AUTONOMOUS DISTRIBUTED SERVICE SYSTEM: CONCEPT, DESIGN AND IMPLEMENTATION

By

Kashif Iqbal
2000-NUST-BIT-211

Project Supervisor
Dr. Farooq Ahmad

Project Report in partial fulfillment of the requirements for the award of Bachelor of Science degree in Information Technology (BIT)

In

NUST Institute of Information Technology (NIIT)
National University of Sciences and Technology (NUST)
Rawalpindi, Pakistan
(2004)

Certified that the contents and form of thesis entitled "Autonomous Distributed service system: Concept, Design and Implementation**" submitted by Kashif Iqbal have been found satisfactory for the requirement of the degree.**

Supervisor: _____
Associate Professor (Dr. H Farooq Ahmad)

Member: _____
Professor (Dr. Arshad Ali)

Member: _____
Associate Professor (Mr. Shahzad Khan)

Member: _____
Lecturer (Mr. Mohammad Aatif)

Member: _____
Lecturer (Mr. Zaheer Abbas Khan)

DEDICATION

In the name of Allah, the Most Beneficent, the Most Merciful

To my dear
Family especially to my Mother,

ACKNOWLEDGEMENTS

I am deeply beholden to my supervisor Associate Professor Dr. H Farooq Ahmad for his continuous assistance, inspiration, and patience. I am highly gratified to Professor Dr. Arshad Ali for his continuous and valuable suggestions and guidance, especially for the provision of all kinds of facilities throughout my thesis work. His ability of management and foresightedness trained me a lot of things which will be more helpful for me in my practical life.

I would like to express my gratitude to Mr. Shahzad Khan (Faculty member), Mr. Mohammad Aatif (Faculty member) and Mr. Zaheer Abbas Khan (Faculty member) for their valuable implications and comments to improve this dissertation.

I am highly thankful to all of my professors especially, Mr. Shahzad Khan, whom had been guiding and supporting me through out my course and research work. Their knowledge, guidance and training enabled me to carry out this research work I would also like to show gratitude to Dr. Hiroki Suguri, Comtec, Japan for his valuable suggestions.

I would like to offer my admiration to all my classmates, and my seniors who had been supporting, helping and encouraging me throughout my thesis project especially, Mr. Abdul Ghafoor, Mr. Mujahid ur Rehman and Mr. Nasir Rasul. I am especially thankful to Mobeena Jamshed who had been working sincerely with me in development of this architecture.

I would like to offer my thanks to all of my colleagues in CERN research lab, whom have had some input or influence over the course of my research. I am also indebted to system administration for their help and support.

I would like to offer appreciation to my parents for their vision and commitment to make me learn from my babyhood, and other family members for their encouragement and support, especially my uncle Dr. Liaqat Ali.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

OGSA	Open Grid services Architecture
OGSI	Open Grid services Infrastructure
DAML-S	DARPA Agent Markup Language for Services
DAML-OIL	DARPA Agent Markup Language – Ontology Inference Layer
WSDL	Web service Description Language
XML	Extensible Markup Language
JADE	Java Agent Development Framework
FIPA	Foundation for Intelligent Physical Agents
UDDI	Universal Description and Discovery Integration
SOAP	Simple Object Access Protocol
CORBA	Common Object Resource Broker Architecture
Java RMI	Java Remote Method Invocation
IDL	Interface Definition Language
NASSL	Network Accessible Service Specification Language
WDS	Well-Defined Service
HTTP	Hyper Text Transfer Protocol
HTTPS	Secure HTTP
LDAP	Lightweight Directory Access Protocol
XSD	XML Schema Definition
ACL	Agent Communication Language

MAS	Multi-Agent System
KQML	Knowledge Query Manipulation Language
SL	Semantic Language
JMS	Java Messaging Service
JAX-RPC	Java API for XML-RPC
RPC	Remote Procedural Call
GSI	Grid Security Infrastructure
GT3	Globus Toolkit 3
GRAM	Grid Resource Allocation Manager
RFT	Reliable File Transfer
B2B	Business to Business
WWW	World Wide Web
IT	Information Technology
PCs	Personal Computers
RDF	Resource Description Framework
P2P	Peer to Peer
QoS	Quality of Service
WSFL	Web service Flow Language
OWL	Ontology Web Language
AI	Artificial Intelligence
SOA	Service Oriented Architectures
EJBs	Enterprise Java Beans
GWSDL	Grid Web service Description Language

GSDL	Grid service Description Language
TCP/IP	Transmission Control Protocol/ Internet Protocol
SAX	Simple API for XML
DOM	Document Object Model
JAXP	Java API for XML Processing
SMTP	Simple Mail Transfer Protocol
FTP	File Transfer Protocol
SAAJ	SOAP with attachment API for JAVA
API	Application Programming Interface
MIME	Multi-purpose Internet Mail Extensions
OS	Operating System
GUI	Graphical User Interface
IIOP	Internet Inter-ORB Protocol
ORB	Object Resource Broker
MTS	Message Transport Service
DF	Directory Facilitator
AMS	Agent Management System
SSL	Secure Socket Layer
XSL	XML Stylesheet Language
WSDL	Web service Description Language
GASS	Globus Access to Secondary Storage
GSDL	Grid service Description Language

LIST OF FIGURES

ABSTRACT

In both e-business and e-science, we often need to integrate services across distributed, heterogeneous, dynamic "virtual organizations" formed from the disparate resources within a single enterprise and/or from external resource sharing and service provider relationships. This integration is technically challenging because of the need to achieve various levels of quality of service (QoS) when running on top of different native platforms and under dynamic workload conditions. We present an *Autonomous Distributed Service System Architecture* that addresses these challenges. Building on concepts and technologies from the Semantic Web, Multi-Agent Systems, Grid and Web services communities, this architecture put together a proposition made to cope with heterogeneous and continuously changing needs of information processing, service provision and utilization in dynamically evolving environment to meet these requirements. Autonomous Distributed Services Architecture also define agents' capabilities in terms of Web services Description Language (WSDL), so that agents can describe and advertise themselves in UDDI (Universal Description Discovery & Integration) as and when required.

There is an inherent communication gap between software agents and other service oriented architectures (SOA) like Web services and Grids, due to different communication protocols, service descriptions, schema definitions and message structures they use. These issues are addressed in thesis where we have proposed a new way of communication and registration of Agent Services in Grid environment.

Regarding the implementation of proposed architecture, both Grid and Web services are successfully invoked using Software Agents developed in JADE. In addition to this task, registering of a Web service in the Grid index service is also achieved successfully so that Web services could be accessed in Grid environment.

CHAPTER 1

1 INTRODUCTION

Computers are fulfilling an increasingly diversified set of tasks in our society. They assume saliently many key jobs and assist us in managing numerous tasks in our daily life. In one or the other form, they process, and provide us information. Many researchers agree that information age is rapidly replacing industrial era.

There have been three major stages in human history. The first stage is said to be the agrarian base, and the economy of this stage was dependent upon the agricultural activity, but with little knowledge. Industrial revolution, which is coming to end now, was the first major change. The economic success of the era had been based on mass-production and quality of the products. Lastly the third era is the age of information technology and services. In this era, information and knowledge are becoming focus of the economic activity of the entire world. This phase has changed the economic, political and cultural values on global scale. This change has forced state-controlled economy to a free economy oriented towards abundant supply and diversity. Global information systems, such as the Internet, are no longer just pathways for digital data rather they generate and process information to create knowledge for commercial activity, education business and research. For the first time, in the history of human kind, non-material resources such as software and information services have become new raw materials and real wealth of knowledge-base society. [19]

Now that we have discussed the new era, namely Information age, we need to explore what new opportunities and challenges are emerging in the near future, in the Internet economy called Electronic commerce (e-commerce). The Internet traffic is distributed as follows: 60 percent for business, 27 percent for research, 8 percent for administration, 5 percent for education. These percentages are likely to change in favor of corporate sector, which is making use of the Internet to penetrate markets around the globe and for interchange with customers. In 1995, Forester Research predicted that the worldwide users of Internet would be 34.9 million in 1998 but the actual number became in excess of 100 million users. In 1995, Jupiter communications predicted US$3.1billion in business-to-consumer revenue for e-commerce by 1998 but the actual figure turned US$13billion. Successful e-commerce requires rapid adaptation and excellent timing for service providers and users, simultaneously. However, the electronic commerce on the Internet is at the stage of infancy. It opens up vast opportunities but a number of fundamental challenges are to be met to bring it into mature discipline of business.

Distributed system is the backbone of information services on the Internet. However, rapidly evolving and highly diversified world of information services requires huge information processing capacity and service provision on the Internet time scale. But the state of the art of distributed systems is human dominated administered, which cannot meet Internet time scale and quality of service for e-commerce. A critical prerequisite for distributed system technology to comply with the new challenge is that it must be completely self-tuning with autonomous adaptation to evolving workload with “zero” human administration [19].

As we know that the Internet was originally designed to share the information between a small numbers of users, with no quality of service requirements. However, due to the emergence of e-commerce, there is an urgent need to change fundamental philosophy of the underlying system. Information services have become mission critical as heavy loss may result if the system does not provide required functionality and resources to achieve QoS under changing conditions, such as changing workload. The system needs to provide guaranteed quality of services at application levels, not at low level like guaranteed packet delivery. There are different concerns in quality of service, such as timeliness, reliability, and fault tolerance for information service utilization and provision.

A system is called a high-assurance system, when heterogeneous and changing requirement levels of QoS are satisfied. In addition to quality of service, we identify that users have two more basic views of customization and situation regarding information services utilization but these do not exist on the current information service systems as well. Consequently, using information services on the Internet is frustrating experience for most of the users. Many information services on the Internet return poor results- inconsistent, arbitrarily inaccurate or completely irrelevant data or the performance is so poor that the whole service becomes useless. We conclude that current information service systems on the Internet do not provide guaranteed quality of services, customization and situation based information services. There is urgent need for new models for information services for e-commerce in the Internet. If the research community fails to provide necessary technology and framework, the success of e-commerce may be delayed or even may become questionable.

This fosters an urgent need to design an information service system with high-assurance that provides information services to meet the above-mentioned requirements.

The rest of this thesis is arranged as follows. Next chapter gives an overview of Software Oriented Architectures in general and Web services in particular. Chapter 3 and chapter 4 cover various aspects of Grid Computing and Multi-Agent Systems including autonomous service discovery/invocation through software agents and application of Semantic Web technologies for Grid infrastructure and applications. Chapter 5 explains the requirements and need that are identified for Autonomous Distributed Service System after consulting related work and literature. Chapter 6 explains the architectural details of the system including its architectural components and concept. Chapter 7 and Chapter 8 will address the list of experiments which are carried out in the form of phases to test various research hypothesis and tools and technologies which are used to carry out experiments respectively. Chapter 9 describes the various outcomes of implementation carried out and on the basis of which conclusions are drawn. In the end Chapter 10 defines some future tasks followed by conclusions, reference and appendix section.

CHAPTER 2

2 SERVICE ORIENTED ARCHITECTURE

With the introduction of Web services over the last year or so, there has been a renewed interest in service-oriented architecture (SOA). An SOA is an architecture that has special properties. It is an architecture made up of components and interconnections that stress interoperability and location transparency. The term service has been used for more than two decades. For example, leading transaction monitoring software has used the term "service" in the early 1990s. Many client-server development efforts in the 90s used the term "service" to indicate the ability to make a remote method call. Web services have given the term service more prominence in the last few months. Services and service-oriented architectures are really about designing and building systems using heterogeneous network addressable software components.

2.1 SOFTWARE ARCHITECTURE IN GENERAL

Before getting too far into the details of SOA, the term software architecture needs to be defined. Software architecture is a fairly new practice in the field of software engineering. The software architecture of a system consists of the large-grained structures of the software. It describes the components of the system and how those components interact at a high level. The interactions between components are called connectors. The configuration of components and connectors provide both a structural and a behavioral view of the system.

