D3.1

UPC CATNETS WP3

Abstract

CATNETS EU IST-FP6-003769 Project Deliverable D1.1

This deliverable describes the work done in task 3.1, “Middleware analysis: Analysis of current middleware used in peer-to-peer and grid implementations for enhancement by catallactic mechanisms” from work package 3, “Middleware Implementation”. The document is divided in four parts: The introduction with application scenarios and middleware requirements, Catnets middleware architecture, evaluation of existing middleware toolkits, and conclusions

Document Id. CATNETS/2005/D3.1/v1.0
Project CATNETS EU IST-FP6-003769
Date 03-03-2005
Distribution Resticted

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 (BWLVII)
95440 Bayreuth
Germany
Tel: +49 921 55-2807, Fax: +49 921 55-2816
Contactperson: 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
Contactperson: 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
Contactperson: 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
Contactperson: Mauro Gallegati
E-mail: gallegati@dea.unian.it

University of Cardiff
School of Computer Science and the Welsh eScience Centre
University of Caradiff, Wales
Cardiff CF24 3AA, UK
United Kingdom
Tel: +44 (0)2920 875542, Fax: +44 (0)2920 874598
Contactperson: 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
Contactperson: Floriano Zini
E-mail: zini@itc.it

Changes

Version	***Date***	***Author***	***Changes***

1 Introduction

This deliverable describes the work done in task 3.1, "*Middleware analysis*: Analysis of current middleware used in peer-to-peer and grid implementations for enhancement by catallactic mechanisms" from work package 3, "*Middleware Implementation"*. The document is divided in four parts: The introduction with application scenarios and middleware requirements, Catnets middleware architecture, evaluation of existing middleware toolkits, and conclusions.

1.1 Middleware toolkit selection objectives

We recall from the Catnets project proposal [Catn04]:

> ***"The main objective of the project is to provide a significant statement about using 'free market' (catallactic) mechanisms in application layer networks. Our conclusions will concern their applicability and implementation possibilities in Grid/P2P middleware, providing a prototype, and performance results and further insight in their behaviour"***

The objective of this deliverable D3.1 is the selection of middleware toolkit as the basis for the Catnets prototype implementation. We address various Application Layer Network (ALN) types, like Grid, Peer-to-Peer (P2P) and Content Distribution Networks (CDN). This document should guide the selection of the tools for the design, implementation and evaluation of middleware using Catallaxy[ERA+03] in real ALN scenarios and given a concrete application. This document should be helpful for the design and implementation of a "proof of concept" prototype.

1.2 Adopted approach

> ***"To achieve our objective the project faces the challenge to combine contributions both from computer science and economics to address the features of coming Grid and P2P applications and infrastructures. Catallaxy has been proposed as a model to describe the behaviour of complex and large scale real world economy. However, its results have not yet been transferred to coordinate large and dynamic computer networks" [Catn04]***

Considering this issue of transference of research results, we have studied which are the new elements that real Grids and P2P applications bring into the catallactic model, and secondly which is the impact on the previous model used in the Catnet assessment project [Catn03]. We have found that composing a prototype for deploying Catallactic agents into real Grid and P2P scenarios is far more complex than just porting the simulation to some convenient middleware toolkit.

The next sections of this document will describe ongoing work in the identification and exposition of requirements in applications, the design of an architecture that meets the found requirements, and the analysis and testing of existing middleware tools that can be used to implement such architecture in Catnets.

This deliverable is organized as follows: Section 1.3 introduces the problematic of Catalaxy applied to information systems. Its purpose is to introduce the vast problematic of Catalaxy applied to ALNs. Its function is to contextualise the ideas described in other sections of the document. Section 1.4 introduces Catallactic middleware requirements.

Section 2 presents the proposed middleware architecture. Section 3 evaluates six different candidate middleware toolkits regarding the identified functional and non-functional requirements of Catnets. Section 4 exposes the conclusions and indicates next steps.

It is important to note the "layered" structure of this deliverable. The architecture introduced in section 2 offers a solution to the Catnets middleware development from the software engineering point of view. The whole picture is presented, but in the context of this deliverable the focus is on the low level layers of the architecture. Section 2 allows to situate the middleware toolkits analysis and evaluation described in section 3, and to understand the way it can be used in conjunction with the architecture to select middleware toolkit for the Catnets prototype implementation, given an ALN scenario.

1.3 Application Scenarios

Application scenarios of interest in the Catnets project are centred in the concept of "Application Layer Networks" (ALN), which includes generic application classes such as Content Distribution Networks (CDN), Peer-to-Peer Networks (P2P) or Grid. An ALN is a collective entity that provides a certain service using a composition of service elements that cooperate among them, organized as an overlay network.

