, 2002.

[3] Nick Jennings and Michael Wooldridge.
Software agents.
IEE Review, pages 17–20, January 1996.

[4] Ian Foster, Nicholas R. Jennings, and Carl Kesselman.
Brain meets brawn: Why grid and agents need each other.
pages 8–15, 2004.

[5] Yolanda Gil.
On agents and grids: Creating the fabric for a new generation of distributed intelligent systems.
Journal of Web Semantics, Volume 4, Issue 2, June, 2006.

[6] Mobeena Jamshed, Kashif Iqbal, Arshad Ali, H. Farooq Ahmad, and Hiroki Suguri.
Integration of agents with web services and grid computing: Design and prototype implementation.
International Workshop on Frontiers of Information Technology, 2003.

[7] David De Roure, Nicholas R. Jennings, and Nigel R. Shadbolt.
The semantic grid: Past, present and future.
July 2004.

[8] Christian Vecchiola, Alberto Grosso, Roberto Podesta, and Antonio Boccalatte.
Design and implementation of a grid architecture over an agent-based framework.
.NET Technologies 2006.

[9] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS.
Fipa abstract architecture specification.

[10] Arturo Avila-Rosas, Luc Moreau, Vijay Dialani, Simon Miles, and Xiaojian Liu.
Agents for the grid: A comparison with web services (part ii: Service discovery).

[11] Joshy Joseph and Craig Fellenstein.
Grid Computing.
IBM Press, December 2003.

[12] Daniel A Reed, Celso L Mendes, Chang da Lu, Ian Foster, and Carl Kesselman.
The Grid 2: Blueprint for a New Computing Infrastructure - Application Tuning and Adaptation.
Morgan Kaufman, San Francisco, CA, second edition, 2003.
pp.513-532.

[13] Ian Foster, Carl Kesselman, and Steven Tuecke.
The anatomy of the grid: Enabling scalable virtual organizations.
2001.

[14] Maozhen Li and Mark Baker.
The Grid: Core Technologies.
Wiley, 2005.

[15] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke.
The physiology of the grid.
June 2002.

[16] Carmela Comito, Domenico Talia, and Paolo Trunfio.
Grid services: principles, implementations and use.
Int. J. Web and Grid Services, 1(1):48–68, 2005.

[17] N. R. Jennings, K. Sycara, and M. Wooldridge.
A roadmap of agent research and development.
Journal of Autonomous Agents and Multi-Agent Systems, 1(1):7–38, 1998.

[18] A. S. Rao and M. P. Georgeff.
BDI-agents: from theory to practice.
In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[19] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS.
Fipa acl message structure specification.

[20] Olivier Gutknecht and Jacques Ferber.
The MADKIT agent platform architecture.
In *Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55, 2000.

[21] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa.
Jade - a white paper.
exp - Volume 3 - n. 3 - September 2003, 2003.

[22] M. Luck, P. McBurney, and C. Preist.
Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing).
AgentLink, 2003.

[23] Jonathan Dale, Akos Hajnal, Martin Kernland, and Laszlo Zsolt Varga.
Agentcities technical recommendation - integrating web services into agentcities recommendation.
Technical report, 28 November, 2003.

[24] *WSDL2Agent Tool http://sas.ilab.sztaki.hu:8080/wsdl2agent/.*

[25] AGENTCITIES.
Web Service Agent Integration Project, http://wsai.sourceforge.net/index.html.

[26] Dominic Greenwood and Monique Calisti.
Engineering web service - agent integration.
IEEE, 2004.

[27] Dominic Greenwood and Monique Calisti.
An automatic, bi-directional service integration gateway.
2004.

[28] Thang Xuan Nguyen and Ryszard Kowalczyk.
Ws2jade: Integrating web service with jade agents.
Faculty of Informtaion and Communication Technologies Centre of Intelligent Agents and Multi-Agent Systems, 30 July 2005.

[29] Laszlo Zsolt Varga, Akos Hajnal, and zsolt Werner.
The wsdl2agent tool.
Software Agent-Based Applications, Platforms and Development Kits - Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 197–222, 2005.

[30] M. Omair Shafiq, Arshad Ali, Hiroki Suguri, and H. Farooq Ahmad.
Agentweb gateway integration of fipa multi agent system and w3c web service system.
GGF16 Semantic Grid Workshop, 2005.

[31] Maruf Pasha, Hafiz Farooq Ahmad, and Hiroki Suguri.
Autonomous semantic grid: Architecture and implementation.
OGF19 - The 19th Open Grid Forum, 2006.

[32] Takuhiro Kimura, Taro Ishikawa, and Hiroki Suguri.
Design and implementation of agent-based semantic grid framework.
Proceedings of the 13th Workshop on Multimedia Communications and Distributed Processing Systems (ISSN 1344-0640, Information Processing Society of Japan Symposium Series Vol. 2005, No.19), pages 97–102, 2005.

[33] Chunlin Li and Layuan Li.
Competitive proportional resource allocation policy for computational grid.
Future Gener. Comput. Syst., 20(6):1041–1054, 2004.

[34] O. Rana and L. Moreau.
Issues in building agent based computational grids.

[35] Junwei Cao, Stephen A. Jarvis, Subhash Saini, Darren J. Kerbyson, and Graham R. Nudd.
Arms: An agent-based resource management system for grid computing.
Sci. Program., 10(2):135–148, 2002.

[36] Zhiyong Feng, Qiming Cheng, and Fei Zhao.
Crutch: An integration framework forweb service and mas.
In *ICEBE '06: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 545–548, Washington, DC, USA, 2006. IEEE Computer Society.

[37] A. Min Tjoa, Peter Brezany, and Ivan Janciak.
Towards grid based intelligent information systems.

[38] Elliotte Rusty Harold.
Sockets for Servers - Java Network Programming, 2nd Edition, Chapter 11.
O'Reilly, 2000.

[39] Ryszard S. Michalski and Kenneth A. Kaufman.
Machine Learning and Data Mining: Methods and Applications, chapter 2 - Data Mining and Knowledge Discovery: A Review of Issues and a Multistrategy Approach.
John Wiley & Sons Ltd, 1997.