2.2 BRIEF HISTORY OF SOFTWARE DESIGN

Over the last four decades, the practice of software development has gone through several different development methods. Each method shift was made in part to deal with greater levels of software complexity. The way we have managed complexity is to continuously invent coarser grained constructs, such as functions, classes, and components. We can think of these constructs as software "black boxes".

A software black box hides its implementation by providing controlled access to its behavior and data through an interface. Think of it as a software integrated circuit. At a fine level of granularity, we use objects to hide behavior and data. At a coarser level of granularity, we use components to do the same. Having information hiding only at the object level works well for small systems, and it allows us to create constructs in software that map onto the real world objects.

A problem arises when we try to group a large number of objects together. Although access to the objects is controlled through their interfaces, the granularity at the object level still makes dependencies between them difficult to control in a large system.

Introducing the component concept gives us a better way of managing these dependencies in large systems. A component is a smaller group of objects working together to provide a system function. For example, a claim, automobile and a claimant object can work together in a claims component to provide the claim function for a large insurance application. The claim component becomes another black box at the level of a large system function. The claim and automobile objects are not known

to any other part of the system, except for the claim component. This means that no other part of the system can become dependent on these objects since they are completely hidden.

This is the point we are at today in software development. Technologies such as Enterprise Java Beans, .NET and CORBA are effective ways of implementing components. The method of component-based development has allowed developers to create more complex, higher quality systems faster than ever before because we have a better way of managing complexities and dependencies within a software system.

2.3 THE NEED FOR SERVICE-ORIENTED ARCHITECTURE

Now that all of these heterogeneous components have developed, for one system or another, we need to be able to use them together. But there is a problem: developers built some components using EJB, some CORBA, and many of the resources we need for our systems are on mainframe computers running COBOL. Using Enterprise Java Beans requires that method invocation be done via RMI and CORBA uses IIOP. Many times the component is located across the Internet, behind a firewall and the message cannot get through. Although each has tried to be interoperable with the other, small inconsistencies cause developers a lot of pain when trying to figure it all out.

In addition, even if these problems lack, the components still have to know too much about each other. For example, if I want to use a Claims component, I have to know not only what transport and payload type to use, I also need to know exactly where it is. I also need to have intimate knowledge about its interface so that if the

interface changes, I have to change the way I call it. Since these components have network addressable interfaces and the number of clients can be large, this creates dependencies on an enormous scale. So what will be the solution to this problem?

2.4 THE SERVICE-ORIENTED APPROACH

A service is behavior that is provided by a component for use by any other component based only on the interface contract. A service has a network-addressable interface. A service stresses interoperability and a service may be dynamically discovered and used.

Web services consist of four technologies in combination that provides an implementation of an SOA. You can use Web services to provide all of the properties necessary to build a service. Web services include HTTP as the primary network protocol, SOAP/XML for the payload format and UDDI for service registry, and WSDL to describe the service interfaces.

However, a service-oriented architecture does not require Web services. An SOA is a design and a way of thinking about building software components.

2.4.1 Use-Based Solely Published Contract

Contract design between components is a critical activity in a service-oriented architecture. The difference between a public interface and a published interface comes into play. A public interface is an interface that can be used by components within a system. The public interfaces of a component are easier to change, because they are only used by known clients. A published interface is one that is exposed to the

network and may not be changed so easily, because the clients of the published interface are not known. The difference is analogous to an intranet-based site only accessible by employees of the company and an Internet site accessible by anyone. Languages and tools have not stepped up yet to enforcing this concept, but it is important to understand the distinction when building published service interfaces.

2.4.2 Network Addressable Interface

A service must have a network addressable interface. This means that a client on a network must be able to invoke a service. A service may be configured for use by a component in the same machine. However, the service must also support a network configuration.

2.4.3 Stresses Interoperability

A service-oriented architecture, first and foremost, stresses interoperability. It means that each component must provide an interface that can be invoked through a payload format and protocol that is understood by all of the potential clients of the service.

2.4.4 Dynamic Discovery and Invocation

A service must be dynamically discovered. This means that a third party mechanism must be used to find the service. Hard coding of a machine location is not consistent with a service-oriented approach. [31]

2.5 WEB SERVICES: A TYPICAL SERVICE ORIENTED ARCHITECTURE

Web services is emerging as the enabling technology that bridges decoupled systems across various platforms, programming languages and applications. They are "self-contained, self-describing modular applications" (Martin 2001) and interoperability among these applications is ensured through the use of standards such as SOAP, XML, and WSDL. Web services standards define the format of the message, specify the interface to which a message is sent, describe conventions for mapping the contents of the message into and out of the programs implementing the service, and define mechanisms to publish and discover Web services interfaces. The specification to publish and discover Web services on the Internet is defined as Universal Description, Discovery, and Integration (UDDI). [22]

2.5.1 Introduction

Previous attempts at distributed computing (CORBA, Distributed Smalltalk, Java RMI) have yielded systems where the coupling between various components in a system is too tight to be effective for low-overhead, ubiquitous B2B e-business over the Internet. These approaches require too much agreement and shared context among business systems from different organizations to be reliable for open, low-overhead B2B e-business.

Meanwhile, the current trend in the application space is moving away from tightly coupled monolithic systems and towards systems of loosely coupled,

dynamically bound components. Systems built with these principles are more likely to dominate the next generation of e-business systems, with flexibility being the overriding characteristic of their success. Service (application) integration becomes the innovation of the next generation of e-business, as businesses move more of their existing IT applications to the Web, taking advantage of e-portals and e-marketplaces and leveraging new technologies, such as XML.

The Web services architecture describes principles for creating dynamic, loosely coupled systems based on services, but no single implementation. There are many ways to instantiate a Web service by choosing various implementation techniques for the roles, operations, and so on described by the Web services architecture.

Various environmental aspects must also be considered when designing Web services. For example, the security requirements for services brokers will vary depending upon the deployment environment. Most intranet deployments have minimal security requirements but in situations where high-value B2B transactions are conducted, much higher security may be necessary. An approach is to take a risk-assessment view of security and design brokers to provide different levels of information based upon an environment's security infrastructure. [2]

2.5.2 Generic Web service Architecture

2.5.2.1 The Web services Platform

So what is the Web service platform? The basic platform is *XML plus HTTP*. HTTP is a ubiquitous protocol, running practically everywhere on the Internet. XML provides a meta-language in which you can write specialized languages to express

complex interactions between clients and services or between components of a composite service. Behind the facade of a web server, the XML message gets converted to a middleware request and the results converted back to XML.

If we think that access and invocation are only the bare bones (this would be like saying CORBA is only IDL plus remote procedure calls). What about the platform support services -- discovery, transactions, security, authentication and so on -- the usual raft of services that make a platform a platform? That's where we step up to the next level.

The Web needs to be augmented with a few other platform services, which maintain the ubiquity and simplicity of the Web, to constitute a more functional platform. The full-function Web services platform can be thought of as *XML plus HTTP plus SOAP plus WSDL plus UDDI.* At higher levels, one might also add technologies such as XAML, XLANG, XKMS, and XFS -- services that are not universally accepted as mandatory.

Below is a brief description of the platform elements. It should be noted that while vendors try to present the emergent Web services platform as coherent, it's really a series of in-development technologies. Often at the higher levels there are, and may remain, multiple approaches to the same problem.

- SOAP (remote invocation)
- UDDI (trader, directory service)
- WSDL (expression of service characteristics)
- XLANG/XAML (transactional support for complex web transactions involving multiple Web services)

- XKMS (XML Key Management Specification) - ongoing work by Microsoft and Verisign to support authentication and registration

2.5.2.2 SOAP

SOAP is a protocol specification that defines a uniform way of passing XML-encoded data. In also defines a way to perform remote procedure calls (RPCs) using HTTP as the underlying communication protocol. SOAP arises from the realization that no matter how nifty the current middleware offerings are, they need a WAN wrapper. Architecturally, sending messages as plain XML has advantages in terms of ensuring interoperability. The middleware players seem willing to put up with the costs of parsing and serializing XML in order to scale their approach to wider networks. [35] Submitted in 2000 to the W3C as a Note by IBM, Microsoft, UserLand, and DevelopMentor, the further development of SOAP is now in the care of the W3C's XML Protocols Working Group. This effectively means that SOAP is frozen and stable until such time as the W3C Working Group delivers a specification.

2.5.2.3 UDDI (Universal Description, Discovery and Integration Service)

UDDI provides a mechanism for clients to dynamically find other Web services. Using a UDDI interface, businesses can dynamically connect to services provided by external business partners. A UDDI registry is similar to a CORBA trader, or it can be thought of as a DNS service for business applications. A UDDI registry has two kinds of clients: businesses that want to publish a service (and its usage interfaces), and clients who want to obtain services of a certain kind and bind programmatically to them. The table below is an overview of what UDDI provides. UDDI is layered over

SOAP and assumes that requests and responses are UDDI objects sent around as SOAP messages. A sample query is included below.

Information	**Operations**	**Detailed information** (supported by lower-level API)
White pages: Information such as the name, address, telephone number, and other contact information of a given business	**Publish**: How the provider of a Web service registers itself.	**Business information**: Contained in a *BusinessEntity* object, which in turn contains information about services, categories, contacts, URLs, and other things necessary to interact with a given business.
Yellow pages: Information that categorizes businesses. This is based on existing (non-electronic) standards	**Find**: How an application finds a particular Web service.	**Service information**: Describes a group of Web services. These are contained in a *BusinessService* object
Green pages: Technical information about the Web services provided by a given business.	**Bind**: How an application connects to, and interacts with, a Web service after it's been found	Binding information**: The technical details necessary to invoke a Web service. This includes URLs, information about method names, argument types, and so on. The *BindingTemplate* object represents this data.** **Service Specification Detail**: This is metadata about the various specifications implemented by a given Web service. These are called *tModels* in the UDDI specification

Figure 1: UDDI Operations and their Detailed Description

There is no near-term plan in UDDI to support full-featured discovery (e.g. geography-limited searches or bidding and contract negotiation supported by vendors like eLance). UDDI expects to be the basis for higher level services supported by some other standard. There are plans for UDDI to support more complex business logic, including support for hierarchical business organizations. UDDI has fairly broad support; IBM, Ariba, and Microsoft are driving it. It's not yet an open standard. [37]

2.5.2.4 WSDL (Web services Description Language)

WSDL provides a way for service providers to describe the basic format of Web service requests over different protocols or encodings. WSDL is used to describe *what* a Web service can do, *where* it resides, and *how* to invoke it. While the claim of SOAP/HTTP independence is made in various specifications, WSDL makes the most sense if it assumes SOAP/HTTP/MIME as the remote object invocation mechanism. UDDI registries describe numerous aspects of Web services, including the binding details of the service. WSDL fits into the subset of a UDDI service description. [36]

WSDL defines services as collections of network endpoints or *ports*. In WSDL the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions of messages, which are abstract descriptions of the data being exchanged, and port types, which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A port is defined by associating a network address with a reusable binding; a collection

of ports define a service. And, thus, a WSDL document uses the following elements in the definition of network services:

- Types -- a container for data type definitions using some type system (such as XSD).
- Message -- an abstract, typed definition of the data being communicated.
- Operation -- an abstract description of an action supported by the service.
- Port Type -- an abstract set of operations supported by one or more endpoints.
- Binding -- a concrete protocol and data format specification for a particular port type.
- Port -- a single endpoint defined as a combination of a binding and a network address.
- Service -- a collection of related endpoints.

So, in simple terms, WSDL is a template for how services should be described and bound by clients.

CHAPTER 3

3 GRID COMPUTING

3.1 INTRODUCTION

WWW has facilitated unprecedented ways of speedy global information sharing. The Grid technologies build on this by allowing facilitating the global sharing of not just information, but also of tangible assets (computational and data-storage resources) to be used at a distance. E-mail and WWW provide basic mechanisms that allow communities that span states, countries and continents to work together. But what if they could link their data, computers and other resources into a single virtual office? - Grid seeks to make this possible by providing the protocols, services and software development kits needed to enable flexible, controlled resource sharing on a large scale.