The rationale for selecting scenarios are those where there may be potential benefit from using a decentralized economic approach, as it was identified in the previous Catnet assessment project. The following are some environments where Catallaxy has potential for success:

- Dynamic: changing environments and the need for adaptation to changes is one of the potential areas where Catallaxy can have a competitive advantage.
- Diverse: requests may have different priorities and responses should be assigned according to them.
- Large: with such number of elements that locality is required to scale.

These environments have the following features:

- Partial knowledge: it is not possible to know everything on time because of its high cost. This can be caused by scale issues such as a large number of elements, number of messages, or communication latency (information arrives too late), which requires locality (that leads to scalability).
- Complex: many parameters must be taken into account, many messages. Learning mechanisms are necessary to self-adjust or adapt to changes, and optimal solutions are not easily computable.
- Evolutionary: open to changes which cannot be take into account in the initial set-up, and able to learn and decide with limited information: neighbours, few parameters that are summarized as a single price value, few historic data, etc.

Taking into account these environments and characteristics, we have selected a few applications representing of certain ALN classes to explore how these applications could be modified to work under the Catallaxy model.

1.3.1 Sample applications

The criteria for selection of these sample applications has been the following: for all potential applications, we have selected applications which are exemplary of their application area, which have a certain degree of popularity, and the source code is available for inspection, modification and experimentation. Ideally, the candidate applications should have been evaluated and characterized in public papers, so we could then compare our results with an external baseline evaluation.

In the next subsections we present three different application scenarios: Grids (Planetlab), CDN (Coral) and P2P file sharing (Bitorrent).

1.3.1.1 Applications for the Grid scenario

In the context of ALN for distributed computing using computing resources across administrative boundaries (Grid computing), we have identified two unique initiatives offering an open infrastructure for the deployment of services and the use of computational resources in the academic or industrial environment. Both are unique in terms of size, availability and relative maturity. They are Planetlab and Globus.

PlanetLab, as described in [BBC+04], is a geographically distributed overlay network designed to support the deployment and evaluation of planetary-scale network services. Two high-level goals shape its design. First, to enable a large research community to share the infrastructure, PlanetLab provides distributed virtualization, whereby each service runs in an isolated slice of PlanetLab's global resources. Second, to support competition among multiple network services, PlanetLab decouples the operating system running on each node from the network-wide services that define PlanetLab, a principle referred to as unbundled management.

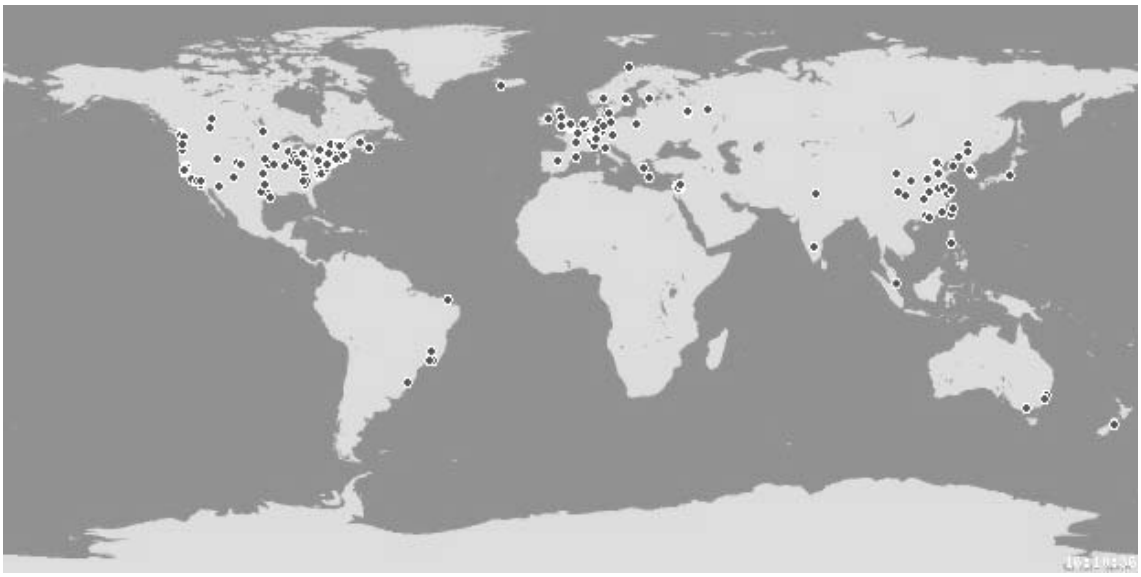


Figure 1.1 Planetlab map of member organizations (as of 2/2005)

PlanetLab currently includes over 500 machines spanning 250 sites (i.e. organizations) and more than 20 countries (figure 1.1). It supports more than 500 research projects which focus on a wide range of services, including file sharing and network embedded storage, content distribution networks, routing and multicast overlays, QoS overlays, scalable object location services, anomaly detection mechanisms, and network measurement tools.

