



Lehrstuhl für
Wirtschaftsinformatik
Information Systems
Management

No. 5

2005

Bayreuther Arbeitspapiere zur Wirtschaftsinformatik

Oscar Ardaiz, Pablo Chacin, Isaac Chao, Juan Carlos Cruellas, Felix Freitag, Manuel Medina, Leandro Navarro, Miguel Valero (Universidad Polytechnica de Catalunya), Omer Rana, Liviu Joita (Cardiff University), Torsten Eymann (University of Bayreuth)



Proof-of-Concept Application - Annual Report Year 1

Bayreuth Reports on Information Systems Management



**UNIVERSITÄT
BAYREUTH**

ISSN 1864-9300

Die Arbeitspapiere des Lehrstuhls für Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

Alle Rechte vorbehalten. Insbesondere die der Übersetzung, des Nachdruckes, des Vortrags, der Entnahme von Abbildungen und Tabellen – auch bei nur auszugsweiser Verwertung.

Authors:

Oscar Ardaiz, Pablo Chacin, Isaac Chao, Juan Carlos Cruellas, Felix Freitag, Manuel Medina, Leandro Navarro, Miguel Valero (Universidad Polytechnica de Catalunya), Omer Rana, Liviu Joita (Cardiff University), Torsten Eymann (University of Bayreuth)

The Bayreuth Reports on Information Systems Management comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

All rights reserved. No part of this report may be reproduced by any means, or translated.

**Information Systems and Management
Working Paper Series**

Edited by:

Prof. Dr. Torsten Eymann

Managing Assistant and Contact:

Raimund Matros
Universität Bayreuth
Lehrstuhl für Wirtschaftsinformatik (BWL VII)
Prof. Dr. Torsten Eymann
Universitätsstrasse 30
95447 Bayreuth
Germany

Email: raimund.matros@uni-bayreuth.de

ISSN 1864-9300



IST-FP6-003769 CATNETS

D 3.1

Implementation of additional services for the economic enhanced platforms in Grid/P2P platform: Preparation of the concepts and mechanisms for implementation

Contractual Date of Delivery to the CEC:	31 August 2005
Actual Date of Delivery to the CEC:	31 August 2005
Author(s):	Oscar Ardaiz, Pablo Chacin, Isaac Chao, Juan Carlos Cruellas, Felix Freigtag, Liviu Joita, Manuel Medina, Leandro Navarro, Omer F. Rana, Miguel Valero
Work package:	WP3
Est. person months:	24
Security:	public
Nature:	final
Version:	1.0
Total number of pages:	113

Abstract:

This deliverable describes the work done in task 3.1, "Implementation of additional services for economic enhanced platforms in Grid/P2P platform: Preparation of concepts and mechanisms for implementation" from the WP 3, "Proof-of-Concept Applications". The document is divided in five parts: the introduction of Cat-COVITE application – motivation, the introduction of the Catallactic middleware and the WS-Agreement concept; requirements and concepts of the Cat-COVITE application with the Catallaxy; the Catallactic middleware; the integration of Catallactic middleware and Cat-COVITE application; and the conclusion and further work.

Keyword list: (optional)

CATNETS Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-FP6-003769. The partners in this project are: LS Wirtschaftsinformatik (BWL VII) / University of Bayreuth (coordinator, Germany), Arquitectura de Computadors / Universitat Politecnica de Catalunya (Spain), Information Management and Systems / University of Karlsruhe (TH) (Germany), Dipartimento di Economia / Università delle merche Ancona (Italy), School of Computer Science and the Welsh eScience Centre / University of Cardiff (United Kingdom), Automated Reasoning Systems Division / ITC-irst Trento (Italy)

University of Bayreuth

LS Wirtschaftsinformatik (BWL VII)
95440 Bayreuth
Germany
Tel: +49 921 55-2807, Fax: +49 921 55-2816
Contact person: Torsten Eymann
E-mail: catnets@uni-bayreuth.de

Universitat Politecnica de Catalunya

Arquitectura de Computadors
Jordi Girona, 1-3
08034 Barcelona
Spain
Tel: +34 93 4016882, Fax: +34 93 4017055
Contact person: Felix Freitag
E-mail: felix@ac.upc.es

University of Karlsruhe

Institute for Information Management and
Systems
Englerstr. 14
76131 Karlsruhe
Germany
Tel: +49 721 608 8370, Fax: +49 721 608
8399
Contact person: Daniel Veit
E-mail: veit@iw.uka.de

Università delle merche Ancona

Dipartimento di Economia
Piazzale Martelli 8
60121 Ancona
Italy
Tel: 39-071- 220.7088 , Fax: +39-071-
220.7102
Contact person: Mauro Gallegati
E-mail: gallegati@dea.unian.it

University of Cardiff

School of Computer Science and the Welsh
eScience Centre
Cardiff University, Wales
Cardiff CF24 3AA, UK
United Kingdom
Tel: +44 (0)2920 875542, Fax: +44 (0)2920
874598
Contact person: Omer F. Rana
E-mail: o.f.rana@cs.cardiff.ac.uk

ITC-irst Trento

Automated Reasoning Systems Division
Via Sommarive, 18
38050 Povo – Trento
Italy
Tel: +39 0461 314 314, Fax: +39 0461 302
040
Contact person: Floriano Zini
E-mail: zini@itc.it

Changes

<i>Version</i>	<i>Date</i>	<i>Author</i>	<i>Changes</i>

1	Introduction.....	7
1.1	<i>Cat-COVITE application and Catallaxy mechanism – motivation.....</i>	7
1.1.1	Description of Service Oriented Architecture (SOA) in COVITE application.....	8
1.2	<i>Catallactic middleware – introduction</i>	9
1.3	<i>Web Service Agreement (WS-Agreement) – introduction.....</i>	10
2	Cat-COVITE application.....	11
2.1	<i>COVITE application – description and architecture.....</i>	11
2.2	<i>Cat-COVITE application – requirements and concepts for Catallaxy.....</i>	12
2.2.1	COVITE - from centralised approach to decentralised approach.....	13
2.3	<i>Layer models – Cat-COVITE model architecture within Catallaxy mechanism... </i>	15
3	Catallactic middleware	18
3.1	<i>Requirements.....</i>	19
3.2	<i>Concepts and design guidelines.....</i>	21
3.3	<i>Architecture Structure</i>	23
3.4	<i>Related Work on Architectures for Economic Based Resource Management</i>	25
3.5	<i>Components and mechanisms</i>	26
3.5.1	Registering resources and agents.....	26
3.5.2	Resource Discovery	27
3.5.3	Negotiating for resources	28
3.6	<i>Implementation of prototype.....</i>	29
3.6.1	Implementation toolkit selection	29
3.6.2	Implementation approach.....	31
3.6.3	Implementation of P2P Agent Communication Layer	32
3.6.4	GT4 as Query Resolver.....	34
4	Integration of middleware and Cat-COVITE application.....	36
4.1	<i>Application and Catallactic middleware interaction description.....</i>	36
4.1.1	Logical view	36
4.1.2	Physical Deployment on GT4 containers	37
4.2	<i>Use cases – description, flow diagrams, sequence diagrams, actions</i>	38
4.3	<i>Concepts and mechanisms for the application implementation – requirements</i>	43
4.4	<i>WS-Agreement - concepts and requirements</i>	45
5	Conclusion and further work	49
5.1	<i>Impact of the Catallactic middleware in the application.....</i>	49
5.2	<i>Conclusions on the adaptation of the middleware to an existent application.....</i>	49
5.3	<i>Roadmap to first CATNETS prototype - open issues and future work.....</i>	50
6	References.....	51
	Annex A – The Architecture Development Process.....	54
	<i>A.1 Architecture Design Process.....</i>	54

<i>A.2 Architecture Specification</i>	55
<i>A.3 References</i>	56
Annex B – Peer-to-Peer Architectures	57
<i>B.1 Star topology</i>	58
<i>B.2 Ring topology</i>	59
<i>B.3 Hierarchical topology</i>	60
<i>B.4 Mesh topology</i>	60
<i>B.5 Centralised + Ring</i>	61
<i>B.6 Centralised + Decentralised</i>	61
<i>B.7 References</i>	62
Annex C – Specifications of P2P Agent layer	63
<i>C.1 Functional Blocks</i>	63
<i>C.2 Requirements</i>	65
<i>C.3 References</i>	67
Annex D - Middleware toolkits evaluation	69
<i>D.1 Identification of candidate middleware toolkits and evaluation process</i>	69
<i>D.2 Presentation of the candidates</i>	71
D.2.1 Web Services JAX-RPC implementations (Axis)	71
D.2.2 WSRF/ OGSA.....	72
D.2.3 J2SE.....	74
D.2.4 JXTA.....	75
D.2.5 JADE	76
D.2.6 Diet Agents	76
<i>D.3. Middleware Evaluation Summary</i>	78
D.3.1 Functional view: Mapping middleware toolkits into the architecture	78
D.3.2 Technical View	79
D.3.3 Development view.....	81
D.3.4 Tests on middleware toolkits integration	84
<i>D.4 Conclusions</i>	85
D.4.1 Conclusions on functional, technical and development views.....	85
D.4.2 Joint selection of middleware and application.....	87
<i>D.5 Middleware Toolkit Evaluation Details</i>	88
D.5.1 Functional View	88
D.5.2 Technical View	93
D.5.3 Development View	95
<i>D.6 Relevant Standards</i>	97
D.6.1 Web Services Standards.....	97
D.6.2 WSRF Related Standards.....	101
<i>D.7 References</i>	102
Annex E – Application framework – concepts and mechanisms	105
<i>E.1 WSDL file</i>	105
<i>E.2 The service source code</i>	107
<i>E.3 Agreement Template examples</i>	108

1 Introduction

This deliverable describes the work done in task 3.1 “Implementation of additional services for economic enhanced platforms in Grid/P2P platform: Preparation of concepts and mechanisms for implementation” from the WP 3, “Proof-of-Concept Applications”. The document is divided in five parts: the introduction of the Cat-COVITE application (Catallactic COLlaborative VIRTual TEams), which make use of concepts and models of the COVITE application [JPB04], and the motivation for Catallaxy mechanism and the WS-Agreement [WS-Ag05] concept; requirements and concepts of the Cat-COVITE application with the Catallaxy; the Catallactic middleware; the integration of the Catallactic middleware and the Cat-COVITE application; and the conclusion and further work.

1.1 Cat-COVITE application and Catallaxy mechanism – motivation

A typical Architecture / Engineering / Construction (AEC) industry project involves many individuals and companies forming a consortium for the duration of a project. Such projects range in size from the design and construction of a single building, to the creation of a large national infrastructure such as airports, dams, and highways. These projects are usually unique, very complex and involve many participants from a number of organizations acting collaboratively. The members are geographically dispersed. The consortia include design teams, product suppliers, contractors and inspection teams who must collaborate and conform to predefined scheduling constraints and standards. These participants also work concurrently, thus requiring real time collaboration between geographically remote participants. Each consortium is in effect a virtual organisation (VO). A typical consortium member is often providing similar services to multiple projects simultaneously involving different partners. Web based communication technology is beginning to play an increasingly important role in supporting collaboration in AEC projects particularly to enable project managers to identify the current state of a project, its activities, and the constraints on these activities and their schedules. The planning, implementation and running of these projects is thus a complex task in which the Grid/P2P will be an important infrastructure.

COVITE supports the establishment of these VOs consisting of virtual design teams in the AEC industry. By Grid-enabling an existing commercial software package, Product Supplier Catalogue Database, an ActivePlan Solutions Ltd. (www.activeplan.co.uk) software package, COVITE enables these virtual teams to plan, schedule, coordinate, and share components between designs and from different suppliers.

The ability of a free-market economy to adjudicate and satisfy the needs of VOs, in terms of services and resources, represent an important feature of the Catallaxy mechanism. Such VOs could require large amount of resources which can be obtained from computing systems connected over simple communication infrastructure such as Internet. There are also possibilities for these VOs to try maximizing their own utilities on the market.

Friedrich August von Hayek [FAvH89] and other Neo-Austrian economists understood the market as a decentralised coordination mechanism opposite to a centralised

command economy. The project will investigate how Catallactic mechanism can be implemented for the resources allocation in the real application layer networks. The application prototype will be based on the COVITE application developed by Cardiff University in the COVITE project [COV04] and will be named Cat-COVITE application. This prototype will provide an assessment of the validity of the economic enhanced middleware used by the Cat-COVITE application. The system architecture within the application has been significantly modified to reflect the new requirements of the Catallactic mechanism, and particular support has been provided to utilise a P2P architecture.

The particular approach adopted in the Cat-COVITE application is employable in a significant number of other industrial applications which make use of distributed databases. In this way, the lessons learned from this application, and integration with the Catallactic middleware may find use by a very wide community.

1.1.1 Description of Service Oriented Architecture (SOA) in COVITE application

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some meaning of connecting services to each other is needed.

The COVITE application is divided into two functional services: Security Service and Multiple Database Search Service (MDSS). Figure 1 gives a conceptual view of the COVITE application and its main components.

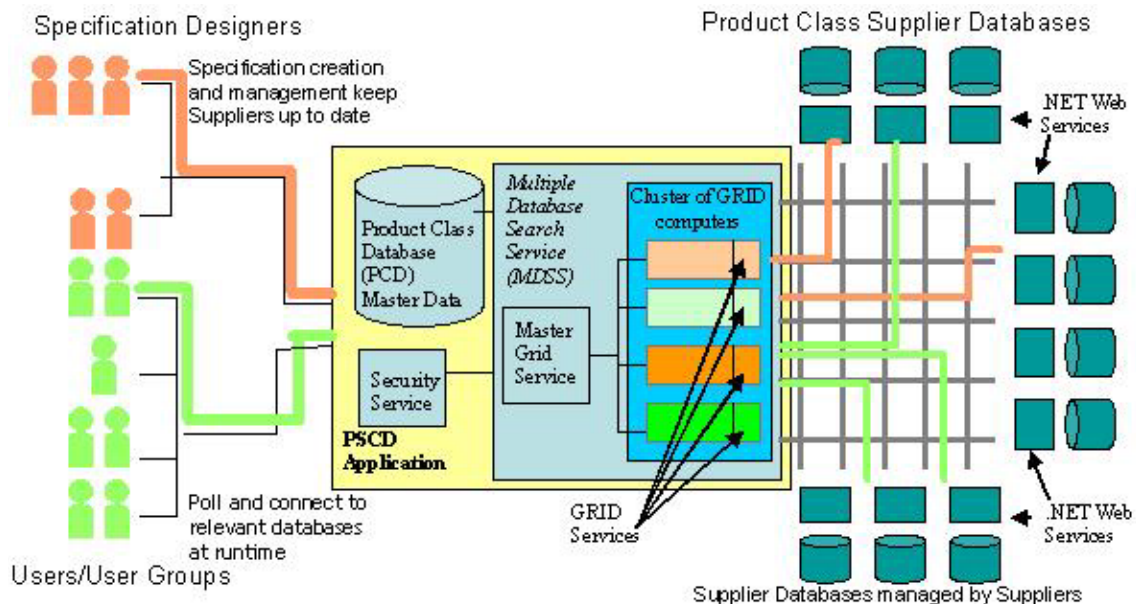


Figure 1 – COVITE application diagram

The security service defines a security framework for the COVITE application using the Globus Security Infrastructure (GSI) [GSI04]. GSI is based upon Public Key Infrastructure (PKI) and requires users to have a private key and an X.509 certificate used to authenticate a user to Grid services. The important feature of GSI is the single sign-on capability and the ability to perform delegation, achieved through the use of a

proxy, to perform the authentication to the COVITE resources on a user's behalf. The security service also provides the capability of role-based privileges within the VO. In this instance, a user is given access to particular services based on their role within a project. Based on this model, a user with a particular identity may be given access rights to different services at different times. Role-based access control is particularly useful for managing and engaging in consortia-based projects.

The Grid enabled Multiple Database Search Service (MDSS) enables searching across a large number of supplier databases using a cluster of machines in a Grid network. When a search is made by the MDSS, the machines in the Grid network should collaborate while retrieving the matching products. In this instance, the query is defined according to a data model that is specific to a given application domain. Arbitrary text queries (as in the Google.com search engine, for instance) are not allowed.

1.2 Catallactic middleware – introduction

We believe the requirements imposed by the application scenarios analyzed demand an innovative approach for the construction of the resource allocation middleware. The proposed approach is the construction of a framework that offers a set of generic negotiation mechanism, on which specialized strategies and policies can be dynamically plugged to adapt to specific application domains or market designs. The middleware should therefore offer a set of high level abstractions and mechanisms to locate and manage resources, locate other trading agents, engage agents in negotiations, learn and adapt to changing conditions. We will first analyze the architectural requirements that need to be addressed to fulfil this vision and then present the proposed architecture.

We propose a layered architecture shown in the figure 2. This layered approach offers the palpable benefit of a clear separation of concerns between the layers, which beside helping in tackling the complexity of the system, also facilitate the construction of a more adaptable system as the upper layers can be progressively specialized (by means of pluggable rules and strategies) into specific application domains.

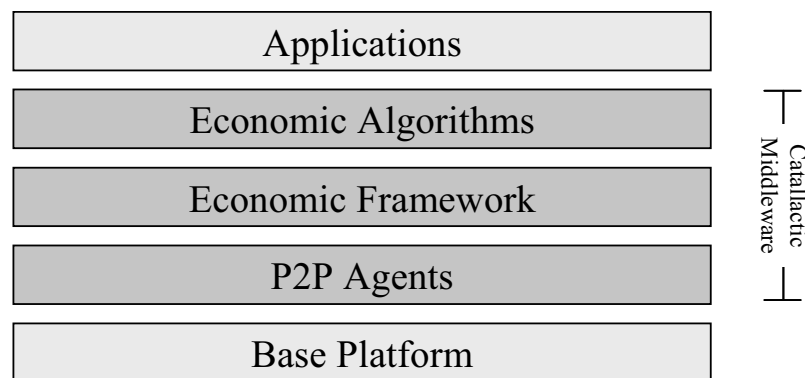


Figure 2 - A layered architecture for resource allocation

Agents in the Economic Algorithms Layer are responsible for implementing the high level economic behaviour contained in the economic algorithms layer (negotiation, learning, adaptation to environment signals, other agent's strategies and its own outcomes). Applications themselves do not participate (and are not actually aware of) the negotiation, but delegate it to the economic agents.

Economic agents rely on a lower level layer, the P2P Agent Layer, for the self-organization of the systems and the interaction with the base platform that ultimately manages the resources being traded. This layer offers key functions like the maintenance of the trading network topology following a P2P paradigm, the decentralized resource discovery and the group communication among agents.

In this context the term “P2P” should be interpreted as a general approach for distributed system design, characterized by the ad-hoc nature of the system topology and the functional symmetry of its components, which can be realised under very different architectures, ranging from unstructured and disperse networks to very hierarchical systems.

Between those two layers, a Framework Layer isolates economic agents from technical complexities; much in the same tenor that modern online trading platform allows non expert users to trade stocks. This framework offers basic functions like searching for suitable providers given a resource specification, handle the exchange of messages during the negotiation process, keeping track of the evolution of the negotiation for further adaptation of strategies.

1.3 Web Service Agreement (WS-Agreement) – introduction

WS-Agreement protocol specification has been developed by the GRAAP Working Group (Grid Resource Allocation and Agreement Protocol WG) of the Scheduling and Resource Management (SRM) Area of the Global Grid Forum (GGF) [GGF05].

WS-Agreement is an XML language protocol for specifying an agreement between a resource/service provider and a consumer [WS-Ag05]. It is generally aimed to be a one-shot interaction, and is not directly intended to support negotiation. However, it can form a useful basis on which negotiation between two parties may be conducted.

WS-Agreement can cover a wide scope of application scenarios and is suitable for use within the CATNETS project in regards to the establishment of an agreement between a service provider and a service consumer. As is specified by the WS-Agreement specification, the establishment of the agreement is achieved by using a single document format and a protocol comprising few states. The service provider acts as an agreement provider, while the service consumer as the agreement initiator. Chapter 4.4 will further detail the concepts and requirements of WS-Agreement in the CATNETS project context.

2 Cat-COVITE application

This chapter is divided in three parts: first part introduces the description and architecture of the COVITE application, the second part introduces the Cat-COVITE application with the requirements and concepts for the Catallaxy mechanism, while the third part presents the layered model of the Cat-COVITE application.

2.1 COVITE application – description and architecture

In the COVITE application (Figure 3), collaboration plays a key role in the procurement of supplies for construction projects. The collaboration can take place in various ways.

The suppliers and purchasers collaborate to procure supplies for a particular construction project by using the COVITE application. The application serves as a platform to bring together large number of suppliers and contractors to negotiate and procure the necessary supplies for the construction projects.

There is collaboration between the Specification Designers when designing the Product Classes and specification types. A Product Class is defined as a template made up of a number of different specification types. Product Classes help product suppliers to populate their databases with their own products in a structured, standardised manner. Collaboration takes place when a number of Specification Designers come together to design a Product Class or when the Product Class is being peer reviewed.

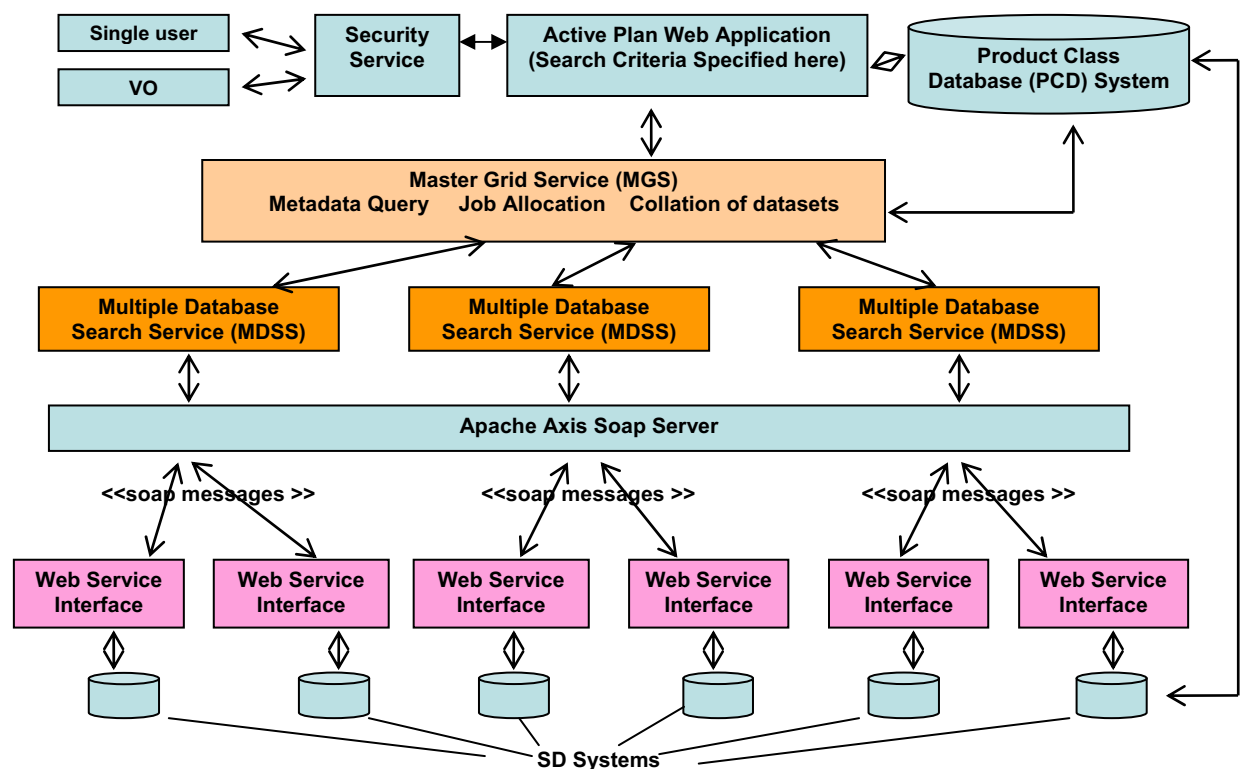


Figure 3 – COVITE flow architecture diagram

Collaboration also takes place when a search is made across a large number of supplier databases to retrieve products matching a criteria set by the purchasers or contractors. The MDSS enables a search to be conducted, making use of the cluster of machines in a Grid network to retrieve the matching products.

Figure 3 presents the flow architecture diagram of the COVITE application. The Security Service and the MDSS have been created as a result of investigating the applicability of Grid technologies in the sphere of AEC projects. Both services reside on the server side of the COVITE application. The security service defines a security framework for the COVITE application using the GSI [GSI04], while the MDSS is based on the Open Grid Services Architecture (OGSA) model [OGSA03] and provides a Grid service solution for processing a large amount of data. MDSS is oriented towards the specific purpose of serving the needs of the members of a VO within an AEC industry community. The Master Grid Service (MGS) distributes jobs to the Grid cluster running MDSS instances that performs the actual work of invoking Supplier Databases (SD) Systems and requesting appropriate data. SOAP messages [SOAP03] are created by the MDSS instances using an Apache Axis SOAP Server [COV04]. The SD Systems provide an XML based Web Service interface for the operations that can be performed by the MDSS [COV04].

2.2 Cat-COVITE application – requirements and concepts for Catallaxy

A centralised and hierarchical topology used in the COVITE project could be combined with a decentralized topology and form a peer-to-peer network architecture.

The centralised + decentralised system can provide great scalability, extensibility and fault-tolerance features. At a glance, this topology has a lack of security support due to the peers are highly dynamics in terms that they can join or leave anytime the peer-to-peer distributed system.

Peer-to-Peer computing architecture allows for decentralised application design, moving from centralised server models (like COVITE architecture) to a distributed model in which each peer can benefit and profit of being connected with other peers. In this type of distributed architecture, clients and servers have horizontal relationships rather than vertical relationships.

There are common features between P2P and Web Services technologies that can be taken into the consideration of the architecture design of Cat-COVITE prototype. Both technologies leverage a Service Oriented Architecture (SOA), with common stack for publishing and discovery across the network. In P2P architecture, a peer can be a provider, a consumer and/or a registrar in the same time, whereas in a Web Services a node is typically a producer and a consumer but not a registrar. The XML-based standards play an important key in the convergence of P2P and Web Services. Web Services can provide a very easy way to handle registration, discovery and content lookup for P2P applications, as well, via the new Web Service security standards, can ensure the integrity of data and services accessed by P2P software. As far as directory services are concerned, in the P2P architecture, the Web Service implementation of using a single central UDDI registry, which contains the service description of the Web Services, can be transformed into a decentralised node. XML-based Web services are an important technology in defining the business processes in P2P systems too, as they

allow easy communication between peers over the Internet in a platform- and software-independent way.

2.2.1 COVITE - from centralised approach to decentralised approach

Such centralised architecture used in the COVITE project [COV04] is considered in the CATNETS project in the context of interacting with other centralised and decentralised systems. Figure 4 presents an architecture scenario of Cat-COVITE application model, in the centralised use case, while Figure 5 shows an architecture scenario of Cat-COVITE application model, in the decentralised use case.

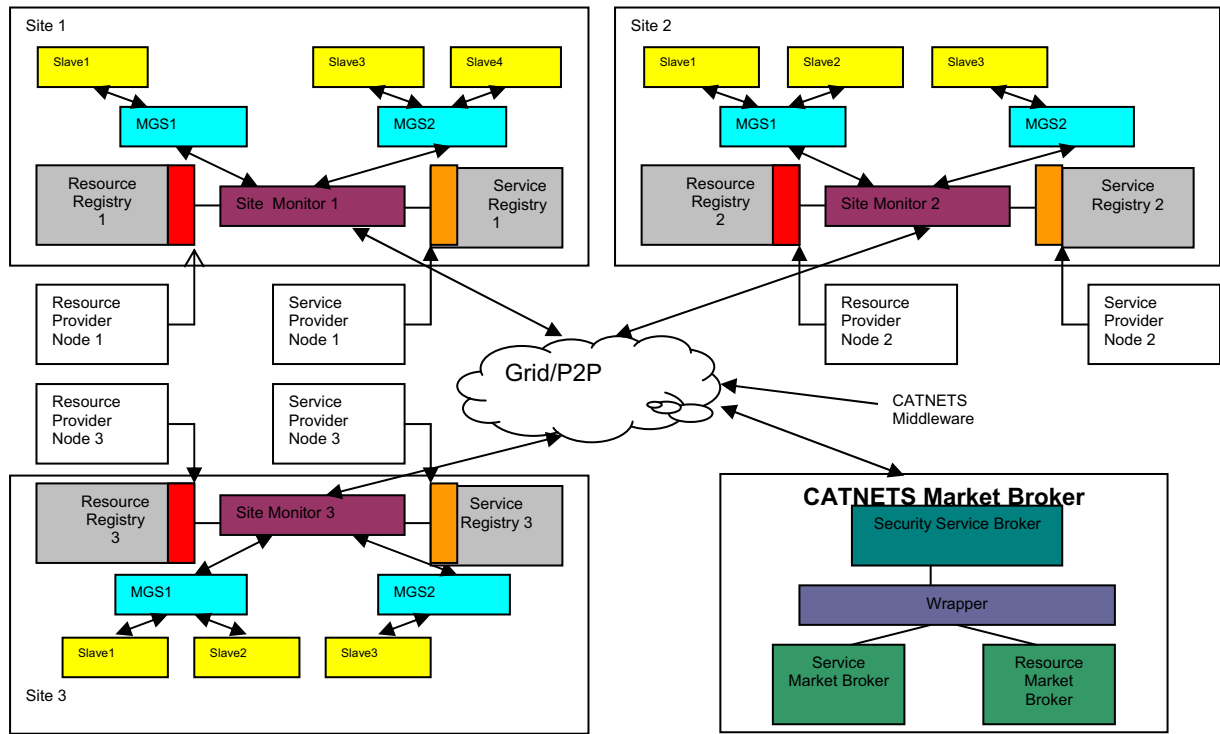


Figure 4 – Cat-COVITE architecture – centralised use case

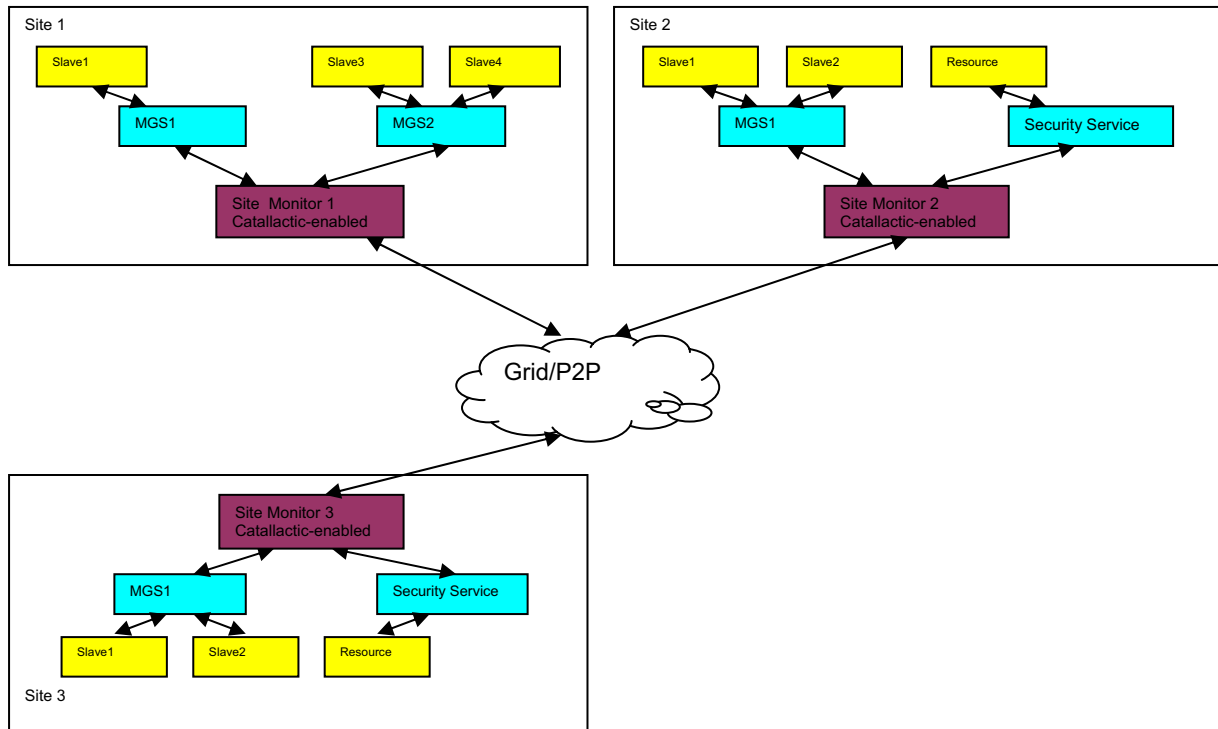


Figure 5 – Cat-COVITE architecture – decentralised use case

Glossary:

- *Site* – the authority that registers and controls all the resources and services made available by the registered Virtual Organisations (VOs).
- *Site Monitor* – the central registry of the Site which keeps control of all VOs created and registered within the Site authority.
- *Master Grid Services (MGS)* – The MGS is allocated to each Virtual Organisation created for an AEC project under the rules of the Site Monitor.
- *Slaves, Resources* - resources allocated to each VOs, registered to the MGS and used to fulfil all jobs within the VO. These Slaves, Resources could be allocated within the Site or could be bought from the market and integrated within the VO.
- *Service Registry i* (where $i = 1$ to n) would be responsible of registering all services available within the Site authority control. These registries could be in the form of UDDI registry specifications or in a database form (MySQL, SQL Server, and Access).
- *Resource Registry i* (where $i = 1$ to n) would be responsible of registering all resources available within the Site authority control. These registries could be in the form of UDDI registry specifications or in a database form (MySQL, SQL Server, and Access).
- *Resource Provider Node i* (where $i = 1$ to n) would be responsible for matching the requests from market resource agents with the available resources on site.