At the heart of Grid is the concept of *virtual organization*. It is a dynamic collection of individuals, institutions and resources bundled together in order to share resources as they tackle common goals. This resource sharing is not primarily file exchange, but rather direct, controlled (i.e. within the authorization, security, copyright, etc. restrictions) access to computers, software, data and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science and engineering.

Grid computing is an innovative approach that leverages existing IT infrastructure to optimize compute resources and manage data and computing

workloads. According to Gartner, "a grid is a collection of resources owned by multiple organizations that is coordinated to allow them to solve a common problem." Gartner further defines three commonly recognized forms of grid:

- Computing grid - multiple computers to solve one application problem
- Data grid - multiple storage systems to host one very large data set
- Collaboration grid - multiple collaboration systems for collaborating on a common issue.

Grid computing is not a new concept but one that has gained recent renewed interest and activity for a couple of main reasons:

1. IT budgets have been cut, and grid computing offers a much less expensive alternative to purchasing new, larger server platforms.
2. Computing problems in several industries involve processing large volumes of data and/or performing repetitive computations to the extent that the workload requirements exceed existing server platform capabilities.

Some of the industries that are interested in grid computing include:

- life sciences,
- computer manufacturing,
- industrial manufacturing,
- financial services, and
- Government.

3.2 DIFFERENCE BETWEEN GRID COMPUTING, CLUSTER COMPUTING AND THE WEB:

Cluster computing focuses on platforms consisting of often homogeneous interconnected nodes in a single administrative domain.

- Clusters often consist of PCs or workstations and relatively fast networks
- Cluster components can be shared or dedicated
- Application focus is on cycle-stealing computations, high-throughput computations, distributed computations

Web focuses on platforms consisting of any combination of resources and networks which support naming services, protocols, search engines, etc.

- Web consists of very diverse set of computational, storage, communication, and other resources shared by an immense number of users
- Application focus is on access to information, electronic commerce, etc.

Grid focus on ensembles of distributed heterogeneous resources used as a platform for high performance computing

- Some grid resources may be shared, other may be dedicated or reserved
- Application focus is on high-performance, resource-intensive applications

3.3 GRID SERVICES

Grid middleware should enable new capabilities to be constructed dynamically and transparently from distributed services. In order to engineer new Grid applications it is desirable to be able to reuse existing components and information resources and to assemble and co-ordinate these components in a flexible manner. Partly for this reason

the Grid is moving away from a collection of protocols to a service-oriented approach: the Open Grid services Architecture (OGSA) [34]. This unites Web services with Grid requirements and techniques.

The Grid's requirements mean that Grid services extend Web services considerably. Grid service configurations are:

- *dynamic and volatile* A consortium of services (databases, sensors, compute servers) participating in a complex analysis may be switched in and out as they become available or cease to be available;
- *ad-hoc.* Service consortia have no central location, no central control, and no existing trust relationships;
- *large.* Hundreds of services could be orchestrated at any time;
- *long-lived.* A simulation could take weeks.

These requirements make strenuous demands on fault tolerance, reliability, performance and security. Whereas Web services are presumed to be available and stateless, Grid services are presumed to be transient and stateful.

Grid services are broadly organised into four tiers:

1. Fabric (security, data transport, certification, remote access, network monitoring, ownership and digital watermarking, authentication);
2. Base (resource scheduling, data access, event notification, metadata management, provenance, versioning);
3. High Level (workflow, database management, personalisation);
4. Application (a gene sequence alignment, a Swiss-Prot database, a gene finding algorithm).

Each tier relies on metadata. To achieve the flexible assembly of Grid services requires information about the functionality, availability and interfaces of the various services. Service discovery and brokering uses metadata descriptions. Service composition is controlled and supported by metadata descriptions. Metadata is the key to achieving the Grid services vision.

The Grid technologies build on Web allows facilitating the global sharing of not just information, but also of tangible assets (computational and data-storage resources) to be used at a distance. Grid seeks to make this possible by providing the protocols, services and software development kits needed to enable flexible, controlled resource sharing on a large scale.

Semantic Grid is an initiative to develop effective methods for enabling such complex resource sharing. The key to this is an infrastructure where all resources, including services, are adequately described in a form that is machine-processable, i.e. knowledge is explicit - in other words, the goal is to provide *semantic interoperability,* based on the technologies of *Semantic Web*.

3.4 SEMANTIC GRID

Until very recently the Grid and the Semantic Web communities were separate, despite the convergence of their respective visions. Both have a need for computationally accessible and sharable metadata to support automated information discovery, integration and aggregation. Both operate in a global, distributed and changeable environment.

The Semantic Web base services can be Grid Base Services. The Semantic Web fabric is the means by which the Grid could represent metadata: both for Grid *infrastructure*, driving the machinery of the Grid fabric, and its base and high level services, and for Grid *applications*, representing the knowledge and operational know-how of the application domain.

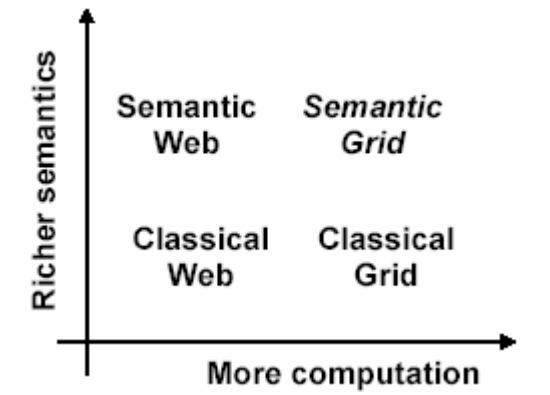

Figure 2: Semantic Grid[1]

3.5 SEMANTIC WEB FOR GRID INFRASTRUCTURE

3.5.1 Semantic Grid services

The description of a service is essential for automated discovery and search, selection, matching, composition and interoperation, invocation and execution monitoring. This choice depends on service metadata. Classification of services based on the functionality they provide has been widely adopted by diverse communities as an efficient way of finding suitable services, e.g. UDDI. Reasoning over service descriptions has a role to play when classifying and matching services. In Condor a matching mechanism is used to choose computational resources. In an architecture

[1] http://www.semanticgrid.org

where the services are highly volatile, and configurations of services are constantly being disbanded and re-organised, knowing if one service is safely substitutable by another is essential.

At the time of writing, the current state of describing Grid services through semantics is by using the names assigned the portType and serviceType elements of a WSDL document, linked to a specification document. Bringing together the Semantic Web and Web services has already attracted attention. DAML+OIL has been explored in myGrid. The myGrid service ontology extends the DAML-S ontologies. Service classifications are more expressive than UDDI's simple hierarchies and services are queried and matched by subsumption reasoning over the service descriptions. However, Grid services dynamically create and destroy service instances, have soft state registration and form long-lived service configurations. How this affects the way Semantic Web technologies can describe and discover Grid services is a challenge yet to be adequately addressed.

3.5.2 Information integration

Complex questions posed by scientists require the fusion of evidence from different, independently developed and heterogeneous resources. In biology, for example, the hundreds of data repositories in active service have different formats, interfaces, structures, coverage. The Web and the Data Grid guarantee a certain level of interoperability in retrieving and accessing data. The next level of interoperability is not just making data available, but understanding what the data means so that it can be

linked in appropriate and insightful ways, and providing automated support for this integration process.

Scientists typically link resources in two ways:

1. *Workflow orchestration*: Process flows, or workflows coordinating and chaining services using a systematic plan, are the manifestation of *in silico* experiments, allowing us to represent the e-Scientist's experimental process explicitly;
2. *Database integration*: dynamic distributed query processing, or the creation of integrated databases through virtual federations or warehouses.

Information mediation is not restricted to traditional scientific databases. Computational resources are discovered, allocated and disbanded dynamically and transparently to the user. The problem of mediation between different Grid compute resource brokering models, such as Unicore and Globus, closely resembles mediation between two database schemas.

Semantic Web and Database technologies offer great possibilities. A common data model for aggregating results drawn from different resources or instruments could use RDF. Domain ontologies for the semantic mediation between database schema, an application's inputs and outputs, and workflow work items could use DAML+OIL/RDF(S). Domain ontologies and rules can be used for constraining the parameters of machines or algorithms, and inferring allowed configurations. Execution plans, workflows and other combinations of services benefit from reasoning to ensure the semantic validity of the composition.

So we can use Semantic Web services for:

- The classification of computational and data resources, performance metrics, job control; schema integration, workflow descriptions;
- Typing data and service inputs and outputs;
- Problem solving selection and intelligent portals;
- Infrastructure for authentication, accounting and access management.

Turning this around, we can envisage that the Base and Application services of the Semantic Web are implemented as Grid services.

3.6 SEMANTIC WEB FOR GRID APPLICATIONS

The ultimate purpose of the Grid is to support knowledge discovery. The Semantic Web is often presented as a global knowledge base. Consider a scenario: A scientist posing the question "what ATPase superfamily proteins are found in mouse?" might get the answers (a) The protein accession number from the Swiss-Prot database she has permission to access; (b) InterPro is a pattern database but needs permission and payment. (c) Attwood's project is in nucleotide binding proteins (ATPase superfamily proteins are a kind of nucleotide binding protein); (d) Smith published a new paper on something similar in Nature Genetics two weeks ago; (e) Jones in your lab already asked this question last week.

A scientist may be advised of equipment or algorithm parameter settings, helped to choose and plan appropriate experiments and resources based on her aims and shared best practice, and ensure that conclusions are not drawn that are not fully justified by the techniques used. These are all applications of, or for, the Semantic

Web, and include personalised agents or services, semantic portals onto services, recommender systems and a variety of other knowledge services.

The scientific community has embraced the Web. The result is commonly publication of information without accompanying accessibility. Many resources have simple call interfaces without APIs or query languages and only "point and click" visual interfaces. Scientific knowledge is often embodied in the literature and in free text "annotations" attached to raw data. The presumption that a scientist will read and interpret the texts makes automatic processing hard and is not sustainable given the huge amount of data becoming available. The Semantic Web is about making the computationally inaccessible accessible and to automate information discovery.

3.6.1 Provenance, Quality, Trust and Proof

Both the results and the way they are obtained are highly valued. Where data came from, who created it, when, why and how was it derived is as important as the data itself for user and service provider. These are applications of the Proof, Trust and Digital Signatures of the Semantic Web. In molecular biology, data is repeatedly copied, corrected and transformed as it passes through numerous databases. Published data is actively curated automatically and by hand. Complex assemblies of programs create results from base data. Annotating results with commentaries, linking results with their sources, asserting which parameters were used when running an algorithm and why, are possible applications of Semantic Web and database technologies.

Assertions are also qualitative. Scientific knowledge is contextual and opinionated. Contexts change and opinions disagree. New information may support or

contradict current orthodoxy leading to a revision of beliefs. Inferences on assertions can give new knowledge but inferences must be exposed or else the scientist will not use them. Dealing with multiple (diverging) assertions over resources, and inference engines capable of tolerating discrepancies, is a challenge of the Semantic Web.

So Semantic Web services can be for:

- annotating results, workflows, database entries and parameters of analyses with: personal notes, provenance data, derivation paths of information, explanations or claims;
- linking *in silico* and 'at the bench' experimental components: literature, notes, code, databases, intermediate results, sketches, images, workflows, the person doing the experiment, the lab they are in, the final paper;
- describing people, labs, literature, tools and scientific knowledge.

Scientific knowledge is replicated and archived for safe-keeping. It is essential to be able to recall a snapshot of the state of understanding at a point in time in order to justify a scientific view held at that time. This raises questions: What does it mean to garbage collect the 'Semantic Grid', and how do we recover a snapshot?