The PlanetLab middleware and API is open and extensible. Distributed applications can use the services offered by the virtualized operating system in each node (currently the Linux API) plus the XML-RPC interface offered by the PlanetLab Central (PLC) administration

The service-resource cycle in PlanetLab is as follows:

- In every node, the node manager is in charge of creating and allocating resources to vservers (virtual machines), and the resource monitor is in charge of tracking node's availability of resources and informing the central agent about available resources.
- The agent tracks nodes' free resources, which are advertised to resource brokers and offered as tickets to services interested in acquiring and using resources. This agent is part of PLC (Planet-Lab Central), a centrally-controlled brokerage service that can be decentralized using a delegation mechanism.
- In every service, the resource broker obtains tickets from agents on behalf of service managers, which are in charge of redeeming tickets with node managers to acquire resources, and if resources can be acquired, start the service in that node.

In terms of the Catnets model, people or processes interested in using a given service have the role of *Client*. They should look for and select a service instance (a *Service Copy* in the *Service Market*). All nodes (represented by node managers and resource monitors; acting as *Resources*), all service instances (represented by resource brokers and service managers; acting as *Service Copies*), both mediated by the central Agent (PLC) belong to the *Resource market*.

The Globus toolkit [Glob05] is the reference implementation of the standard Grid protocols and APIs that the Global Grid Forum (GGF) is defining for different aspects of distributed computing, such as security, resource management, data management, and information discovery. The status of this implementation is now on a public version 3.0 release and working on version 4.0. The Globus middleware has been adopted by most of the Grid projects world-wide.

In comparison with the Globus Grid implementation which offers a higher level homogeneous API, Planetlab offers a less coupled and simpler API based on the idea of virtualization. While the grid offers an ample collection of middleware services unified in a single architecture (as exemplified by the Globus, Planetlab offers an API for the basic service for creating slices, and associating people and nodes to them. Slices appear to users as a set of virtual Linux machines (i.e. offering a multiple Linux API instead of a higher level and abstract API: virtualization in contrast to abstraction). Additionally, there may be competing services providing additional functionality that also run using the Planetlab infrastructure (multiplicity in contrast to homogeneity). There is another difference to emphasize: While the grid is primarily interesting in gluing together a modest number of high-performance computing resources connected by high performance networks, Planetlab is focused on scaling less bandwidth and CPU intensive applications offering innovative services across a wider collection of nodes [PACR02]. Finally, both worlds can be combined: There are pilot experiments where Globus based applications are run on top of PlanetLab (in a slice, on several nodes or slivers).

1.3.1.2 Applications for Content Distribution

In the context of content distribution, the selection criteria applies to two academic CDN which by coincidence both run on the PlanetLab infrastructure: Coral [FFM04] and CoDeeN [PWP+03]. CoDeeN is a proxy based CDN with some restrictions and limitations, and in contrast Coral provides the typical service that a CDN does with some very interesting properties, and focusing on redirecting clients requests to the "best" copy in terms of load, locality, proximity, offloading work from web origin servers.

Coral CDN is a decentralized, self-organizing, peer-to-peer web-content distribution network (use illustration in figure 1.2). Coral CDN leverages the aggregate bandwidth of volunteers (typically PlanetLab slivers) running the software to absorb and dissipate most of the traffic of web sites using the system. In doing so, CoralCDN replicates content in proportion to the content's popularity, regardless of the publisher's resources, in effect democratizing content publication [FFM04].