- *Service Provider Node i* (where $i = 1$ to n) would be responsible for matching the requests from market service agents with the available services on site.

The architecture consists a number of Site Monitors (SM) with a number of Master Grid Services (MGS) under their control, each of MGS having a cluster of Grid computers acting as slaves that perform the received jobs. In the context of the COVITE project, the communication between MGS and the cluster of Grid computers is XML-based using SOAP messages. Cat-COVITE application will keep the same characteristic of communication based on SOAP messages.

Site Monitors undertake an important role within the system, and are responsible for:

- Establishing and maintaining a P2P communications infrastructure.
- The point of propagation of “call for bids” market messages between Grids/P2P sites and other peers. An example is the use of an auction protocol, where such site monitors act as auctioneers for their own sites. The site monitor therefore acts as a control authority for a particular site participating in the market, as well as nodes in a P2P/Grid topology.
- They also act as a rendezvous point in a P2P topology – essentially supporting the caching of messages that are propagated in the network.
- They can also provide a service or resource registries – responsible for registering all services available within their site.

A resource provider node i would be responsible for matching the requests from market resource agents with the available resources on site. Similarly, a service provider node j would be responsible for matching the requests from market service agents with the available services on site. Both resource and service provider nodes will be agent based nodes capable of hosting agents that interact within the CATNETS market. Another function of these nodes will be to send notification messages, such as forwarding requests for resources/services to their neighbouring nodes. They may negotiate directly with the nearest nodes for resources/services.

2.3 Layer models – Cat-COVITE model architecture within Catallaxy mechanism.

The Figure 6 shows the layer architecture of the Catallaxy mechanism, while Figure 7 shows the mapping between Cat-COVITE architecture and the Catallaxy mechanism.

The Basic Service is a standardized service for query job execution. At the application layer, the Basic Service consists of:

- Seller entity on the service market
- Query Job Execution Environment (offers the deployment of „slaves“, which are able to execute the query)
- Translate query to resource demand

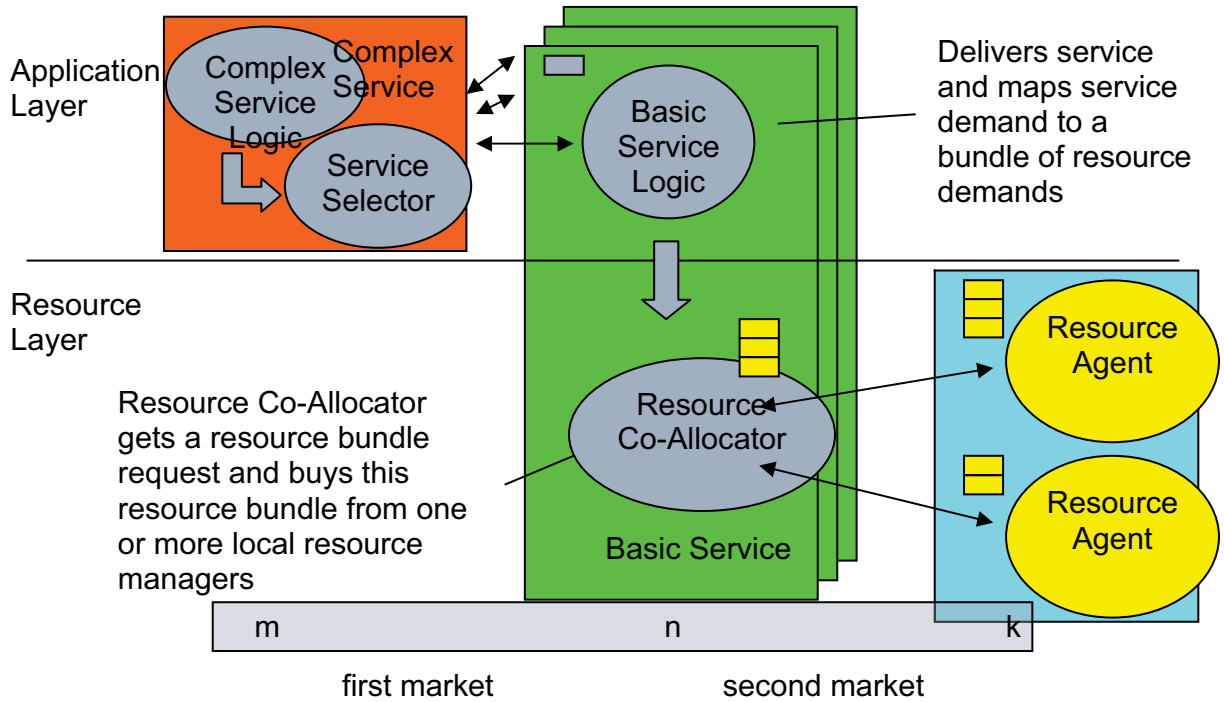


Figure 6 – CATNETS Catallactic Scenario

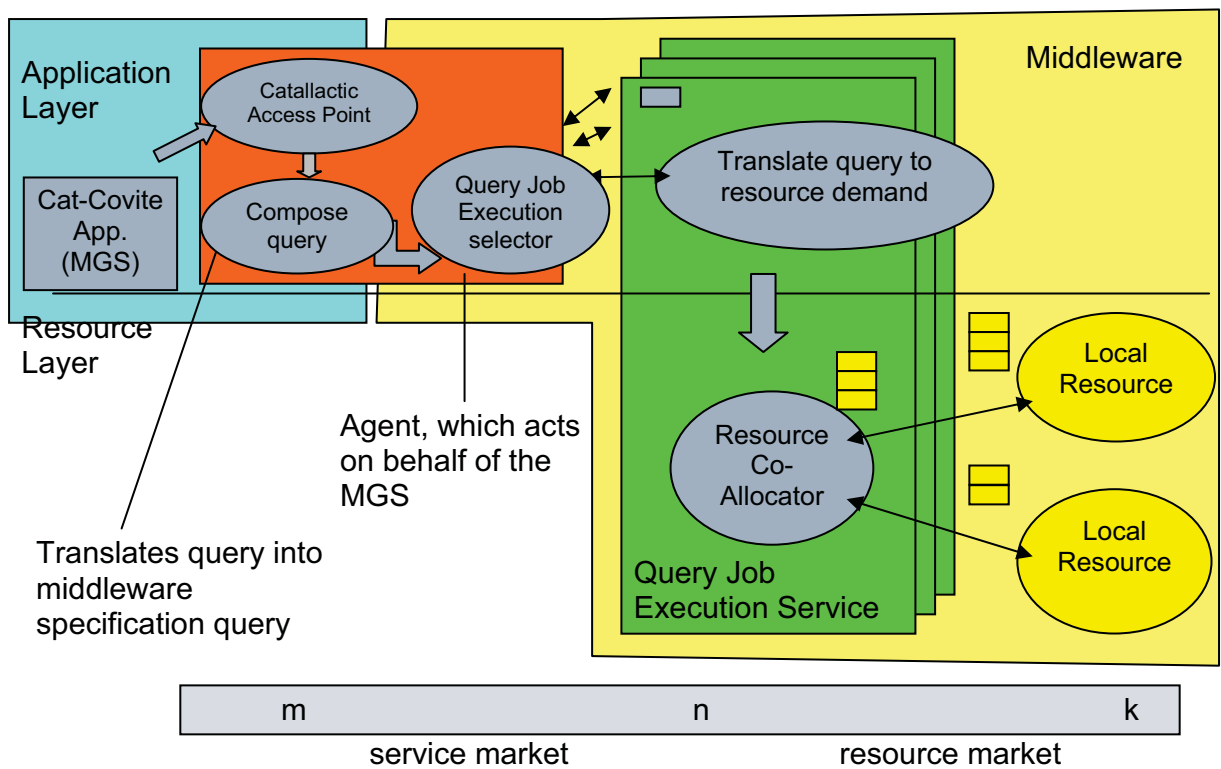


Figure 7 – Cat-COVITE CATNETS-enabled service overview

From the Cat-COVITE application point of view, the Basic Service Logic issues are the response time and the quality and quantity of the search.

The main functionalities of the Resource Co-Allocator, at the resource layer level, are:

- Represents the buyer entity on the resource market
- Co-allocator of resources (resource bundles) by parallel negotiation with different resource providers (local resource manager entities)
- Informs the Basic Service Logic about the outcome of the resource negotiation

Sellers' entities on the resource market are able to provide a set of resources via the Local Resource Manager (LRM). The Resource Agents act on behalf of these LRMs, which hide the physical resources behind them. More details about how markets are working can be found in the Deliverable D1 (WP1.1) [D1-WP1.1-05].

The term of Catallactic Access Point (CAP) represents the access point for the "on-demand" application requests. It makes use of WS-Agreement protocol to interact with the Master Grid Service (MGS) module of the application.

The term of complex service logic describes the following actions:

- Translates a request to a Basic Service - query service
- Starts parallel negotiation with a number of agents representing Query Job Execution Services (Basic Services) to fulfil any performance constraints identified in its contract (implemented via WS-Agreement)
- Sends a query to a list of Query Job Execution Services (Basic Services)

The term of service selector describe the following actions:

- The service selector is a query job execution selector.
- The service selector sends this query to a list of Query Job Execution Services (Basic Service Copies)
- Then starts parallel negotiation with them to fulfil the QoS constraints.

3 Catallactic middleware

WP3 Middleware Implementation focuses on the technical requirements of ALN middleware. It evaluates the available middleware toolkits used in Peer-to-Peer and grid implementations, identifies and implements specific components of the infrastructure required for the integration of economic enhanced components developed in WP2. These additional components will include extensions to the market environment, new and extended components for network agents, and new components for measuring performance of the ALN. A major deliverable of WP3 and a milestone for the project is a “ready-to-use” middleware, which will be used in the prototype application.

The term “application layer networks” (ALN) integrates different overlay network approaches, like Grid and P2P systems, on top of the Internet. Their common characteristic is the redundant, distributed provisioning and access of data, computation or application services, while hiding the heterogeneity of the service network from the user’s view [ERA+03].

The characteristics of the resource allocation scenarios being considered might be very variable, with very different usage scenarios like collaborative P2P networks, scientific P2P grids and P2P Content Distribution Networks (CDNs). Even when all those systems are P2P, they have a great variability in terms of key characteristics.

Table 1 summarizes some of those characteristics in an abstract ANL model (based on [Catn03] and [IaFo01]) and evaluates how it could impact the Catallactic middleware.

Characteristic	Description	Impact on middleware
Overlay topology	Topology of the logical network that ALN components uses to communicate (e.g. random, hierarchical, power law)	Communication mechanisms must adapt to diverse topologies to guarantee an efficient message routing
Configuration Dynamism	to what extent the ALN configuration is maintained in terms of participant nodes and overlay structure.	Information regarding resource location and network topology must be updated frequently
Resource Distribution	Resources in the network might be highly distributed among nodes or concentrated in few nodes	The overlay network architecture and the request forwarding algorithm for centralized resources can be hierarchical, whereas for distributed resources a flooding style might be more efficient (mostly when considered in highly dynamic environments)
Resource Diversity	From commodity resources to highly specialized, unique resources	Commodity resources can be easily located by local broadcasts or DHT-like mechanisms. Unique resources might need efficient network-wide discovery and match making algorithms
Usage Patterns	Clients might request same resources recurrently or each request might be unique	Recurrent request might benefit from caching information from previous requests (resource location, for example)

Table 1 - Characteristics of ANL applications

The main challenge is therefore to build a middleware architecture that could be adapted to different ALN architectures, what will define aspects like the logical topology used for communication, the characteristics of the nodes and the physical distribution of components.

To address this challenge, the Catallactic middleware has been envisioned as a set of economic agents that interact between them and with the software components of the underlying ALN, to coordinate, in a decentralized way and using economic criteria, the assignment of resources, as can be seen in the Figure 8.

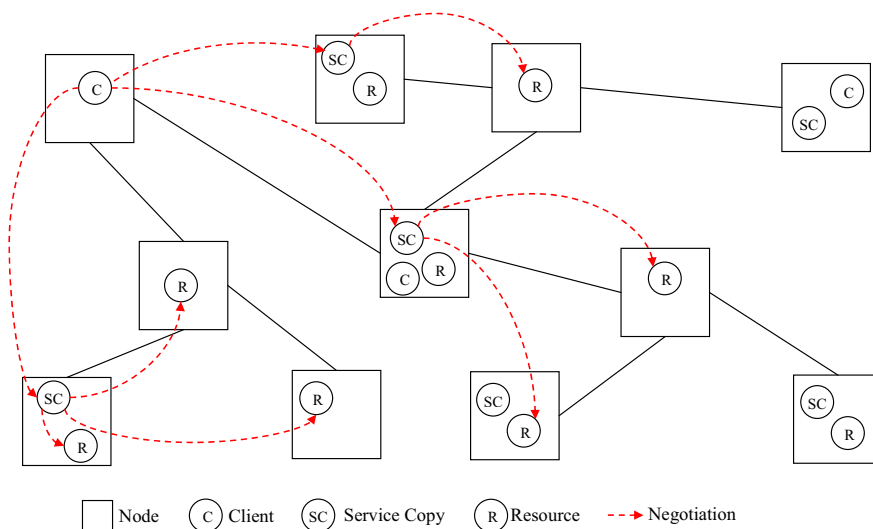


Figure 8 - CATNETS as a P2P network of agents.

In that vision, those agents interact under a P2P architecture. The term P2P should be interpreted not as an specific system architecture, but as a general approach for distributed system design ([Pepi03a]) that can be realized under very different architectures and topologies, ranging from unstructured and disperse networks to very centralized systems ([P2p02a], [MKL+02]). Annex B “P2P Architectures” surveys some architecture styles that can be adopted by P2P systems.

In the following sections we present the specific requirements that the Catallactic middleware should satisfy and the design principles that guided the process of designing its architecture. We also analyze their impact on the selection of the middleware architecture and implementation options.

The definition of the middleware has followed an architecture based development process, as described in annex A “The Architecture Development Process”, which can be summarized in three main activities: requirement analysis, design and validation. This deliverable covers the first two, while the last one, validation, will be addressed in future deliverables.

3.1 Requirements

- Scalability

The very essence of the CATNETS project is to use Catallactic mechanism to manage the resource allocation in very large ALNs, so the possibility to scale cannot

be limited by any design decision. The Catallactic middleware should be able to address scenarios with thousands of nodes in a highly dynamic environment, where nodes enter and leaves the network frequently.

The dynamism in the network configuration implies that information about the system should be maintained at a minimum (avoiding global topological information) and that updates must be easy and efficient. Also, under this scenario, the common assumption that nodes and resources are organized in well known and trusted administrative domains might not apply. So, excessive dependency of existing services on each administrative domain should be avoided. Scale also implies a high level of heterogeneity in applications, the underlying platform, resources, QoS of providers, reliability of any middleware service and availability of nodes (some will be quasi permanent, other will enter and leave).

- Compatibility with different base platforms

The design of the middleware should consider a generic design that allows the integration of different base platforms. This might lead to the definition of generic APIs and the definition of very flexible and extensible models to represent the platform's information (resources, for example). Also, some adaptors would be needed to translate this generic model to the specific model used for each platform. This translation mechanism could harm the performance of the system if transformations are complex or frequent.

- Allow the self-organization of components

The exact characteristics of the P2P architecture for the Catallactic middleware will be one of the key issues to be addressed in the design and implementation phases. However, all P2P systems exhibit a set of characteristics that are relevant from the architectural point of view [P2P02b, Pepi03a]:

- Decentralization: there is no single or centralized coordination nor administration point
- Symmetric interaction between peers: all peers are simultaneously clients and servers requesting service of, and providing service to, their network peers
- Non-deterministic topology: At any moment in time, the overall topology of a P2P network is completely unpredictable. The set of nodes that makes up the network varies constantly
- Heterogeneity: The devices contributing in P2P applications can differ in many respects, including communication bandwidth, available memory and the persistence of their network connections.
- Dynamic and virtual allocation of communication paths: due to communication paths between peers are created dynamically based on various factors, like network conjunction or intermediate peers state.

These characteristics, when considered together, lead to a set of astringent architectural requirements for self-organization. The dynamicity of the network prevents an a priori configuration of the peers or the maintenance of centralized

configuration files. Peer need to discover continuously the network characteristics and adapt accordingly, what requires a distribution of some important system functions like security, resource management, topology management, among other, which have been traditionally reserved to very specialized nodes.

As all the system function should be implemented in all peers and there have heterogeneous properties and configurations, all these self-organization functions should make little assumptions about the underlying platform's features.

- Support different implementation architectures

The Catallactic mechanisms could be implemented in different platforms and for a diversity of applications, each with its unique architecture regarding the organization of clients, service providers and brokers, as well as with respect to the communication topology.

Therefore, the Catallactic Agents (that implement the behavior of Clients, Services, Service Copies, and the Catallactic middleware components (that implements supporting functions like resource discovery, resource management, request processing, etcetera) will be deployed under different configurations and will use different communication patterns.

Different architecture will lead to different ways to organize and deploy the Catallactic components. Therefore, each component should not make any assumptions about a specific distribution. Basic functions of the Catallactic middleware should be implemented as independent agents instead of subroutines into a complex agent. This will facilitate their redistribution across the different components of the underlying platform and the applications that use it.

Different architecture models will lead to different interaction patterns between the base platform, the applications and the Catallactic middleware. Under some scenarios, the applications will make request for resources to the base platform, which will in turn, forward it to the Catallactic middleware (probably, using a component specifically modified to interact with it). In other scenarios, the application will make request directly to the Catallactic middleware (probably, using a component specifically modified to interact with it) which will interact with the base platform to fulfil it.

3.2 Concepts and design guidelines

To address the architectural requirements defined in the previous sections, we have defined a series of strategies that allows us to separate the different concerns and manage them individually without missing the coherence of the architecture as a whole. These strategies are summarized as follows:

- Isolate economic agents from the underlying ALN

Agents should rely in its ability to discover other agents and to efficiently communicate with them. However, due to the potential variability of the ALN's topology as well as the discovery and communication mechanisms, agents should neither be aware of the overlay topology nor make any assumption about its communication mechanisms.

On the other hand, the scalability of CANTNETS will be determined to great extent by the ability of the Catallactic middleware to efficiently handle a huge amount of nodes and resources in very dynamic environments.

Middleware will probably need to implement different algorithms to adapt to different scenarios (for example, adaptation to sudden changes in the network or disruptions). Also, different algorithms could be used simultaneously to search resources, combining strategies and increasing the success. It is therefore expected that discovery will be one of the components more likely to change.

However, isolating the economic agents from the agent discovery process should not limit the ability of agents to learn about the best peers to negotiate with, neither should it preclude the integration of agent level information (for example, success ratio of negotiations with other agents) into the adaptation mechanisms used by the middleware.

- Allow pluggable mechanisms and strategies

When implementing the Catallactic middleware, it is very probable that different mechanisms, strategies and policies might be considered to adapt the system to very different environments. Those components might even coexist, to allow a dynamic adaptation to the changing conditions. This will allow, for example, using two completely different requests forwarding algorithms to find local and remote peers, and deciding to use one or the other depending on the type of request, past experience or other environmental conditions.

For all major components, like resource discovery, request processing, negotiation and resource allocation, consider the separation of the basic mechanism from the decision making of how (and when) to use them.

- Use APIs with Xml based parameters

Many of the APIs for the different Catallactic middleware layers will handle information that will depend on the specific application domain and base platform used for implementation. For example, the resource discovery will return a list of resource descriptions, which depends on the kind of resources used by application: processors for a Grid, bandwidth for a CDN, and so forth.

Therefore, we found very restrictive to specify those APIs with concrete data types for their parameters, which will very probably be changed in each implementation scenario, and might require a massive software actualization.

This limitation can be overcome using Xml based data types in the middleware APIs, which can be extended or specialized on each specific implementation. Whenever possible, standard Xml formats should be used.

XML is highly flexible and interoperable but imposes a considerable runtime overhead in the manipulation of data.

- Create complex behaviour by interaction of simple agents

Traditional agent development approaches are based on the implementation of complex agents that exhibit sophisticated capabilities like learning or reasoning. However, this imposes some limitations on the protocols and algorithms that can be used in key concerns like agent discovery and negotiation, which are expected to

change during the design and implementation phase of the project, as the requirements are refined.

Instead, we propose that agent behaviour and negotiation algorithms should be expressed in terms of the interaction of multiple simple, specialized and efficient agents. These agents are responsible for basic functions like agent discovery, managing individual negotiations, message routing, message format handling, exception handling and message encryption ([ZaPa04], [HWBM02], [MKL+01]).

These agents will require a minimum execution platform, it will be easy to be implemented and collective behaviour could be adapted changing interaction patterns and including new agents.

Also, designing the system as a set of cooperating agents makes easier to change the distribution of functions among different nodes, either statically at deployment time or dynamically depending on the environment (work-load, requirement patterns), including the possibility of dynamic agent creation and agent migration.

Depending on the capabilities of the implementation platform and the performance issues that the interaction among many agents might generate, some these simple agents could be aggregated as specialized behaviours of one “heavy” agent. However, efficiency must be balanced with the flexibility of the implementation, because the specification of some key functions is expected to change along the implementation phase.

3.3 Architecture Structure

We structure the architecture in terms of the separation of two fundamental layers: the Middleware Services and the Framework [Bers96]. A Middleware Service is a general-purpose service that sits between platform and applications and is defined by the APIs and protocols it supports [Emme00].

The framework is a software environment, defined by a set of programming interfaces and tools, designed to simplify application development for a specific application domain.

This architecture, shown in Figure 9, is composed of five different layers:

- **Application Layer:** is given by the domain specific end user applications like collaboration tools, problem solving environments, and many others. Applications rely on the base platform for functions like communication and platform level resource management. However, applications can have application level resources, like a virtual meeting room in a collaboration tool or a matrix resolution algorithm in a scientific environment.

The interaction model between the application layer and the Catallactic middleware is application and middleware dependent. Application can interact directly with the Catallactic middleware (becoming Catallactic enabled applications) to manage their resources or they can interact transparently by means of the base platform they are built on.

- **Catallactic Algorithms Layer:** Implements economic algorithms for resource allocation. These algorithms should be domain independent and platform independent.

This layer is structured as a set of interacting agents that play the roles of Sellers and Buyers in service and resource markets. Also, in this layer are extensions and specializations of the functionalities provided by the underlying framework, to adapt them to the specific ALN and the resource allocation policies in place.

- **Catnets Framework Layer:** offers the primitives that supports the implementation of Catalactic algorithms, like find peers agents to negotiate, start negotiation, make a bid, wait for a bid. It is dependent on the agent platform being used, but should be independent of the application domain and the base platform.

This layer is structured in a set of basic entities that model the interaction of trading agents in a market to exchange goods. These abstract entities are the building blocks of the Catalactic algorithms.

- **P2P Agent Layer:** Platform that hosts the Catalactic agents offering a generic P2P application model with abstractions for the discovery and communication mechanism, and a generic interface with the underlying platform.

This layer offers a rich development environment, covering the basic functions that will be used by all implementations; it is responsible for interfacing with the underlying platform and complementing it when necessary.

- **Base Platform Layer:** Supports applications and Catalactic middleware. It is (potentially) domain specific.

The model of interaction with the Catalactic middleware depends on the architecture of the base platform, but in general will require the implementation of a connector, which routes the request for resources to the corresponding economic agents. In some cases, this might even require the re-implementation of some core platform components, like the GRAMs (Globus Resource Allocation Managers) in Globus [FKL+99].

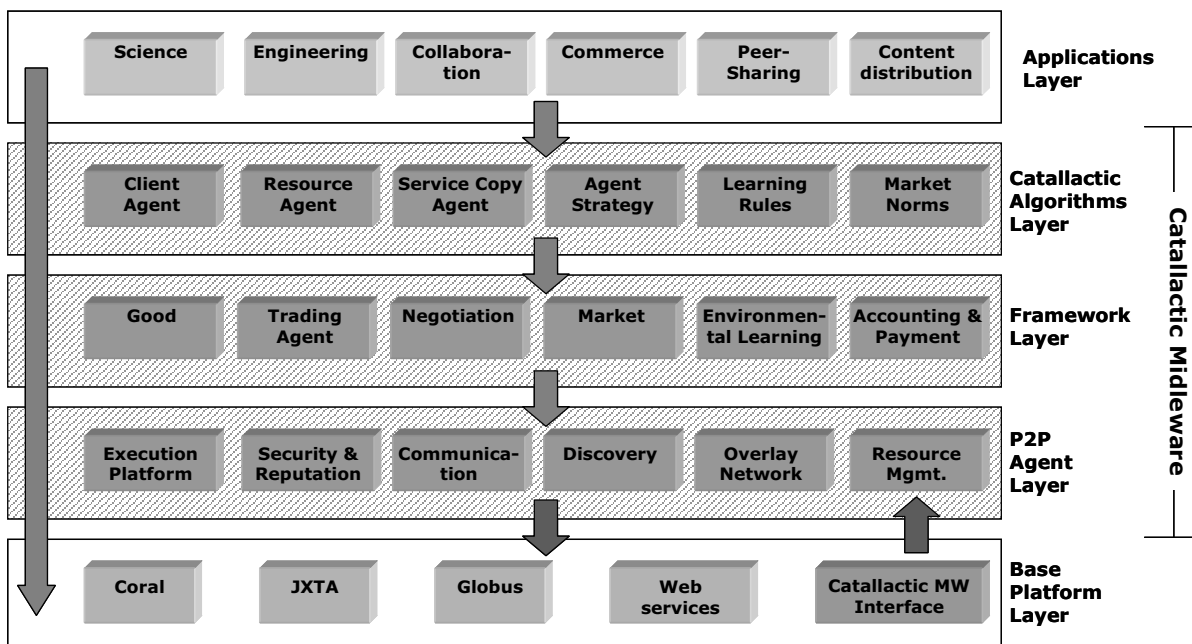


Figure 9 - CATNETS architecture – Layered View

3.4 Related Work on Architectures for Economic Based Resource Management

Both P2P and economy based resource allocation have received a great deal of attention in the last couple of years. Therefore, there are some projects that have coincidences in their goals with CATNETS and whose architectures can have some similarities.

MMAPPS project ([Mmap04a], [Mmap04b]) aimed to provide a toolkit for the development of P2P applications that uses economic based incentive mechanism that allows the coordination and optimization of these applications. The base architecture considered a set of applications that users employ to access services distributed in a P2P overlay network. These applications and services use a middleware which offers functions like group management, search, service management, security, negotiation, rules and policies enforcement, accounting and pricing. This architecture allows a transparent integration of economic mechanism for service negotiation into the application. MMAPPS considered that all applications and services would be developed using this framework, so integration of already existing applications was not considered (at least explicitly) in the design. This is a fundamental departure from CATNETS approach, where integrability to heterogeneous ALNs is a key design objective. However, we have found MMAPPS pluggable rules and policies an interesting approach for handling the adaptation of the Catallaxy market's rules to different environments and needs, which will be studied in detail during the design stage.

Related to the idea of economic based resource allocation is the GridBus Project ([BuVe04], [VBW05]), which applies concepts from the utility markets (e.g. power market) for resource allocation in grid applications. GridBus is based on a Service Market Directory, where application services are published, and a Service Broker, which matches the requests from users to the available resources considering the execution cost and diverse QoS parameters and looking for the optimization of the system wide utility. Our model, on the contrary, is a fully decentralized direct bargaining between producers and consumers and does not require any centralized market mechanism. This decentralization brings a higher scalability and a better adaptability to local resource requirements and to highly dynamic environments. The drawback is, however a less than optimal allocation of resources [Catn03].

Some few decentralized frameworks have been proposed in the literature, remarkably OCEAN [PHP+03] and Tycoon [LHF04]. OCEAN (Open Computation Exchange and Network) provides an open and portable software infrastructure to automated commercial buying and selling of computing resources over the Internet. Each OCEAN node that wants to buy resources uses a Matching service, which implements an optimized P2P search protocol, to find a set of potential sellers based on the description of the resources being requested. Then, an automatic negotiation process starts with each seller, based on the rules dynamically defined in a XML format. The ability to define negotiation rules is a remarkable characteristic of OCEAN that allows the adaptation of the economic model to diverse applications. The main limitation we found in this rule based approach is the lack of mechanisms for learning and adaptation to evolving environments. We found an agent based approach more suitable to achieve this level of adaptiveness.

Tycoon is a distributed market-based allocation architecture based on a local auctioning for resources on each node. Auctioneers receive fine grained requests of local resources from agents acting on behalf of applications and schedule them using efficient sealed bid auctions in a way that approximates proportional share, allowing high resource utilization rates and the adaptation to changes in demand and/or supply. One interesting feature of Tycoon is that it separates the allocation mechanism from the agents which interprets application and user preferences. This allows the specialization of agent different applications. Tycoon however doesn't offer any framework for the construction of those agents.

A major limitation of Tycoon is that the resource allocation mechanism is already fixed in the system design and no extension or adaptation methods are offered. To overcome this limitation, our proposed framework is capable to plug key components to adapt to specific application domain in environments with heterogeneous or changing resource allocation requirements. Also, we offer a set of high level tools to develop those components, alleviating the implementation burden for new market designs.

3.5 Components and mechanisms

To understand the interrelationships between the components of the architecture, it is necessary to see how they interact in different scenarios, being the more relevant: the initial registry of resources in markets, the distributed resource search, and the bargaining process.

3.5.1 Registering resources and agents

Negotiation for resources is carried out by *Broker Agents* representing the client requesting a resource, and *Resource Agents* representing resource providers. How those agents are actually created is very dependant on the scenario and the architecture of the systems requesting the resource and offering it. Figure 10 shows a generic situation.