Grid services come and go, which is why event notification is a Grid base service. As data collections and analytical applications evolve, keeping track of the impact of changes is difficult. Scientists rerun their queries if base data changes or new knowledge questions the underlying premise of an analysis. Mistakes or discredited information are propagated and difficult to eliminate. The ontologies and rules change. When an ontology changes in line with new beliefs, this does not wipe the old inferences that no longer hold (and how do we propagate those changes?). They must

continue to co-exist and be accessible. Monitored events and items can be described using ontologies; database triggers can implement the notification mechanism. [33]

CHAPTER 4

4 SOFTWARE AGENTS AND MULTI-AGENT SYSTEMS

4.1 INTRODUCTION

In computer science, as in any other science, several new ideas, concepts and paradigms emerged over time and became the “Big idea” or “Big excitement” of the discipline. The ‘90s brought the concept of agents in computer science and this term is now as fashionable as object-oriented was in the ‘80s or artificial intelligence in the ‘70s. Being fashionable means that anyone who wants to be “en vogue” will use it, that maybe more expectation than needed will be put in the new concept and that there is the great risk of having an overused word.

Then why agents in computer science and do they bring us anything new in modeling and constructing our applications? The answer is definitively YES and the papers in this volume contribute to justify this answer.

It would certainly not be an original thing to say that the notion of agent or agency is difficult to define. There are an important number of papers on the subject of agent and multi-agent system definition and a tremendous number of definitions for agents, ranging from one line definitions to pages of agent attribute descriptions. The situation is somehow comparable with the one encountered when defining artificial intelligence. Why was it so difficult to define artificial intelligence and why is it so difficult to define agents and multi-agents systems, when some other concepts in

computer science, as object-oriented, distributed computing, etc., were not so resistant to be properly defined.

The answer is that the concept of agent, as the one of artificial intelligence, steams from people, from the human society. Trying to emulate or simulate human specific concepts in computer programs is obviously extremely difficult and resists definition.

More than 30 years ago, computer scientists set themselves to create artificial intelligence programs to mimic human intelligent behavior, so the goal was to create an artifact with the capacities of an intelligent person. Now we are facing the challenge to emulate or simulate the way human act in their environment, interact with one another, cooperatively solve problems or act on behalf of others, solve more and more complex problems by distributing tasks or enhance their problem solving performances by competition.

Artificial intelligence (AI) put forward high expectations and the comparison of actual achievements with the initial hopes brought some disappointment. But AI contributed computer science with some very important methods, concepts, and techniques that strongly influenced other branches of the discipline, and the results obtained by AI in real world applications are far from being negligible.

As many researchers think that agents and multi-agent systems will be one of the landmark technologies in computer science of the years to come, that will bring extra conceptual power, new methods and techniques, and that will essentially broaden the spectrum of our computer applications. The technology has the chances to compensate

the failures of AI just because this new paradigm shifts from the single intelligent entity model to the multi-intelligent entity one, which is in fact the true model of human intelligence acting.

One possible question about agents can be if there is any difference between a computer program and a computational agent. To answer this question, we shall examine some agent definitions and identify the most relevant features of agents. One primary characteristic that differentiate agents from an ordinary program is that the agent must be *autonomous*. Several definitions of agents include this characteristic, for example:

- "Most often, when people use the term 'agent' they refer to an entity that functions continuously and autonomously in an environment in which other processes take place and other agents exist." (Shoham, 1993);
- "An agent is an entity that senses its environment and acts upon it" (Russell, 1997);
- "The term agent is used to represent two orthogonal entities. The first is the agent's ability for autonomous execution. The second is the agent's ability to perform domain oriented reasoning." (the MuBot Agent);
- "Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program, with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires." (the IBM Agent);

- "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, in pursuit of its own agenda and so as to effect what it senses in the future." (Franklin, Gasser, 1997).

Although not stated explicitly, Russell's definition implies the notion of autonomy as the agent will act in response to perceiving changes in the environment. The other four definitions explicitly state autonomy. But all definitions add some other characteristics, among which *interaction with the environment* is mentioned by most. Another identified feature is the property of the agent to *perform specific tasks on behalf of the user*, coming thus to the original sense of the word agent, namely someone acting on behalf of someone else.

One of the most comprehensive definitions of agents is the one given by Wooldridge and Jennings (1995) in which an agent is:

- " A hardware or (more usually) a software-based computer system that enjoys the following properties: *autonomy* - agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state; *social ability* - agents interact with other agents (and possibly humans) via some kind of agent-communication language; *reactivity*: agents perceive their environment and respond in a timely fashion to changes that occur in it; *pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking initiative."

Social dimension: Comparing the definitions above, we may identify two main trends in defining agents and agencies. Some researchers consider that we may talk and

define an agent in isolation, while some others view agents mainly as entities acting in a collectively of other agents, therefore the *multi-agent system (MAS) paradigm*. Even if we stick to the single agent type of definition it is rather difficult to expect that an agent will exist only as a stand alone entity and will not encounter other agents (be they artificial or human) in its environment. Personal agents, or information agents, which are not mainly supposed to collectively work to solve problems, will certainly have much to gain if interacting with other agents and soon, with the wide spread of agent technology, will not even be able achieve their tasks in isolation. Therefore, *the social dimension* of an agent is one of its essential features.

Mobility: Some researchers consider mobility as being one of the characteristic features of computational agents but mobility is an aspect connected mainly to implementation or realization, for software agents and hardware ones, respectively, and may be included in the capacities of interacting with the environment. [1]

4.2 INTELLIGENT AGENTS

Although almost all of the above characteristics of agents may be considered as sharing something with intelligent behavior, researchers have tried to define a clear cut between computational agents and *intelligent agents*, sliding in the world of agents the much searched difference between programs and intelligent programs. From one point of view, it is clear that, if in the design of an agent or multi-agent system, we use methods and techniques specific to artificial intelligence then the agent may be considered intelligent. For example, if the agent is able to learn from examples or if its

internal representation is knowledge-based, we should see it as an intelligent agent. If the agent has an explicit goal to pursue and it uses heuristics to select the best operations necessary to achieve its goal, it then shares one specific feature of AI programs and may be considered intelligent. But is this all that intelligence implies in the world of artificial agents or did this new paradigm bring some new characteristics to artificial intelligence?

To apply the model of human intelligence and human perspective of the world, it is quite common in the community of artificial intelligence researchers to characterize an intelligent agent using *mentalistic notions* such as knowledge, beliefs, intentions, desires, choices, commitments, and obligation (Shoham, 1993). One of the most important characteristics of intelligent agents is that they can be seen as *intentional systems,* namely systems "whose behavior can be predicted by the method of attributing belief, desires and rational acumen" (Dennett, 1987). As Shoham points out, such a mentalistic or intentional view of agents is not just another invention of computer scientists but is a useful paradigm for describing complex distributed systems. The complexity of such a system or the fact that we can not know or predict the internal structure of all components seems to imply that we must rely on animistic, intentional explanation of system functioning and behavior. We thus come again to the idea presented in the beginning: try to apply the model of human distributed activities and behavior to our more and more complex computer-based artifacts.

4.3 CLASSIFICATION

We could thus view the world of agents as being categorized as presented bellow:

Among computational agents we may identify also a broad category of agents, which are in fact nowadays the most popular ones, namely those that are generally called software agents (or weak agents, as in Wooldridge and Jennings, 1995, to differentiate them from the cognitive ones, corresponding to the strong notion of agent): information agents and personal agents. An *information agent* is an agent that has access to one or several sources of information, is able to collect, filter and select relevant information on a subject and present this information to the user. *Personal agents* or interface agents are agents that act as a kind of personal assistant to the user, facilitating for him tedious tasks of email message filtering and classification, user interaction with the operating system, management of daily activity scheduling, etc.

Last, but not least as predicted for the future, we should mention *emotional agents* (called also believable agents). Such agents aim at further developing the import of human-like features in computer programs, trying thus to simulate highly specific human attributes such as emotions, altruism, creativity, giving thus the illusion of life. Although at present they are mainly used in computer games and entertainment in general, it is believed that such agent models might contribute at developing the general concept of computational agents and further evolve our problem solving capabilities.

4.4 ACL

Multiagent systems (MAS) represent one of the most promising technological paradigms for the development of distributed, open and intelligent software systems. Moreover agent technology is beginning to be used to produce real solutions to real business problems. However, typically agent based systems have been implemented with adhoc solutions (communication languages, protocols, and so on) to meet the specific requirements of the application. In this respect a need rises for standards governing structure of and communication between heterogeneous agent communities. Now the challenge is to have our agent talk to any other agent, not just to our own. The obvious solution is a common language - ideally, all the agents that implement the common language will be mutually intelligible. Such a common language needs an unambiguous syntax, so the agents can all parse sentences the same way. It should have a well-defined semantics or meaning, so the agents can all understand sentences the same way. It should be well known, so different designers can implement it and so it has a chance of encountering another agent who knows the same language. And it should have the expressive power to communicate the kinds of things agents may need to say to one another. This is a nontrivial list of requirements. Coming up with an unambiguous syntax is the easiest. Being well known is not so much a technical as a political requirement on a language— committees and consortia are expected to ensure that their results will be widely adopted. Ensuring the expressive power of a language is potentially very difficult, but a lot of good ideas can be borrowed from the study of

human language. That leaves the question of meaning—this seems to be the most difficult problem with the current ACLs. [4]

4.4.1 Overview of FIPA ACL

FIPA's agent communication language, like KQML, is based on speech act theory: messages are actions or communicative acts, as they are intended to perform some action by virtue of being sent. The FIPA ACL specification consists of a set of message types and the description of their pragmatics—that is, the effects on the mental attitudes of the sender and receiver agents. The specification describes every communicative act with both a narrative form and a formal semantics based on modal logic. It also provides the normative description of a set of high-level interaction protocols, including requesting an action, contract net, and several kinds of auctions.

FIPA ACL is superficially similar to KQML. Its syntax is identical to KQML's except for different names for some reserved primitives. Thus, it maintains the KQML approach of separating the outer language from the inner language. The outer language defines the intended meaning of the message; the inner, or content, language denotes the expression to which the interlocutors' beliefs, desires, and intentions, as described by the meaning of the communication primitive, apply. KQML has been criticized for using the term performative to refer to communication primitives. In FIPA ACL, the communication primitives are called communicative acts, or CAs for short. Despite the difference in naming, KQML performatives and FIPA ACL communicative acts are the same kind of entity. The FIPA ACL specification document claims that FIPA ACL (like KQML) does not make any commitment to a particular content language. This

claim holds true for most primitives. However, to understand and process some FIPA ACL primitives, receiving agents must have some understanding of Semantic Language, or SL. [5] [7]

CHAPTER 5

5 LITERATURE REVIEW

The convergence of computers and telecommunication has enhanced the processing and transmission of information from one location to other on the globe. The World Wide Web (WWW) popularized enormously the markup languages (HTML primarily) that allow the file content to be enriched with additional meta-information. The most important aspect of this meta-information is the connectivity that a file maintains with other files that are related to it. This gave rise to the global hyperlinked environment that exists over the Internet and that is typically referred to as the WWW. The web started out supporting human interactions with textual data and graphics. But text based web does not support software interactions very well, especially transfers of large amount of data. Web service architectures build on standardized taxonomies and vocabularies that exhibit little flexibility and expressiveness restricts the usability of Web services mostly to human users rather than machine agents. For the latter, one would need Web service description languages that support semi-structured data, constraints, types and inheritance.

The Web is now evolving into a medium for providing a wide array of e-commerce, business-to-business, business-to-consumer and other information based services. Service-driven architectures promise a new paradigm providing an extremely flexible approach for building complex information systems. However, at the current moment, service architectures go little way beyond standardized remote procedure

calls and textual directories to locate and describe a service provider based on human intervention. But rapidly evolving and highly diversified world of information services requires huge information processing capacity and service provision on the Internet time scale. A critical prerequisite for distributed system technology to comply with the new challenge is that it must be completely self-tuning with autonomous adaptation to evolving workload with "zero" human administration. Clearly applications, executing on behalf of providers and users, in such an environment have the following requirements, which make obligatory to change the way currently the web is operating.