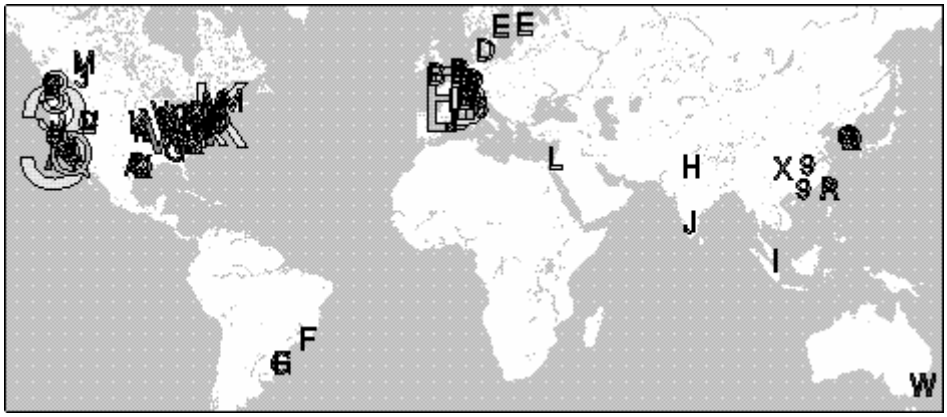


Figure 1.2. Coral's deployment and clusters based on network round-trip-time (letter identifies cluster).

To use Coral CDN, a content publisher —or someone posting a link to a high-traffic portal— simply appends ".nyud.net:8090" to the hostname in a URL. Through DNS redirection, oblivious clients with unmodified web browsers are transparently redirected to nearby Coral web caches. These caches cooperate to transfer data from nearby peers whenever possible, minimizing both the load on the origin web server and the end-to-end latency experienced by browsers.

This requires two mechanisms: finding a close peer, finding a close copy of the requested object. The first is achieved by mapping Coral servers and clients into clusters based on latency. The second is done using a locality-aware request routing algorithm or indexing abstraction (also know as a Distributed Sloppy Hash Table or DSHT). Every Coral peer is running three elements: a DNS server, a HTTP proxy and a DSHT element.

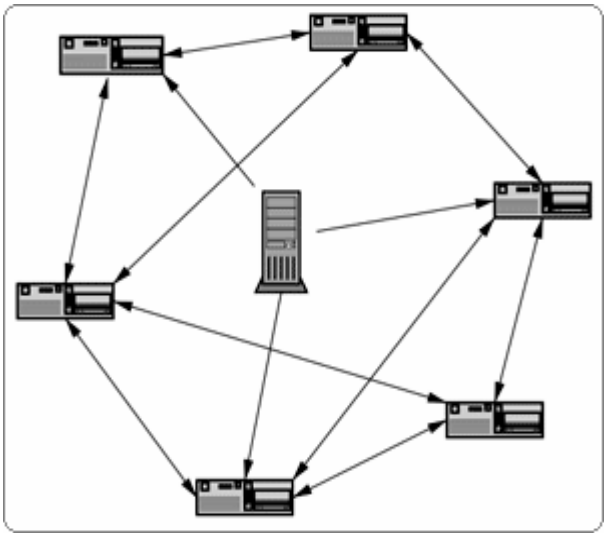
The Coral CDN is implemented on top of a very simple middleware based on RPC over UDP, structured in terms of events and callbacks, with a module for clustering nodes, mapping client locations, routing requests by proximity (Coral DSHT), and modified DNS and HTTP proxy servers.

In terms of the Catnets model, people interested in downloading a file, running an unmodified web browser (or one with a Coral plug-in to "coralize" URLs) has the role of a *Client*. They request a coralized URL, thus going to a Coral DNS server where a response, the IP address of a close-by Coral proxy will be selected among many of

them, based on the location of the client (this is the *Service market* and the Coral http proxy has the role of *Service Copy*). The client web browser will contact the http proxy with the given IP address. Then the proxy will look for the requested file in its own store or it will look for a close copy of the file in other peers using the Coral DSHT routing algorithm. Proxies belong to the *Resource market*, the election in the market is determined by the DSHT algorithm looking for a close copy of a file, and proxies are acting as *Resources*.

1.3.1.3 Applications for peer-to-peer networks

In the context of peer-to-peer networks, we have selected a P2P protocol which has a clearly specified protocol, that is popular enough, and that is used for clearly useful and legal purposes (some other P2P networks are almost only used for sharing copyrighted content). This protocol is BitTorrent [Bitt05].



With BitTorrent [Cohe03], when multiple people are downloading the same file at the same time, they are also uploading pieces of the file to each other. This redistributes the cost of upload to downloaders, thus making hosting a file with a potentially unlimited number of downloaders affordable.

Following [IUB+04], a torrent consists of a central component, called tracker and all the currently active peers. BitTorrent distinguishes between two kinds of peers depending on their download status: clients that have already a complete copy of the file and continue to serve other peers are called seeds; clients that are still downloading the file are called leechers. The tracker is the only centralized component of the system. The tracker is not involved in the actual distribution of the file; instead, it keeps meta-information about the peers that are currently active and acts as a rendez-vous point for all the clients of the torrent.