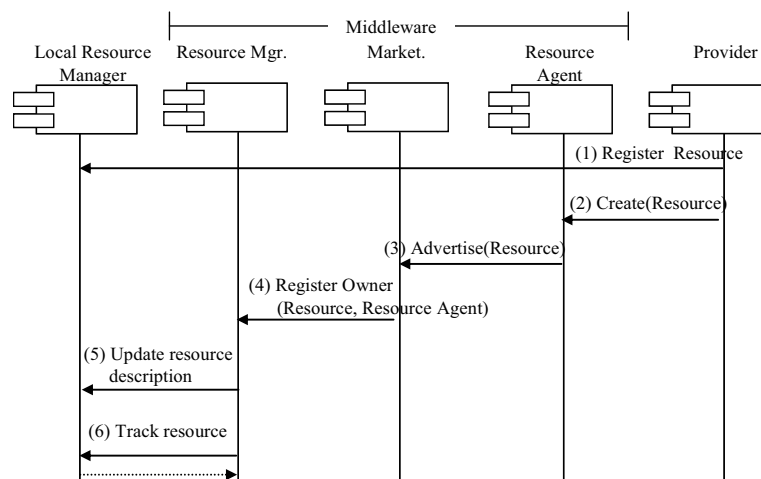


Figure 10 - Registering Agents and resources

The *Service Provider* is responsible for creating and registering a resource with its platform specific *Local Resource Manager*. It then instantiates a *Resource Agent*, which registers itself to the *Market Agent* to handle negotiations for that resource. The *Market*

Agent uses the *Resource Manager Agent (RMA)* to associate the *Resource Agent* with the specific resource. The *RMA* can, optionally, update the resource's information in the *Local Resource Manager* to reflect, for instance, that the resource is already reserved by the middleware and cannot be offered to other application. Finally, the *RMA* keeps track of the resource state (e.g. availability and usage level) and uses this information to answer queries for resources given a certain characteristics.

3.5.2 Resource Discovery

One of the more critical functions in the middleware is the discovery of resources in a fully decentralized way, within a dynamic set of nodes (that can enter and leave the network at any time) which must cooperate in the location of objects.

To address these requirements, the proposed architecture separates three critical functions: the *Overlay Network Agent (ONA)* manages the construction and maintenance of the logical topology of the network, keeping track of other nodes. *Communication Agent (CA)* manages the complexities of multicasting messages over the overlay network and the *Object Discovery Agent (ODA)* handles the search requests using a set of pluggable *Query Resolver Agents (QRA)*, which specializes in the location of a specific kind of objects. *ODA* is also responsible to coordinate the search with other nodes when no local information is available. Figure 11 shows how all these components interact to fulfil a search request.

First, *ODA* registers to the *CA* as a listener of multicast messages send to the "ObjectDiscovery" group; the *QRA* register to the *ODA* as a handler for queries of a specific object type. When the *ODA* receives a request, forwards it to the corresponding local *QRA* (if any) and to other remote *ODA* sending a multicast message with the query.

Before sending the multicast, the *CA* requests a list of known nodes to the *ON* agent. Before sending the message, the *CA* agent asks the *ODA* to filter and prioritize this list, allowing the introduction of heuristics in the search (for example, based on the results from previous requests [IaFo01]). For each selected node, *CA* agent requests the *ONA* to route the message using its knowledge of the logical network's topology.

On each node, the *ODA* receives the multicasted message and forwards to the locally registered *RA* (if any) and returns the response to the originating node, where the *ODA* sends back to the requestor. It can also, optionally, store this response in a local cache to enhance performance of subsequent requests.

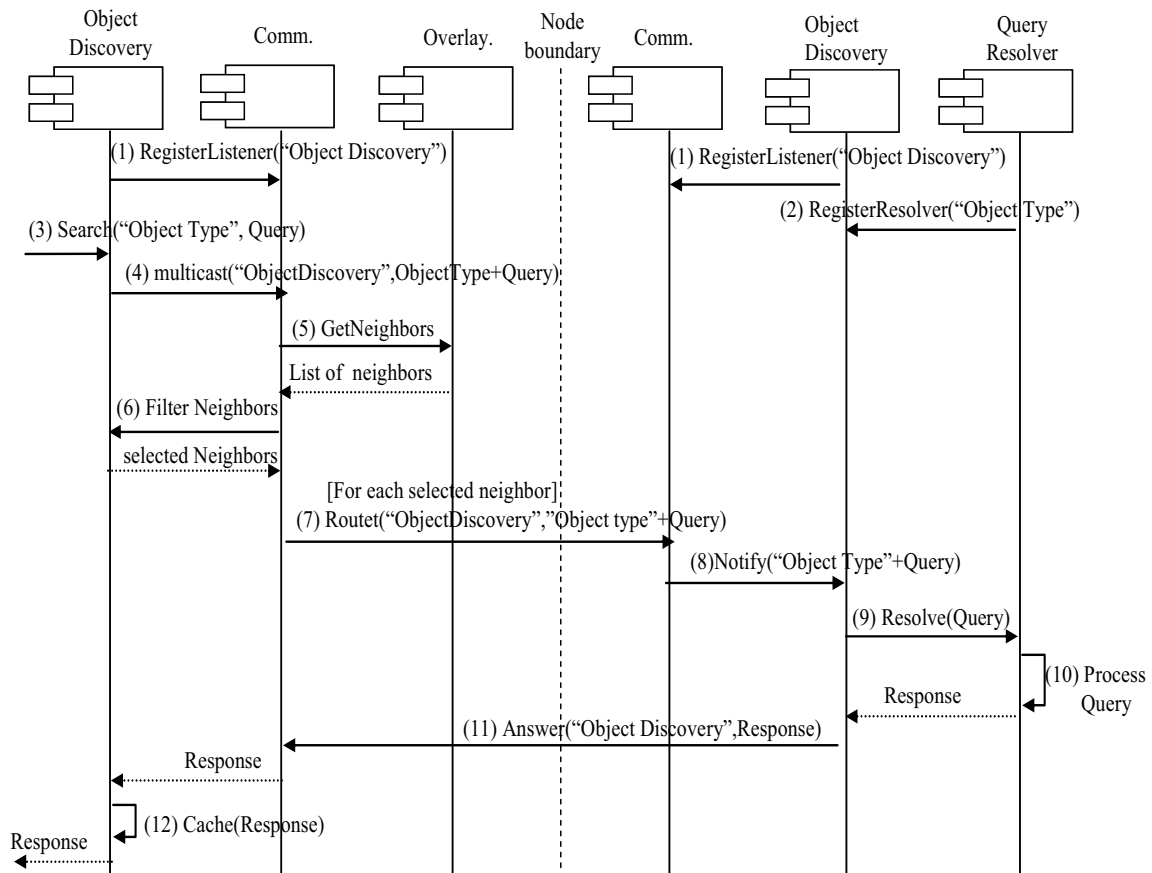


Figure 11 - Fully decentralised object discovery

3.5.3 Negotiating for resources

Negotiation process begins when a *Client Application (CA)* request a resource to the *Broker Agent (BA)*, giving some contractual conditions (e.g. available budget) and technical specifications. Figure 12 shows this process.

How the *CA* communicates with the *BA* depends on the application scenario. The *CA* can be fully aware of the *BA* or this agent can be invoked by a component in the *CA*'s platform (a local resource manager, for instance. Also, the conditions and specifications can be explicitly given by the *CA* to the *BA*, be part of the middleware's configuration parameters or a result of the *BA* learning process.

After receiving the request, the *BA* asks the *Market Agent (MA)* for a list of potential *Resource Agents (RA)*. The *MA* performs a distributed search to find the resources that match the specifications and the *RAs* that handle the negotiation for those resources. Then the *MA* filters the *SA* according to the contractual conditions. The *BA* selects from the given list the *RA(s)* it want to trade with (based on previous experience, for instance) and starts the negotiation process. The *MA* has the right to allow or not the negotiation based on rules governing the market.

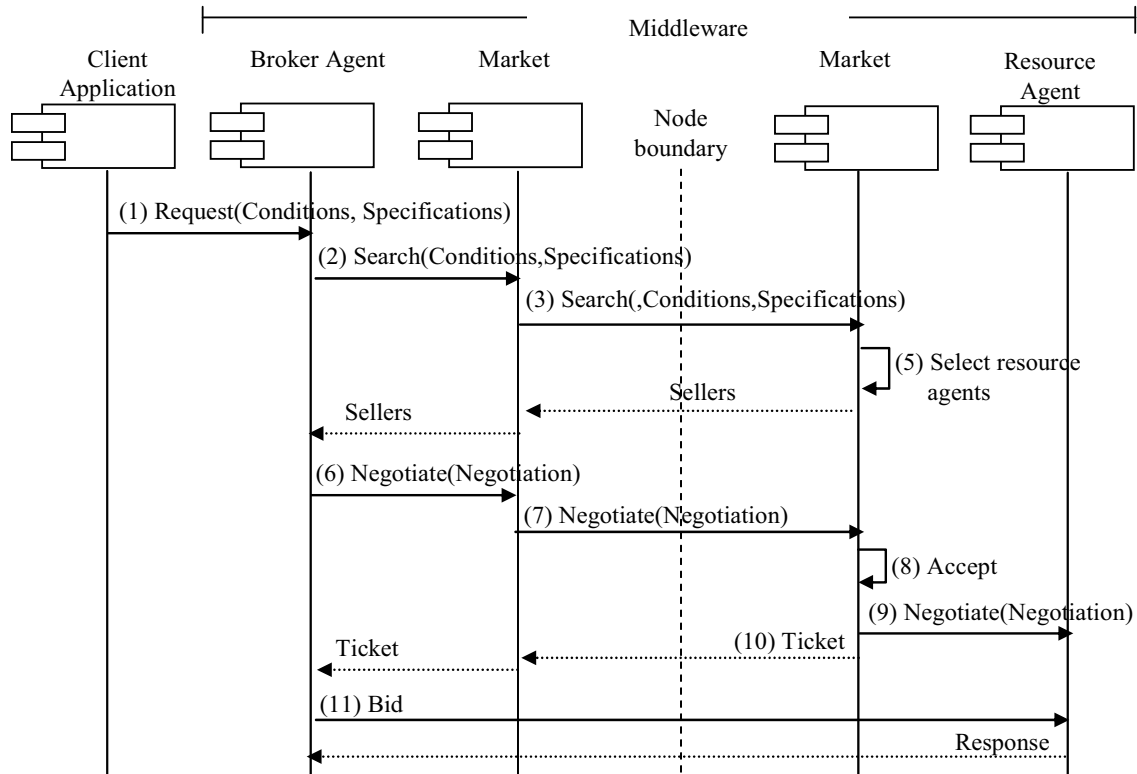


Figure 12 - Negotiating for resource

3.6 Implementation of prototype

3.6.1 Implementation toolkit selection

The middleware toolkits selection process was carried out taking into account three different but related aspects, which are potential application scenarios, software architecture, and the evaluation of a number of middleware toolkits.

Concerning the evaluation, six toolkits were selected and reviewed: DIET and JADE agent platforms, J2SE, WSRF/OGSA, Web Services and JXTA. The evaluation includes their functional properties according to the software architecture defined for Catnets, their technical characteristics and their suitability as a development toolkit.

From the functional view we conclude that complementary features between the middleware toolkits exist, which should be exploited to build the CATNETS middleware. The flexibility of the proposed architecture should allow to use it for different ALN domains.

Concerning technical features, the solutions provided by the different candidates could also be complementary, in terms of scalability, messaging performance, discovery performance and interoperability. Therefore, to address the above requirements, it would be necessary to compose an architecture that integrates the best implementation approaches offered by the different toolkits. For example, performance enhancements could be achieved by a light weighted agent implementation as in DIET, interoperability would benefit from a web services based communication and scalability could be achieved by a strong decentralization of key functions, as in JXTA.

We discard JADE for its lack of architectural flexibility, forced by FIPA standards compliance, and its problems to scale. With respect of J2SE imposes a huge load of low level implementation which should be avoided if possible using what is already available in existing toolkits. Finally, we consider that proposed Web Services standards are still immature and many lack any reference implementation, what will lead to a high implementation risk and probably will require a lot of implementation effort. However, basic Web Services standards like SOAP and WSDL, offer a good deal of interoperability and will therefore still considered in the implementation.

A condensed view of all requirements, in **functional, technical and development** views is obtained considering the following criteria:

- **Modularity** to achieve architectural flexibility required to implement the Catalactic middleware into different platforms and using diverse middleware toolkits
- **Amenability**: The middleware toolkit should be able to cover as much as possible of the ALN domains, like Grid, P2P and CDN
- **Performance & Scalability**: The middleware toolkit should allow the organization of a huge number of software agents in a decentralized way, and their interactions.
- **Completeness**: The set of functionalities provided by the middleware toolkit should allow covering as much as possible of the desired requirements of the P2P Agent Layer).
- **Development**: In order to support the CATNETS middleware development, the middleware toolkits should provide be mature, have a rich set of development tools and good documentation.

Figure 13 shows an illustration of this unified view. Each of the pentagon axis represents one of the criteria. For each criteria the two middleware toolkits best covering it are indicated.

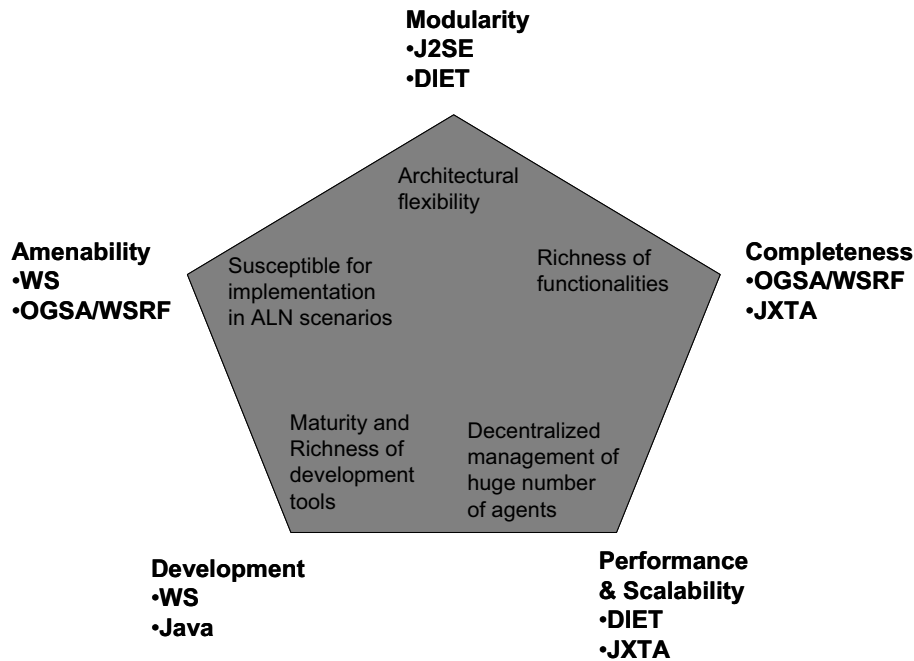


Figure 13 - Condensed view of the key middleware toolkit evaluation criteria

It can be seen in the previous figure that CATNETS middleware will be composition of different middleware toolkits, like DIET with JXTA and WSRF/OGSA, which achieve a good balance between the functional and non functional requirements.

Tests we carried out on middleware toolkits confirm the feasibility of this composition, in the sense that we could integrate JXTA Discovery with the DIET Agents platform. Also the invocation of Grid Services from Java applications using the Java Globus API has been tested.

The actual composition of the “proof of concept” middleware depends on the characteristics of the application to be used in CATNETS. Once the requirements imposed by such application are determined, middleware toolkits can be matched with application requirements.

3.6.2 Implementation approach

Previously to any implementation effort, an extensive survey on implementation tools has been conveyed. A detailed description of the process and its results is described in annex D “Middleware toolkits evaluation”. The middleware was implemented as a set of simple, specialized DIET agents. The details on the interaction of these agents are organized in a way that resembles the separation of concerns proposed by the architecture, as is shown in Figure 14. Framework agents supports the basic functions needed to implement economic algorithms, like access to markets. Peer Agent Layer agents implement the low level functionalities to support system execution. Service Provider agents interface with the implementation platforms (JXTA and GT4).

The Overlay Network, Object Discovery and Communication functions were implemented using JXTA Peer Resolver Protocol in a network of Rendezvous Peers that uses a DHT to maintain and route messages among nodes [TAP03]. The

management of local resources, in this case services offered by the service providers, uses the WSRF framework offered by GT4.

In the proposed implementation, Trading Agents implementing economic algorithms share the supporting agents from lower levels. This separation allows for scalability as the number of supporting agents can be dynamically adapted to workload. This separation also offers location transparency for agents.

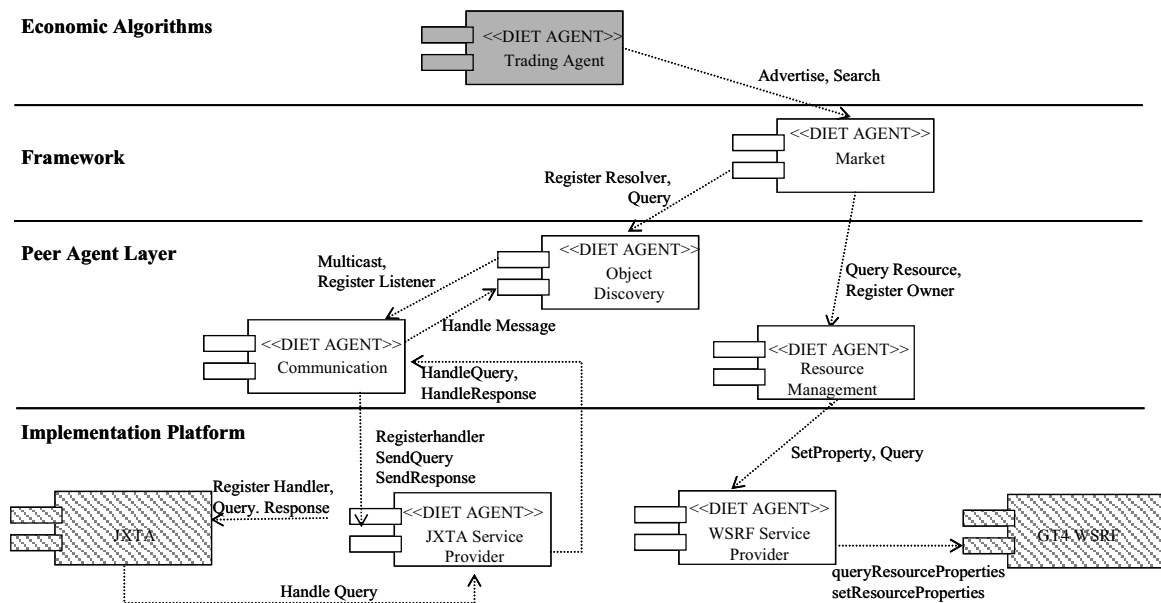


Figure 14 - Organisation of middleware components

3.6.3 Implementation of P2P Agent Communication Layer

The current implementation uses the JXTA platform to maintain a self-organizing overlay network. JXTA has been selected for P2P communication for its architectural flexibility and the rich set of features offered by such a large community-oriented project.

P2P communication in CATNETS requires both high scalability and complex queries resolution, which is currently beyond the state of the art in P2P Search. We expect to leverage current simulation results [SSPS05] and exploratory research on P2P search [LiWu05], but considering the features of currently available toolkits for P2P discovery, we decided to implement our P2P communications using JXTA Resolver Service. This service allows for complex queries resolution better than the higher level JXTA Discovery Service. We decided to schedule for a latter stage in the project the introduction of novel query diffusion algorithms on the self-organized JXTA topology, leading to a more efficient message routing and re-propagation schemes and resulting in the required high scalability level.

The JXTA ResolverService is used to implement the following functionalities:

- Multicast of a query with responses
- Multicast of messages (no response)

- Handle queries sent by other nodes and send back responses to the issuer

Figure 15 shows a scenario where a QueryIssuer can multicast a query requesting a response for nearby (according to the overlay network structure) nodes.

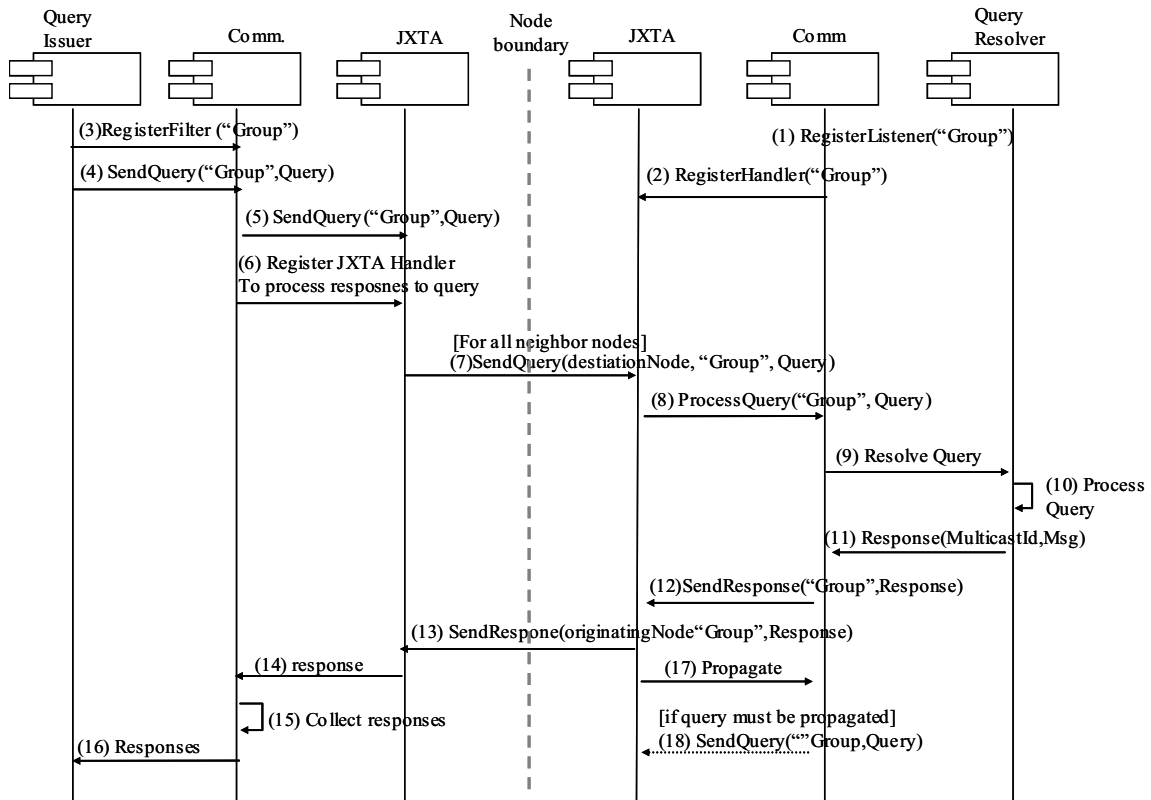


Figure 15 - Query resolution using multicast with response service

The *Query Resolver* (QR) registers in the *Communications Service* (CS) as a listener to receive queries multicasts directed to a given group. CS registers in JXTA a *QueryProcessing* handler to process incoming queries directed to the group.

Query Issuer request a query to be multicast to a given group. The *Communications Service* send a query message to each neighbour using the *JXTA ResolverService*.

On each node, the *QueryProcessing JXTA Handler* registered for the group receives the query and forwards it to the *Communication Service*, which in turn passes it to any registered listener.

Listener process the query and send it back the response correlated to the received query to the requesting node, where the *Communication Service* gathers response Messages from all remote nodes, composes a response and sends it back to the *Query Issuer*.

On each target node, after processing the query, the *Communication Service* depending on multicast options and query TLL, will propagate or not the query to nearby nodes. Also, if during this process a duplicated query message is received, the *Communication Service* discards the message.

The proposed model of the multicast process can be optimized at the routing level to improve the scalability reducing number of messages needed (by means of tree-based broadcast models and some caching of results), and the efficiency of the query matching efficiency using heuristics that drives the diffusion of query messages (based on past results, for example).

3.6.4 GT4 as Query Resolver

So far we have been exploring the integration of the P2P communication mechanism described in the previous section with GT4 information services, in order to provide Grid services P2P search mechanism. We use GT4 as a resolver for queries on services and resources. The local resource manager gets registered in its local market as a resolver for incoming request related to services and resources. The Resource Manager translates the query coming from the P2P communication layer into an Xpath expression suitable for a WSRF-query to be issued against the targeted Grid Services.

Figure 16 shows the interaction between the middleware components and base platform GT4 in order to address resource management. It is important to remember that queries arrive to local nodes in a P2P fashion and all the query flow is managed as depicted in figure 15. Figure 16 just details the concrete resource management issues for a Grid environment using GT4 and ganglia. This basic resource management setting will be modified in next development iterations in order to better use the GT4 Information Services tools.

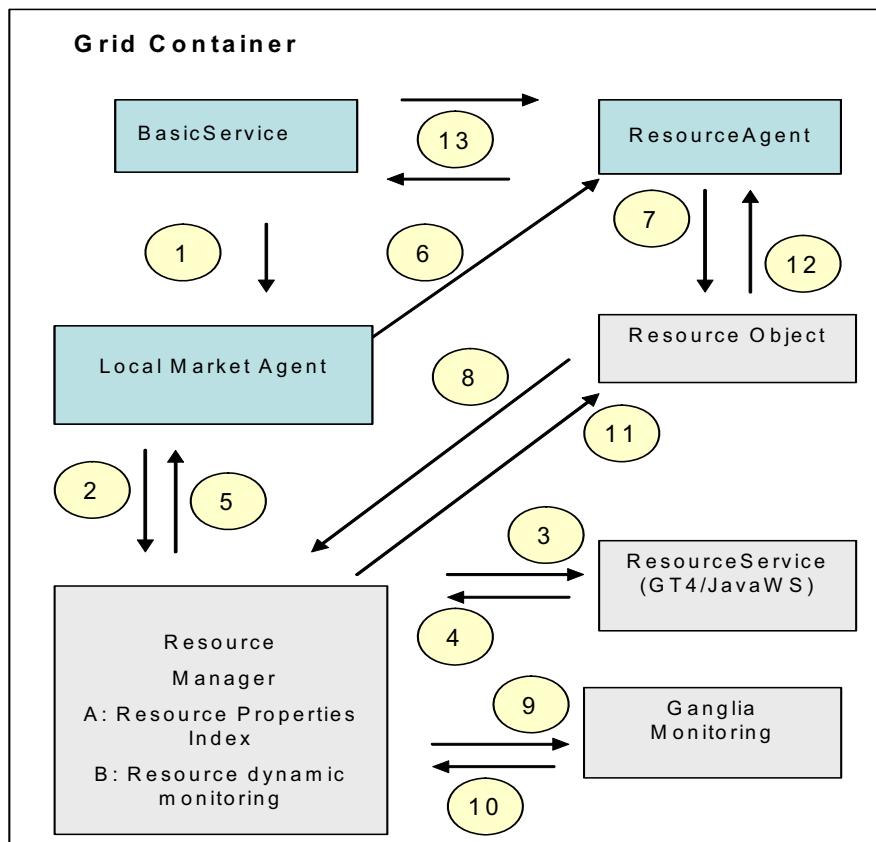


Figure 16 – Resource Management

1. Query arrives to Local Market agent, previously registered as resolver
2. Query resource manager to perform function A, providing access to resource properties of registered GT4 Resources
3. Query resource properties on GT4 resource (XPath)
4. Answer with resource properties values
5. Provide LocalMarketAgent with query answers
6. Instantiate ResourceAgent to negotiate for the resource:
InstantiateResourceAgent(resource)
7. Query resource object for dynamic resource properties in order to set resource prices
8. Query ResourceManager to perform function B, querying for dynamic properties of a given resource
9. Query dynamic resource properties of resource using ganglia monitoring
10. Answer with dynamic resource properties values
11. Provide resource object with dynamic resource properties values
12. Return dynamic resource properties values to Requesting ResourceAgent
13. Perform negotiation (steps from 7 to 12 can be performed again at any step during negotiation process).

4 Integration of middleware and Cat-COVITE application

4.1 Application and Catallactic middleware interaction description

4.1.1 Logical view

In this section we present the logical architecture of the integration between the application and the middleware, describing the flow of information between the components. Figure 17 depicts at a high level view of the architecture, highlighting the placement of logical components along the three layers, the application layer, the catallactic middleware layer and the Base platform layer, as explained in Chapter 3.

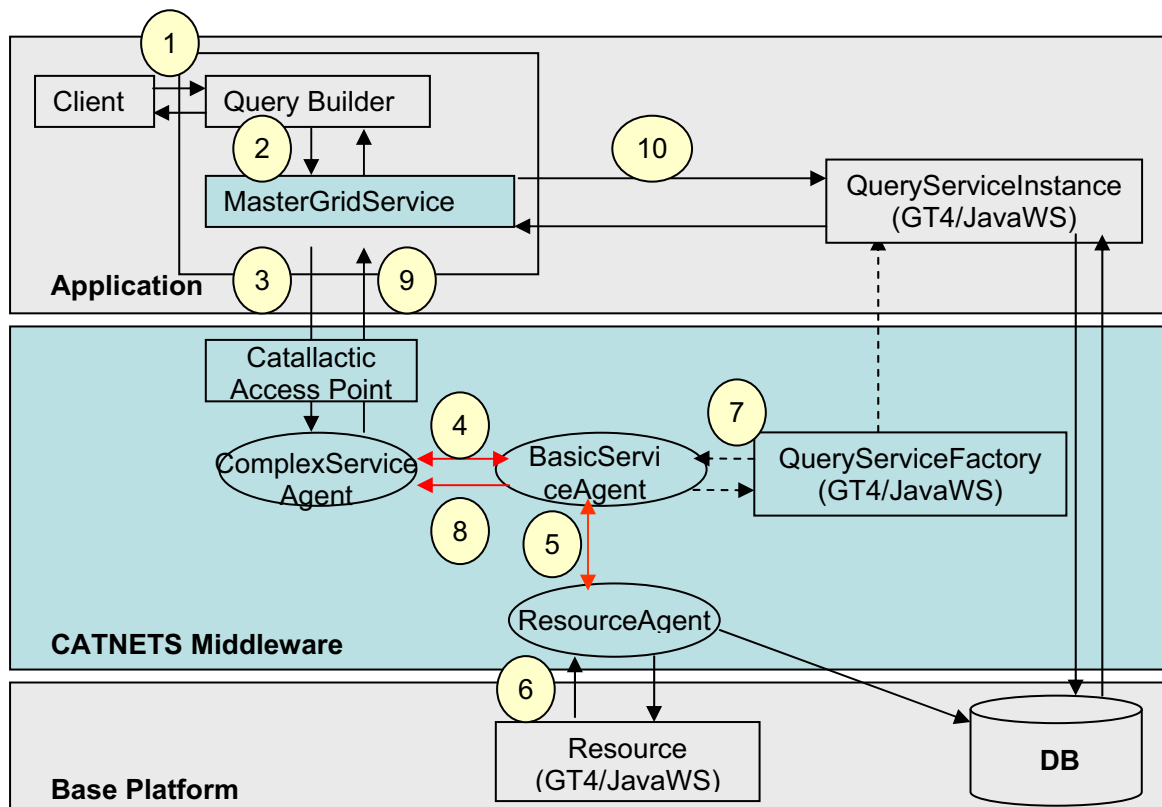


Figure 17 – Integration of Catallactic Middleware and Cat-COVITE Grid Application

At the application layer, the application must provide an interface to the middleware which must issue the requests for services to the middleware and use the references to service instances provided by the middleware to execute such services.

At the middleware layer, a set of agents provide the capabilities to negotiate for services and the resources needed to execute them. The Complex Service Agent acting on behalf of the application initiates the negotiation. Basic Service and Resource agents manage the negotiation for services and resources, respectively. Also, a Service Factory is provided to instantiate the service on the execution environment selected during the negotiation process.

Finally, at the Base Platform layer, a Resource is created to manage the allocation of resources to the service. This resource represents the “state” of the service from the perspective of the middleware (notice, this doesn’t mean the service is statefull from the perspective of the application).