- Components that coordinate through negotiation in highly complex and dynamic environment
- Resource availability on supply demand basis
- Visibility of services on large and public scale

Currently Web service technology around UDDI, WSDL, and SOAP does not yet provide a mature technology. Elements need to be added around document structures, semantics of data, business logics, message exchange sequences, and formalization. Combining Ontology technology with workflow approaches is required to enrich Web service technology enabling their use in mission-critical applications. Mechanized support is needed in discovering services and their offer is required. Currently, nearly all of this work is done manually which seriously hampers the scalability of electronic commerce. A Web service discovery framework that goes beyond simple key-word-based registration means providing full-fledged Semantic Web-driven service discovery has to be defined based on approaches such as XML,

XML Schema, RDF(S), DAML+OIL, and OWL. Bringing Web service for E-commerce to its full potential requires a Peer-to-Peer (P2P) or Grid approach combined with Semantic Web technology.

Web was originally designed to share information among small number of users but now information services are becoming more mission critical as heavy loss may result if the system does not provide required functionality and resources to achieve QoS under changing conditions, such as changing workload. The system needs to provide guaranteed quality of services at application levels, not at low level like guaranteed packet delivery. A system is called a high-assurance system, when heterogeneous and changing requirement levels of QoS are satisfied. Current information service systems on the Internet do not provide guaranteed quality of services, customization and situation based information services. There is urgent need for new models for information services for e-commerce in the Internet. If the research community fails to provide necessary technology and framework, the success of e-commerce may be delayed or even may become questionable.

This fosters an urgent need to design an information service system with high-assurance that provides information services to meet the above-mentioned requirements. Autonomous distributed service system is a proposition made to cope with heterogeneous and continuously changing needs of information processing, service provision and utilization in dynamically evolving environment to meet these requirements. The proposed distributed system architecture is based on synergy of software agents integrated with Web services and Grid computing to bring high-assurance of services through supply-demand basis.

Grid computing has two aspects that make it differ from older meta-computing and distributing computing efforts. One is the scale of the data handled. Other is the use of computing sources and data sources that are not controlled by the user or his organization. To achieve both, a dynamic way to define and use a computer or data service is required. This is the goal of the OGSA effort. Similarly, data and resources should be defined in a way that is understandable and usable by the target user community. This is the goal of "ontologies", part of the Semantic Web effort.

Our vision of the integration of agents with Web services and Grid computing is to lay foundation for a self-regulating system, for e-business realization. In Autonomous Semantic Grid, Web services and Grid (OGSA) will provide an open system for dynamic resource sharing and agents will be the provider or consumer of resources while acting as proxy for humans (autonomy). Ontologies will bridge the gap between agents and Grid by bringing semantics into Grid. The whole system will build upon sheer trust among the entities i.e. (Service Providers and Service consumers).

The concept that agents can be closely aligned with that of a Web service, is like that an agent can be described as a Web service and discovered using a standard mechanism such as UDDI. XSD is expressive enough to describe ontologies, and through validation process, ontology-based pattern-matching can be implemented in order to adapt UDDI to the sort of searching that is required to find an agent service. Using WSDL gives the agent the ability to describe and to advertise its capabilities. Agent Web service descriptions and its terms, which are expressed using ontologies as a semantic enhancement to WSDL and UDDI, enable dynamic discovery and

invocation of services by software through common terminology and shared meaning. This is a vital property in an open system such as the Grid. We are working to look at the orchestration of agent services and Grid services, based on workflow languages such as WSFL and XLANG. This will allow us to study the suitability of WSDL for describing semantically composable agent services. Such an orchestration activity can make heavy usage of ontology based metadata about the quality of service offered by agents, for which we consider formalisms such as OWL, DAML-S and RDF. The integration of agent technology and ontologies with both Web and Grid services will make significant impact on the use of services and the ability to extend programs to more efficiently perform tasks for users with less human intervention.

Bringing Web services for e-commerce to its full potential requires a Peer-to-Peer (P2P) or Grid approach combined with Semantic Web technology. Both Semantic Web and Semantic Grid initiatives build heavily on the utilization of ontologies. In our proposed architecture various computer programs would next depict from these ontologies, the necessary knowledge in order to enable the most effective resource sharing over a Grid. The integration of agent technology and ontologies can make significant impact on the use of Web services or Grid services and the ability to extend programs to more efficiently perform tasks for users with less human intervention. all these technologies point up that unifying these research areas and bringing to fruition a Web teaming with complex, "intelligent" agents is both promising and practical, although a number of research challenges still remain. The pieces are coming together, and thus, the Semantic Web and Semantic Grid of agents is no longer a science fiction future. It is a practical application on which to focus current efforts.

CHAPTER 6

6 SYSTEM ARCHITECTURE

6.1 ARCHITECTURAL COMPONENTS

6.1.1 DAML-S Matchmaker

The Matchmaker is also a Web service that helps make connections between service requesters and service providers. The Matchmaker serves as a "yellow pages" of service capabilities. The Matchmaker allows users and/or software agents to find each other by providing a mechanism for registering service capabilities. Registration information is stored as advertisements. When the Matchmaker agent receives a query from a user or another software agent, it searches its dynamic database of advertisements for agents that can fulfill the incoming request. Thus, the Matchmaker also serves as a liaison between a service requester and a service provider.

Our DAML-S Matchmaker employs techniques from information retrieval, AI, and software engineering to compute the syntactical and semantic similarity among service capability descriptions. The matching engine of the matchmaking system contains five different filters for namespace comparison, word frequency comparison, ontology similarity matching, ontology subsumption matching, and constraint matching. The user configures these filters to achieve the desired tradeoff between performance and matching quality.

6.1.2 DAML-S

Web services are defining a new paradigm for the Web in which a network of computer programs becomes the consumers of information. The growing infrastructure for Web services is based on SOAP and WSDL assumes XML as unifying language to guarantee Web services interoperability. XML guarantees syntactic interoperability by providing a standard for a common syntax that is shared across the Web, with the result that Web services can parse each other message, verify whether they adhere to the expected formats, and locate each piece of information within the message. Unfortunately, the two Web services do not have any means to extract the meaning of the messages exchanged. The two Web services are in the awkward position of understanding the structure of each other message, but not understanding the content of those messages. The limitation requires programmers to hardcode Web services with information about their interaction partners, the messages that they exchange and the interpretation of the messages that they receive. The result is a set of rigid Web services that cannot reconfigure dynamically to adapt to changes without direct human intervention.

Ideally, we would like Web services to act autonomously, to require the minimal human intervention as possible. Web services should be able to register autonomously with infrastructure Registries such as UDDI, in addition they should use the infrastructure Registries to locate providers of services that they need, and finally, they should be able to transact with these Web services sending them information formatted in a way that they can understand, and be able to interpret the information that they

receive as a response. Autonomous Web services not only minimize the human intervention by automating interaction with other Web services, allowing programmers to concentrate on application development, but also they should be able to recover from failures more efficiently by automatically reconfiguring their interaction patterns. For example, if one of their partners is failing or it is becoming unreliable, they may be able to find other more reliable partners, similarly, if a new and cheaper, or anyway better, provider comes on line, Web services should be able to switch to work with the new provider.

Autonomous Web services need to be able to find partner Web services, in order to do that they need to be able to describe and register their own capabilities with public registries, as well as locate other Web services with specified capabilities. Capability information is crucial for Web services to locate each other on the bases of the services that they provide rather than on the bases of their name or of the name of the company that deploy the Web service. In addition, a Web service should have information on how to interact with the provider, which means that it should know the interaction protocol of the provider, and binding information. Most crucially, this information should allow the requesting Web services as well as the provider to decode the information exchanged, so it should specify not only the format of the messages to exchange or the remote procedures to call, but also the semantic type of the information to exchange. This view is embraced by DAML-S which defines DAML ontology for the description of Web services that attempts to bridge the gap between an infrastructure of Web services based essentially on WSDL and SOAP, and

the Semantic Web. In other words, DAML-S bridges the gap between the specification of the format of the information to be exchanged and the specification of its meaning.

DAML-S assumes a view of Web services that is wildly shared in the community. It assumes that a transaction between Web services involves at least three parties: a provider of the service, a requester of the service, and some infrastructure component such as UDDI that facilitates the location of the provider and possibly facilitates the transaction between provider and requester. Furthermore, DAML-S allows for a flexible assignment of roles in which a Web service can be both, a provider in a transaction and a requester in another, and also it allows for a role switch within the same transaction. DAML-S is constructed in three modules that provide a description of different aspects of Web services. The first one, called Profile, is an abstract description of the Web service and of the transformation it implements described as a transformation from the inputs the Web service requires to the outputs it generates. The second module is the Process Model that characterizes the Web service, specifically; it describes the interaction flow with the Web service, what function is produced by each step. The third module, called Grounding, specifies how the input/outputs of each step are mapped on WSDL specifications of messages that the two Web services exchange.

DAML-S provides all the information Web services need to interact on the Web. DAML-S supports discovery by allowing Web services to describe their capabilities in the Service Profile so that they can be matched with requests of capabilities. DAML-S capability description and the capability matching, extends the UDDI registry allowing Web services to register their own capabilities and to locate providers of the

functionality they seek. Once the provider is located, the requesting agent can use the Process Model and the Grounding to interact with the provider. The Process Model describes the interaction workflow of the provider so the requester can derive what information the provider needs at any given time. Through the Grounding the requester compiles the messages to exchange with the provider.

6.1.3 Grid Index Service

Grid technologies enable large-scale sharing of resources within groups of individuals and/or institutions. In these settings, the discovery, characterization, and monitoring of resources, services, and computations are challenging problems due to the considerable diversity, large numbers, dynamic behavior, and geographical distribution of the entities in which a user might be interested. Consequently, information services are a vital part of any Grid software or infrastructure, providing fundamental mechanisms for discovery and monitoring, and hence for planning and adapting application behavior.

The Globus Project is developing the fundamental technologies needed to build these computational Grids. Globus research focuses not only on the issues associated with building computational Grid infrastructures, but also on the problems that arise in designing and developing Grid-based applications.

The Globus Project provides software tools that make it easier to build computational Grids and Grid-based applications. These tools are collectively called the Globus Toolkit. The Toolkit is used by many organizations to build computational Grids that can support their applications.

The composition of the Globus Toolkit can be pictured as the following three pillars. Security is the foundation common to all three pillars.

The first pillar of the Globus Toolkit provides *Resource Management*, which involves the allocation of Grid resources. It includes such packages as the Globus Resource Allocation Manager (GRAM) and Globus Access to Secondary Storage (GASS).

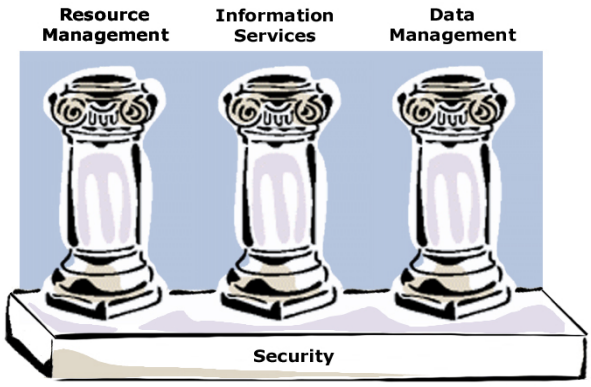

Figure 3: Composition of Globus Toolkit[2]

The second pillar of the Globus Toolkit is for *Information Services*, which provide information about Grid resources and are the focus of this document. Such services include the GT3 Index Service.

The third pillar of the Globus Toolkit is for *Data Management*, which involves the ability to access and manage data in a Grid environment. This involves such utilities as GridFTP and the Reliable File Transfer (RFT) service, which are used to move files between Grid-enabled devices.

[2] http://www.globus.org/ogsa/releases/final/docs/infosvcs/indexsvc_overview.html#ISandGT

In the context of the Globus Toolkit, Information Services have the following requirements:

- A basis for configuration and adaptation in heterogeneous environments
- Uniform, flexible access to static and dynamic information
- Scalable, efficient access to data
- Access to multiple information sources
- Decentralized maintenance

As part of this information infrastructure, the Index Service uses an extensible framework for managing static and dynamic data for Grids built using the Globus Toolkit 3.0. The functionality provided includes the following:

- Dynamic service data creation and management via Service Data Provider programs
- Aggregation of service data from multiple instances
- Registration of Grid service instances

The Globus Toolkit 3.0 is based on Open Grid service Architecture (OGSA) mechanisms. OGSA integrates Grid computing and Web services technologies by using the Web services Description Language (WSDL) to achieve self-describing, discoverable services and interoperable protocols, with extensions to support multiple coordinated interfaces and change management. Within OGSA, everything is represented as a *Grid service*: 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.