A user joins an existing torrent by downloading a torrent file (usually from a Web server), which contains the IP address of the tracker. Generic or specialized web search engines usually lead to pages where a file can be downloaded from one or several trackers. The user has to select one torrent file (and thus the tracker) to start downloading the file which will let him connect to the tracker and an initial seed with a complete copy of the file. In case of multiple trackers available for the same object, statistics about every tracker are published to help the visitor choose the right tracker. To update the tracker's global view of the system, active clients periodically (every 30 minutes) report their state to the tracker or when joining or leaving the torrent. Upon

joining the torrent, a new client receives from the tracker a list of active peers to connect to.

Typically, the tracker provides 50 peers chosen at random among active peers while the client seeks to maintain connections to 20−40 peers. If ever a client fails to maintain at least 20 connections, it recontacts the tracker to obtain additional peers. The set of peers to which a client is connected is called its peer set.

The clients involved in a torrent cooperate to replicate the file among each other using swarming techniques: the file is broken into equal size chunks (typically 256kB each) and the clients in a peer set exchange chunks with one another. The swarming technique allows the implementation of parallel download where different chunks are simultaneously downloaded from different clients. Each time a client obtains a new chunk, it informs all the peers it is connected with. Interactions between clients are primarily guided by two principles. First, a peer preferentially sends data to peers that reciprocally sent data to him. This "tit-for-tat" strategy is used to encourage cooperation and ban "free-riding". Second, a peer limits the number of peers being served simultaneously to 4 peers and continuously looks for the 4 best downloaders (in terms of the rate achieved) if it is a seed or the 4 best uploaders if it is a leecher.

In terms of the Catnets model, people interested in downloading a file, running a web browser and a Bittorrent client has the role of *Client*. They look for a torrent file (a tracker) on a search engine and looking at the statistics of several trackers offering the same file they manually select one tracker (the *Service market* and the tracker has the role of *Service Copy*). The tracker joins the client in a swarm of peers exchanging fragments of the file of common interest. All Bittorrent clients in the swarm belong to the *Resource market* and are acting as *Resources*.

Bittorrent has monolithic software architecture, not intended to be extensible in the form required for Catnets. Nevertheless, JXTA [Jaxt05] as perhaps the most known middleware for P2P will be evaluated in section 3.

1.4 Middleware Requirements

We start this requirements introduction recalling what is pointed out in the Catnets proposal:

> ***"Compared to the assessment project, the implementation of a real system with Catallactic mechanisms might need a higher complexity of service discovery and management as in the assessment and will thus carry out resource discovery (and also topology maintenance) based on overlay structures like DHTs, and multiple stateful servers. The performance of the system will be measured to evaluate the contribution of using Catallactic mechanisms"***

We knew that the "real world" was going to introduce new challenges to the applicability of Catallaxy. We knew as well from our previous experience in distributed systems that key issues like decentralization, resource discovery, scalability, fault-tolerance and computational efficiency are very rarely fully addressed by just one middleware tool. We decided to tackle this complexity by composing the requirements of the ideal middleware for Catnets, regardless if some existent middleware toolkit could fully meet these requirements or not. But these requirements can be used as a guide for the architecture and evaluation of existent middleware toolkits.

We decompose the analysis of middleware requirements into 3 different views:

- The Functional View considers the required functionalities to implement a distributed platform.
- The Technical View reflects the technical requirements for the middleware toolkit.
- The Development View reflects the properties concerning the ease of development with the middleware toolkit.

The Functional View defines the functions that the Catallactic middleware should provide to support the implementation of Catallaxy. These functions cover the hosting of the agents, the coordination of their execution in a decentralized way and the integration with the underlying platform. Section 2 of this document explores these functionalities in detail, taking into account the identified scenarios and their architectural requirements.

Catallactic systems may be formed by thousands of agents engaged in numerous negotiations that must be completed in a reasonable time. The Technical View evaluates the general requirements to accomplish technical issues in an efficient way under diverse implementation scenarios. One of the principal requirements is the scalability to huge number of software agents and their coordination in a decentralized way. Closely related is the need for handling massive parallel negotiations among those agents. Another set of technical requirements are related to the architectural flexibility to allow the implementation of the Catallactic middleware into different platforms using diverse middleware toolkits, and the openness of these toolkits concerning the interoperability with external applications.

The Development View evaluates to what extend it is feasible to implement the designed architecture using the selected middleware toolkits within the time and resource constrains of the project. Factors like API complexity, documentation, tutorials and support for development are critical to successfully achieve the planned goals. Also the maturity of the platform and the availability of reference implementations are important aspects concerning the Catallactic middleware.