The flow of information among the logical components can be summarized as follows:

1. Client issues a request to the application
2. Application builds a query and requests the execution of query to the Master Grid Service (MGS)
3. MSG requests to the Catallactic Access Point a Query Service to execute the query
4. Complex Service Agent starts the negotiation with different Basic Service Agents, each representing an instance of the Query Service on a different container.
5. Each Basic Service Agent negotiates with the resource agent(s) for the resource(s) needed to execute the query. Basic Service Agent uses the terms agreed with the Resource Agent to negotiate with the Complex Service Agent.
6. When an agreement with a Basic Service is reached, the Resource Agent instantiate a Resource to keep track of the allocated resources and returns to the Basic Service Agent a handle for this resource.
7. Basic Service Agent uses the Query Service Factory to instantiate the Query Service on the selected GT4 container
8. Basic Service Agent returns to the Complex Service Agent the reference to the newly instantiated Query Service and the related resource(s)
9. The reference to the Query Service is returned to the MSG, which uses it to invoke the service, passing the query to be executed.
10. Query Service is invoked and results are returned to the application.

4.1.2 Physical Deployment on GT4 containers

The logical architecture depicted in the previous section can be implemented in different ways depending on the architecture of the application and the base platform. In this section we present a specific implementation architecture on a GT4 based platform. Figure 18 shows the proposed physical distribution of the application and middleware components.

In this architecture some assumptions are made. First, the services are previously deployed on a set of GT4 containers. Second, the only “resource” considered in the negotiation are the “rights” to execute the service on a specific container. Finally, the service can be instantiated on a container using a generic factory.

The Application resides in a host (or series of hosts) where also resides the Master Grid Service (interface with the middleware) and the Complex Service Agent, which represents the application in the negotiation process. On each Grid Container (GT4)

where the Query Service is deployed, resides the corresponding Basic Service Agent, which negotiates with the Complex Service Agent for access to the Query Service. In this container also resides the Resource Agent, which negotiates with the Service Agent for the rights to execute the Query Service in this container. Finally, a Resource is created as result of the negotiation process, which represents the “rights” to execute the service in this container.

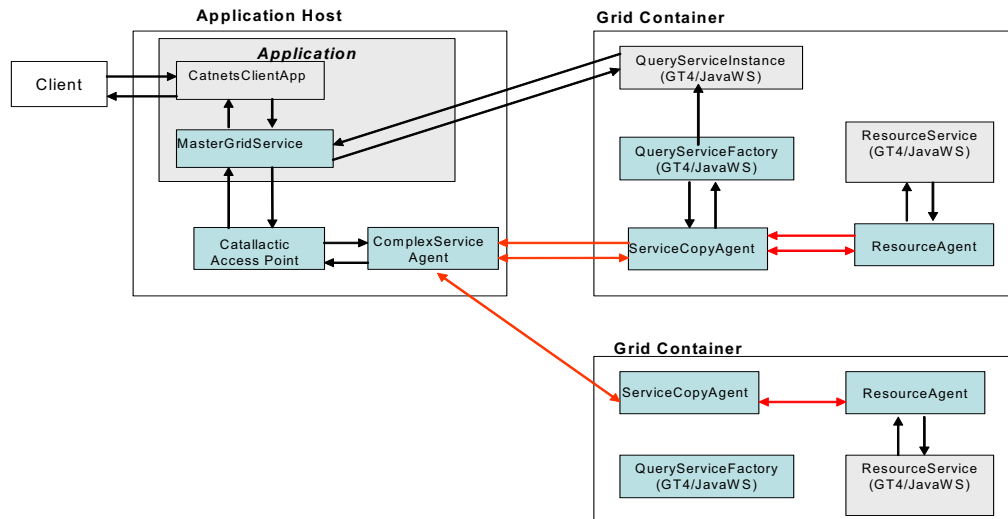


Figure 18 – Physical deployment of application and middleware components

4.2 Use cases – description, flow diagrams, sequence diagrams, actions

Three use cases are to be taken into consideration:

1. Complex Service agent contacts Basic Services on the Service Market to requesting plain resources for a limited time to deal with execution of the application service (a query job service). The first scenario is about just plain service brokers, as Basic Services, which find plain resources or bundle of resources on the resource market. The application has to have the permission to deploy and instantiate instances of Grid Service Handles (GSHs) services on these resources in order to deal with the application job request.
2. Complex Service agent contacts Basic Services on the Service Market to requesting query job execution services. Basic Services will request resources, on the Resource Market, having already available application service oriented instances (Grid Service Handle - GSH). These GSHs will execute the query job initiated by the client input request and distributed by the Master Grid Service via the QueryServiceInstance (see Figure 17).
3. Complex Service contacts Basic Services to requesting the query job execution service. Complex Service look up in the local Service Registry to see if there are local service instances (GSHs) – Basic Services, with bundled resources that can deal with the application job execution (query job).

1. Use case 1

Basic Services are just plain services which find resources or bundle of resources on the Resource Market. The following two sub-scenarios could be further consider:

- The application has to have the permission to deploy and instantiate instances of application services (GSHs) on these resources. In the service market, the tasks of Basic Services will basically be of offering resources or bundle of resources at the requested parameters or attributes specified by the Complex Service (e.g. numbers of CPUs, type of CPUs, period of time, price etc.). In this scenario, the XML messages between the Complex Service and the Basic Service could have two sets of attributes, one related to the service requests, and the other related to the resource requests.
- The Basic Services have to have the permission to deploy and instantiate instances of Grid Service Handles (GSHs) services on these resources in order to deal with the application job request. In this scenario, the XML messages between the Complex Service and Basic Service could have one set of attributes related to the job execution of a query job service.

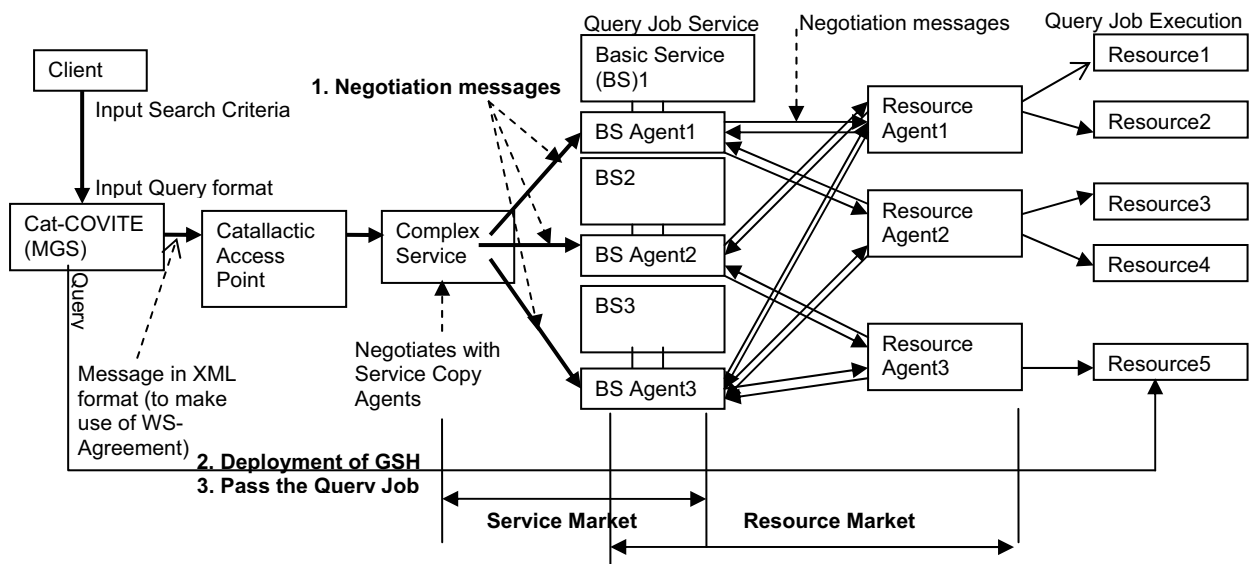


Figure 19 – Flow diagram – use case 1

Glossary:

- Catallactic Access Point (CAP)* = access point for the “on-demand” application requests of job execution
- Complex Service (CS)* = brokers that offer basic services or bundle of basic services
- Basic Service (BS)* = brokers that offer plain resource or bundle of resources
- Basic Service Agents* = agents acting on behalf of Basic Services

- *Resource Agents* = Agents acting on behalf of a resource or bundle of resources (in the Cat-COVITE application example the bundle consist of machine and database), which negotiate with Basic Service Agents

The flow of actions among the logical components is shown in Figure 19, while Figure 20 presents an UML sequence diagram of the interaction between application and middleware.

The flow of information is summarized as follows:

1. Client inputs a search criterion.
2. Cat-COVITE (MGS - “left side”) transforms the search criteria in a query format.
3. Cat-COVITE (MGS – “right side”) contacts the *Catallactic Access Point*, via a XML message, requesting a complex job execution.
4. On the Service Market – *Complex Service Agent* negotiates with different *Basic Service Agents* (which offer only plain resources) to find resources to execute the requested by the application.
5. On the Resource Market – *Basic Service Agents* negotiate with *Resource Agents* to buy resources.
6. Resources are embedded within the application, through the deployment of the Grid Service Handles, which execute the query job based on the client search criteria.

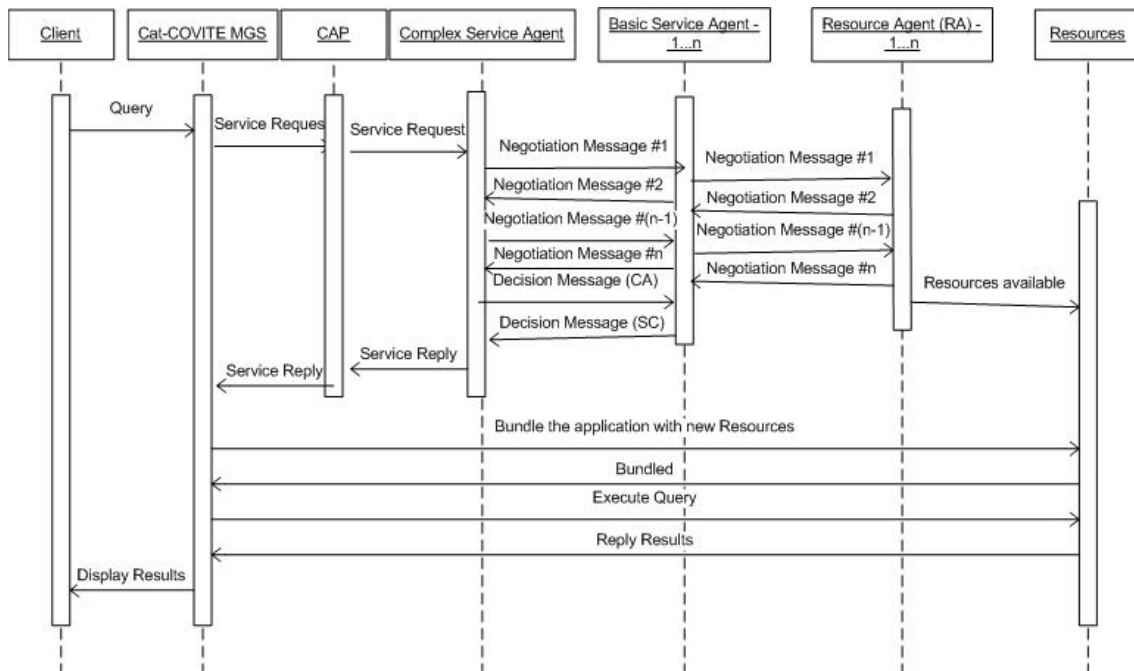


Figure 20 – Use case 1 - UML Sequence diagram of the interaction between Cat-COVITE application and middleware

2. Use case 2

The flow of actions among the logical components is shown in Figure 21, while Figure 22 presents an UML sequence diagram of the interaction between application and middleware. This use case differs than use case 1 as the resources have already deployed services needed by the application.

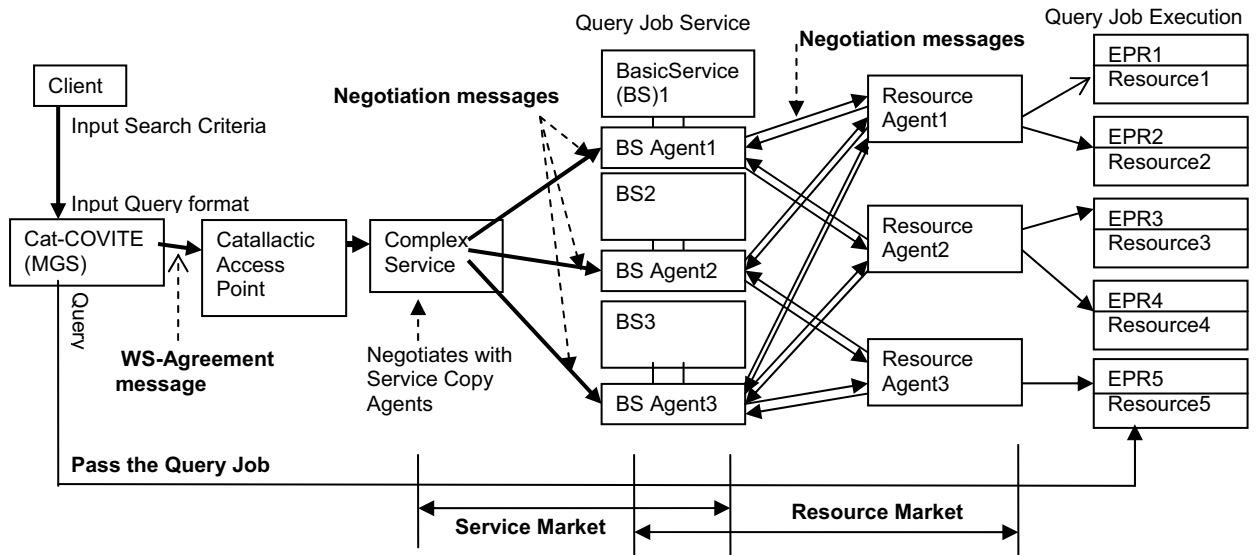


Figure 21 – Flow diagram - use case 2

Glossary:

- *Catallactic Access Point* = access point for the “on-demand” application requests
- *Service Copies* = Offer a Query Job Service
- *Basic Service Agents* = Agents acting on behalf of Basic Services
- *Resource Agents* = Agents acting on behalf of a resource or bundle of resources (identified by its IP addresses), which negotiate with the *Basic Service Agents*
- *EPR* = End point reference is the address of the particular resource

The flow of information is summarized as follows:

1. Client inputs a search criterion.
2. Cat-COVITE (MGS - “left side”) transforms the search criteria in a query format.
3. Cat-COVITE (MGS - “right side”) contacts the Catallactic Access Point, via a XML message, requesting a Query Job Service.
4. On the Service Market – *Complex Service Agent* negotiates with different *Basic Service Agents*, which offer *Query Job Services*.
5. On the Resource Market – *Basic Service Agents* negotiate with *Resource Agents* to buy resources to executing the job.

6. The application pass the query job to the query service instance and distributes the search job on the resources bought from the resource market.

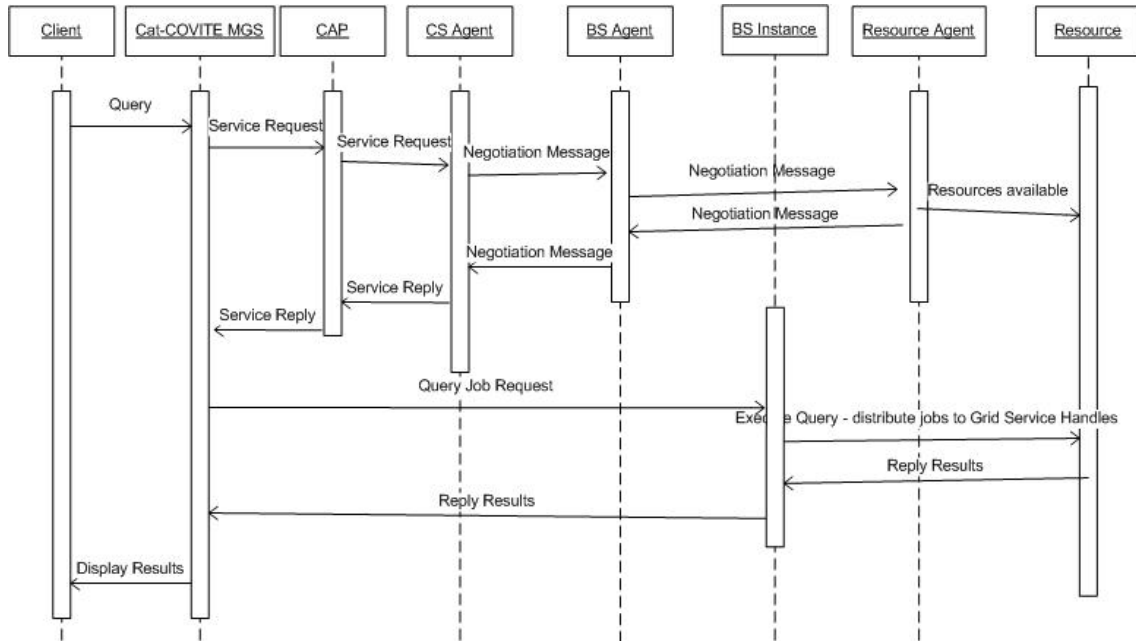


Figure 22 – Use case 2 - UML Sequence diagram of the interaction between Cat-COVITE application and middleware

3. Use case 3

The flow of actions among the logical components is shown in Figure 23. The main difference of this use case in comparison with the other two is that there is use the local site monitor to look up locally for resources availability.

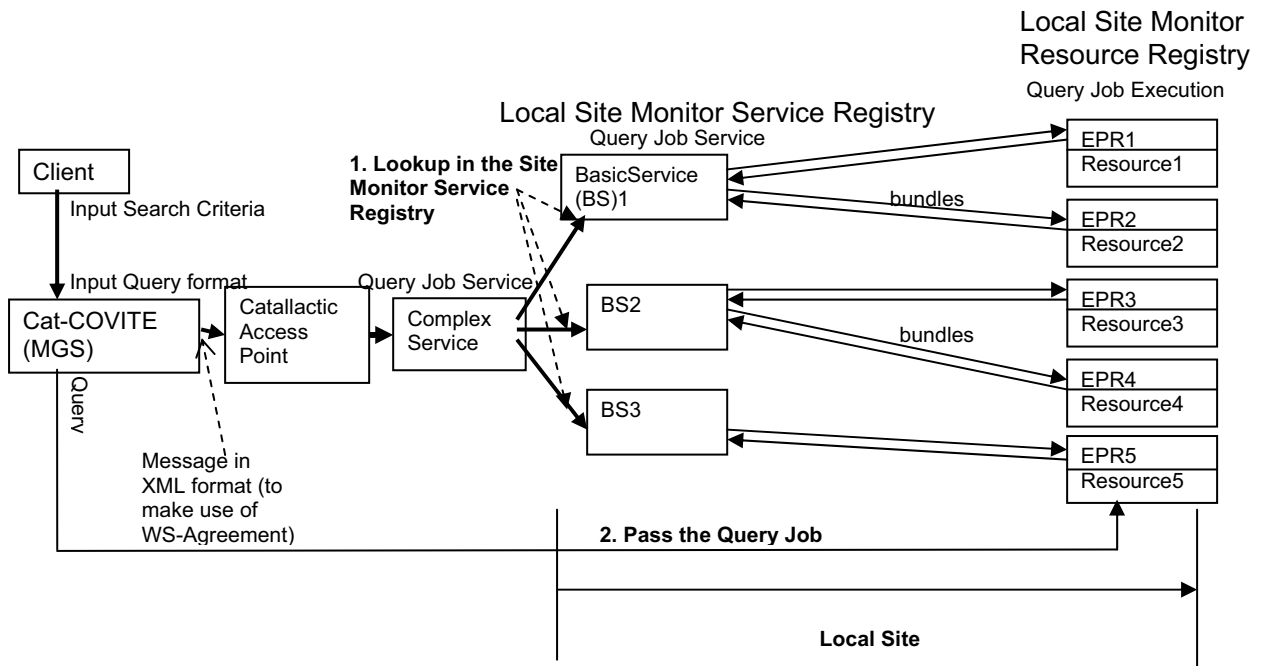


Figure 23 – Flow diagram - use case 3

Glossary:

- *Catallactic Access Point* = access point for the “on-demand” application requests
- *Basic Service* = Offer a Query Job Service
- *Basic Service Agents* = Agents acting on behalf of Basic Services
- *EPR* = End point reference is the address of the particular resource

The flow of information is summarized as follows:

1. Client inputs a search criterion.
2. Cat-COVITE (MGS - “left side”) transforms the search criteria in a query format.
3. Cat-COVITE (MGS – “right side”) contacts the Catallactic Access Point, via a XML message, requesting a Query Job Service.
4. On the Local Site Monitor - Complex Service Agent lookup in the Local Site Monitor Service Repository to choose one of the Basic Services, which offer Query Job Service, based on the parameters requested by the application.
5. Basic Services have already resources or bundle of resources with instances to execute the query job, or they lookup in the Local Site Monitor Resource Repository to choose plain resources or bundle of resources to be used by the application (as in use case 1).
6. The application pass the query job to the query service instance, which offers the query job service, and distributes the search job to the instances on the resources.

At the initial development stage, we consider use case 2. All initial implementations of the prototype follow this pattern.

4.3 Concepts and mechanisms for the application implementation – requirements

At the initial stage of the application prototype development, two models are considered. Figure 7 details a diagram integration of Catallactic middleware and Cat-COVITE Grid Application. The prototype application is based on GT4 services, as the first model, and Java Web Services, as the second model.

The first prototype application is based on the following information flow: the MGS pass the query string in xml format to the QueryService (JavaWS/GT4 Service); this contacts the wrapper .NET Web Service, on the supplier site, and passes the xml string query. On the supplier site, the query runs over the database and returns the results to the QueryService, which pass back the results to the client. The results returned from the .NET Web Service wrapper is formatted in the document literal encoding. GT4 service uses this format too. Java Web Service uses RPC format encoding. The problem of interoperability between GT4/.NET Web Service and Java Web Service is the matter of marshalling / un-marshalling issue.

1. Initial prototype application based on GT4 services

In the case of the first model, using the GT4 services, below are the meanings of the first prototype files:

- Search.java is the Resource. This class actually manages all the actions we are going to take with regard to the Search action (the Query job), such as setting property values.
- SearchHome.java is the ResourceHome. This class is responsible: 1) for creating: a) single instance of the resource (in the case of singleton), or b) multiple instances of the resources (in the case of multiple resources); and 2) for helping locate a specific instance, as requested.
- SearchService is the actual web service that interacts with the resource. This class contains the methods exposed by the web service (in our case the "write" method, which tasks is to accept a string as an input and return a string), when is invoked by the "service client".
- search_port_type.wsdl is the WSDL file that defines what the service can do. It also defines the resource properties document for the resource.
- deploy-jndi-config.wsdd is the file which explains how to link the service with the actual class that implements it.
- deploy-client.wsdd - this file is not actually used, but has to be there.
- NStoPkg.properties - also needed, but actually is empty. This file specifies how to derive the package names from the namespaces if they need to be different from what can be derived from the service's WSDL file.
- build.xml - contains the instruction that guide the entire build process.

Many of these files are standard files that come with the WSRF. Below are examples of some of these files and the meaning in the context of the first prototype application.

a. Create the WSDL file

The WSDL file has an important role as it contains information of how we define what the prototype expects to see and what it will return. The WSDL file defines the message pattern used by the WS-Resource too. In annex E “Application framework – concepts and mechanisms” is presented the detailed WSDL file.

This file creates a basic WS-Resource with a resource properties document called *SearchResourceProperties* and the basic operations required of WS-Resource: *GetResourceProperty*, *Write* and *Create*. *Create* operation is responsible for creating the resources via the service factory, while *Write* operation is responsible for passing the query from the MGS to the .NET Web Service wrapper on the supplier side, as well as keeping the result as a state information.

b. Create the service

The service provides a way to access the capabilities built in to a WS-Resource. In annex E “Application framework – concepts and mechanisms” is presented the file code of the query service.

c. *Create the constants*

One goal of the prototype application is to have a code that it is easily translatable. There is an easy way by creating this object which keeps track of all constants defined within the prototype.

```
package org.catnets.tutorial.search;
import javax.xml.namespace.QName;

public interface SearchConstants
{
    static final String SEARCH_NS= "http://tutorial.catnets.org/search";
    static final QName RP_SET = new QName(SEARCH_NS, "SearchResourceProperties");
    static final QName MESSAGE_RP = new QName(SEARCH_NS, "StatusMessage");
    static final QName SEARCH_RP = new QName(SEARCH_NS, "CurrentSearch");
}
```

All these above examples are only a part of the first Cat-COVITE application prototype implementation and the interaction with the Catallactic middleware. This first prototype presents a base of further integration and development of a Catallactic enabled Cat-COVITE application prototype.

2. Initial prototype application based on the Java Web Services

The architecture and functionalities (see Figure 7) of this model are similar to the other presented above using the GT4 services. The interoperability between a Java Web Service and a .NET Web Service is solve using marshalling and un-marshalling concepts, as Java Web Service uses RPC format encoding, while .NET Web Service uses the document literal format.

4.4 WS-Agreement - concepts and requirements

An agreement consists of several parts. According with the WS-Agreement draft [WS-Ag05], these are:

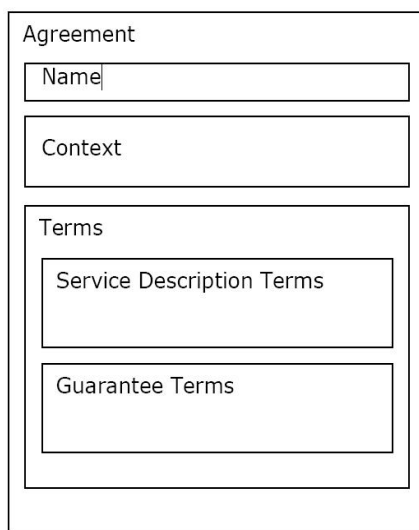


Figure 24 - Structure of an agreement [WS-Ag05]

The XML representation of an agreement or an agreement creation offer has the following structure:

```
<wsag:Agreement>
  <wsag:Name>
    xs:NCName
  </wsag:Name> ?
  <wsag:AgreementContext>
    wsag:AgreementContextType
  </wsag:AgreementContext>
  <wsag:Terms>
    wsag:TermCompositorType
  </wsag:Terms>
</wsag:Agreement>
```

The section of the agreement name is optional. The agreement context includes the parties to an agreement, reference to the service(s) provided in support of the agreement, the lifetime of the agreement. The next section contains the agreement terms, which describe the agreement itself, and could contain:

- The *service description terms*, which provide information needed to instantiate or otherwise identify a service to which this agreement pertains
- The *guarantee terms*, which specify the service levels that the parties are agreeing to.

The example scenario in terms of the Cat-COVITE application proposed for the CATNETS prototype interacting with the middleware it is as: “I (MGS) need to run a query search job. I (MGS) send an Agreement Offer (AO1), based on the agreement template downloaded from the *Catallactic Access Point*, to the *Catallactic Access Point* to execute a query job. I (MGS) need, for example, *NoOfQueriesInPeriodOfTime*, *ResponseTimePerQuery*, *PeriodOfUsage*, and *PayForService*. The *Complex Service Agent*, acting on behalf of the *Complex Service* chosen by the *Catallactic Access Point*, negotiates with the *Basic Service Agents* (in the CATNETS environment) for query services to fulfil the job”.

MasterGridService (MGS) receives the client request for executing a query over a supplier database (see Figure 21 – Flow diagram – use case 2). The client’s input search criterion is transformed by the application (via the *MGS*) in an input query in the following format:

```
SELECT IDProduct, ManufacturerName, ProductName, Price FROM Product ORDER
BY Price DESC
```

The *MGS* creates an xml string format as follows:

```
<?xml version="1.0"?><searchCriteria> SELECT IDProduct, ManufacturerName,
ProductName, Price FROM Product ORDER BY Price DESC</searchCriteria>
```

This is the xml string message that *MGS* is passing to the *SearchServiceInstance* in order to fulfil the client requirements (searching for products on the supplier databases).

The *MGS* contacts the *Catallactic Access Point* in order to finding resource to execute the search. The communication between the *MGS* and *Catallactic Access Point* is WS-Agreement based.

Technical Specifications of the request from the *MGS* to the *Catallactic Access Point*:

1. MGS looks for a service provider which offers query service (services with the possibilities of running queries).
2. (WHAT TO EXECUTE) - to run a query job.

The query is of this format:

```
SELECT IDProduct, ManufacturerName, ProductName, Price FROM Product ORDER BY Price DESC
```

The query job message passed by the MGS will be of an xml format as follows:

```
<?xml version="1.0"?><searchCriteria> SELECT IDProduct, ManufacturerName, ProductName, Price FROM Product ORDER BY Price DESC</searchCriteria>
```

3. (WHERE TO EXECUTE) the supplier(s) database address(es) (URLs). (This will be part of the next stage of the prototype development). This will be an xml string containing the URLs of the supplier database to search for according with the input search criterion. The search will be distributed only to suppliers who supply the products that the client is interested in.

The *Catallactic Access Point* is an agreement provider, while the *MGS* “right side” is an agreement initiator. The *MGS* initiates an agreement offer by retrieving the agreement template, filling it in and sending it back to the *Catallactic Access Point*. The *Catallactic Access Point*, as the agreement provider, could accept or deny. If the agreement is denied than the *MGS* “right side” has to submit a new agreement offer that will be analysed by the *Catallactic Access Point*.

4. Provides the URL of the input file (the query in xml string format)
5. Provides the URL of the output file (where the result of the search - in xml string format- should be returned)

Quality of Service parameters

Example of what the *MGS* is interested in and quality of service parameters that a service provider has to offer in order to fulfil the application tasks:

1. Running minimum “n” number of queries in a period of time.

Example: not less than n = 100 queries / minute

2. Response time not more than “m” seconds per query.

Example: not more than m = 3 seconds / query response time.

3. Period of time “t” of usage of the service.

MGS needs a dedicated service provider for a “t” period of time. Example: t = 8 hours.

4. Starting time of the use of service. Example: date:hour:minute:second
5. End time of the use of service. Example: date:hour:minute:second
6. The amount of money “a” MGS is willing to pay for the dedicated service provided

Example: a = 30 £ / 8 hours

Agreement Template

The agreement template specifies the service description elements that are allowed by the factory which advertises it. In the annex E “Application framework – concepts and mechanisms”, point E.3, are presented: an example format of an agreement template and an agreement template lite.

Agreement Offer

In the annex E “Application framework – concepts and mechanisms”, point E.4, are presented an example of an agreement offer that is compliant with the agreement template presented in annex E.3, and an agreement offer lite. This agreement offer is initiated by the agreement initiator (the *MGS*).

There are guarantees which express constraints on the time by which *QueryJob* service must start and must finish. The metric is refers to is `job:startTime` and `job:endTime` which are the type of `xsd:dateTime`.

Agreement

The agreement acceptance is the same as the agreement offer if the agreement provider accepts the conditions of the offer. If the agreement provider doesn't accept the offer, the agreement initiator has to send another agreement offer.

5 Conclusion and further work

5.1 Impact of the Catallactic middleware in the application.

The first implementation of a real system, as COVITE prototype, with a Catallactic mechanism, has raised interesting conclusions outlined below:

- COVITE prototype has been designed and developed as a centralised system, where all services and resources are under one authority control. All VOs created via the central system use the resources of the authority. A decentralised solution of the COVITE prototype will create great benefits to such system, as the demand of services and resources from VOs will be better solved via the CATNETS markets. The Cat-COVITE prototype will follow these patterns.
- The CATNETS markets expect to provide VOs the possibility to gain profits out of their spare resources and services.
- The Catallactic mechanism expects to help systems in discovering and selecting resources and services on demand and just in time, as application processes can make use of third parties services or can demand more and more resources.

5.2 Conclusions on the adaptation of the middleware to an existent application

From this first attempt to integrate the Catallactic middleware to an existent application (in this case a grid application), we can extract interesting conclusions: First and foremost we confirm that the proposed architecture captures all the required aspects from a grid scenario, providing for a natural integration into an existing grid application. These are the achievements we have identified so far:

- We have succeeded plugging a service/resource discovery and selection mechanism on a pre-existent grid application. Thus we convert a “classical” centralized, monolithic data-grid engine into a decentralized, more flexible Catallaxy-enabled grid application, based on a shared infrastructure (the middleware).
- We fulfil the requirements described in above sections using standards and standard tooling. In this sense, undoubtedly WS-Agreement and GT4/WSRF help lowering modelling and implementation/integration costs.
- In particular the connection application-middleware using the WS-Agreement has been quite straightforward and did not enforce any architectural change on the middleware side.