These Grid services are not only a static set of persistent services; they can also be transient service instances such as a query against a database, a data mining operation, a network bandwidth allocation, a running data transfer, and an advance reservation for processing capability. There may be one or more *instances* of a particular Grid service.

6.1.3.1 Uses/Benefits of the Index Service

Each Grid service instance has a set of service data associated with it, and this data is represented in a standardized way. There is also a standard operation for retrieving this service data from individual Grid service instances, as well as standard interfaces for registering information about Grid service instances.

Discovery often requires instance-specific, perhaps dynamic information. Service data offers a general solution in that every service must support some common service data, and may support any additional service data desired.

The Index Service does not beget specific data types. The types of data available from the Index Service for queries instead depend on how the Service is configured; that is, what sorts of Service Data Provider programs it uses to aggregate data.

The Index Service provides the following key capabilities:

- An interface for connecting external Service Data Provider programs to service instances

The Index Service provides a standard mechanism for dynamic generation of service data via external programs. These external provider programs can be the core providers that are part of GT3 or user-created, custom providers.

- **A generic framework for aggregation of service data**

Service data coming in from various Service Data Provider programs can be aggregated in different ways and indexed to provide efficient query processing. The Index Service also provides a standard mechanism for registration, polling, and notification/subscription of service data.

- **A Registry of Grid services**

A set of available Grid services is maintained in a Registry. A Registry allows for soft-state registration of Grid services, in that a set of services can be registered and periodically updated as required. A Registry then can be used to support query or other operations on a given service. [25]

6.2 ARCHITECTURE: CONCEPT AND DETAILS

6.2.1 Concept

The main concern of the information service system in the past has been to efficiently retrieve from enormous repositories the relevant information for a particular request. Due to the emergence of mission critical applications, such as e-commerce, the main focus of our research on information service system is to provide high assurance in information services to satisfy users' and providers' requirements as outlined in the previous chapters. We abstract the system based on the following two concepts: [22]

• **Resources:** The system can be viewed as a collection of resources. Resources may be physical, such as hardware, networks, and logical resources such as software, a piece of document and so on.

• **Services:** Resources either individually or in combination of more than one resource offer services.

In the system, we identify two co-existing active entities:

• **Service Provider (SP)** is the provider of services as well as resources as resources are consumed as a service.

• **User** is the consumer of services and resources.

Service provider and user may be human, an organization or a software component, which may poses negotiating capabilities. In order to provide high-assurance of services in dynamically changing environment, the system needs to assure functional as well as non-functional requirements i.e. QoS in continuously evolving environment of the system.

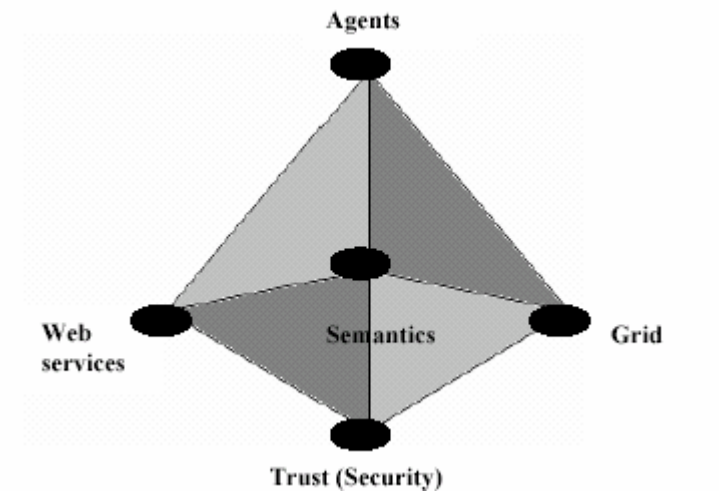

Figure 4: Autonomous Semantic Grid

The abstraction of the system based on resources and services, and conceptualizing providers and users as actors provides the foundation to meet the

requirements of autonomous distributed service system, as will be made clear in the section 6.2.3, on the system architecture. [22]

6.2.2 Synergy of Technologies

Grid computing has two aspects that make it differ from older meta-computing and distributing computing efforts. One is the scale of the data handled. Other is the use of computing sources and data sources that are not controlled by the user or his organization. To achieve both, a dynamic way to define and use a computer or data service is required. This is the goal of the OGSA effort. Similarly, data and resources should be defined in a way that is understandable and usable by the target user community. This is the goal of "ontologies", part of the Semantic Web effort.[22]

Our vision of the integration of agents with Web services and Grid computing is to lay foundation for a self-regulating system, for e-business realization as shown in Figure 3.

In Autonomous Semantic Grid, Web services and Grid (OGSA) will provide an open system for dynamic resource sharing and agents will be the provider or consumer of resources while acting as proxy for humans (autonomy). Ontologies will bridge the gap between agents and Grid by bringing semantics into Grid. The whole system will build upon sheer trust among the entities i.e. (Service Providers and Service consumers).

The concept that agents can be closely aligned with that of a Web service, is like that an agent can be described as a Web service and discovered using a standard mechanism such as UDDI. XSD is expressive enough to describe ontologies, and

through validation process, ontology-based pattern-matching can be implemented in order to adapt UDDI to the sort of searching that is required to find an agent service. Using WSDL gives the agent the ability to describe and to advertise its capabilities. Agent Web service descriptions and its terms, which are expressed using ontologies as a semantic enhancement to WSDL and UDDI, enable dynamic discovery and invocation of services by software through common terminology and shared meaning. This is a vital property in an open system such as the Grid. We are working to look at the orchestration of agent services and Grid services, based on workflow languages such as WSFL and XLANG. This will allow us to study the suitability of WSDL for describing semantically composable agent services. Such an orchestration activity can make heavy usage of ontology based metadata about the quality of service offered by agents, for which we consider formalisms such as OWL, DAML-S and RDF. The integration of agent technology and ontologies with both Web and Grid services will make significant impact on the use of services and the ability to extend programs to more efficiently perform tasks for users with less human intervention. [22]

6.2.3 Architectural Details

The system architecture approach attempts to identify key principles and layers of abstractions for interactions between actors and resources to provide services on supply demand basis to meet requirements. The system realization strategy builds on and reuses as much of the architectural foundations in related standards and emerging technologies as possible.

Resource virtualization is the fundamental tenet that leads toward sharing of resources on supply and demand basis. Providers willing to share resources use mechanisms to publish them along with the terms of usage, and users discover the required resources using some mechanism. Resources can be aggregated dynamically to complete a task to provide a service. Alternatively the task may be executed in distributed way on some resources, and then integrated into a single service, if required. Consequently, software developer may not know at design time the type or the number of nodes the application will execute on.

Inspired from Autonomous Decentralized System (ADS) concept, we define an Autonomous Distributed Service System that assures continuous service provision and utilization on supply-demand basis with the following two principles:

- Autonomy: active entities of the system are self-regulatory, and can manage their behavior.
- Adaptability: system is capable to adapt itself in response to dynamic working conditions.

In Autonomous Distributed Service System architecture both Web services and Grid services would be described in such a way that agent can consume them while exploiting their description. But some protocol transformations will be required in order to make it possible for agents to communicate with Web or Grid services while using SOAP, as a communication protocol. Likewise, agent services can be made available in Web or Grid world by describing them through WSDL and their ontologies by using XSD.

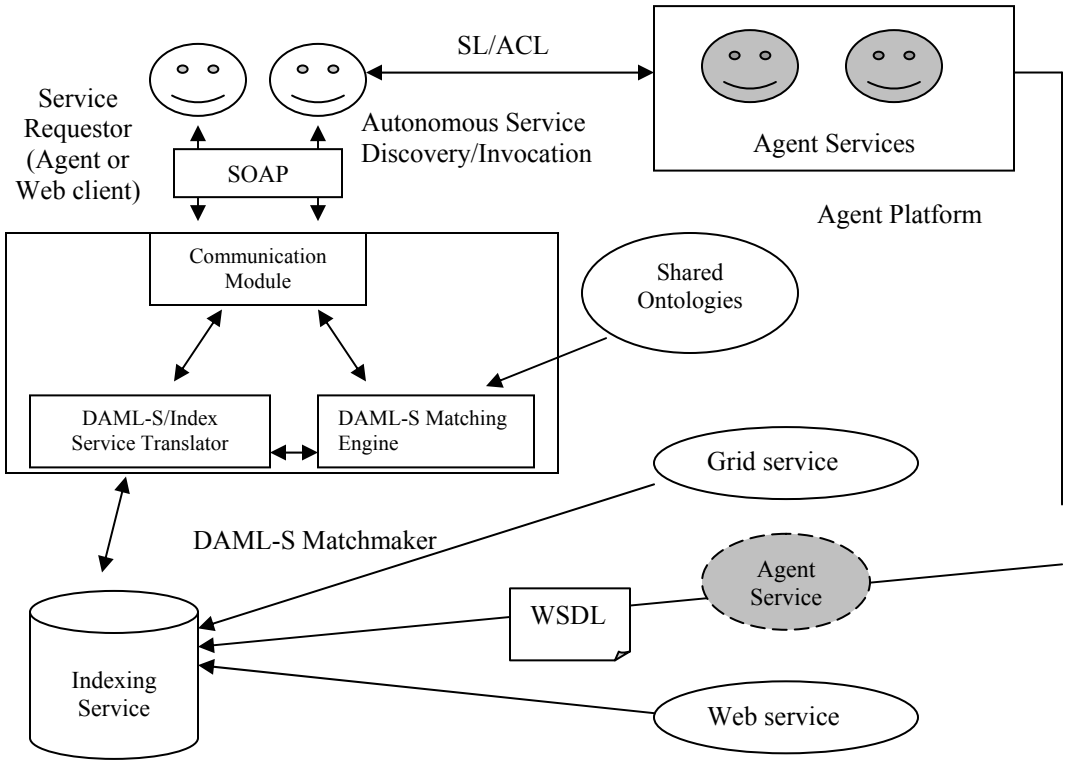

Figure 5: Autonomous Distributed Service System Architecture

The sole perception behind the availability of Web and Grid services in agent world and agent services in Grid and Web environment (as shown in figure 1) is to keep this architecture flexible and compatible up to a level where it can work with both existing as well as emerging technologies. We have sought related areas that could provide concepts, technology, and standards to reuse or adapt to Autonomous Distributed Service Systems` realization. Web services, Grid computing, and Software agents provide useful basis for the proposed system. [22]

CHAPTER 7

7 IMPLEMENTATION

7.1 PHASE I – BACKGROUND STUDY, REQUIREMENTS IDENTIFICATION AND SOLUTION

During this phase the following tasks were planned:

- Identification of requirements for future applications
- Problem areas
- What would be the possibilities for solution?

7.2 PHASE II – ARCHITECTURE

During the architecture phase various architectural components had been studied in order to find the correct proposition which will full fill the requirements identified in the previous phase. Detailed architecture and various architectural components are already explained in Chapter 6.

7.3 PHASE III – FORMULATION OF METHODOLOGY

At the end of this phase a complete methodology for the project was formulated. All the major tools and technologies were explored and on the basis of results a test bed was formed. Details related to methodology will be covered in the following chapter.

7.4 PHASE IV – TEST BED FORMATION

During this phase different toolkits and platforms were configured in order to provide a test bed for experiments. The tools, platforms and technologies which are going to be used in experiments, are explained in Chapter 8.

7.5 PHASE V – INTEGRATION

Phase V was further divided into two sub-phases:

1. Integration of Agents and Web services
2. Integration of Agents and Grid services

7.5.1 Integration of Agents and Web services

The goal of this sub-phase was to somehow bridge the communication gap between agent system and Web services. While Web services use the SOAP protocol and agents use FIPA Agent communication language, it is important that some kind of gateway will be provided in between that can interpret messages passed between multi-agent and Web services world.