Both Technical and Development Views requirements are used, developed and evaluated in a set of candidate middleware toolkits in of section 3.

1.5 References

[BCC+04] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, Operating System Support for Planetary-Scale Network Services. *First Symposium on Networked Systems Design and Implementation (NSDI)* (March 2004), 253-266

[Bitt05] http://bittorrent.com/ (2005)

[Catn04] CATNETS project (2004),"Annex I - Description of Work", IST-FP6-003769

[Catn03] CATNET Project (2003), "Catallaxy Simulation Study. Report No. D2", ITS-2001-34030

[Cohe03] Bram Cohen, Incentives Build Robustness in BitTorrent (May 2003)

[ERA+03] T. Eymann, M. Reinickke, 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

[FFM04] Michael J. Freedman, Eric Freudenthal, and David Mazières , Democratizing Content Publication with Coral,. In *Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation* (NSDI '04) San Francisco, CA, March 2004.

[Glob05] http://www.globus.org/ (2005)

[IUB+04] Mikel Izal, Guillaume Urvoy-Keller, Ernst W. Biersack, Pascal A. Felber, Anwar Al Hamra and Luis Garces-Erice Dissecting BitTorrent: Five Months in a Torrent's Lifetime.. In Proceedings of Passive and Active Measurements (PAM), 2004

[Jaxt05] http://www.jaxta.org (2005)

[PACR02] L. Peterson, T. Anderson, D. Culler, and T. Roscoe A Blueprint for Introducing Disruptive Technology into the Internet.. *First Workshop on Hot Topics in Networking (HotNets-I)* (October 2002)

[PWP+03] VS Pai, L Wang, K Park, R Pang, L Peterson , The dark side of the web: An open proxy's view, Proceedings of the 2nd Workshop on Hot Topics in Networking, 2003

2 CATNETS Architecture

"The project will investigate how "Catallactic" mechanisms can be implemented for resource allocation in real application layer networks. This requires a specification of the components that use "Catallactic" mechanisms and how these components can be integrated in the middleware of current Grid and P2P platforms. Their functionality covers resource brokerage, resource discovery, and re-deployment of services. The prototype will be an important result to demonstrate the feasibility and applicability of the approach."[Catn04]

The main goal of WP3 during this stage of the project is the selection of middleware toolkits and options for the integration of Catallactic Middleware mechanisms in current ALN applications. This requires a specification of the components that use Catallactic mechanisms, which cover resource brokerage, resource discovery, and re-deployment of services, and how these components can be integrated into diverse ALN architectures.

In this document we have collected the requirements for the Catallactic Middleware, as well as the potential constraints imposed by diverse deployment scenarios, depending on the ALN architecture. It is clear from the identified heterogeneity that the problem of integration of Catallactic mechanisms and components into current Grid and P2P platforms is far from being a simple one. In fact we face a very complex design problem which needs from a highly structured design approach.

2.1 The Need for an Architecture

"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

2.1.1 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 2.1.

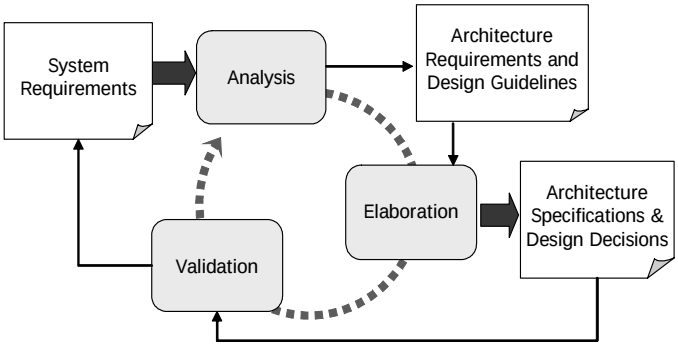


Figure 2.1. The Architecture Design Process

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].

2.1.2 Architecture Specification

We use a model composed of multiple views or perspectives to describe a software architecture ([Kruc95]), see Table 2.1. 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.

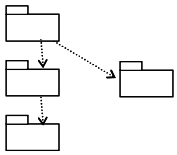
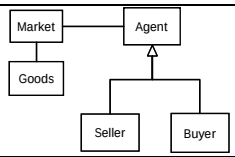
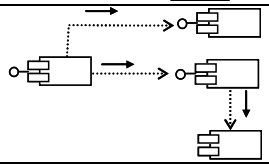
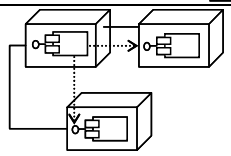
Development	• Specifies how code is organized • Defined in terms of software **packages** and their **dependencies** • Helps to manage the development process	
Logical	• Specifies what the system does • Defined in terms of **logical entities** (classes) and their **relationships** • Helps to understand 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 2.1. 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.