However other issues have been detected as the most problematic, namely the shared components between application and middleware, and the connection middleware-base platform:

- There is a need to better abstract the “service component” on the middleware side in order to alleviate conflicting requirements with application. To that point we believe that the correct way to proceed is using the WSRF notion of service instance, which is created following a factory pattern. This factory will be provided by the middleware as a point of policy enforcement, accounting and monitoring. There are still some open questions about the management of the lifecycle of this service/resource instances, as the application might not be designed to handle them.
- During the integration efforts it has been identified some conflicting roles assumptions, since application designers supposed that catalactic middleware would be taking care to some extent of service execution. It is important to stress the exclusive role of the middleware as a service/resource discovery and selection tool. From the amenability point of view (broader application of the middleware to other grid, p2p and CDN applications) it is required a very low coupling with the application. The middleware should never be providing application specific services as this would decrease modularity and complicate any further usage in different scenarios.
- The connection with the base platform is still to be properly designed. To that point the “resource management” component in the P2P Agent Layer on the architecture must be further developed. We envisage resource virtualization in the WSRF context as the main driver for a proper connection middleware-base platform (on the broad scenario of grid applications).

5.3 Roadmap to first CATNETS prototype - open issues and future work

An important part of the work on integrating Cat-COVITE application and catalactic middleware has been accomplished. Several remaining issues will be targeted in the following months, following this roadmap:

- Implement the lifecycle management for service instances, using the factory model and WSRF mechanism (e.g. WS-Notificaton)
- Develop a model for computational resources virtualization (e.g. CPU, disk) using WSRF
- Complete integration and functioning tests.
- Develop measurement components on the middleware
- Define metrics relevant both for middleware and application
- Design experiments and test prototype, extracting and analysing results
- Start next iteration on the prototype evolution, leading to a more complex, realistic grid scenario
- Further development of the Cat-COVITE application prototype by searching multiple supplier databases

6 References

- [Bers96] P. Berstain (1996), “Middleware: A Model for Distributed System Services”, Communications of the ACM, 39(2):86-98
- [BuVe04] R. Buyya, S. Venugopal (2004), “The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report, Technical Report, GRIDS-TR-2004-2”, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, April 2004.
- [Catn03] CATNET project (2003), “Catallaxy Simulation Study, Report No. D2” http://research.ac.upc.es/catnet/pubs/D2_Simulation_Study.pdf
- [Catn04] CATNETS project (2004), “Annex I - Description of Work”, IST-FP6-003769
- [COV04] COVITE project (2004). <http://www.wesc.ac.uk/projectsite/covite/index.html>
- [D1WP1.1-05] Environmental Analysis for Application Layer Networks (2005), Deliverable D1, WP1.1, CATNETS project
- [Emme00] W. Emmerich (2000), “Software engineering and middleware: a roadmap”. In Proceedings of the conference on The Future of Software Engineering (ICSE 2000)
- [ERA+03] T. Eymann, M. Reinicke, O. Ardaiz, P. Artigas, F. Freitag, L. Navarro (2003), “Self-organizing resource allocation for autonomic network”, Proceedings. 14th International Workshop on Database and Expert Systems Applications, Germany, 656- 660
- [FAvH89] F.A.Hayek, W. Bartley, P. Klein, B. Cadwell – “The collected works of F.A. Hayek”, University of Chicago Press, 1989
- [FKL+99] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy “A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation”, Intl Workshop on Quality of Service, 1999.
- [GGF05] Global Grid Forum (2005), <http://www.ggf.org/>
- [Glob05] <http://www.globus.org/>
- [GRAAP05] Grid Resource Allocation and Agreement Protocol Working Group (2005), <https://forge.gridforum.org/projects/graap-wg>
- [GSI04] Grid Security Infrastructure (2004), <http://www-unix.globus.org/security/>
- [HWBM02] C. Hoile, F. Wang, E. Bonsma , P. Marrow (2002), “Core specification and experiments in DIET: a decentralised ecosystem-inspired mobile agent system “, Proceedings of the first international joint conference on Autonomous agents and multiagent systems”, ACM Publishing, 623 – 630

- [IaFo01] A. Iamnitchi, I. T. Foster (2001), "On Fully Decentralized Resource Discovery in Grid Environments", Proceedings of the Second International Workshop on Grid Computing, p.51-62
- [J2se05] <http://java.sun.com/>
- [Jade05] <http://jade.tilab.com/>
- [JPB04] Liviu Joita, Jaspreet Singh Pahwa, Pete Burnap, Alex Gray, Omer Rana, John Miles - "Supporting Collaborative Virtual Organisations in the Construction Industry via the Grid", Proceedings of the UK e-Science All Hands Meeting 2004, Nottingham, United Kingdom, 31st August - 3rd September 2004, © EPSRC Sept 2004, ISBN 1-904425-21-6,
<http://www.allhands.org.uk/2004/proceedings/papers/182.pdf>
- [Jxta05] <http://www.jxta.org/>
- [LHF04] K. Lai, B. A. Huberman, and L. Fine (2004), "Tycoon: A Distributed Market-based Resource Allocation System," HP Lab, Palo Alto, Technical Report cs.DC/0404013, Apr. 2004.
- [LiWu05] X. Li and J. Wu, "Searching Techniques in Peer-to-Peer Networks," accepted to appear in Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks, J. Wu (ed.), CRC Press, 2005
- [MKL+01] P. Marrow, M. Koubarakis, R.H. van Lengen, F. Valverde-Albacete, E. Bonsma, J. Cid-Suerio, A.R. Figueiras-Vidal, A. Gallardo-Antolin, C. Hoile, T. Koutris, H. Molina-Bulla, A. Navia-Vazquez, P. Raftopoulou, N. Skarmas, C. Tryfonopoulos, F. Wang, C. Xiruhaki (2001), "Agents in Decentralised Information Ecosystems: The DIET Approach". Symposium on Information Agents for E-Commerce, AISB'01 Convention, 21st - 24th March 2001 University of York, United Kingdom
- [MKL+02] D. Milojevic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, Z. Xu (2002), "Peer-to-Peer Computing", HP Laboratories Palo Alto, Research report PL-2002-57
- [Mmap04a] MMAPPS Project (2004), "Deliverable 5: Peer-to-Peer Services Architecture", September, 2004
- [Mmap04b] MMAPPS Project(2004), "Deliverable 27- Final Project Summary Report", November 2004
- [OGSA03] Open Grid Service Architecture (OGSA) (2003)
<http://www.globus.org/toolkit/docs/3.0/physiology.pdf>
- [P2P02a] P2P Architect Project (2002), "Deliverable D1 - Comprehensive Survey of contemporary P2P technology", IST-2001-32708
- [P2P02b] P2P Architecture Project (2002), "Deliverable D4 – P2P applications development process overview", IST-2001-32708
- [Pepi03a] Pepito Project (2003a), "Deliverable D1.1 –Required foundations for Peer-to-Peer systems", IST-2001-33234

[PHP+03] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M. P Frank and C. Chokkareddy (2003), "OCEAN: The Open Computation Exchange and Arbitration Network, A Market Approach to Meta computing", In proceedings of the International Symposium on Parallel and Distributed Computing (ISPDC'03), Oct 2003

[SOAP03] Simple Object Access Protocol (SOAP) Version 1.2, W3C Recommendation 24 June 2003, World Wide Web Consortium.
<http://www.w3.org/TR/2003/REC-soap12-part1-20030624/#intro>

[SSPS05] Sujoy Basu, Sujata Banerjee, Puneet Sharma, Sung-Ju Lee, NodeWiz (2005), " Peer-to-peer Resource Discovery for Grids", In the 5th IEEE International Symposium on Cluster Computing and the Grid Cardiff, Wales, UK 9th – 12th May 2005

[TAP03] Traversat, M. Abdelaziz, and E. Pouyoul, Project JXTA: Loosely-Consistent DHT Rendezvous Walker, Sun Microsystems, Inc.,
<http://www.jxta.org/project/www/docs/jxtadh>

[VBW05] S. Venugopal, R. Buyya, L. Winton (2005), "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids", Journal of Concurrency and Computation: Practice and Experience, Wiley Press, USA

[WS-Ag05] Web Services Agreement Specification (WS-Agreement), 28 June 2005
https://forge.gridforum.org/docman2/ViewCategory.php?group_id=71&category_id=659

[ZaPa04] F. Zambonelli, V. Parunak (2004), "Towards a Paradigm Change in Computer Science and Software Engineering: a Synthesis", The Knowledge Engineering Review, 2004

Annex A – The Architecture Development Process

“As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem”. [GaS93]

To address this complexity, a different approach for developing is needed. Architecture based development focuses on reasoning about the structural issues of a system. “Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives” [GaS93]

The importance of the architecture goes far beyond the simple documentation of technical elements. According to [BBC+00], the architecture serves as “the blueprint for both the system and the project developing it” and therefore it helps in the definition of how the work can be organized. Also, the authors state that the architecture “is the carrier of system qualities such as performance, modifiability, and security, none of which can be achieved without a unifying architectural vision”. Finally, it is “a vehicle for early analysis to make sure that the design approach will yield an acceptable system”.

Considering the importance of the architecture and its impact in the overall system and project organization, the main objectives of the architecture design process in CATNETS project can be summarized as:

- Define a set of common design concepts that bring coherence to the architecture
- Define a set of sound design and implementation principles that assure the quality of the resulting middleware
- Identify key architectural requirements that allow the evaluation of potential implementation options, including the adoption of already existing middleware toolkits and development platforms
- Separate design concerns to facilitate the division of work in the different work packages, giving each group the freedom to experiment with implementation options, but avoiding the risk of incompatibilities
- Structure the system in a way that allows future experimentation in specific areas like negotiation protocols and basic middleware mechanisms and policies (peer location, resource replication, etcetera) with minimal impact on the rest of the system

A.1 Architecture Design Process

The architecture design process goes from the system’s requirements to the architecture specifications. We have adapted the methodological approaches proposed in [HNS99] and [KaBa99] to define an architecture design process that considers three steps, as shown in figure A.1.

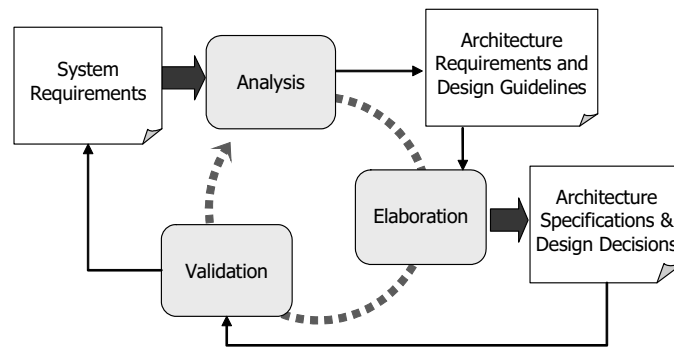


Figure A.1. The Architecture Design Process

The **Analysis** process translates the various systems requirements to architecture requirements. It consists of three main tasks:

- Review the functional requirements (described in usage scenarios) that the architecture must address, along with the nonfunctional requirements (performance, scalability, security) that it must meet.
- Identify the architectural requirements that will constrain the design options
- Define strategies to deal with identified factors and state design guidelines that will guide the elaboration phase.

The **Elaboration** phase consist in the construction of a set of specifications of the architecture, covering different levels of detail and from diverse perspective, so that the complexity of the system can be properly reflected to different stakeholders (e.g. project managers and developers). One important part of the design process is the documentation of the design decisions that lead to this architecture, so that future reviewers (architects, developers) can understand the rationale behind those decisions.

Finally, the **Validation** phase consists in the exploration of different usage scenarios to verify the compliance of the architecture with the requirements [KABC96].

This process is iterative and it is expected to continue even when the detailed design of the system is well advanced [KaBa99].

A.2 Architecture Specification

We use a model composed of multiple views or perspectives to describe a software architecture ([Kruc95]), see Table A.1. Each model covers a set of relevant aspects of the specification from one stake holder's point of view (for example, project manager or developers) during a stage in the development process (for example, design or implementation). The following table resumes the different architectural views we consider.

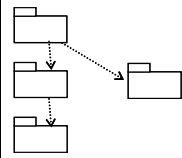
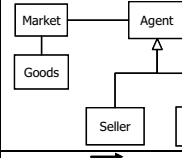
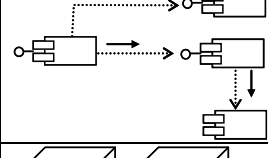
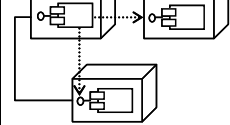
Development	Specifies how code is organized Defined in terms of software packages and their dependencies Helps to manage the development process	
Logical	Specifies what the system does Defined in terms of logical entities (classes) and their relationships Helps to understand the problem and the proposed solution	
Dynamic	Specifies how functions are distributed at run time Defined in terms of executable components (processes, threads, agents) and their interactions Helps to understand how the system behaves	
Deployment	Specifies how software is deployed Defined in terms of physical components (modules and nodes) and their references Helps to manage the system's operation	

Table A.1. Architectural Views

As the initial phase of middleware development focuses on the evaluation of the implementation alternatives, we have delimited the architecture description to the identification of the overall organization (*Development View*) and the identification of the main functionalities it must cover (a partial *Logical View*). As we proceed with the project, the remaining views shall also be covered.

A.3 References

[BBC+00] F. Bachmann, L. Bass, J. Carriere, P. Clements, D. Garlan, J. Ivers, R. Nord, R. Little (2000), "Software Architecture Documentation in Practice: Documenting Architectural Layers", Technical Report, Software Engineering Institute, Carnegie Mellon University

[GaS93] D. Garlan, M. Shaw (1993), Also published as "An Introduction to Software Architecture," *Advances in Software Engineering and Knowledge Engineering, Volume I*, edited by V.Ambriola and G.Tortora, World Scientific Publishing Company, New Jersey

[HNS99] C. Hofmeister, R. Nord, D. Soni (1999), "Applied Software Architecture", 397 pages, Addison-Wesley Professional 1st edition

[KaBa99] R. Kazman, L. Bass, (1999), "Architecture Based Development", Software Engineering Institute, Carnegie Mellon University, Technical report CMU/SEI-99-TR-007

[KABC96] R. Kazman, G. Abowd, L. Bass, P. Clements (1996), "Scenario-Based Analysis of Software Architecture", *IEEE Software*, 13 (6):47 - 55

[Kru95] P. Kruchten (1995), "Architectural Blueprints—The '4+1' View Model of Software Architecture", *IEEE Software* 12 (6), 42-5

Annex B – Peer-to-Peer Architectures

In order to support the application within a Catallactic market, it is useful to review existing architectures that impact on the development of this application. Currently, many distributed technologies are using a network-based computing style that support mixed models of the network topologies. The main debate about distributed systems topologies is focused on the centralized or decentralised systems. Our focus in the application is on the creation of a decentralised system. Generally, there are a number of reasons to support such a model:

- there is a need to scale the system to support increased demand of resources and services;
- there is generally a need to move resources and services closer to their access point;
- to support better fault tolerance and network resilience;
- to enable better sharing of resources and facilitate collaboration.

P2P systems are highly decentralized and distributed, the peers being highly autonomous, independent and self-organised. In a client-server model, the server controls and manages the relationship a client has with resources, services and other clients. The topology within a distributed system can be considered at different levels: physical, logical, or organizational. Often a centralised system is preferred, as the advantages of a centralised system are ease of use in the operation, administration and management of a distributed computing environment. Resources and services can be administered from a central server, while the functionality can be deployed to complement the physical structure of the network topology. The decentralised system, however, provides challenges that have to be addressed: the management of the network is more difficult, failures are not always detected immediately, response time and latency issues are difficult to predict in a remote communication. A P2P distributed system should be able to address these decentralized system issues and try to eliminate or mitigate them.

Peers in a P2P system are entities used to provide services or to find other peers that can provide the services requested. A peer can be considered a client when is requesting a service, and can be considered a server when it is providing a service. The predominant architecture of a P2P system is a decentralized model augmented with centralised control nodes at key points, which are important performance characteristics, such as discovery and content management.

Recently, Grid computing provides an approach for building large-scale and cost-effective distributed computing infrastructures. Grid computing generally involves the integration of geographically distributed resources like compute engines, storage, and data, and enables the virtualization of these distributed computing and data resources to create a single system image, granting users and applications seamless access to vast IT capabilities.

Often, the debate between centralised and decentralised systems is fundamentally about topology, i.e. how the nodes in the system are connected. Four common topologies can be found in distributed systems:

- Star
- Ring
- Hierarchical
- Mesh

In addition there are hybrid topologies that may also be considered. The most interesting topologies for a P2P/Grid-enabled application are:

- Centralised + Ring
- Centralised + Decentralised

Minar [Minar02] considers seven properties for evaluating topologies:

- Manageability – evaluating the difficulties of keeping a system working. Updating and logging are factors to be considered.
- Information coherence: how correct is information within the system? Non-repudiation and consistency are particular important aspects.
- Extensibility: reaction of the system when grows, especially when more resources are added.
- Fault Tolerance: reaction of the system when fails. This is a very important criterion in large distributed systems.
- Security: how hard is it to threaten the system?
- Resistance to lawsuits: how hard is it for an authority to shut down the system? This property is not important to be considered in the CATNETS project.
- Scalability: how large can the system grow? Scalability is a key advantage of decentralized systems over centralized, although the reality is more complex.

The list above provides some of the main properties which are often considered in the preliminary architecture and design, and in decision-making criteria of the application topology. These criteria are also very useful in the evaluation of advantages or disadvantages of the decentralized systems (or P2P systems) architecture.

B.1 Star topology

Star topology (Figure B.1), known also as a *client-server* or *centralised* topology, involves connecting each device or node to a central point of control (the server). In this case, the server makes services available over a network to all clients.

The main advantage of centralized systems is their simplicity. As all data is concentrated in one place, centralized systems are easily managed and have no questions of data consistency or coherence. Centralized systems are also relatively easy to secure: there is only one host (the central server) that needs to be protected. The drawback of centralization is that everything is in only one place. If the central server goes down, everything does. There is no fault tolerance. Centralized systems are also often hard to extend, resources can only be added to the central system. The scalability of centralized systems is subtle. Scale is limited by the capacity of the server, and so centralized systems are often thought of as not scalable.

Nowadays computers are very fast and a single computer can often support all the demands of its users. Unlike more complex topologies, the scalability of a centralized system is very easy to measure. When there is a need of complex processes from many clients to be handled by the server, then a central system is not fast enough to run all these clients' requests. [Minar02]

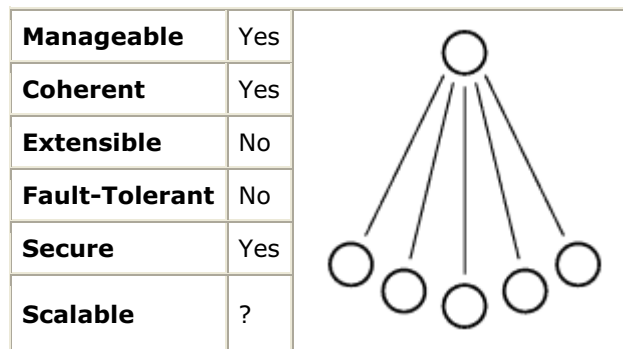


Figure B.1 – Star topology [Minar02]

B.2 Ring topology

In a ring topology (Figure B.2), such as a Token Ring, each node is connected to two other nodes, forming a loop. This topology is based on the concept of passing a single token around to the computers connected to the ring. Ring systems typically have a single owner. This concentration gives them many of the same advantages of centralized systems: they are manageable, coherent, and relatively secure from tampering.

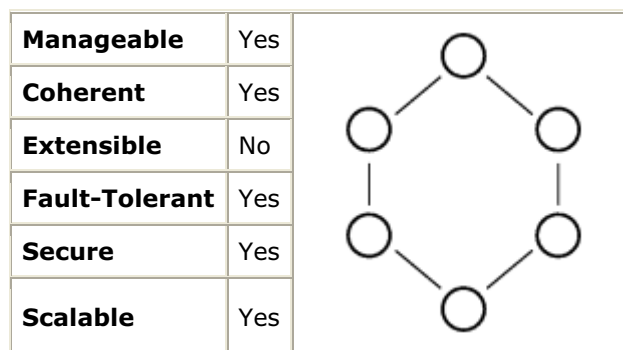


Figure B.2 – Ring topology [Minar02]

The added complexity of the ring is mitigated by fairly simple rules for propagating state between the nodes in a ring. But the single-owner restriction means rings are also not extensible: a user still needs the owner's permission to add a resource into the ring.

The advantages of rings over centralized systems are fault tolerance and scalability. If a host goes down in a ring, failover logic makes it a simple matter to have another host cover the problem. Well-designed rings are scalable; one can simply add more hosts to the ring and expand the capacity. The drawback of the ring topology is that it isn't used much anymore for common networks.

B.3 Hierarchical topology

One of the most common topology is the *hierarchy* (Figure B.3), which provides some advantages in comparison with the rings. Hierarchical systems are somewhat manageable in that they have a clear chain of action. But because these systems have such a broad scope, it can be hard to correct a host with a problem. Coherence is usually achieved with a cache consistency strategy; effective, but not complete. Hierarchical systems are extensible in that any host in the system can add data, but the rules of data management may limit what information can be added. Hierarchical systems are also more fault-tolerant than centralized systems, but the root is still a single point of failure. They tend to be harder to secure than centralized systems. If a node high in the hierarchy is subverted or spoofed, the whole system suffers. And it is not just the root that is a risk: if data travels up the branches to the root, then leaf nodes may be able to inject bad information to the system. The primary advantage of hierarchical systems is their scalability. New nodes can be added at any level to cover for additional workload. The relative simplicity and openness of hierarchical systems, in addition to their scalability, make them a desirable option for large distributed Internet systems.

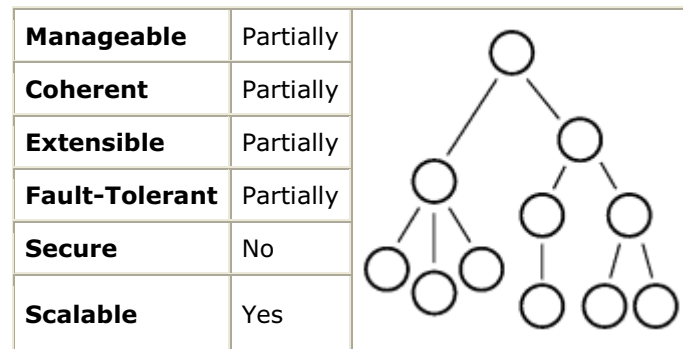


Figure B.3 – Hierarchical topology [Minas02]

B.4 Mesh topology

A mesh topology (Figure B.4), also known as a *decentralised topology*, requires all the nodes to have dedicated paths to all other nodes on that network. A mesh network topology also resembles the Internet routing topology. Decentralized systems come closest to being truly P2P. Examples of decentralised systems are: Gnutella and KaZaA. The characteristics of such systems are opposite to the centralized systems. These systems have a tendency to being difficult to manage and data in the system is never fully authoritative. The security is also an issue as it is very easy for a node to join the network and start putting bad data into the system. A main advantage of decentralized

systems is their extensibility. Decentralized systems also tend to be fault-tolerant. The scalability of decentralized systems is hard to evaluate. In theory, the greater the number of hosts, the greater the capability offered by the system. In practice, the algorithms required to keep a decentralized system coherent often carry a lot of overhead. If that overhead grows with the size of the system, then the system may not scale well. Scalability of decentralized systems remains an active research topic.

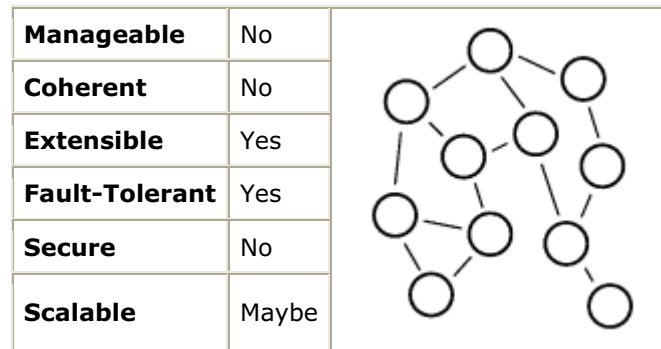


Figure B.4 – Mesh topology [Minar02]

B.5 Centralised + Ring

The hybrid system takes advantages of using a ring topology for its central server, with the benefit of simplicity of centralisation and redundancy of a ring. Such systems are easy to managed and secure. The main advantage of using a ring is that server adds fault-tolerance and scalability.

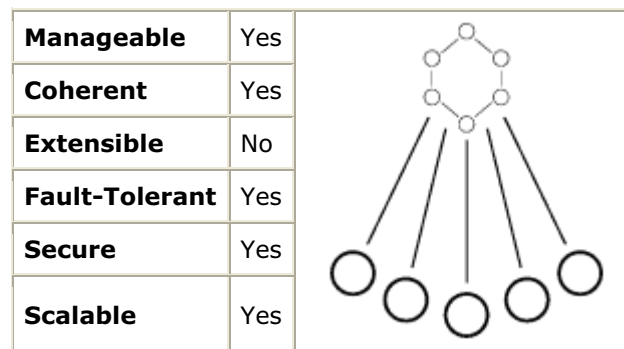


Figure B.5 – Centralised + Ring topology [Minar02]

B.6 Centralised + Decentralised

A system, as shown in Figure B.6, can involve components that are centralized and others that are not. Decentralization plays an important role to the extensibility and fault-tolerance of the system. The partial centralization makes the system more coherent than a purely decentralized system, as there are relatively fewer hosts that are holding authoritative data. Manageability is as difficult as a decentralized system, and the system is no more secure than any other decentralized system.

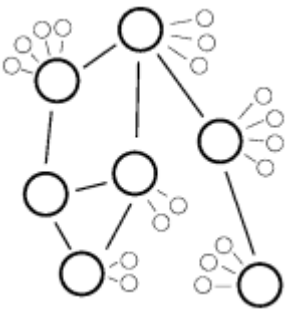
Manageable	No	
Coherent	Partially	
Extensible	Yes	
Fault-Tolerant	Yes	
Secure	Partially	
Scalable	Apparently	

Figure B.6 - Centralised + Decentralised topology [Minar02]

Nowadays the security within such a system may be enhanced by the use of X509-based digital certificates, as used in the Grid Security Infrastructure (GSI), or through the description of security credentials via the Security Assertion Markup Language (SAML), for instance. The most important advantage is the scalability of this hybrid.

Most peers have a centralized relationship to a “super node”, forwarding all file queries to this server. But instead of super nodes being standalone servers, they band themselves together in a *Gnutella-like* decentralized network, propagating queries. Internet email also shows this kind of hybrid topology. Mail clients have a centralized relationship with a specific mail server, but mail servers themselves share email in a decentralized fashion.

We make use of a decentralized application architecture in this project, and one which has been adapted from a previous centralized version.

B.7 References

[Minar02] Nelson Minar, “Distributed Systems Topologies: Part 2”, 08.01.2002
http://www.openp2p.com/pub/a/p2p/2002/01/08/p2p_topologies_pt2.html

Annex C – Specifications of P2P Agent layer

The P2P Agent Layer encompasses the basic functionalities that supports all the Catallactic middleware, providing the basic mechanism that will allow the system to self-organize according to the policies implemented in the upper layers ([EAB+99], [BJM04]). Therefore, this layer has the responsibility to address the critical requirements of interoperability, flexibility and scalability required by the project. In this section we will offer a detailed explanation of this layer, its functionalities and the requirements it should fulfil.

The P2P Agent layer also provides a rich execution interface to speed the implementation of the Catallactic agents, providing a set of common functions and complementing the base middleware when necessary. This layer also isolates the rest of the Catallactic middleware from the particularities of the underlying base middleware, promoting more portable components in the upper layers.

The P2P Agent Layer is built based on the basic abstraction of a set of agents, each of them implementing a basic function within the system, and interacts using a logical topology. It is important to notice that there is no one-to-one correspondence between the trading agents in the Catallactic Algorithms layer (for example, Clients, Service copies and Resources) and the agents in the P2P Agent Layer. Actually, we expect that for each trading agent there will be several agents supporting them, carrying with low level tasks like optimizing the logical topology, handling failures, and many others.

One additional consideration regarding this layer is the need for a great deal of flexibility to allow the experimentation with diverse mechanisms like discovery and agent migration, to explore the adaptation of the Catallactic middleware to different ALNs. Therefore, the architecture should support pluggable component architecture (BCG04], [BJM04])

C.1 Functional Blocks

This layer encompasses the following main functional blocks:

- **Execution Platform:** will provide hosting for the efficient execution of agents. It should permit coexistence of a number of agents on an execution node and facilities the creation, monitoring, scheduling, and management of agents. Besides, the implementation of agents will be much simplified if such functionality is provided by the platform. In scenarios when node failures are possible, good failure management features are essential and the persistence of agent's state or even the mobility of agents to other nodes, could be required to increase its availability ([HoB02], [PNC02])
- **Resource Management:** offers a generic interface to base platform's local resource management to permit its allocation and de-allocation of resources to requesting agents. In some scenarios, it could be necessary to handle efficiently the assignment of different types of resources in a single location (co-allocation) and the reservation of resources for future usage. Finally, some resource monitoring mechanism is required to control the state of resource allocations for management, auditing, etc. ([ADG+04], [PBC03])

- **Communication:** abstracts the basic communication mechanisms and isolates agents from the complexities of the communication protocols. Since network topology will be very dynamic and agent location could vary frequently, a logical addressing to distinguish communications among different agents regardless of its location is required. In general, we are considering a dynamic scenario where communication and node failures are not just possible, but very likely; therefore communications should be assumed as unreliable and delivery guarantee is not a requirement. Also, a robust failure management is required ([CJK+03])
- **Overlay Network:** manages the logical communication topology to efficiently communicate cooperating agents regardless of their physical location. The project is considering very large scale and highly dynamic scenarios where logical communication topology can not be maintained in a single server, a hierarchy of servers, and direct discovery of nodes is not feasible. It is required a distributed mechanism that provides overlay network construction and maintenance. P2P topology construction mechanisms could be optimized for fast location or fast information dissemination. Finally, in such a huge network of agents, the possibility of grouping them could be appealing to make communication more efficient and management easier. ([GCB+04], [HCW04], [DZD+03])
- **Object Discovery:** offers mechanism for the location of objects (agents and resources). Catalytic middleware will be used in very large scale and dynamic scenarios where resources, services and the agents which represent them can not be maintained in a table. Therefore it required mechanisms to discover resource and the agents which manage them. Discovery can be performed either by resource advertising or by resource query and matchmaking mechanisms. Besides, information changes can be published and subscribed to. In order to diminish communication cost, some information cache management system could be used. Finally, it might be required mechanism that permit complex queries (multiple resources, multiple attributes, partial matches, range matches), independent of the semantic of the resource description. ([TsRo03], [LCC+02], [BHPW04])
- **Security and Trust:** We consider an open system where communication attacks are possible, and agents are autonomous agents which could be malicious. Therefore, mechanisms for agent authentication, agent access authorization (e.g. trade on a given market), encryption of agent-to-agent communications, non-repudiation of settled agreements, are required. Agent reputation mechanisms could also be considered, since they have been proven to diminish fraudulent operations ([FCC+03], [YHF+03]).

One important consideration with respect to the P2P Agent layer is the dependence of its implementation on the functionalities provided by the underlying platform. This can be observed in the figure C.1.