The first choice can be the implementation of SOAP-based calls in agents willing to communicate with Web services but experience with this approach reveals that this will increase the complexity of the code at client side and it will be difficult for a client to communicate with several Web services.

Then a proxy agent was created by analyzing the WSDL interface provided by the Web service and after deploying that agent in JADE platform it can accept client agent requests to call a Web service, invokes the appropriate operation of the Web service, and send the results back to client agent. Now the advantage with this

approach is that instead of implementing SOAP calls, what client agent needs to do is, to send an appropriate formed agent message to the agent wrapping the Web service.

7.5.2 Integration of Agents and Grid services

Integration of Agents with Grid services have never been done before. But with OGSA, as discussed earlier that Web services standards have been integrated with Grid computing, so it becomes obvious that if we can integrate Agents and Web services then in principle integration of agents with Grid services is quite possible using same tools and techniques. This was the hypothesis which was tested in this sub phase.

Every Grid service is a Web service but reverse is not true as Grid service implement some interfaces which provide them some capabilities like (factory service, transient behavior, lifetime management etc) which Web services lack. Another thing is current Web services conform to WSDL 1.1 standard, while Grid services use GWSDL for Grid service description. But WSDL 1.2 will incorporate all the features of GWSDL as well.

These issues made it slightly challenging to integrate Agents and Grid using the same techniques which were used for Web services. But with slight manipulations, careful observations and analysis of intermediate results, finally the objectives were accomplished.

CHAPTER 8

8 MATERIALS AND METHODS

8.1 APPROACH

Divide and conquer approach will be followed during the whole life cycle of the project. As seen in previous section the whole project is divided and subdivided in work packages and sub phases respectively. Experimentation will be carried out to test our hypothesis and implementation will be done for the development of new tools and application.

8.2 TOOLS

8.2.1 Apache-Axis

Apache Axis is an implementation of the SOAP ("Simple Object Access Protocol"). SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses.

Axis is essentially a *SOAP engine* -- a framework for constructing SOAP processors such as clients, servers, gateways, etc. The current version of Axis is

written in Java, but a C++ implementation of the client side of Axis is being developed. [29]

But Axis isn't just a SOAP engine -- it also includes:

- a simple stand-alone server,
- a server which plugs into servlet engines such as Tomcat, extensive support for the *Web service Description Language (WSDL)*
- emitter tooling that generates Java classes from WSDL.
- some sample programs, and
- a tool for monitoring TCP/IP packets.

Axis now delivers the following key features:

- Speed. Axis uses SAX (event-based) parsing to achieve significantly greater speed than earlier versions of Apache SOAP.
- Flexibility. The Axis architecture gives the developer complete freedom to insert extensions into the engine for custom header processing, system management, or anything else you can imagine.
- Stability. Axis defines a set of published interfaces which change relatively slowly compared to the rest of Axis.
- Component-oriented deployment. You can easily define reusable networks of Handlers to implement common patterns of processing for your applications, or to distribute to partners.
- Transport framework. We have a clean and simple abstraction for designing transports (i.e., senders and listeners for SOAP over various protocols such as

SMTP, FTP, message-oriented middleware, etc), and the core of the engine is completely transport-independent.

- WSDL support. Axis supports the Web service Description Language, version 1.1, which allows you to easily build stubs to access remote services, and also to automatically export machine-readable descriptions of your deployed services from Axis.
- extensive support for the *Web service Description Language (WSDL)*
- emitter tooling that generates Java classes from WSDL.
- some sample programs, and
- a tool for monitoring TCP/IP packets.

Axis now delivers the following key features:

- Speed. Axis uses SAX (event-based) parsing to achieve significantly greater speed than earlier versions of Apache SOAP.
- Flexibility. The Axis architecture gives the developer complete freedom to insert extensions into the engine for custom header processing, system management, or anything else you can imagine.
- Stability. Axis defines a set of published interfaces which change relatively slowly compared to the rest of Axis.
- Component-oriented deployment. You can easily define reusable networks of Handlers to implement common patterns of processing for your applications, or to distribute to partners.
- Transport framework. We have a clean and simple abstraction for designing transports (i.e., senders and listeners for SOAP over various protocols such as

SMTP, FTP, message-oriented middleware, etc), and the core of the engine is completely transport-independent.

- WSDL support. Axis supports the Web service Description Language, version 1.1, which allows you to easily build stubs to access remote services, and also to automatically export machine-readable descriptions of your deployed services from Axis.

8.2.2 JAX-RPC

The Java API for XML-based RPC (JAX-RPC) enables Java technology developers to develop SOAP based interoperable and portable Web services. JAX-RPC provides the core API for developing and deploying Web services on the Java platform. [30]

Developer Benefits

- Portable and interoperable Web services
- Ease of development of Web services endpoints and clients
- Increased developer productivity
- Support for open standards: XML, SOAP, and WSDL
- Standard API developed under Java Community Process
- Support for tools
- RPC programming model with support for attachments
- Support for SOAP message processing model and extensions
- Secure Web services
- Extensible type mapping

Developers use the standard JAX-RPC programming model to develop SOAP based Web service clients and endpoints. A Web service endpoint is described using a Web services Description Language (WSDL) document. JAX-RPC enables JAX-RPC clients to invoke Web services developed across heterogeneous platforms. In a similar manner, JAX-RPC Web service endpoints can be invoked by heterogeneous clients. JAX-RPC requires SOAP and WSDL standards for this cross-platform interoperability. JAX-RPC is about Web services interoperability across heterogeneous platforms and languages. This makes JAX-RPC a key technology for Web services based integration.

JAX-RPC requires SOAP over HTTP for interoperability. JAX-RPC provides support for SOAP message processing model through the SOAP message handler functionality. This enables developers to build SOAP specific extensions to support security, logging and any other facility based on the SOAP messaging. JAX-RPC uses SAAJ API for SOAP message handlers. SAAJ provides a standard Java API for constructing and manipulating SOAP messages with attachments. JAX-RPC provides support for document based messaging. Using JAX-RPC, any MIME encoded content can be carried as part of a SOAP message with attachments. This enables exchange of XML document, images and other MIME types across Web services. JAX-RPC supports HTTP level session management and SSL based security mechanisms. This enables developers to develop secure Web services. More advanced SOAP message level security will be addressed in the roadmap of JAX-RPC technology.

8.2.3 OGSA, OGSI and Globus Toolkit 3

The Globus Toolkit is a software toolkit that allows us to program grid-based applications. The third and latest version of the toolkit is based on something called *Grid services*. Before defining Grid services, we're going to see how Grid services are related to a lot of acronyms you've probably heard (OGSA, OGSI ...), but aren't quite sure what they mean exactly. The following diagram summarizes the major players in the Grid service world:

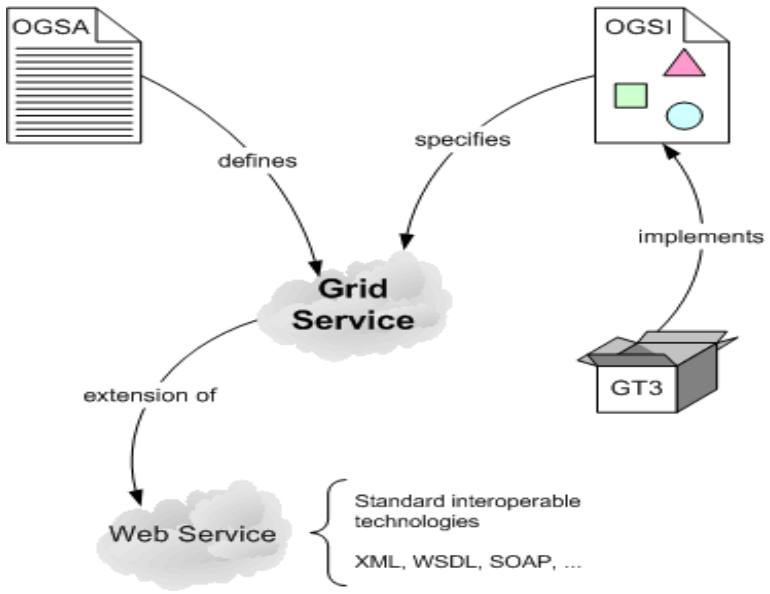

Figure 6: Relationship between OGSA, OGSI and GT3[3]

Grid services are defined by OGSA. The Open Grid services Architecture (OGSA) aims to define a new common and standard architecture for grid-based applications. Right at the center of this new architecture is the concept of a Grid service. OGSA *defines* what Grid services are, what they should be capable of, what

[3] http://www.casa-sotomayor.net/gt3-tutorial/start/key/ogsa_ogsi.html

types of technologies they should be based on, but doesn't give a technical and detailed specification (which would be needed to implement a Grid service).

Grid services are specified by OGSI. The Open Grid services Infrastructure is a formal and technical specification of the concepts described in OGSA, including Grid services.

The Globus Toolkit 3 is an implementation of OGSI. GT3 is a usable implementation of everything that is specified in OGSI (and, therefore, of everything that is defined in OGSA).

Grid services are based on Web services. More specifically we can say that Grid services are an *extension* of Web services. [28]

8.2.4 Agent Development Framework

JADE (Java Agent Development Framework) is a software framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that claims to comply with the FIPA specifications and through a set of tools that supports the debugging and deployment phase. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. The only system requirement is the Java Run Time version 1.2.

The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent; agents can access their queue via a combination of several modes: blocking, polling,

timeout and pattern matching based. The full FIPA communication model has been implemented and its components have been clearly distincted and fully integrated: interaction protocols, envelope, ACL, content languages, encoding schemes, Ontologies and, finally, transport protocols. The transport mechanism, in particular, is like a chameleon because it adapts to each situation, by transparently choosing the best available protocol. Java RMI, event-notification, and IIOP are currently used, but more protocols can be easily added and integration of SMTP, HTTP and WAP has been already scheduled. Most of the interaction protocols defined by FIPA are already available and can be instantiated after defining the application-dependent behaviour of each state of the protocol. SL and agent management ontology have been implemented already, as well as the support for user-defined content languages and Ontologies that can be implemented, registered with agents, and automatically used by the framework. JADE has also been integrated with JESS, a Java shell of CLIPS, in order to exploit its reasoning capabilities.

JADE is being used by a number of companies and academic groups, both members and non-members of FIPA, such as BT, CNET, NHK, Imperial College, IRST, KPN, University of Helsinky, INRIA, ATOS and many others. It has been recently made available under Open Source License. [27]

CHAPTER 9

9 RESULTS AND DISCUSSION

The results based on the prototype implementation of proposed system architecture are listed:

1. WSDL (and GWSDL) are better than starting from an interface language (such as Java or IDL). As they are semantically richer and offer more control.
2. WSDL is not rich enough to specify the semantics of the composition or of the interaction protocol needed for composition. In contrast to WSDL, DAML-S, rather than describing Web services in terms of their ports or the messages that they receive, it describes the capabilities of Web services in terms of the abstract function that they provide, their Process Model and the Grounding, which describes how services interact. So, WSDL and DAML-S are complementary to each other: DAML-S provides the abstract information about composition of operations and information exchange, while WSDL describes how such abstract information is mapped into actual messages and how these messages are transmitted.
3. Service Data is a structured collection of information that can be associated to a Grid service or a Web service; it facilitates the process of service discovery by providing query support over it by the client. But while automating the process of service discovery by entities like software agents, this service data cannot be mapped and shared between two negotiating agents to provide context

matching. So, ontologies should be used to define service characteristics like that of DAML-S profile which provides both provenance and non-functional information about a service and it can be mapped and shared as well.

4. The wrapper (proxy) agent approach for each web or Grid service is not a scalable solution in long run. As it introduces extra overhead in terms of communication traffic and SOAP calls. What would be ideal is that Agents and their services can be made accessible through some global registry like Grid Index Service. So, that they can communicate on the same level as Web or Grid services are doing.