2.2 Architecture Analysis

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 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 2.2.

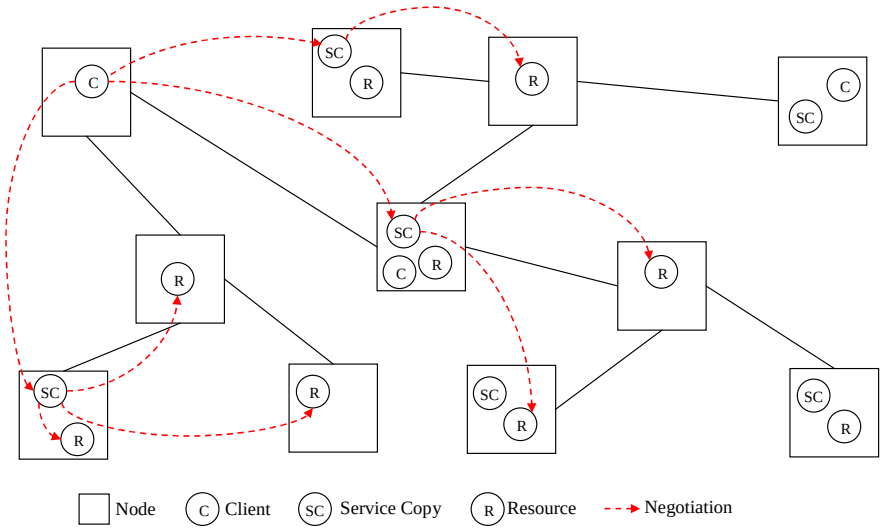


Figure 2.2 CATNETS as a P2P network of agents.

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]).

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.

2.2.1 Architectural 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).

- Variability of ALN characteristics: One of the objectives of CATNETS is to provide a clear statement of the applicability of the Catallactic mechanism to a wide variety of ANLs.

  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 CDNs. Even when all those systems are P2P, they have a great variability in terms of key characteristics

  Table 2.2 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 extend 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 2.2. Characteristics of ANL applications.

It is possible that different policies and algorithms need to be applied to adapt the system to different scenarios. Our vision is a system modular enough to considerer the replacement, following a plug-and-play metaphor, of key components like:

- Overlay network management (peer node, topology maintenance and heuristic learning)
- Resource discovery
- Resource-Request Match making
- Resource allocation
- Request processing/forwarding
- Negotiation protocols
- Message format handling
- Security mechanisms (message encryption, agent authentication)

- 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, as described in section 1.3 of this document) 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.

2.2.2 Architecture Design Strategies

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 relay 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 extend 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.

- Implement middleware using lightweight agents

  Scalability, efficiency and flexibility are the main issues to consider when implementing the Catallactic mechanisms. Handling a potentially high number of negotiations simultaneously requires an efficient implementation of economic agents.

  Traditional agent development approach requires sophisticated platforms where key functions like discovery and ontology handing are tightly integrated and difficult to replace or modify. The resulting implementations usually impose a high overhead in terms of memory footprint and CPU consumption [HWBM02].

  On the other hand, a "bare java code" will impose the heavy burden of implementing mechanisms like thread, memory and communication management, which requires a great deal of low level work to obtain the required efficiency. Also, this approach might lead to tightly coupled code, which will result hard to maintain and extend.

  Therefore, the Catallactic Middleware requires a minimum platform to handle efficiently low level functions like message routing and thread management without imposing an overhead or limiting the flexibility of the solution.

  Using a lightweight agent approach might limit the complexity of the functions and algorithms that could be implemented in the expected time frame of the project and with the estimated resources. Functions such as learning should be

carefully considered and a trade off between feature richness, efficiency and development effort must be established.

- 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 impose 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 patters 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.

- 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 coexists, 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.

  The separation of mechanisms from strategies requires a great deal of design in the basic abstractions and interfaces between those two components. Some design patterns [GHJV95] should be considered:

    - Chain of responsibility (allow more than one object to handle one request, adding functionality dynamically to the process)

- Strategy (encapsulate one object's behaviours as independent, pluggable objects)
- Command (encapsulate requests as objects)
- Decorator (adds functionality to an object by encapsulating it in a richer interface)

- Use APIs with opaque data types

  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 opaque data types in APIs, which can be extended or specialized on each specific implementation. These opaque data types can by language dependent, like abstract parameter objects in Java or can be language neutral, like XML documents.