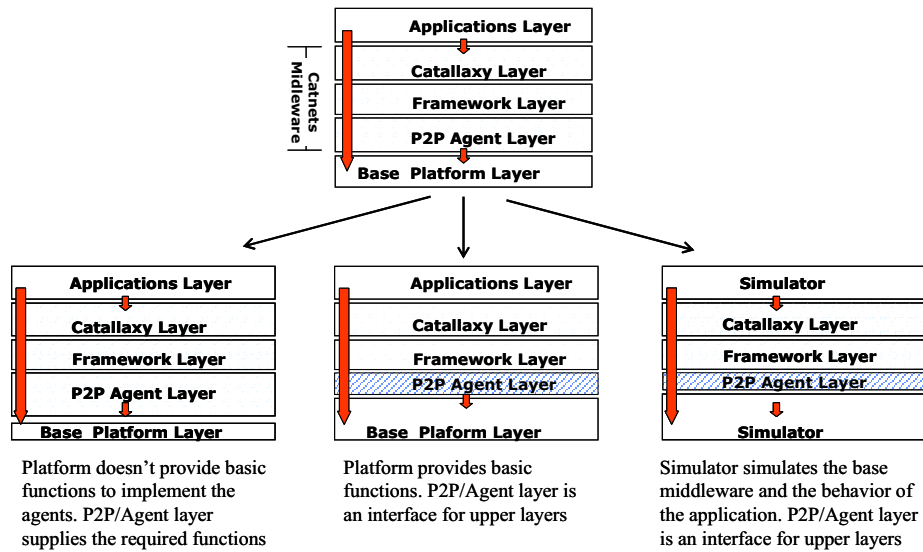


Figure C.1 - Implementation scenarios for P2P Agent Layer.

For example, if the base platform already provides a distributed location, the P2P Agent will implement a simple “pass through” interface to this functionality, instead of duplicating it. If the base platform’s functionality is incomplete, the P2P Agent layer will complement it to guarantee the required level of functionality. In any case, it will provide a standardized interface to the upper layer, regardless of the implementation details.

C.2 Requirements

The table C.1 shows the detailed functionalities of this layer and the key requirements that should be considered during the design stage and for the evaluation of implementation alternatives. Each function is classified as “Required” (●), “Convenient” (◐) or “Optional” (○).

Component	Functions	Key requirements
Execution Platform		
Hosting for the efficient execution of agents	<ul style="list-style-type: none"> ● Agent life cycle management, execution resource management (thread dispatching, memory, comm. channels) ◐ Exception notification and management ○ Agent state persistence ○ Migration and mobility of agents ○ Distributed activation (objects, components, agents) ○ Distributed transaction management 	Manage short lived agents (very frequent creation and destruction of agents) Do not expose the thread and memory management issues to programmers Provide mechanism for agent chaining and composition Do not impose negotiation or communication protocols
Resource Management		
Generic interface to base platform’s local	<ul style="list-style-type: none"> ● Resource discovery and query ◐ Resource allocation and 	Extensible representation of resource properties

resource management	<ul style="list-style-type: none"> ● deallocation ● Resource reservation (future allocation) ● Resource monitoring ○ Resource usage accounting ○ Resource related alarms (e.g. malfunctioning) 	<p>Support both direct queries and Publish/Subscribe models for resource information actualization</p> <p>Support monitoring of frequently changing attributes (e.g CPU workload or network bandwidth)</p>
Communication		
Abstracts the basic communication mechanisms and isolates agents from the complexities of the communication protocols.	<ul style="list-style-type: none"> ● Communication primitives (send, receive, multicast) ○ Logical addressing (global naming) of agents ○ Failure management 	<p>Best effort message delivery</p> <p>Easy coordination of many parallel conversations by a single agent</p> <p>Synchronous and asynchronous communication primitives</p> <p>Support for mobile nodes (location independent addressing)</p> <p>Support efficient group and system wide multi and any-casts</p>
Overlay Network		
Logical communication topology	<ul style="list-style-type: none"> ● Overlay network construction and maintenance ● Key based routing ● Peer grouping 	<p>Location awareness</p> <p>Enable both local and system wide information and request dissemination</p> <p>Support very frequently changing topologies (node membership and communication paths)</p>
Object Discovery		
Localization of catallactic middleware's objects based on attributes	<ul style="list-style-type: none"> ● Resource advertising and location ● Resource query and matchmaking ● Information cache management ○ Publication/subscription of information changes 	<p>Decentralized; do not requires global repositories</p> <p>Independent from the semantic of the resource description</p> <p>Complex queries (multiple resources, multiple attributes, partial matches, range matches)</p> <p>Scalable to the millions of objects and very frequent updates</p>
Security & Trust		
Assurance interacting agents' identities and rights	<ul style="list-style-type: none"> ● Agent authentication ● Agent access authorization (e.g. trade on a given market) ● Encryption of agent-to-agent communications ○ Non repudiation ○ Generic interface to base platform's security mechanism ○ Agent Reputation 	<p>Compliant with standards</p> <p>Decentralized/Federated to work in multi-domain environments</p> <p>Extensible to allow protection of new kind of objects</p> <p>Auditable</p>

Table C.1 - P2P Agent Layer functionality

Besides providing these functional requirements, the Catallactic middleware should also meet some technical requirements concerning performance, quality of service, scalability, availability, etc. Such technical requirements vary from one implementation to another even when providing the same functionality, due to the technology used to implement such functionality and the specific usage scenario (application and environment).

Therefore, in this initial analysis, we have limited the analysis to identify those requirements without quantifying them. During the design and implementation of the prototype, we will refine this analysis and provide specific metrics. In the table C.2 the

principal metrics are listed with an expected range for an “average” scenario (one that would not be atypical to find).

Factor	Functional Component	Description	Expected Range
Scalability	Hosting	Number of agents per node	> 1000
	Communications	Number of concurrent conversation per agent	> 10
	Resource Mgmt.	Number of resource information updates (per second)	> 100
	Object Discovery	Number of object queries issued by node (per second)	> 100
		Total number of objects registered	> 10 ⁶
	Overlay	Number of active nodes	> 1000
Responsiveness	Hosting	Maximum agent creation time (milliseconds)	< 100
		Maximum state persistence time (milliseconds)	< 500
		Maximum migration time (seconds)	< 2
	Communications	Maximum message round trip (milliseconds)	< 250
	Object Discovery	Maximum search time (seconds)	< 1
	Resource Mgmt.	Maximum allocation time (seconds)	< 1
		Maximum resource information update time (seconds)	< 1
Efficiency	Hosting	Maximum memory footprint (Mb)	< 20

Table C.2 - Performance requirements for P2P Agent Layer

C.3 References

[ADG+04] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schütt, E. Seidel, B. Ullmer (2004) “The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid” to appear in Proceedings of the IEEE, 93(3)

[BCG04] G. Blair, G. Coulson, P. Grace (2004), "Research Directions in Reflective Middleware: the Lancaster Experience", Proceedings of the 3rd Workshop on Reflective and Adaptive Middleware (RM2004) co-located with Middleware 2004, Toronto, Ontario, Canada, October 2004.

[BHPW04] D. Bauer, P. Hurley, R. Pletka, M. Waldvogel (2004), “Bringing Efficient Advanced Queries to Distributed Hash Tables”, 29th Annual IEEE International Conference on Local Computer Networks (LCN'04), November 16 - 18, 2004

[BJM04] O. Babaoglu, M. Jelasity, A. Montresor (2004), “Grassroots Approach to Self-Management in Large-Scale Distributed Systems”, In Proceedings of the EU-NSF Strategic Research Workshop on Unconventional Programming Paradigms, Mont Saint-Michel, France, September 2004.

[CJK+03] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, A. Wolman (2003), “An Evaluation of Scalable Application-level Multicast Built Using Peer-to-Peer Overlays”, In Proceedings. of IEEE INFOCOM, March-April 2003.

[DZD+03] F. Dabek, B. Zhao, P. Druschel, J. Kubiawicz, I. Stoica, (2003), “Towards a Common API for Structured Peer-to-Peer Overlays”, In the Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Berkeley, CA, 2003.

- [EAB+99] F. Eliassen, A. Andersen, G.S. Blair, F. Costa, G. Coulson, V. Goebel, O. Hansen, T. Kristensen, T. Plagemann, H.O. Rafaelsen, K.B. Saikoski, Y. Weihai Yu, (1999), "Next generation middleware: requirements, architecture, and prototypes", Proceedings. 7th IEEE Workshop on Future Trends of Distributed Computing Systems, Cape Town , South Africa, 1999
- [FCC+03] Y. Fu, J. Chase, B. Chun, S. Schwab, A. Vahdat (2003), "SHARP: an architecture for secure resource peering", In Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing, NY, USA, 2003
- [GCB+04] P. Grace, G. Coulson, G. Blair, L. Mathy, D. Duce, C. Cooper, W. Yeung, W. Cai, "GRIDKIT: Pluggable Overlay Networks for Grid Computing", Proceedings of International Symposium on Distributed Objects and Applications (DOA), Larnaca, Cyprus, October 2004.
- [HCW04] D. Hughes, G. Coulson, I. Warren, (2004), "A p2p Network with inherent support for adaptation", Technical Report, (comp-006-2004), Lancaster University, Lancaster, LA1 4YR.
- [HoB02] C. Hoile and E. Bonsma (2002), "Towards a minimal hosting specification for open agent systems : the lessons of IP" 1st International Workshop on "Challenges in Open Agent Systems", AAMAS2002, July 2002, Bologna, Italy.
- [LCC+02] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker (2002), "Search and replication in unstructured Peer-to-Peer networks", In Proceedings of the 16th international conference on Supercomputing table of contents, New York, USA, Pages: 84 – 95, 2002
- [PBC03] Parlavantzas, N., Blair, G.S., Coulson, G. (2003), "A Resource Adaptation Framework for Reflective Middleware", Proc. 2nd Intl. Workshop on Reflective and Adaptive Middleware (located with ACM/IFIP/USENIX Middleware 2003), Rio de Janeiro, Brazil, June, 2003.
- [PNC02] M. Purvis, M. Nowostawski, S. Cranefield (2002), "A multi-level approach and infrastructure for agent-oriented software development", Proceedings of the first international joint conference on Autonomous agents and multiagent systems, Bologna, Italy, 2002
- [TsRo03] D. Tsoumakos, N. Roussopoulos (2003), "A Comparison of Peer-to-Peer Search Methods" In Proceedings of the Sixth International Workshop on the Web and Databases, June 12-13 2003, San Diego, USA
- [YHF03] Y. Yan, Y. Huang, G. Fox, S. Pallickara, M. Pierce, A. Kaplan, A. Topcu. (2003), "Implementing a Prototype of the Security Framework for Distributed Brokering Systems", Proceedings of the 2003 International Conference on Security and Management. Volume I pp 212-218.

Annex D - Middleware toolkits evaluation

D.1 Identification of candidate middleware toolkits and evaluation process

We start this section recalling passages from the CATNETS proposal:

“For proper classification of this proposal, it should be noted that CATNETS surpasses the objectives of the ‘Grids for complex problem solving’ call (FP6-2.3.2.8), as its goals are not directly aimed at Grid technology but likewise at Autonomic Computing, Peer-to-Peer Computing, Web Services etc., and it does not intent to produce ready-to-use software tools, but aims at more fundamental understanding of the transferability of an economic concept to information systems in general”

It is clear from this statement, from the rich set of functional and non-functional requirements given in annex C of this document, that CATNETS middleware has no direct match with any existing middleware toolkit, but will rather integrate a set of features currently applied in separate approaches. The tools to be analyzed and evaluated for CATNETS middleware are thus taken from Web Services (WS), Grid, P2P, Content Distribution Networks (CDN), and from agent toolkits

The list of candidate tools to be examined is the following:

- J2SE [J2SE05] (including RMI [RMI05] and JNDI [JNDI05])
- Web Services [WeSe05], JAX-RPC [JaRp05], Axis implementation [Axis05]
- WSRF [WSRF05] / OGSA [OGSA05]
- JXTA [JXTA05]
- JADE [JADE05]
- Diet Agents [DIET05]

We select these tools for evaluation considering the following criteria:

- How well do they fit the identified CATNETS requirements
- Which is the current strength of the platform (support and maintenance, community commitment)
- Availability as open source
- Sources of information available, like bibliography (surveys and performance comparison papers), platform’s websites documentation and mailing lists, and our own and third parties/colleagues real experience with them.

Agent platforms are considered for the P2P-Agent Layer. There are important reasons for that. Decentralized negotiations in CATNETS will need support for these negotiations. Such support will provide maintaining several states, and the efficient performance of parallel conversations. Explicit support to agent's mobility could also be required. Those functionalities are addressed by agent platforms, but are not explicitly addressed by Grid or P2P platforms, like the Globus and JXTA implementations, neither by current WS specifications.

We have considered several alternatives to cover the P2P Agent Layer like JADE and DIET agent toolkits, pure Java, and others. JADE was selected because it is actually the "de facto" reference implementation of the FIPA standard, widely adopted in MAS community. Also it was shown by surveys to outperform alternative agent platforms (BGN04) like Tryllian [Tryl05] (commercial platform based on JXTA) and SAP [SAP05], a new platform close to but not fully implementing FIPA standard. DIET was selected for its novel bottom-up and light-weighted approach which we found very appropriate to comply with the identified CATNETS requirements. Another performance-oriented agent platform has been considered, Cougaar [Coug05], a java-based architecture for the construction of large-scale distributed agent-based applications. Even considering its interesting technical properties (Wrig04), we discarded it due to the fact that their objectives were far from the ones in CATNETS.

An alternative to using agent platforms is developing in Java the low level functionalities required by the P2P Agent Layer, namely thread management, state management, event management and agent/node mobility. The inconvenience is the high implementation cost of doing so; therefore our focus is first evaluating existing agent toolkits. The previously introduced tools are the ones we are going to carefully evaluate in the next sub-sections. Other platforms to be taken into account during the design and implementation phases are:

- P2P DHTs (Free Pastry[Past05], Coral DHT[Coral05], CHORD[Chor05], CAN[RFH+01])
- Peer-to-Peer-Simplified (P2PS) [P2PS05]

For the present evaluation P2P DHTs are too "low level" from the view of most of the considered properties. In fact they are could be suitable to cover one or only a few functions on the architecture, but not the complete P2P Agent Layer or Base Platform Layer. However, we see them as "complementary tools" to potentially provide some specific functionality in a later stage.

P2PS is a P2P platform which appeared recently is. It aims to provide a simplified version of JXTA. P2PS focus on the provision of basic P2P functions, an expressive and extensible P2P discovery mechanism and pipe based communication, without caring about more complicated functionalities commonly not required [Wang05]. P2PS currently is in its early stages, and lacks from some interesting features like peer groups and security (planned to be incorporated in the future). However, we appreciate its light weighted orientation and once further developed we should take it into account as a candidate for the Agent P2P Layer.

In the rest of this section we describe the evaluation of the selected six middleware toolkit candidates, regarding the properties relevant from CATNETS requirements point of view, and organized in three separated evaluations:

- **Functional view:** to what extent does the middleware toolkit cover the functionalities identified in the architecture?
- **Technical view:** which is the performance cost associated with the basic operations. Which are the technical limitations of the platform?
- **Development view:** community support, available resources and other aspects that important for the implementation

D.2 Presentation of the candidates

We will take here much more time describing what we refer to by “WebServices” and “Globus toolkit/WSRF” than describing the rest of platforms. The reason for that is that for the rest of platforms the current releases are stable and it is easy to get a common agreement to what their properties are. We cannot say the same of Web Services, which are currently merged in a complex and hard to follow standardization process. Since GT4 implements WSRF, it is also involved in the same unstable process, moving from OGSi to WSRF and re-factoring the whole specification. To clarify what exactly we consider inside of our Web Services evaluation and GT4/WSRF evaluation, we explicitly state which specifications we take into account for the evaluation. For the rest of the evaluated platforms we just give some architectural details.

D.2.1 Web Services JAX-RPC implementations (Axis)

Web Services (WS) [W3c04] is an interoperability architecture that provides a standard means for interaction between different software applications, running on a variety of platforms and/or frameworks. In this architecture, a Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. WS today goes far beyond the initial SOAP/WSDL/UDDI standards triad. Figure D.1 presents a basic Web Services Stack. Section D.6 describes in detail the WS specifications taken into account.

Business Domain Specific extensions	Various	Business Domain
Distributed Management	WSDM, WS-Manageability	Management
Provisioning	WS-Provisioning	
Security	WS-Security	Security
Security Policy	WS-SecurityPolicy	
Secure Conversation	WS-SecureConversation	
Trusted Message	WS-Trust	
Federated Identity	WS-Federation	
Portal and Presentation	WSRP	Portal and Presentation
Asynchronous Services	ASAP	Transactions and Business Process
Transaction	WS-Transactions, WS-Coordination, WS-CAF	
Orchestration	BPEL4WS, WS-CDL	
Events and Notification	WS-Eventing, WS-Notification	Messaging
Multiple message Sessions	WS-Enumeration, WS-Transfer	
Routing/Addressing	WS-Addressing, WS-MessageDelivery	
Reliable Messaging	WS-ReliableMessaging, WS-Reliability	
Message Packaging	SOAP, MTOM	
Publication and Discovery	UDDI, WSIL	Metadata
Policy	WS-Policy, WS-PolicyAssertions	
Base Service and Message Description	WSDL	
Metadata Retrieval	WS-MetadataExchange	

Figure D.1 - Web Services Stack (from [Cdbi04])

Apache Axis is an implementation of the JAX-RPC [JaRp05], specification, which defines a mapping between WSDL [WSDL05] and Java architecture, and supports communications based on SOAP [SOAP05]. One important feature of the JAX-RPC architecture is its extensibility by means of handlers that can be chained in the SOAP request processing for additional processing, like encryption.

Axis has proven itself to be a reliable and stable base on which to implement Java Web Services. There is a very active user community, which is part of the Apache Project [APAC] and there are many companies which use Axis for Web Service support in their products

There are some extensions to Axis that support additional WS related specification:

- WS-Addressing (Addressing)
- Support for WS-Security (WSS4J)
- Support for WS-ReliableMessaging (project still in incubation stage, with codename “Sandesha”)

D.2.2 WSRF/ OGSA

The WS-Resource Framework [WSRF05] is inspired by the work of the Global Grid Forum's Open Grid Services Infrastructure (OGSI) Working Group [OGSI05]. Indeed, it can be viewed as a straightforward refactoring of the concepts and interfaces developed in the OGSI V1.0 specification in a manner that exploits recent developments in Web services architecture (e.g. WS-Addressing).

OGSA design [Ggf04] is intended to facilitate the seamless use and management of distributed, heterogeneous resources. In this architecture, the terms “distributed,” “heterogeneous” and “resources” are used in their broad sense. For example:

“distributed” could refer to a spectrum from geographically-contiguous resources linked to each other by some connection fabric to global, multi-domain, loosely- and intermittently-connected resources. “Resources” refers to any artifact, entity or knowledge required to complete an operation in or on the system

OGSA pretends to enable interoperability between diverse, heterogeneous, and distributed resources and services as well as reduce the complexity of administering them. The need to support heterogeneous systems leads to requirements that are amenable to CATNET’s needs:

- Resource virtualization: management of diverse resources in a unified way.
- Common management capabilities: mechanisms for uniform and consistent management of heterogeneous systems
- Resource discovery and query: Mechanisms for discovering resources with desired attributes and for retrieving their properties
- Standard protocols and schemas: to allow platform and language neutral interoperability

Some of the functions required in distributed environments, such as security and resource management, may already be implemented by stable and reliable legacy systems. Therefore the integration of external components is a key design consideration.

The primary assumption is that work on OGSA both builds on, and is contributing to the development of the growing collection of technical specifications that form the emerging Web Services Architecture. Indeed, OGSA can be viewed as a particular profile for the application of core WS standards.

Even when OGSA emerged to address resource intensive scenarios related to e-Science (computational grids, data grids) it has evolved to a more general architecture and aims to include other scenarios like interaction from mobile devices and P2P systems.

To close the gap between those two worlds, the Commodity Grid Toolkit (CoG Kit) defines and implements a set of general components that map Grid functionality into commodity environment/framework, the like J2EE and DCOM.

The Globus Toolkit [Glob05] can be used to program grid-based applications. The toolkit, first and foremost, includes quite a few *high-level services* that we can use to build grid applications. These services, in fact, meet most of the abstract requirements set forth in OGSA. In other words, the Globus Toolkit includes a resource monitoring and discovery service, a job submission infrastructure, a security infrastructure, and data management services. Globus uses Axis SOAP engine and incorporates a Tomcat [Tomc05] Web Server.

OGSA has recently evolved to adhere to *WSRF* [WSRF05] as a fully WS based architecture. The more relevant specifications from WSRF are detailed in section D.6

The soon-to-be-released Globus Toolkit 4 (GT4) [GT405] (figure d.2), in fact, includes a complete implementation of the WSRF and Web Services Notification specifications.

GT4 provides an API for building stateful Web services targeted to distributed heterogeneous computing environments.

Since the working groups at GGF are still working on defining standard interfaces for these types of services, we cannot say at this point that GT4 is an implementation of OGSA (although GT4 does implement some security specifications defined by GGF). However, it *is* a realization of the OGSA requirements and a sort of *de facto* standard for the Grid community while GGF works on standardizing all the different services.

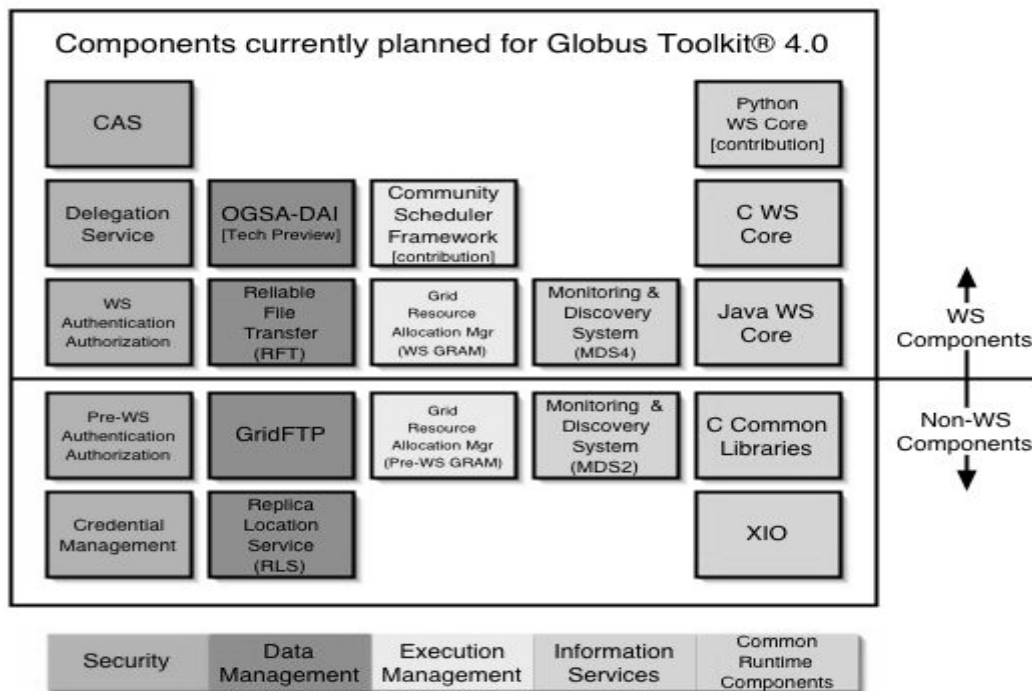


Figure D.2 - GT4 Architecture (from [GT4])

D.2.3 J2SE

Java technology [J2SE] is a portfolio of products that are based on the power of networks and the idea that the same software should run on many different kinds of systems and devices. In addition to the core and extension Java language libraries, J2SE includes the following

- RMI: Java Remote Method Invocation (Java RMI) [RMI05] enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines*, possibly on different hosts. RMI uses object serialization to marshal and un-marshal parameters and does not truncate types, supporting true object-oriented polymorphism.
- JNDI: The Java Naming and Directory Interface (JNDI) [JNDI05] is part of the Java platform, providing applications based on Java technology with a unified interface to multiple naming and directory services. It is possible to build powerful and portable directory-enabled applications using this industry standard.

D.2.4 JXTA

JXTA™ [JXTA05] technology is a set of open protocols that allow any connected device on the network to communicate and collaborate in a P2P manner. JXTA peers create a virtual network in which any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls and NATs or are on different network transports. Figure D.3 presents the basic architecture of JXTA, including the JXTA services and protocols.

The Project JXTA virtual network allows a peer to exchange messages with any other peers independently of its network location (firewalls, NATs or non-IP networks). Messages are transparently routed, potentially traversing firewalls or NATs. The Project JXTA virtual network standardizes the manner in which peers discover each other, self-organize into peergroups, discover peer resources, and communicate with each other.

Project JXTA builds upon the 5 virtual network abstractions. First, a logical peer addressing model that spans the entire JXTA network. Second, peergroups that let peers dynamically self-organize into protected virtual domains. Third, advertisements to publish peer resources (peer, peergroup, endpoint, service, content). Fourth, a universal binding mechanism, called the resolver, to perform all binding operations required in a distributed system. Finally, pipes as virtual communication channels enabling applications to communicate between each other.

All network resources in the Project JXTA network, such as peers, peergroups, pipes, and services are represented by advertisements. Advertisements are language-neutral metadata structures resource descriptors represented as XML documents. Project JXTA standardizes advertisements for the following core JXTA resource: peer, peergroup, pipe, service, metering, route, content, rendezvous, peer endpoint, transport.

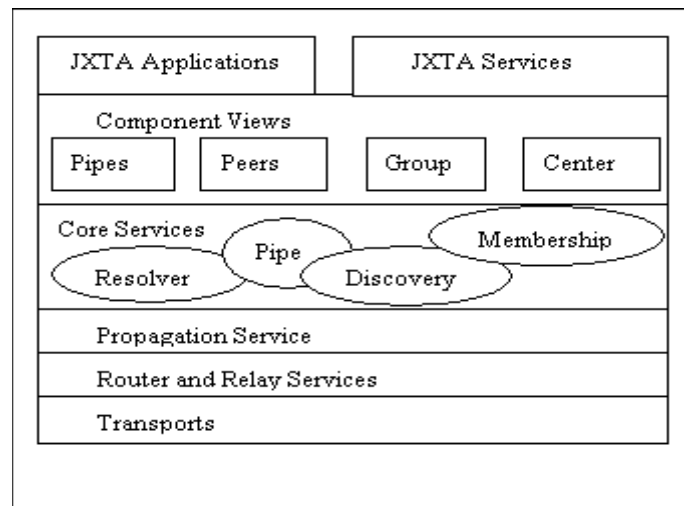


Figure D.3 - JXTA Architecture (from [Li01])

The JXTA 2.x release introduces the concept of a rendezvous peer view (RPV) to propagate resolver queries, and a shared resource distributed index (SRDI) to index advertisements on the rendezvous peer view for efficient advertisement query lookups.

D.2.5 JADE

JADE (Java Agent Development Framework) [JADE05] is a software framework implemented in the Java language. It simplifies the implementation of multi-agent systems by a middleware toolkit that complies with the FIPA [FIPA05] specifications and by a set of graphical tools that support the debugging and deployment phases.

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 even be changed at run-time by moving agents from one machine to another one, as and when required

The communication architecture of JADE offers flexible and efficient messaging, where it 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 differentiated and fully integrated: interaction protocols, envelope, ACL, content languages, encoding schemes, ontologies and transport protocols. The transport mechanisms 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 HTTP has been integrated. 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. Service level and agent management ontology are available, as well as the support for user-defined content languages and ontologies registered with the 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. Figure 4.4 illustrates the main elements of the JADE platform.

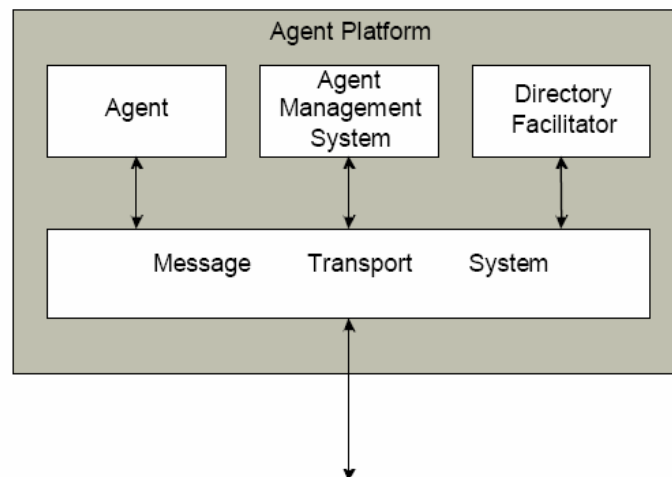


Figure D.4 - FIPA/JADE architecture (from [JadA04])

D.2.6 Diet Agents

DIET Agents [DIET05] is a multi-agent platform in Java. It was developed as part of the DIET project [DIET05] and released as Open Source at the end of the project. A bottom-up design was used to ensure that the platform is lightweight, scalable, robust,

adaptive and extensible. It is especially suitable for rapidly developing Peer-to-Peer prototype applications and/or adaptive, distributed applications that use bottom-up, nature-inspired techniques. The basic architecture is presented in figure D.5.

The platform has been designed to be scalable, robust and adaptive using a "bottom-up" design approach:

- It is scalable at a local and at a global level. Local scalability is achieved because DIET agents can be very lightweight. This makes it possible to run large numbers of agents in a single machine. The DIET Agents platform is scalable in the sense that the architecture does not impose any constraints on the size of distributed DIET application. The architecture is fully decentralized, thus does not impose centralized bottlenecks.
- It is robust and supports adaptive applications. The DIET kernel itself is robust to hardware failure and/or system overload. The effects of these failures are localized, and the kernel provides feedback when failure occurs allowing applications to adapt accordingly. The decentralized nature of the DIET Agents platform also makes it less susceptible to failure.
- It is based on a bottom-up, nature-inspired design approach. DIET agents are not assumed have artificial intelligence features and/or use complex communication protocols. Instead, agents can be very small and simple, allowing intelligent behaviour to emerge from the interactions between large numbers of agents
- Lightweight: The agents have a minimal memory footprint and inter-agent communication can be very fast. Agents can be thought of as small, mobile processes.
- Extensible: A high quality Object-Oriented design ensures that the code is general, modular and extensible. The Application Reusable (ARC) Layer provides support for plug-and-play agent behaviours, enabling modular construction of agents.

Application Layer	Application components
Application Reusable Layer (plugged into agents as jobs)	Application reusable components - remote communication - agent behaviours - events scheduling
Core Layer (minimal agent hosting environment)	DIET Agents kernel - Basic messaging - Thread Management - Mobility

Figure D.5 - DIET Agents Architecture (from [DietA04])

D.3. Middleware Evaluation Summary

D.3.1 Functional view: Mapping middleware toolkits into the architecture

In the functional view we evaluate which roles in the architecture could cover each middleware toolkit, mapping toolkits into the architecture low level layer boxes. This mapping requires considering characteristics like centralization/decentralization, discovery types support, degree of modularity, persistence, communication functions and security. A detailed evaluation concerning each of the identified functional requirements is presented in section D.5.1.

The middleware toolkits are evaluated in view of a set of functional requirements, which are: 1) execution platform, 2) resource management, 3) interoperable communication, 4) overlay network, and 5) security.

DIET and JADE provide the best *execution platform* for agents, due to the fact that they are developed to manage Multi-Agent Systems (MAS). JADE implements the FIPA standard and provides all the functionalities of the Agent Management System. A JADE platform relies on a main container, which contains the AMS and the yellow pages service or Directory Service (DF). Additional secondary containers in remote hosts are linked to the main one. The architecture is rather centralized concerning the agent management. Some support to decentralisation is given by the federation of replicated DFs. Significant support for the construction and management of reasoning agent is given (ontologies, integration with rule-engines and rich built-in behaviours templates). Contrarily, DIET concentrates in offering a much more light-weighted core with agent management being fully decentralized. It provides basic support for messaging and thread management. The goal of DIET is to achieve high performance, scalability and fault tolerance, promoting self-organization rather than those hard-coded reasoning agents. Thus, no extra support for the construction of reasoning agents is provided,

Resource Management is offered by WSRF/OGSA, since Grid resources management is one of the design goals. Within GT4 extensive support is given for local resource management, resource data collection and resource monitoring. The other middleware toolkits evaluated here do not give support to Resource Management, with exception of JXTA, which has some support for peer information management by means of the resolver service and peer monitoring service. Out of the evaluated middleware toolkits only GT4 considers the support for multiple competing applications (instead of cooperative applications), in which resource sharing becomes a critical issue.