The Semantic Web should enable access not only to content but also to services on web. Users and software agents should be able to invoke, discover, compose and monitor web resources offering particular services and having particular properties. OWL-S (former DAML-S) supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. OWL-S markup of Web services will facilitate the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring. The extensibility elements of WSDL allow for a straightforward means of using OWL-S and WSDL together. This, in turn, allows service developers to take advantage of the complementary strengths of these two specification languages. [24]

OGSI is concerned primarily with creating, addressing, inspecting, and managing the lifetime of stateful *Grid services* [Physiology]. The OGSI version 1.0 specification [OGSI-Spec], released in July 2003, defines a Grid service to be a Web service that

conforms to a set of conventions (interfaces and behaviors) that define how a client interacts with a Grid service. These conventions and other OGSI mechanisms associated with Grid service creation and discovery, provide for the controlled, fault resilient, and secure management of the distributed and often long-lived state that is commonly required in advanced distributed applications.

Since development started on OGSI in early 2002, the Web services world has evolved significantly. Specifically, a number of new specifications and use patterns have emerged that simplify and clarify the ideas expressed in OGSI. The following briefly outlines this evolution.

WS-Addressing provides transport-neutral mechanisms to address Web services. Specifically, WS-Addressing specification defines XML elements to identify Web service endpoints and to include endpoint identification in messages. This specification enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner. The end point reference information provides not only the address of Web service itself, but can also serve to identify state instances “behind” the service by using endpoint reference properties.

Although less central to the WS-Resource definition, WS-MetaDataExchange provides a collection of mechanisms for obtaining information about a published service, such as its WSDL description, XML Schema definitions and any other policy information necessary to use the service.

Since WS-Addressing and WS-MetaDataExchange provide several capabilities that are also defined OGSI, it is beneficial to exploit those Web services specifications

rather than maintaining a specification that defines the same functionality redundantly.[5]

CHAPTER 10

10 FUTURE WORK

10.1 PHASE VI – UNIVERSAL ACCESSIBILITY

There are two sub-phases in Phase VI:

1. Tool Development for the description of Agent services using WSDL, so that they can be published in any globally accessible registry mechanism, like Grid Index Service.
2. Agents Service Discovery/Invocation through some globally accessible index service like Grid Index Service using service Ontologies like DAML-S.

10.2 PHASE VII – AUTONOMY

Phase VII will be consisting two sub-phases:

1. Autonomous and Intelligent Agents can search Discover and Consume/Invoke Web, Grid and Agent Services.
2. Integration of DAML-S match-maker module with UDDI.

10.3 PHASE VIII – APPLICATION DEVELOPMENT

Development of an application that require autonomous resource Discovery/Invocation through negotiation, based on supply and demand in highly dynamic, heterogeneous and resource intensive environments e.g. Human Genome Decoding.

11 CONCLUSIONS

Grid computing is a promising emerging technology that is growing in mindshare and relevance in both industry and society. But future Grid applications' requirements enforce new architectures to be built on top of Grid e.g. bringing Web services for e-commerce to its full potential requires a Peer-to-Peer (P2P) or Grid approach combined with Semantic Web technology. Both Semantic Web and Semantic Grid initiatives build heavily on the utilization of ontologies. In our proposed architecture various computer programs would next depict from these ontologies, the necessary knowledge in order to enable the most effective resource sharing over a Grid. The integration of agent technology and ontologies can make significant impact on the use of Web services or Grid services and the ability to extend programs to more efficiently perform tasks for users with less human intervention. Our experience with all these technologies point up that unifying these research areas and bringing to fruition a Web teaming with complex, "intelligent" agents is both promising and practical, although a number of research challenges still remain. The pieces are coming together, and thus, the Semantic Web and Semantic Grid of agents is no longer a science fiction future. It is a practical application on which to focus current efforts. Nonetheless; in addition to ontologies, alternative methods must be developed, as development and maintenance of ontologies is extremely complex and resource consuming and ontologies suffer from problems related to the intrinsic complexity.

12 REFERENCES

[1]. Introduction to Multi-Agent Systems, International Summer School on Multi-Agent Systems, Bucharest, 1998, Adina Magda Florea, “Politehnica” University of Bucharest, Email: adina@cs.pub.ro

[2]. A Web services Primer,, by Venu Vasudevan April 04, 2001

[3]. Michael N. Huhns, Munindar P. Singh, "Conversational Agents", *IEEE Internet Computing*, March-April 1997, pp. 73-75.

[4]. J. Mayfield, Y. Labrou and T. Finin, "Desiderata for Agent Communication Languages," in: *Working Notes of the AAAI Spring Symposium Series*, pp. 122-127, Stanford University, 1995.

[5]. From Open Grid services Infrastructure to WSResource Framework: Refactoring & Evolution, Version 1.0, 2/12/2004, Karl Czajkowski, Don Ferguson (IBM), Ian Foster (Globus Alliance / Argonne National Laboratory), Jeff Frey (IBM), Steve Graham (IBM), Tom Maguire (IBM), David Snelling (Fujitsu Laboratories of Europe), Steve Tuecke (Globus Alliance / Argonne National Laboratory)

[6]. http://www.cs.umbc.edu/kqml/

[7]. http://www.cselt.it/fipa/

[8]. Tim Finin, Richard Fritzson, Don McKay and Robin McEntire, "KQML as An Agent Communication Language ", *Proceedings of the third international conference on Information and knowledge management*, 1994, Gaithersburg, MD USA, pp. 456-463.

[9]. P Charlton, R. Cattoni, A. Potrich, and E. Mamdani, "Evaluating the FIPA Standards and Its Role in Achieving Cooperation in Multi-agent Systems", *Proc. 33rd Hawaii Int'l Conf. on System Sciences*, HICSS-33, 2000, Maui, HI, USA.

[10]. Fabio Bellifemine, Giovanni Rimassa, and Agostino Poggi, "JADE - A FIPA-compliant Agent Framework", *The Fourth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents* (PAAM99), 1999, London, UK.

[11]. Marian H. Nodine and Amy Unruh, "Constructing Robust Conversation Policies in Dynamic Agent Communities", *Third International Conference on AUTONOMOUS AGENTS* (Agents '99) Seattle, Washington, 1999.

[12]. H. Dong, J.H. Ding, X. Li and J. Lu, "On Open Communication Frameworks for Software Agents", *Proc. of the Technology of Object-Oriented Languages and Systems*, 1998, Beijing, China.

[13]. Chelliah Thirunavukkarasu, Tim Finin, and James Mayfield, "Secret Agents - A Security Architecture for KQML", in: *Proc. ACM CIKM Intelligent Information Agents Workshop*, Baltimore, Maryland, USA, 1995.

[14]. James Mayfield and Tim Finin, "A security architecture for agent communication languages", *Fourth International Workshop on Agent Theories, Architectures, and Languages, Providence*, Rhode Island, USA, 1997.

[15]. Mundidar P. Singh, "Agent Communication Languages: Rethinking the Principles", *IEEE Computer*, December, 1998, pp. 40-47.

[16]. P., Sadri, F., Toni, F., "*Communicating Agents*", *Proc. International Workshop on Multi-Agent Systems in Logic Programming*, in conjunction with ICLP'99, Las Cruces, New Mexico, 1999.

[17]. J.A. Pastor, T. S. Liebowitz., D.P. McKay, and R McEntire, "An architecture for intelligent resource agents", *Proceedings of the 2nd IFCIS International Conference on Cooperative Information Systems* (CoopIS '97), Kiawah Island, SC ,1997.

[18]. The Grid: An Application of the Semantic Web, Carole Goble Department of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, UK, carole@cs.man.ac.uk, David De Roure, Dept of Electronics & Computer Science, University of Southampton, Southampton, SO17 1BJ, UK dder@ecs.soton.ac.uk

[19]. H. Farooq Ahmad, Kashif Iqbal, Naveed Baqir, Arshad Ali, Hiroki Suguri, "Integration of Agents with Web services and Grid computing Environment", Proc. 9th Assurance System Symposium, pp. 65-71, June 2003, Tokyo, Japan.

[20]. H. Farooq Ahmad, Kashif Iqbal, Arshad Ali, Hiroki Suguri, "Autonomous Distributed Service System: Basic Concepts and Evaluation", Proc. The Second International Workshop on Grid and Cooperative Computing, GCC 2003, pp. 432-439, Shanghai, China.

[21]. Kashif Iqbal, Mobeena Jamshed, H. Farooq Ahmad, Arshad Ali, Hiroki Suguri, "Integration of Agents with Web services and Grid Computing

Environment: Design and Implementation", International Workshop on Frontiers of Information Technology, December 23-24, 2003, Islamabad, Pakistan.

[22]. Kashif Iqbal, H. Farooq Ahmad, Arshad Ali, Hiroki Suguri, Mobeena Jamshed, "Autonomous Distributed Service System Implementation", ADSN 2004, Tokyo, Japan.

[23]. Kashif Iqbal, H. Farooq Ahmad, Arshad Ali, "Autonomous Distributed Service System: Concept, Architecture and Implementation", GIKITech 2004, Ghulam Ishaq Khan Institute, Topi, Pakistan.

[24]. Describing Web services using OWL-S and WSDL, DAML-S Coalition working document; October 2003, David Martin, Mark Burstein, Ora Lassila, Massimo Paolucci, Terry Payne, Sheila McIlraith

[25]. http://www.globus.org/ogsa/releases/final/docs/infosvcs/indexsvc_overview.html

[26]. WSDL2JADE TOOL, http://sas.ilab.sztaki.hu:8080/wsdl2jade/index.html

[27]. JADE, Java Agent Development Framework, http://sharon.cselt.it/projects/jade

[28]. Globus Toolkit 3, Programmers' Tutorial, http://www.casa-sotomayor.net/gt3-tutorial/

[29]. Apache-axis, http://ws.apache.org/axis/

[30]. JAX-RPC, http://java.sun.com/xml/downloads/jaxrpc.html

[31]. http://www.developer.com/design/article.php/10925_1010451_1

[32]. Foster, I., Kesselman, C., Nick, J. and Tueske, S., "The Physiology of the Grid: An Open Grid services Architecture for Distributed Systems Integration", Globus, Project, www.globus.org/research/papers/ogsa.pdf (2002).

[33]. http://www.semanticgrid.org/documents/sigmod/ami9.pdf

[34]. Foster, I., Kesselman, C., Nick, J. and Tueske, S., "The Physiology of the Grid: An Open Grid services Architecture for Distributed Systems Integration", Globus, Project, www.globus.org/research/papers/ogsa.pdf (2002).

[35]. http://www.w3.org/SOAP

[36]. http://www.w3.org//WSDL1.1

[37]. http://www.UDDI.org

13 APPENDIX

13.1 PUBLICATIONS

- *H. Farooq Ahmad, Kashif Iqbal, Naveed Baqir, Arshad Ali, Hiroki Suguri,* "Integration of Agents with Web services and Grid computing Environment", Proc. 9th Assurance System Symposium, pp. 65-71, June 2003, Tokyo, Japan.
- *H. Farooq Ahmad, Kashif Iqbal, Arshad Ali, Hiroki Suguri,* "Autonomous Distributed Service System: Basic Concepts and Evaluation", Proc. The Second International Workshop on Grid and Cooperative Computing, GCC 2003, pp. 432-439, Shanghai, China.
- *Kashif Iqbal, Mobeena Jamshed, H. Farooq Ahmad, Arshad Ali, Hiroki Suguri, "*Integration of Agents with Web services and Grid Computing Environment: Design and Implementation", International Workshop on Frontiers of Information Technology, December 23-24, 2003, Islamabad, Pakistan.
- *Kashif Iqbal, H. Farooq Ahmad, Arshad Ali, Hiroki Suguri, Mobeena Jamshed,* "Autonomous Distributed Service System Implementation", ADSN 2004, Tokyo, Japan.
- *Kashif Iqbal, H. Farooq Ahmad, Arshad Ali,* "Autonomous Distributed Service System: Concept, Architecture and Implementation", GIKITech 2004*,* Ghulam Ishaq Khan Institute, Topi, Pakistan.

13.2 TUTORIALS AND GUIDES:

- Apache-axis Installation Guide
- JWSDP Tutorial
- GT3 Tutorial