Web Services provide support for *interoperable communication* with SOAP and a huge and continuously growing set of specification for WS-everything. This feature is very important because of the worldwide adoption and industrial support of WS standards for component interaction in loosely coupled distributed systems. JXTA also is oriented towards this direction providing pipes for point-to-point communication and XML based messaging. Ongoing projects of JXTA incorporate SOAP based communication in JXTA pipes. WSRF/OGSA leverages interoperability from WS. As for JADE and DIET, both accept extensions to support SOAP communication by means of a convenient java API. Though important, SOAP may not be always the best solution,

especially if performance becomes an issue. RMI or TCP/UDP may become much more adequate for those cases.

Overlay network functionalities and decentralized resource discovery is best provided by JXTA. The JXTA 2.x super-peer network [Trav03] provides a powerful and scalable key-based routing engine, enriched with JXTA expressive XML advertisements discovery. In contrast, centralized, or at best federated, discovery and notifications are supported in GT4/WSRF and Web Services by Index Service and UDDI respectively, which however limits their use in large-scale decentralized environments.

Security issues are best supported by standard WS-specifications in Web Services and GT4. JXTA provides also integrated but inflexible support for security. J2SE itself provides flexible APIs for security which can in fact turn much more modular for application integration by a good support for delegation.

It becomes clear from this functional analysis, that strong complementarities exist between the different middleware toolkits, and no single toolkit will cover all of the desired functionalities for CATNETS.

D.3.2 Technical View

We consider in this section technical parameters related to scalability, supported protocols, messaging channels, messaging types and performance, naming services and yellow pages performance, robustness and fault-tolerance. The comparison of the middleware toolkits is given in section D.5.2. The parameters taken into account are the following:

Standards & Protocols: Which standards are supported by the platform, and which protocols does it implement (if any)

Messaging Channels: Possible message channels for inter-platform messaging

Messaging Types: Considers synchronous and asynchronous messaging. Synchronous means that buffers are not used. Request from an agent/peer A to an agent/peer B are supposed to be followed by an answer from B before the start of any other conversation. Asynchronous means using buffers for messages and the ability to perform parallel conversations. Also covers if P2P multicast style communication is supported or not.

Messaging Performance: This is based on the Round Trip Time (RTT) measured in ms for the sending and reception of a message. The test beds are taken from existent surveys. This performance measure does not indicate the overall performance, since that also depends on scalability, naming services and memory issues. In order to get a comparative performance measure, we give the RMI messaging performance in ms, and express the other platforms relative to the RMI case, indicating $N \times$ RMI time.

Resource Discovery Performance: How well perform the publishing and discovery services on the platform.

Scalability N° Agents/Noses/Resources: Maximum number of agents/nodes/resources that can be instantiated without platform crash or heavy performance degradation.

The results of our study concerning the technical view are:

Standards & Protocols: Axis and GT4 leverage basically WS technologies and protocols. These standards have important supported in industrial settings. JXTA specifies a set of P2P standard protocols, targeting the full spectre of P2P applications (from file-sharing to corporative P2P applications).

JADE implements FIPA specification for MAS architectures, providing also standard behaviours. It is bound to that standard and it has been proved it is quite complicated for JADE-based multi-agent systems to interoperate with agent platforms not complying with FIPA. Some research efforts have been trying to interoperate FIPA with WS and J2EE architectures [LRCN03], but still remains unclear the future of those approaches. More and more Web Services are growing as the standard for “loosely couple open systems”, and that includes most of multi-agent applications. FIPA standard probably won't be continued and agent platforms will be based on extensions upon the WS-standards such as Web Services Conversation Language (WSCL) and WS-Agreement (the path followed by [PaJe05]). Another important conclusion from this paper is that WSCL and WS-Agreement are suitable for closing deals, but they don't give explicit support for auctioning and/or bargaining. That is an important fact to be taken into account by CATNETS if aiming to develop negotiating agents.

With a totally opposed philosophy, DIET Agents is standard agnostics and concentrates in providing modularity and a bottom-up design. DIET agents are not assumed to be highly intelligent and/or to use complex communication protocols. Instead, agents can be very small and simple, allowing intelligent behaviour to emerge from the interactions between large numbers of agents.

Messaging channels: We have here two kinds of middleware toolkit. On one side, the ones with XML envelope and commonly transported on top of HTTP (SOAP in WS and GT4, pipes in JXTA), which focuses on functionality. They bring loosely coupled interoperability and are also firewall and NAT friendly. On the other side the ones which focus more on performance. RMI brings an efficient invocation compared to SOAP. The penalty here is the loose in interoperability, since the applications need to talk Java RMI. In general JADE and DIET approaches to messaging are more flexible, since basic messaging is provided (RMI, TCP/UDP sockets), and XML based messaging however is either given as a plugging or expected to be implemented and plugged by the developer when needed.

Messaging Types: WS SOAP messaging is not explicitly specified to be synchronous, but current implementations are so. It is expected that JAX-RCP 2.0 will support asynchronous messaging. GT4 leverages from the AXIS engine and incorporates the same synchronicity. JXTA pipes provide more flexible communication by means of asynchronous messaging and P2P multicast. JADE and DIET, being agent platforms, provide asynchronous handling of messages such that agents are able to maintain parallel interactions with other agents.

Message Performance: SOAP and pipes as XML-based messaging mechanism perform considerably slower than RMI (~10 times slower) [Juri04]. Although different techniques exist in order to increase the performance of XML messaging [Chiu04], it needs to be considered in which cases SOAP messaging is actually required. JXTA messaging based on JXTA pipes also has a higher overhead than RMI. In general,

starting a platform in JXTA and performing peers discovery and messaging is a computationally demanding task [HaDe03a, HaDe03b]. JADE communication is based on RMI and its performance is close to this underlying technology [VQC02]. DIET agents default for remote communication is using TCP/UDP sockets, but other mechanisms could be plugged given a suitable Java API.

Resource Discovery Performance: jUDDI [JUDD05] was the implementation of WS UDDI repository evaluated due to its extensive presence. It has an average performance as centralized registry and suffers from performance degradation in the case of concurrent publications [SSB04]. Globus Index Service is expected to improve performance in GT4 from previous GT3 release. GT3 Index Service had problems to scale to thousands of nodes, but Globus developers expects from GT4 information Services (MDS4) to be able to scale up to 10000 nodes without performance degradation. JXTA super-peers network with the rendezvous nodes and the SRDI distributed hash table is expected to perform well for key-based routing [Sing03]. The provided DHT is considered as a compromise to perform well in most typical P2P configurations [Trav03]. JXTA rendezvous super peer network performance is analysed in deep and found to be good compared to both centralized and previous P2P flooding approaches [HDT04]. JADE Directory Facilitator performance is good for small platforms, but has been reported by some users to be very problematic when using federated registries [Jadx04].

Scalability N° Agents/Noses/Resources: UDDI and Globus Index Service central directories have limited scalability. Nevertheless, GT4 developers claim GT4 Index Service to be able to scale up to 10000 nodes, but no empirical test is available. JXTA P2P overlay network is expected to be able to gracefully scale using rendezvous nodes. Thus the current JXTA super peer network provides increased scalability (Trav03). For JADE, performance degrades when platform size scales, as pointed out by several colleagues when performing direct experimental testing. Additionally, some tests have been performed on JADE containers distributed across 8 nodes. The platform was not able to manage more than 600 agents, and some uncontrolled complex interactions JVM-JADE were detected (CTGK+04). DIET, which is specifically designed for scalability, is scalable at a local and at a global level. Local scalability is achieved because DIET agents can be very lightweight. This makes it possible to run a large numbers of agents in a single machine. The DIET Agents platform is also globally scalable, because the architecture does not impose any constraints on the size of distributed DIET applications. This is mainly achieved because the architecture is fully decentralized. An example of more than 100000 DIET agents successfully living in a 16 nodes cluster is given in [BoHo03].

D.3.3 Development view

We evaluate the middleware toolkit as a development platform for a complex development tasks. This is an important view since the challenge to “make real” the designed middleware architecture depends also on the ease of development and support provided by the platform and tools. The complete evaluation of development view is given in section D.5.3. The criteria considered in this evaluation are:

- **Languages:** The programming languages supported for developing with this platform

- **Community Support:** Strength of the community built around the platform. Universities and companies involved, deployed real applications, the platform and website maintenance are taken into account.
- **API:** Richness and complexity of the API provided. How easy is to program with the given API?
- **Modularity & integrability:** How amenable to be integrated within different architectures is the platform?
- **Available tooling:** Are there any tool developed to ease the use of this platform from the designer and implementer point of view?
- **Specifications & Documentation:** Is it understandable for the developers the specification? Which is the quality of the documentation provided with the software package or online?
- **Tutorials, books Developers sup. & mailing-list:** Available tutorials or books. Mailing lists are helpful? We evaluate also here to what extent platform developers are involved in the support to the platform users.

The results of our study concerning the development view are:

Languages: All 6 middleware toolkits are java based. The JXTA protocols specification has only one complete reference implementation, the Java one. WS has also a C++ implementations, but we consider Axis, developed by the Apache foundation and it is Java based. JADE and DIET are also fully developed in Java. It is clear that Java provides many facilities for distributed systems middleware programming which are not present in any other OO language. The most salient is platform independence due to the Java Virtual Machine. C++ libraries for networking are clearly inferior to the ones provided by Java. The only drawback for Java is the performance, and JIT technology for bytecode compilation is narrowing step by step the difference with machine compiled languages. In CATNETS we assume Java as the language most suitable for the project purposes.

Community Support: WS is clearly leading on the industrial support. That is certainly true, to the point that JXTA is adding a plugging to its pipe-based communication mechanism in order to provide SOAP invocation support. JADE also offers SOAP plugging, and in the future the FIPA standard which JADE implements will probably be replaced by some new multi-agent systems standards, focused in WS-family standards. Apart from that fact, JADE community is big and important within the agent community. Globus went even farer and moved from GT2 to the OGSi approach. WS standards are a first attempt, which finally embraced into WS with GT4 and WSRF. J2SE has the Java community behind, Sun support and a currently generalised and even increasing uptake of Java for distributed computing worldwide. JXTA is also supported by Sun and the community behind is quite big and very active. Nevertheless, their initial attempt to become the de-facto P2P standard has been far from succeeding. DIET is the weakest platform in this aspect, since it is a product developed by British Telecom and later released open-source. There are several academic projects using it, and BT is continuing industrial development with the platform as well as platform maintenance and enhancement.

API: We find that current WS (Axis), GT4 and JXTA APIs are definitely complex. XML-based interoperability has converted these platforms in something very abstract and the APIs are not easy to learn. The learning curve is high for those three technologies. It is also true that the objectives of these platforms are definitely broad and part of the complexity they exhibit comes directly from the complexity of the real problem they address.

A quick view on the GT4 architecture presented in section D.2.3 can give us an idea of the API complexity. The huge number of OGSA (from GT3 and GT4) and non-OGSA services provided (from GT2) increases the platform complexity.

As we will document with more detail in the next section, we consider specially complicated the JXTA API. If we look at JXTA architecture in section D.2.5 we can see support for groups, pipes, peer monitoring and security on the core itself, which turns into a quite complicated API. We believe that alternative projects like P2PS may suit better most developers needs. It is not as clear if the API complexity comes up directly from the problem complexity. Concerning JADE and DIET, they focus much more on providing a simple (richer in the case of JADE) API for easy developing MAS applications.

Modularity & integrability: WS and Globus provide very good integration between loosely coupled systems. JXTA and JADE aim to provide P2P and agent systems standards respectively, but failed in some sense since the adoption of their standards is not as popular as WS. JXTA is based on XML, and is in a better position in this sense. . As for DIET, its standard agnostic condition is another point of view to address the open systems problematic. The levels of modularity achieved by DIET are mostly due to its property of minimal core provided [HoBo02]. All extra communication, security, etc functionalities are plugged into the Application Reusable Component (ARC) layer. It is very lightweight, and comparing its architecture with the JXTA API we see that the core supports just basic messaging, thread management and agent mobility. Remote communication, a framework for pluggable agent behaviours and support for scheduling events are provided in the ARC Layer, promoting modularity and making it easy to plug additional features into the platform.

Available tooling: WS has currently extensive tools for the support of creation, management and orchestration of Web Services. GT4 expects leveraging all these WS tools via WSRF. JXTA provides few tools aside the JXTA platform, but that may be due to the fact that built-in JXTA protocols cover almost any P2P functionality needed by the developer. JADE provides debuggers and sniffers for monitoring of agents conversations. DIET provides some support for agent's interaction visualization. J2SE has been enriched by numerous IDEs, debugging tools and performance measurement tools. All this rich tooling for Java programming can be leveraged by the rest of the middleware toolkits since they are all Java based.

Specifications & Documentation: WS specifications are generally too dense, which is specially unpleasant for developers since there is no clear knowledge about which specifications are draft, which standard and which between them are available in the development platform selected. This is in part consequence of the novelty of WS, but also due to the un-stability of the open domain it addresses. GT3 had the same problem, but aggravated by adding the Grid-specific issues. With WSRF, specifications have been separated into 5 different sets of documents, covering the different subsets of

issues. A main problem in GT3 was the fact that documentation for practical development with the platform was very poor. It was really hard to get support for practical Grid application development with GT3, with the honourable exception of Borja Sotomayor Globus tutorial [Borj05]. That lack of support is not expected to be solved in GT4, since documentation is supposed to be developed by volunteers. This is a very negative point from the developer's point of view. JXTA documentation is better, but it covers just very simple cases. It is very hard to get support for more complicated application development, while the API itself is fairly complex. JADE documentation is much more extensive and useful from the developer's point of view, but still lacks support for the large scale MAS deployment step. DIET has a simple documentation, but programming with the platform is quite easy, such that the provided documentation is enough to understand the basics of programming with DIET. Like in JADE, support for the large scale MAS deployment is also missing in the DIET documentation.

Tutorials, books Developers sup. & mailing-list: Papers and books on WS, Globus and JXTA are extensively available. One problem with books is that code gets quickly outdated. The problem with papers is that they give a good overview, but few resources for practical implementation. In general, good tutorials for the practical development with platforms are very rare. Users and developers mailing-lists are very active for these platforms. JADE also has a very active mailing-list. DIET mailing lists are much less active, one reason might be the small size of the community, but DIET platform developers themselves have given support to DIET users. As for Java, the Java Tutorial covers almost all need for basic development, and countless books and advanced tutorials provide support for development.

D.3.4 Tests on middleware toolkits integration

We evaluate in this section the ease of integration with other platforms. This is especially important for CATNETS since expect to build the CATNETS middleware from a combination of middleware toolkits. Most of the evaluation work in this section is first-hand, conducted by the CATNETS WP3 members through tests and implementations using the toolkits of.

We have tested DIET – JXTA Discovery Service integration. Both toolkits are complementary, since DIET does not provide P2P Discovery mechanism, but expect the developer to plug one himself taking advantage of DIET decentralized architecture. JXTA Discovery Service has promising features, incorporating a built-in DHT (SRDI). The result is that DIET reusable jobs in the ARC layer provide a useful placeholder for such discovery mechanism. It could be seen that the integration of this functionality of JXTA into DIET was fairly straightforward. From the tests it appears that integrating another discovery or remote communication mechanism into DIET (for example an UDDI registry for centralized discovery, or a SOAP engine for Web/Grid services invocation) would also use the corresponding Java API (UDDI4J, AXIS, etc). That feature comes from the minimal DIET core, which does not impose any standards for transport channels, remote communications, directory management or semantics.

As for JXTA, we found it very hard to decouple the discovery service from the rest of JXTA protocols in order to use it as a ready-to-use service. In the standard usage one is forced to launch the JXTA platform using the graphical JXTA configuration tool, including security settings. Flexibility is clearly not a feature in JXTA. Developers aiming to use just the discovery service apparently need to start the whole JXTA

platform and use the XML-based advertisements, which can lead to a performance problem in some applications. From this test we identified the need of considering alternative P2P discovery implementations, as for example the earlier mentioned P2PS. P2PS could be much more light-weighted, allowing easier integration with other platforms. P2P DIET [StMk04], developed also within the DIET project could be another interesting alternative to provide P2P discovery service.

Another test we have carried out is the integration of GT3 Grid Services invocation into JADE. It revealed to be quite straightforward since both kits are Java, so importing the Globus API into JADE application was enough to provide a clean interaction with GT3 Grid Services from JADE. We did not attempt to address Discovery issues merging Globus Index Service with JADE, but we believe this will be quite a hard issue. JADE Directory Facilitator has been reported by other colleagues at UPC to be very inflexible. There are several attempts to provide a P2P-aware DF for JADE, most of them integrating JADE and JXTA, to our knowledge without success [Jadx04]. The FIPA specification for DF is too rigid and centralized and its integration with P2P architectures leads to bad performance.

We will continue performing tests on middleware toolkits integration, since this is an important issue for CATNETS and can also give us practical clues on the feasibility of the proposed architecture for CATNETS

D.4 Conclusions

D.4.1 Conclusions on functional, technical and development views

The conclusion of our study on middleware toolkits for CATNETS implementation is that no single middleware toolkit fulfils all the requirements. However, exploiting complementarities of different middleware toolkits and integrating them in the proposed architecture we could get a Catallactic middleware, which potentially can be:

- Flexible and robust, being able to cope with heterogeneity and dynamics
- Efficient in order successfully implement and reproduce expected behaviour of the Catallactic model
- Complete in the sense that it can cover several ALN domains.

We provided a classification of the middleware toolkits with regards to the P2P Agent Layer. JADE, DIET Agents or just pure java with J2SE are candidates to cover the P2P Agent Layer. WS, WSRF/OGSA, and JXTA are also able to cover the Base Platform Layer. Additionally, several functions on the P2P Agent Layer may be also covered by the middleware toolkit from the Base Platform Layer. For example JXTA can be used for P2P Discovery of DIET agents; GT4 can cover security for a Grid application, and so on. We have evaluated to what extent the middleware toolkits cover each of the identified functionalities. From that analysis it becomes explicit that no single middleware toolkit provides all the desired functionalities. Then, CATNETS middleware will be a composition, guided by the developed architecture.

We have considered performance issues, since the number of negotiations required by agents in CATNETS may constraint the type of messaging, discovery, or both,

depending on the application. Thus, to implement the prototype of CATNETS it is not enough porting the algorithms from the simulator into a prototype.




We identify current ALN middleware WS, GT4, and JXTA to be very complex: Huge and dense specifications, complex architectures and XML-messaging could not provide lead the modularity desired by the CATNETS architecture.

The P2P Agent Layer is the proposed solution to address the ALN middleware integration. We need from the middleware toolkit covering that layer to be modular enough to provide a reasonable integration with the Base Platforms. To that respect DIET clearly outperforms JADE, since the FIPA specification is too rigid to delegate functionalities into the Base Platform. DIET Agents gives support where we need it (basic messaging and platform management, thread and memory management, context support for negotiations and mobility) without imposing communication transports, semantics or centralized discovery mechanisms. An alternative for the P2P Agent Layer implementation is using directly J2SE, which gives as total freedom, but also would require considering low-level platform management implementation issues.

Considering documentation and development support, industrial standards like Web Services have extensive support, while other technologies as GT4 and JXTA lacks support for development when regarding their platform complexity. JADE has a good documentation, and DIET has a too simple documentation, but good platform developer's support. J2SE is best rate on support issues since Sun and the Java communities are behind taking care on comfortable Java language adoption by developers.

In figure D.6 we graphically summarize the scoring of each middleware toolkit regarding the set of functional, technical and development views properties.

	MIDDLEWARE TOOLKIT/ PROPERTY	JXTA	WS	WSRF / OGSA	JADE	DIET	J2SE
Functional	Execution platform						
	Overlay Network						
	Object Discovery						
	Communication Primitives						
	Resource Management						
	Security						
Technical	Standards support						
	Messaging types						
	Messaging performance						
	Object Discovery Performance						
Development	Scalability						
	Maturity & Support						
	API complexity						
	Modularity and Integrability						
	Specifications and documentation						
	Available tooling						

Figure D.6: Middleware toolkit scoring on the 3 views  Good  Average  Bad

D.4.2 Joint selection of middleware and application

As a result of our findings we state:

- CATNETS middleware must be a flexible composition of existent middleware toolkits in order to handle the inherent complexity of real Grid and P2P scenarios and to implement the catallactic model.
- Part of the middleware used in a particular implementation depends on the application selected. The strengths identified in each of the candidate middleware toolkits are then used to decide upon the set of implementation toolkits.

D.5 Middleware Toolkit Evaluation Details

D.5.1 Functional View

Requirement/Toolkit	JXTA	WS (JAX-RPC Imp)	WSRF /OGSA	JADE	DIET	J2SE
Execution Platform						
Agent Hosting (lifecycle & Execution resource management)	Built-in /inefficient	None	Built-in /WSRF	Built-in /efficient	Built-in/ lightweight	None
Exception notification and management	Built-in	None / depends on implementation	Built-in /WS – BaseFault	Built-in	Built-in	None
Agent state persistence	None	None / Stateless	Built-in /WSRF Stateful	Built-in	Built-in	None
Migration and mobility	Built-in	None	None	Built-in	Built-in	
Resource Management						
Resource allocation and deallocation	None	None	Built-in /WS-GRAM	None	None	None

Requirement/Toolkit	JXTA	WS (JAX-RPC Imp)	WSRF /OGSA	JADE	DIET	J2SE
Resource Monitoring	Built-in / Resolver Service	None	Built-in WS-ResourceProperties	None	None	None
Resource Discovery and Query	Built-in/ Monitoring Service	None	Built-in /MDS Centralized	None	None	None
Communication						
Agent addressing and location	Built-in / Discovery Service	Built-in /UDDI	Built-in /WS-Addressing	Built-in / Agent IDs	Built-in /Agent Tags and IDS	Built-in / JNDI
Basic mechanism (send, receive, multicast)	Built-in / by Pipes	Built-in / SOAP Supported/WS-Notification	Built-in /WS-Notification	Built-in / RMI-IOP No multicast	Built-in / ARC Layer	Built-in / RMI No multicast
XML message handling (marshalling and unmarshaling)	Built-in	Built-in	Built-in /WSRF	Supported / plugging extension	None	Built-in / Java XML tooling

Requirement/Toolkit	JXTA	WS (JAX-RPC Imp)	WSRF /OGSA	JADE	DIET	J2SE
Failure Management in communication	Built-in	Built-in / SOAP faults	Built-in / WS – Base Fault	Built-in/ but may block	Built-in/ fast-fail	Built-in / RMI remote exceptions
Overlay Network						
Overlay Network construction & maintenance	Built-in	None	None	None	None	None
Peer grouping	Built-in / JXTA Groups	None	Built-in /WS- Service Group	None	Built-in / Family Tags	None
Key based routing	Built-in / SRDI	None	None	None	Supported / plugging	None
Resource Discovery						
Advertisement & search	Built-in / Jxta Advs	Built-in / WSDL/UDDI	Built-in / Globus Information Services	Built-in / FIPA DF	Supported /plugging ARC Layer	Built-in / JNDI
Matchmaking	Built-in / Rich	Built-in / Basic	None	Built-in / Rich	None	Built-in / JNDI

Requirement/Toolkit	JXTA	WS (JAX-RPC Imp)	WSRF /OGSA	JADE	DIET	J2SE
Cache management	Built-in / efficient	None	Built-in / GASS	None	None	None
Publication/subscription of information changes	None	Supported / WS-Notification	Built-in / WS-Notification	Built-in	None	None
Security & Reputation						
Agent authentication	Built-in	Supported / Ws-Security	Built-in / WSRF Security	Supported / Plugging	Built-in/ SSL sockets that require authentication	JAAS
Access authorization (to trade in a given market)	Built-in	Supported /Ws-Security	Built-in /WS-Security, SAML	Supported / Plugging	None	JAAS / certificates
No repudiation	Built-in /JXTA Security	Supported / WS-Signature	Supported/WS-Agreement	None	None	Built-in / Digital Signature

Requirement/Toolkit	JXTA	WS (JAX-RPC Imp)	WSRF /OGSA	JADE	DIET	J2SE
Interface to Base platform security mechanisms	None	None	None	None	None	Built-in / JAAS
Encryption of communications	Built-in	Supported /Ws-Security	Built-in /WS-Security	Supported / Plugging	Built-in / JADE-S	Built-in / JSSE
Trust/reputation mechanisms	None	Supported /Ws-Trust (limited)	WS-Trust (draft specification)	None	None	None

D.5.2 Technical View

Requirement /Toolkit	JXTA	Web Services (JAX-RPC Imp)	WSRF / OGSA	JADE	DIET	J2SE
- Standards & Protocols	<ul style="list-style-type: none"> - peers, groups - JXTA pipes - XML advertisements 	<ul style="list-style-type: none"> - SOAP - WSDL - UDDI - XML 	<ul style="list-style-type: none"> - Web Services - Index Service - GridFTP - Grid Security 	<ul style="list-style-type: none"> - FIPA ACL - AMS & DF - Behaviours - Interaction - Protocols 	<ul style="list-style-type: none"> - bottom-up design - decentralized - standard agnostic 	<ul style="list-style-type: none"> - JVM - RMI - JNDI
- Messaging Channels	<ul style="list-style-type: none"> - JXTA pipes, various protocols - Firewall and NAT friendly 	<ul style="list-style-type: none"> • Any (normally HTTP) - Firewall friendly 	<ul style="list-style-type: none"> - Same as WS (Axis) 	<ul style="list-style-type: none"> - RMI, ORB - HTTP, JMS by existent plugging 	<ul style="list-style-type: none"> - UDP and TCP - Mobile agents - Any other can be plugged 	<ul style="list-style-type: none"> - RMI - Sockets
- Messaging Types	<ul style="list-style-type: none"> - Asynchronous - P2P /unicast or multicast 	<ul style="list-style-type: none"> - Synchronous 	<ul style="list-style-type: none"> - Synchronous 	<ul style="list-style-type: none"> - Asynchronous 	<ul style="list-style-type: none"> - Asynchronous - P2P / mechanism need to be plugged 	<ul style="list-style-type: none"> - Synchronous
- Messaging Perform. (ms)	<ul style="list-style-type: none"> - 10 x RMI 	<ul style="list-style-type: none"> - 9 x RMI 	<ul style="list-style-type: none"> - Relies on WS invocation (Axis) - GT4 improves 4 x GT3 performance 	<ul style="list-style-type: none"> - 2xRMI 	<ul style="list-style-type: none"> - Fast for UDP or TCP sockets - Rest depends on transport plugged 	<ul style="list-style-type: none"> - Round Trip Time (RTT) average : - 0.25ms
- Resource Discovery Performance	<ul style="list-style-type: none"> - Average, XML processing penalizes - Good for key based routing 	<ul style="list-style-type: none"> - Average (JUDDI) - Degradation for concurrent publications 	<ul style="list-style-type: none"> - Expected to be Average - Index Service refactored in GT4 (with JNDI) 	<ul style="list-style-type: none"> - Good for small and medium sized MAS - Average/Bad for federated DFs 	<ul style="list-style-type: none"> - Good for known hosted agents - Discovery depends on plugged mechanism 	<ul style="list-style-type: none"> - Depends on plugged service - (e.g. JNDI over LDAP)

Requirement /Toolkit	JXTA	Web Services (JAX-RPC Imp)	WSRF / OGSA	JADE	DIET	J2SE
- Scalability - N° Agents - /Nodes /Resources	- Expected to be Very Good	- ----	- Expected to be limited due to central management - GT4 developers expect >> 10000	- Expected to be >10000 - Proved to be >500 -	- Expected to be >>100000 - Proved to be >10000	- Depends on plugged service - (e.g. JNDI over LDAP)

D.5.3 Development View

Requirement/ Toolkit	JXTA	Web Services (JAX-RPC Imp)	WSRF /OGSA	JADE	DIET	J2SE
- Programming - Language	- Java	- Language neutral	- Java	- Java	- Java	- Java
- Maturity and Community Support	- Sun support and active website and community	- Very good, many companies and projects - Industrial standard	- Many involved companies - GT3 still buggy - Active website and community	- Agent community support - FIPA standard	- BT product - Very small community	- Sun and very active website and community - Popular for networked app
- API	- Complex API with many protocols	- Java API	- Complex API - WSRF refactorization makes it clearer	- Not modular in general - Allows new transports to be plugged	- Simple API -	- Complete and functional - Specific support for networked app
- Modularity & integrability	- P2P standard protocols - Allows few extensions	- Interoperability - HTTP /XML - Lacks standards for composite or federated UDDIs	- Grid Services interoperability -	- P2P standard protocols - Allows few extensions	- Modular: minimum core + reusable	- Platform independence - Object oriented and rich class library
- Available tooling	- Few or no tooling	- Extensive -	- Few or inexistent - WSRF leverages WS tooling	- Sniffer and debugger for MAS provided	- Some support for MAS visualization	- A lot of tools - Many IDEs - Debuggers
- Specifications & Documentation	- Complex specification - Abundant documentation	- Extensive - Sometimes too complex specifications	- Too complex specifications - WSRF alleviates this problem	- Complex FIPA Specification - Abundant documentation	- Too simple documentation	- Extensive and tested documentation

Requirement/ Toolkit	JXTA	Web Services (JAX-RPC Imp)	WSRF /OGSA	JADE	DIET	J2SE
- Tutorials, books Developers sup. & mailing-lists	-- Some existent -- Too simple examples, few troubleshooting sections	-- A lot to try -- Many developers in WS community	-- Very poor -- Too many description papers, few code examples	-- Some existent -- Covers only basic issues -- Active mailing list	-- Just one tutorial -- Very good	-- Extensive and well proved -- Many reusable code available online

D.6 Relevant Standards

D.6.1 Web Services Standards

- SOAP: W3C (www.w3c.org) standard

SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.

Two major design goals for SOAP are simplicity and extensibility (see XMLP Requirements [XMLP Requirements]). SOAP attempts to meet these goals by omitting, from the messaging framework, features that are often found in distributed systems. Such features include but are not limited to "reliability", "security", "correlation", "routing", and "Message Exchange Patterns" (MEPs). While it is anticipated that many features will be defined, this specification provides specifics only for two MEPs. Other features are left to be defined as extensions by other specifications.”

(<http://www.w3.org/TR/soap12-part1>)

- WSDL: W3C (www.w3c.org) standard

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

A WSDL document 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: 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, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

(<http://www.w3.org/TR/wsdl>)

- UDDI: Oasis (www.oasis-open.org) standard

Universal Description, Discovery and Integration, or UDDI, is the name of a group of web-based registries that expose information about a business or other entity and its

technical interfaces (or API's). These registries are run by multiple Operator Sites, and can be used by anyone who wants to make information available about one or more businesses or entities, as well as anyone that wants to find that information. There is no charge for using the basic services of these operator sites.

By accessing any of the public UDDI Operator Sites, anyone can search for information about web services that are made available by or on behalf of a business. The benefit of having access to this information is to provide a mechanism that allows others to discover what technical programming interfaces are provided for interacting with a business for such purposes as electronic commerce, etc. The benefit to the individual business is increased exposure in an electronic commerce enabled world.

The information that a business can register includes several kinds of simple data that help others determine the answers to the questions "who, what, where and how". Simple information about a business – information such as name, business identifiers (D&B D-U-N-S Number®, etc.), and contact information answers the question "Who?" "What?" involves classification information that includes industry codes and product classifications, as well as descriptive information about the services that the business makes available. Answering the question "Where?" involves registering information about the URL or email address (or other address) through which each type of service is accessed. Finally, the question "How?" is answered by registering references to information about interfaces and other properties of a given service. These service properties describe how a particular software package or technical interface functions. These references are called tModels in the UDDI documentation (<http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm>)

- WS-Security: is a OASIS (www.open-oasis.org) standard

This specification proposes a standard set of SOAP [SOAP11, SOAP12] extensions that can be used when building secure Web services to implement message content integrity and confidentiality. This specification refers to this set of extensions and modules as the "Web Services Security: SOAP Message Security" or "WSS: SOAP Message Security".

This specification is flexible and is designed to be used as the basis for securing Web services within a wide variety of security models including PKI, Kerberos, and SSL. Specifically, this specification provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies. The token formats and semantics for using these are defined in the associated profile documents.

This specification provides three main mechanisms: ability to send security tokens as part of a message, message integrity, and message confidentiality. These mechanisms by themselves do not provide a complete security solution for Web services. Instead, this specification is a building block that can be used in conjunction with other Web service extensions and higher-level application-specific protocols to accommodate a wide variety of security models and security technologies.

These mechanisms can be used independently (e.g., to pass a security token) or in a tightly coupled manner (e.g., signing and encrypting a message or part of a message and providing a security token or token path associated with the keys used for signing and encryption).

(<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>)

- WS-Trust (non standard specification proposed by Microsoft and IBM)

WS-Trust will describe the model for establishing both direct and brokered trust relationships (including third parties and intermediaries).

This specification describes a model for brokering trust through the creation of Security Token Services (STS). These security token issuance services build on WS-Security to transfer the requisite security tokens in a manner that ensures the integrity and confidentiality of those tokens.

(<http://www-106.ibm.com/developerworks/library/ws-trust/>)

- WS-Notification: W3C (www.w3c.org) standard not yet approved.

The Event-driven, or Notification-based, interaction pattern is a commonly used pattern for inter-object communications. Examples exist in many domains, for example in publish/subscribe systems provided by Message Oriented Middleware vendors, or in system and device management domains.

The WS-Notification family of specifications defines a standard Web services approach to notification. It defines the normative Web services interfaces for two of the important roles in the notification pattern, namely the NotificationProducer and NotificationConsumer roles. This specification includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with operational requirements expected of them

In addition, this specification defines the Web services interface for the NotificationBroker. A NotificationBroker is an intermediary, which, among other things, allows publication of messages from entities that are not themselves service providers. It includes standard message exchanges to be implemented by NotificationBroker service providers along with operational requirements expected of service providers and requestors that participate in brokered notifications.

(<http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf>,

<http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BrokeredNotification-1.2-draft-01.pdf>)

- WS-Addressing: is a W3C (www.w3c.org) standard.

WS-Addressing provides transport-neutral mechanisms to address Web services and messages. Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end 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.

Web Services Addressing (WS-Addressing) defines two interoperable constructs that convey information that is typically provided by transport protocols and messaging systems. These constructs normalize this underlying information into a uniform format that can be processed independently of transport or application. The two constructs are endpoint references and message information headers.

A Web service endpoint is a (referenceable) entity, processor, or resource where Web service messages can be targeted. Endpoint references convey the information needed to identify/reference a Web service endpoint, and may be used in several different ways: endpoint references are suitable for conveying the information needed to access a Web service endpoint, but are also used to provide addresses for individual messages sent to and from Web services. To deal with this last usage case this specification defines a family of message information headers that allows uniform addressing of messages independent of underlying transport. These message information headers convey end-to-end message characteristics including addressing for source and destination endpoints as well as message identity.

Both of these constructs are designed to be extensible and re-usable so that other specifications can build on and leverage endpoint references and message information headers.

(<http://www.w3.org/Submission/ws-addressing/>)

- WSRF: Oasis (www.oasis-open.org) family of standards to manage stateful services
WS-Resource specification defines what the relationship between Web services and stateful resources is. This relationship is described as the WS-Resource Access Pattern. In the WS-Resource Access Pattern, messages to a Web service may include a component that identifies a stateful resource to be used in the execution of the message. The composition of a stateful resource and a Web service is a WS-Resource.

For more information see WSRF/OGSA subsection D.2.3, or available links:

(<http://docs.oasis-open.org/wsr/2004/11/wsr-WS-Resource-1.2-draft-02.pdf>,

<http://docs.oasis-open.org/wsr/2004/11/wsr-WS-ResourceProperties-1.2-draft-05.pdf>,

<http://docs.oasis-open.org/wsr/2004/11/wsr-WS-ServiceGroup-1.2-draft-03.pdf>)

- WS-Agreement: Global Grid Forum (www.ggf.org)
The objective of the WS-Agreement specification is to define a language and a protocol for advertising the capabilities of service providers and creating agreements based on creational offers, and for monitoring agreement compliance at runtime.
The goals of WS-Agreement are to standardize the terminology, concepts, overall agreement structure with types of agreement terms, agreement template with creation constraints and a set of port types and operations for creation, termination and monitoring of agreements, including WSDL needed to express the message exchanges and resources needed to express the state.
(http://www.ggf.org/Public_Comment_Docs/Documents/Public_Comment_2004/WS-AgreementSpecification_v2.pdf)
- WS-Reliability: Oasis (www.oasis-open.org) standard for web services reliable messaging.
WS-Reliability is a SOAP-based specification that fulfils reliable messaging requirements critical to some applications of Web Services. SOAP over HTTP is not sufficient when an application-level messaging protocol must also guarantee some

level of reliability and security. This specification defines reliability in the context of current Web Services standards.

Reliable Messaging (RM) is the execution of a transport-agnostic, SOAP-based protocol providing quality of service in the reliable delivery of messages. There are two aspects to Reliable Messaging; both must be equally addressed when specifying RM features: (1) The “wire” protocol aspect. RM is a protocol, including both specific message headers and specific message choreographies, between a sending party and a receiving party. (2) The quality of service (QoS) aspect. RM defines a quality of messaging service to the communicating parties, viz., the users of the messaging service. This assumes a protocol between these users and the provider of this service (i.e., the reliable messaging middleware). This protocol is defined by a set of abstract operations: Submit, Deliver, Notify, and Respond.

Reliable messaging requires the definition and enforcement of contracts between:

The Sending and Receiving message processors (contracts about the wire protocol)

The messaging service provider and the users of the messaging service (contracts about quality of service).

<http://docs.oasis-open.org/wsrn/2004/06/WS-Reliability-CD1.086.pdf>

D.6.2 WSRF Related Standards

The WSRF specification: The Web Services Resources Framework is a collection of five different specifications.

- **WS-ResourceProperties:** A resource is composed of zero or more *resource properties*. For example, in the figure shown above each resource has three resource properties: Filename, Size, and Descriptors. WS-ResourceProperties specifies how resource properties are defined and accessed. As we'll see later on when we start programming, the resource properties are defined in the Web service's WSDL interface description.
- **WS-ResourceLifecycle:** Resources have non-trivial lifecycles. In other words, they're not a static entity that is created when our server starts and destroyed when our server stops. Resources can be created and destroyed at any time. The WS-ResourceLifecycle supplies some basic mechanisms to manage the lifecycle of our resources.
- **WS-RenewableReferences:** Once we have a WS-Resource's endpoint reference, there might be some cases where we'll need to renew that reference if it becomes invalid. The WS-RenewableReferences specification defines the mechanisms to do this.
- **WS-ServiceGroup:** We will often be interested in managing groups of Web Services or groups of WS-Resources, and performing operations such as 'add new service to group', 'remove this service from group', and (more importantly) 'find a service in the group that meets condition FOOBAR'. The WS-ServiceGroup specifies how exactly we should go about grouping services or WS-Resources together. Although the functionality provided by this specification is very basic, it is nonetheless the base of more powerful discovery services (such as GT4's IndexService) which allow us to

group different services together and access them through a single point of entry (the service group).

- WS-BaseFaults: Finally, this specification aims to provide a standard way of reporting faults when something goes wrong during a WS-Service invocation.

D.7 References

[Apac05] <http://ws.apache.org/>

[Axis05] <http://ws.apache.org/axis/>

[BGN04] K. Burbeck, D. Garpe, and S. Nadjm-Tehrani, Scale-up and Performance Studies of Three Agent Platforms, in *Proceedings of International Performance, Communication and Computing Conference, Middleware Performance workshop.*, (Phoenix, Arizona, USA), pp. 857--863, Apr. 2004

[BoHo03] E. Bonsma and C. Hoile, "A distributed implementation of the SWAN Peer-to-Peer look-up system using mobile agents", 1st International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002), AAMAS2002, July 2002, Bologna, Italy.

[Borj05] The Globus Toolkit 4 Programmer's Tutorial, <http://gdp.globus.org/gt4-tutorial/>

[Cdbi04] CDBI Forum(2004), "Web Services Stack", <http://roadmap.cdbiforum.com/reports/protocols/index.php>

[Chiu04] Chiu, Kenneth, Web Services Performance: A Survey of Issues and Solutions, 8th Multi-conference on Systemics, Cybernetics and Informatics: SCI 2004

[Chor05] <http://www.pdos.lcs.mit.edu/chord/>

[Cora05] <http://www.scs.cs.nyu.edu/coral/>

[Cort02] E. Cortese, F. Quarta, G. Vitaglione, "Scalability and Performance of JADE MessageTransport System", AAMAS Workshop on Agencies, Bologna, July 2002.

[Coug05] <http://www.cougaar.org/>

[CTGK+04] Chmiel, D. Tomiak, M. Gawinecki, P. Karczmarek, M. Szymczak M. Paprzycki, Testing the Efficiency of JADE Agent Platform, in: *Proceedings of ISPDC 2004*, IEEE CS Press, Los Angeles, 2004, 49-56

[DieA04]DIET Overview <http://dietagents.sourceforge.net/PlatformOverview.html>

[DIET05] <http://diet-agents.sourceforge.net/Index.html>

[FIPA05] <http://www.fipa.org/>

[Ggf04] Globus Grid Forum (2004) [url]

[Glob05] <http://www.globus.org/>

[GT405] <http://www-unix.globus.org/toolkit/docs/development/4.0-drafts/GT4Facts/index.html>

[HaDe03a] Halepovic, E. and Deters, R. The Costs of Using JXTA. To appear at The Third IEEE International Conference on Peer-to-Peer Computing, Linköping, Sweden, 2003

[HaDe03b] E. Halepovic and R. Deters, JXTA Performance Study. PACRIM'03 Conference, Victoria, BC, Canada, 2003

[HDT04] E. Halepovic, R. Deters, and B. Traversat, Performance Evaluation of JXTA Rendezvous. DOA 2004 Conference, Agia Napa, Cyprus, 2004

[HoBo02] C. Hoile and E. Bonsma, *Towards a minimal hosting specification for open agent systems : the lessons of IP*, 1st International Workshop on "Challenges in Open Agent Systems", AAMAS2002, July 2002, Bologna, Italy.

[J2SE05] <http://java.sun.com/>

[JadA04] Jade programmers guide, <http://jade.tilab.com/doc/programmersguide.pdf>

[JADE05] <http://jade.tilab.com/>

[Jadx04] The Jadex project at Distributed Systems and Information Systems Hamburg University. (See <http://vsiis-www.informatik.uni-hamburg.de/projects/jadex/>)

[JaRp05] <http://java.sun.com/xml/jaxrpc/index.jsp>

[JNDI05] <http://java.sun.com/products/jndi/>

[JUDD05] <http://ws.apache.org/juddi/>

[Juri04] M. Juric et al. Java RMI, RMI Tunneling and Web Services Comparison and Performance Analysis. *ACM SIGPLAN Notices*, 39(5), May 2004.

[JXTA05] <http://www.jxta.org/>

[LFGL01] G. von Laszewski, I. Foster, J. Gawor, P. Lane (2001), "A Java Commodity Grid Toolkit", *Concurrency: Practice and Experience*, 13

[Li01] Li, S. 2001. *JXTA: Peer-to-Peer Computing with Java*, WROX: Birmingham

[LRCN03] M. Lyell, L. Rosen, M. Casagni-Simkins, D. Norris, On Software Agents and Web Services: Usage and Design Concepts and issues, International Joint Conference on Autonomous Agents and Multiagent Systems, Workshop on Web Services and Agent-based software engineering - Melbourne (Australia) 2003

[OGSA05] <http://www.globus.org/ogsa/>

[OGSI05] <http://www.globus.org/ogsa/>

[P2PS05] <http://www.trianacode.org/p2ps/download/index.html>

[PaJe05] Paurobally, S. and Jennings, N. R. (2005) Protocol engineering for web services conversations, *Int J. Engineering Applications of Artificial Intelligence* 18(2).

[Past05] <http://freepastry.rice.edu/FreePastry/README-1.3.2.html>

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, A Scalable Content-Addressable Network, Proceedings of ACM SIGCOMM 2001

[RMI05] <http://java.sun.com/products/jdk/rmi/>

[SAP05] <http://www.ist-safeguard.org>

[Sing03] <http://www-106.ibm.com/developerworks/java/library/j-jxta2/>

[SOAP05] <http://www.w3.org/TR/soap/>

[SSB04] Saez, G., Sliva, A.L., Blake, M.B. "Web Services-Based Data Management: Evaluating the Performance of UDDI Registries" Proceedings of the International Conference on Web Services (ICWS 2004), San Diego, CA, July 2004

[StMk04] Stratos Idreos, Manolis Koubarakis: P2P-DIET: One-Time and Continuous Queries in Super-Peer Networks. EDBT 2004: 851-853

[Tomc05] <http://jakarta.apache.org/tomcat/>

[Trav03] Bernard Traversat et al (Project JXTA, May 2003), Project JXTA 2.0 Super-Peer Virtual Network, white paper.

[Tryl05] <http://www.tryllian.com/>

[VQC02] G. Vitaglione, F. Quarta, E. Cortese, *Scalability and Performance of JADE Message Transport System*. Presented at AAMAS Workshop on AgentCities, Bologna. July the 16th, 2002.

[W3c04] Web Services Architecture W3C Working Group Note, D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, 11 February 2004 (See <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.)

[Wang05] Ian Wang. Peer-to-Peer Simplified), in *Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies*. To be published, 2005.

[WeSe05] <http://www.w3.org/2002/ws/>

[Wrig04] Todd Wright BBN Technologies, Naming Services in Multi-Agent Systems: A Design for Agent-Based White Pages, Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3, New York(USA) 2004

[WSDL05] <http://www.w3.org/TR/wsdl>

[WSFR05] <http://www.globus.org/wsrf/>

Annex E – Application framework – concepts and mechanisms

This annex contains parts of the source code of the first prototype of the Catallactic enabled Cat-COVITE application, as well as examples of the agreement templates and offers proposed.

E.1 WSDL file

```
<definitions name="Search"
  targetNamespace="http://tutorial.catnets.org/search"
  xmlns:tns="http://tutorial.catnets.org/search"
  xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
  xmlns:wslw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
  xmlns:wslpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
  xmlns:gtwsdl1="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.wsdl"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
  xmlns:wntw="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
  xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
01.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-01.wsdl"
    location="../wsrf/faults/WS-BaseFaults.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
    location="../wsrf/lifetime/WS-ResourceLifetime.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    location="../wsrf/properties/WS-ResourceProperties.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
    location="../wsrf/notification/WS-BaseN.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.wsdl"
    location="../wsrf/servicegroup/WS-ServiceGroup.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
    location="../wsrf/notification/WS-BaseN.wsdl" />

<types>
  <schema
    targetNamespace="http://tutorial.catnets.org/search"
    xmlns:tns="http://tutorial.catnets.org/search"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <import
      namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
      schemaLocation="../ws/addressing/WS-Addressing.xsd"/>
    <import
      namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-
draft-01.xsd"
      schemaLocation="../wsrf/servicegroup/WS-ServiceGroup.xsd"/>

    <element name="StatusMessage" type="xsd:string"/>
    <element name="CurrentSearch" type="xsd:string"/>
    <element name="SearchResourceProperties">
      <complexType>
        <sequence>
          <element ref="tns:StatusMessage"/>

```

```

        <element ref="tns:CurrentSearch"/>
    </sequence>
</complexType>
</element>

<!--===== Add Create Support to WSDL (see SearchService.java)
=====-->
    <element name="Create">
        <complexType/>
    </element>
<element name="CreateResponse" type="wsa:EndpointReferenceType" />

<!--===== Add Write Support to WSDL (see SearchService.java) =====-->
<element name="Write" type="xsd:string" />
    <element name="WriteResponse">
        <complexType/>
    </element>

    <!-- ===== End =====>
</schema>
</types>
<!-- ===== MESSAGES ===== -->
<message name="CreateRequest">
    <part name="CreateRequest"
        element="tns:Create" />
</message>
<message name="CreateResponse">
    <part name="CreateResponse"
        element="tns:CreateResponse" />
</message>
<message name="WriteRequest">
    <part name="WriteRequest"
        element="tns:Write" />
</message>
<message name="WriteResponse">
    <part name="WriteResponse"
        element="tns:WriteResponse" />
</message>

<!-- ===== END OF MESSAGES ===== -->
<portType name="SearchPortType" wsrp:ResourceProperties="SearchResourceProperties">
    <operation name="GetResourceProperty">
        <input name="GetResourcePropertyRequest"
            message="wsrpw:GetResourcePropertyRequest"
            wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetResourceProperty"/>
        <output name="GetResourcePropertyResponse"
            message="wsrpw:GetResourcePropertyResponse"
            wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties/GetResourcePropertyResponse"/>
        <fault name="InvalidResourcePropertyQNameFault"
            message="wsrpw:InvalidResourcePropertyQNameFault"/>
        <fault name="ResourceUnknownFault"
            message="wsrpw:ResourceUnknownFault"/>
    </operation>

    <operation name="Create">
        <input name="CreateRequest"
            message="tns:CreateRequest" />
        <output name="CreateResponse"

```

```

        message="tns:CreateResponse" />
    </operation>
    <operation name="Write">
        <input name="WriteRequest"
            message="tns:WriteRequest" />

        <output name="WriteResponse"
            message="tns:WriteResponse" />
    </operation>
</portType>
</definitions>

```

E.2 The service source code

```

package org.catnets.tutorial.search;

import java.rmi.RemoteException;
import org.globus.wsrf.ResourceContext;
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.globus.wsrf.ResourceKey;
import org.globus.wsrf.utils.AddressingUtils;

import org.catnets.SupplierWS.*;

/**
 * This is the code for the SearchService (QueryService in Catnets project)
 */
public class SearchService
{
    /** implementation of the 'create' operation, to create a new
    search resource
    */
    public EndpointReferenceType create(Create c) throws RemoteException
    {
        ResourceContext ctx = ResourceContext.getResourceContext();
        SearchHome home = (SearchHome)ctx.getResourceHome();
        ResourceKey key = home.create();

        // form an EPR that points to the search
        EndpointReferenceType epr;

        try
        {
            epr = AddressingUtils.createEndpointReference(ctx, key);
        } catch(Exception e)
        {
            throw new RemoteException("Could not form an EPR to new search: "+ e);
        }
        return epr;
    }

    /** Implementation of the 'write' operation, to write text onto the
    the search.
    */
    public WriteResponse write(String text) throws RemoteException
    {
        // get appropriate resource
    }
}

```

```

Search resource = (Search)ResourceContext.getResourceContext().getResource();

String responseDotNETWS;

try
{
    // create an instance of the desired web service (aka the .NET web service)
    SupplierWSLocator service = new SupplierWSLocator();

    //use the factory method on the service to get the portType stub
    SupplierWSSoap port = service.getSupplierWSSoap();

    // make the actual call
    responseDotNETWS = port.getProductAsString(text);

    // set the text on it
    resource.setStatusMessage(responseDotNETWS);
} catch(Exception e)
{
    throw new RemoteException("Could not write the text to new search: "+ e);
}
return new WriteResponse();
}
}

```

E.3 Agreement Template examples

```

<wsag:Template xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\catnets\Catnets_Query_Agreement_Template_1.xsd">
  <wsag:Name>QueryComplexServiceTemplate</wsag:Name>
  <wsag:Context>
    <wsag:AgreementInitiator>
      <local:Name>NameOfTheInitiator</local:Name>
      <local:Address>AddressOfTheInitiator</local:Address>
    </wsag:AgreementInitiator>
    <wsag:AgreementInitiatorIsServiceConsumer>true</wsag:AgreementInitiatorIsServiceConsumer>
    <wsag:StartingTime>2005-09-01T00:00:00</wsag:StartingTime>
    <wsag:TerminationTime>2005-12-25T00:00:00</wsag:TerminationTime>
    <wsag:TemplateName>QueryServiceTemplate</wsag:TemplateName>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm wsag:Name="BasicServiceType"
wsag:ServiceName="QueryJobExecutionService">
        <job:BasicServiceType>"BasicServiceInstance"</job:BasicServiceType>
      </wsag:ServiceDescriptionTerm>

      <wsag:ServiceDescriptionTerm wsag:Name="NumberOfBasicServiceNodes"
wsag:ServiceName="QueryJobExecutionService">
        <job:NumberOfBasicServiceNodes>1</job:NumberOfBasicServiceNodes>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm wsag:Name="NoOfQueriesInPeriodOfTime"
wsag:ServiceName="QueryJobExecutionService">
        <job:NumberOfQueriesInPeriodOfTime>100
      </job:NumberOfQueriesInPeriodOfTime>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm wsag:Name="PeriodOfUsage"
wsag:ServiceName="QueryJobExecutionService">
        <job:PeriodOfUsage>8
      </job:PeriodOfUsage>
      </wsag:ServiceDescriptionTerm>
    </wsag:All>
  </wsag:Terms>
</wsag:Template>

```

```

        <wsag:ServiceDescriptionTerm wsag:Name="PayForService"
wsag:ServiceName="QueryJobExecutionService">
            <job:PayForService>30
        </job:PayForService>
    </wsag:ServiceDescriptionTerm>
    <wsag:ServiceDescriptionTerm wsag:"DBType"
wsag:ServiceName="QueryJobExecutionService">
        <job:DBType>Architectural/Engineering/Construction
    </job:DBType>
        <job:ResponseTimePerRequest>10
    </job:ResponseTimePerRequest>
    </wsag:ServiceDescriptionTerm>
    <wsag:ServiceProperties wsag:ServiceName="QueryJobExecutionService">
        <wsag:VariableSet>
            <wsag:Variable wsag:Name="startTime"
wsag:Metric="job:startTime">
                <wsag:Location/>
            </wsag:Variable>
        </wsag:VariableSet>
        <wsag:VariableSet>
            <wsag:Variable wsag:Name="endTime"
wsag:Metric="job:endTime">
                <wsag:Location/>
            </wsag:Variable>
        </wsag:VariableSet>
    </wsag:ServiceProperties>
    <wsag:GuaranteeTerm wsag:Name="MinStartTime">
        <wsag:ServiceScope>
            <wsag:ServiceName>QueryJobExecutionService</wsag:ServiceName>
        </wsag:ServiceScope>
        <wsag:ServiceLevelObjective>startTime IS_NOT_AFTER 2005-09-
25T08:00:00</wsag:ServiceLevelObjective>
        <wsag:BusinessValueList>
            <wsag:Penalty>
                <wsag:AssessmentInterval>
                    <wsag:Count>1</wsag:Count>
                </wsag:AssessmentInterval>
                <wsag:ValueExpression>5</wsag:ValueExpression>
            </wsag:Penalty>
        </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
    <wsag:GuaranteeTerm wsag:Name="MaxEndTime">
        <wsag:ServiceScope>
            <wsag:ServiceName>QueryJobExecutionService</wsag:ServiceName>
        </wsag:ServiceScope>
        <wsag:ServiceLevelObjective>endTime IS_BEFORE 2005-09-
25T16:00:00</wsag:ServiceLevelObjective>
        <wsag:BusinessValueList>
            <wsag:Penalty>
                <wsag:AssessmentInterval>
                    <wsag:Count>1</wsag:Count>
                </wsag:AssessmentInterval>
                <wsag:ValueExpression>5</wsag:ValueExpression>
            </wsag:Penalty>
        </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
</wsag:All>
</wsag:Terms>
<wsag:CreationConstraints>
    <wsag:Item wsag:Name="BasicServiceType">

```

```

<wsag:Location>/wsag:Template/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm/job:basicService
Type</wsag:Location>
    <!-- for each domain-specific service description <job:basicServiceType>,
    constrain the value of that element (i.e. reduce list of possible
    BasicServiceTypes)
    -->
    </wsag:Item>
    <wsag:Item wsag:Name="NumberOfBasicServiceNodes">
    <wsag:Location>/wsag:Template/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm/job:numberOfB
asicServiceNodes</wsag:Location>
        <!-- <job: numberOfBasicServiceNodes> is allowed, but must be within the range
        <xs:minInclusive xs:value="1"/>
        <xs:maxInclusive xs:value="5"/>
        </wsag:Item>
        <wsag:Item wsag:Name="NoOfQueriesInPeriodOfTime">
        <wsag:Location>/wsag:Template/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm/job:numberOfQ
ueriesInPeriodOfTime</wsag:Location>
            <!--<job: numberOfQueriesInPeriodOfTime> is allowed; but must not be
            greater than 150 queries/minute
            -->
            <xs:maxInclusive xs:value="150"/>
            </wsag:Item>
            <wsag:Item wsag:Name="PeriodOfUsage">
            <wsag:Location>/wsag:Template/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm/job:periodOfUs
age</wsag:Location>
                <!--<job: periodOfUsage> is allowed; but must be within a range (number of hours of
                usage the service)
                -->
                <xs:minInclusive xs:value="4"/>
                <xs:maxInclusive xs:value="10"/>
                </wsag:Item>
                <wsag:Item wsag:Name="PayForService">
                <wsag:Location>/wsag:Template/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm/job:payForServi
ce</wsag:Location>
                    <!--<job: payForService> is allowed; no constrain on its value (amount of money pay
                    for dedicated the period of service)
                    -->
                    </wsag:Item>
                    <wsag:Item wsag:Name="DBType">
                    <wsag:Location>/wsag:Template/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm/job:dbType</ws
ag:Location>
                        <!--<job: dbType> is allowed; no constrain on its value (the type of database where
                        the query job is executed)
                        -->
                        </wsag:Item>
                        <wsag:Item wsag:Name="ResponseTimePerQuery">
                        <wsag:Location>/wsag:Template/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm/job:responseTi
mePerQuery</wsag:Location>
                            <!--<job: responseTimePerQuery> is allowed; but must be within a range (number of
                            seconds per response)
                            -->
                            <xs:minInclusive xs:value="2"/>
                            <xs:maxInclusive xs:value="10"/>
                            </wsag:Item>
                        </wsag:CreationConstraints>
                    </wsag:Template>

```

An agreement template lite is presented below:

```

<?xml version="1.0" encoding="UTF-8"?>
<AgreementTemplateLite>

```



```

<Name>QueryComplexService</Name>
<Context>
  <AgreementInitiator>Your Name
  </AgreementInitiator>
  <StartingTime>2005-09-01T00:00:00
  </StartingTime>
  <TerminationTime>2005-10-01T00:00:00
  </TerminationTime>
</Context>
<Terms>
  <BasicServiceType>QueryJobExecutionService
  </BasicServiceType>
  <NumberOfBasicServiceNodes>1 to 10
  </NumberOfBasicServiceNodes>
  <BasicServiceConstraints>
    <DBType>Architectural/Engineering/Construction
    </DBType>
    <ResponseTimePerRequest>10
    </ResponseTimePerRequest>
  </BasicServiceConstraints>
  <PayForService>100
  </PayForService>
</Terms>
</AgreementTemplateLite>

```

E.4 Agreement Offer examples

```

<wsag:Agreement xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Catnets_Query_Agreement_Offer_1.xsd">
  <wsag:Name>QueryComplexServiceOffer</wsag:Name>
  <wsag:Context/>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm wsag:Name="BasicServiceType"
wsag:ServiceName="QueryJobExecutionService">
        <job:BasicServiceType>"BasicServiceInstance"</job:BasicServiceType>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm wsag:Name="NumberOfBasicServiceNodes"
wsag:ServiceName="QueryJobExecutionService">
        <job:NumberOfBasicServiceNodes>1</job:NumberOfBasicServiceNodes>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm wsag:Name="NoOfQueriesInPeriodOfTime"
wsag:ServiceName="QueryJobExecutionService">
        <job:NumberOfQueriesInPeriodOfTime>100
      </job:NumberOfQueriesInPeriodOfTime>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm wsag:Name="PeriodOfUsage"
wsag:ServiceName="QueryJobExecutionService">
        <job:PeriodOfUsage>8
      </job:PeriodOfUsage>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm wsag:Name="PayForService"
wsag:ServiceName="QueryJobExecutionService">
        <job:PayForService>30
      </job:PayForService>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm wsag:"DBType"
wsag:ServiceName="QueryJobExecutionService">
        <job:DBType>Architectural/Engineering/Construction
      </job:DBType>
        <job:ResponseTimePerRequest>10

```

```

        </job:ResponseTimePerRequest>
    </wsag:ServiceDescriptionTerm>
    <wsag:ServiceProperties wsag:ServiceName="QueryJobExecutionService">
        <wsag:VariableSet>
            <wsag:Variable wsag:Name="startTime"
wsag:Metric="job:startTime">
                <wsag:Location/>
            </wsag:Variable>
        </wsag:VariableSet>
        <wsag:VariableSet>
            <wsag:Variable wsag:Name="endTime"
wsag:Metric="job:endTime">
                <wsag:Location/>
            </wsag:Variable>
        </wsag:VariableSet>
    </wsag:ServiceProperties>
    <wsag:GuaranteeTerm wsag:Name="MinStartTime">
        <wsag:ServiceScope>
    <wsag:ServiceName>QueryJobExecutionService</wsag:ServiceName>
        </wsag:ServiceScope>
        <wsag:ServiceLevelObjective>startTime IS_NOT_AFTER 2005-09-
25T08:00:00</wsag:ServiceLevelObjective>
        <wsag:BusinessValueList>
            <wsag:Penalty>
                <wsag:AssessmentInterval>
                    <wsag:Count>1</wsag:Count>
                </wsag:AssessmentInterval>
                <wsag:ValueExpression>5</wsag:ValueExpression>
            </wsag:Penalty>
        </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
    <wsag:GuaranteeTerm wsag:Name="MaxEndTime">
        <wsag:ServiceScope>
    <wsag:ServiceName>QueryJobExecutionService</wsag:ServiceName>
        </wsag:ServiceScope>
        <wsag:ServiceLevelObjective>endTime IS_BEFORE 2005-09-
25T16:00:00</wsag:ServiceLevelObjective>
        <wsag:BusinessValueList>
            <wsag:Penalty>
                <wsag:AssessmentInterval>
                    <wsag:Count>1</wsag:Count>
                </wsag:AssessmentInterval>
                <wsag:ValueExpression>5</wsag:ValueExpression>
            </wsag:Penalty>
        </wsag:BusinessValueList>
    </wsag:GuaranteeTerm>
    </wsag:All>
</wsag:Terms>
</wsag:Agreement>

```

An agreement offer lite is presented below:

```

<?xml version="1.0" encoding="UTF-8"?>
<AgreementOfferLite>
    <Name>QueryComplexService</Name>
    <Context>
        <AgreementInitiator>Your Name
        </AgreementInitiator>
        <StartingTime>2005-09-01T00:00:00
        </StartingTime>
        <TerminationTime>2005-10-01T00:00:00
        </TerminationTime>
    </Context>
</AgreementOfferLite>

```

```
</Context>
<Terms>
  <BasicServiceType>QueryJobExecutionService
</BasicServiceType>
  <NumberOfBasicServiceNodes>1
</NumberOfBasicServiceNodes>
  <BasicServiceConstraints>
    <DBType>Architectural/Engineering/Construction
  </DBType>
    <ResponseTimePerRequest>10
  </ResponseTimePerRequest>
  </BasicServiceConstraints>
  <PayForService>100
</PayForService>
</Terms>
</AgreementOfferLite>
```

In this document the Cat-COVITE Application for use in the CATNETS Project is introduced and motivated. Furthermore an introduction to the catallactic middleware and Web Services Agreement (WS-Agreement) concepts is given as a basis for the future work. Requirements for the application of Cat-COVITE with in catallactic systems are analysed. Finally the integration of the Cat-COVITE application and the catallactic middleware is described.