  Abstracts parameters are generic objects capable of introspection, that is, to inform on runtime what their structure is (for example, the list of attributes) and to access the parameter's content in some "neutral" data type like Strings. These abstracts objects can be specialized to handle the specific parameter's structures and data types needed by each implementation scenario, allowing their optimization.

  XML documents can be considered a kind of abstract parameters, because they offer methods to introspect their structure and access their content. The main difference is that the interface is very standardized in the DOM programming interface and there are many implementations available. Also, they are based on text formats that are platform and language independent.

  Language dependent abstract parameters are more efficient but impose interoperability problems. XML is more interoperable but imposes a considerable runtime overhead in the manipulation of data.

2.3 Architecture Design

In this stage of the project we have focused on the definition of the Development View of the architecture, which will allow us a better organization of the work and a clear separation of design and implementation concerns.

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.

2.3.1 Proposed Architecture

This architecture, shown in figure 2.3, 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 polices 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 Catallactic algorithms.

- **P2P Agent Layer:** Platform that hosts the Catallactic 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 Catallactic middleware. It is (potentially) domain specific.

The model of interaction with the Catallactic 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]

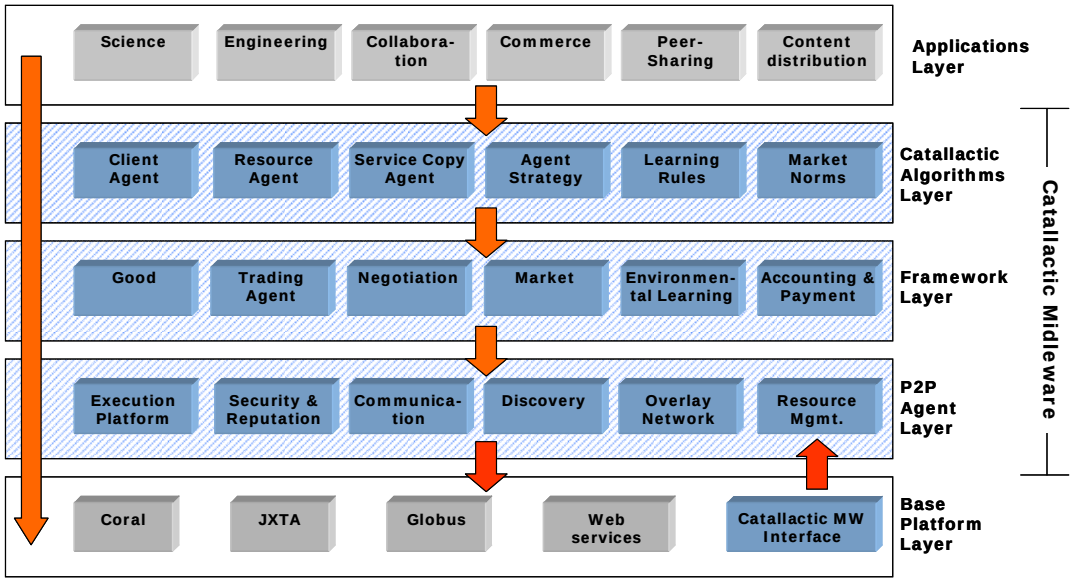


Figure 2.3 Catnets architecture – Development View.

2.3.2 Related Work

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.

PEPITO project [Pepi03a, Pepi03b] address the P2P computing model in general, from the formal perspective, the programming paradigms and the middleware services to develop distributed applications under this model. The architecture proposed by this project is based on the vision of a distributed virtual machine. CATNETS, on the other hand, focus on the development of generic middleware components needed for decentralized resource allocation. It makes no special emphasis on P2P application development, because it should adapt to different ALN architectures. However, we expect to take advantage of the results from PEPITO project during the design stage, in areas like the design of DHTs, fault tolerance and mobility.

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]). However, its strong emphasis on computational intensive grids, (just one of the several ALNs considered by CATNETS) and the hierarchical nature of some of the proposed components, like the Grid Market Directory, diverges from the proposed CATNETS's architecture, which promote a fully decentralized resource allocation mechanism for diverse ALNs.

2.4 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, their 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, as was stated in the section 2.2.2 "Architecture Design Strategies".

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 a pluggable component architecture (BCG04], [BJM0])

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 form 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 it 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 maintain in a single server, a hierarchy of servers, and direct discovery of nodes is not feasible. It is required a distributed mechanisms that provides overlay network construction and maintenance. P2P topology construction mechanisms could be optimised for fast location or fast information dissemination. Finally, in such a huge networks 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). Catallactic 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 has 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 2.4.

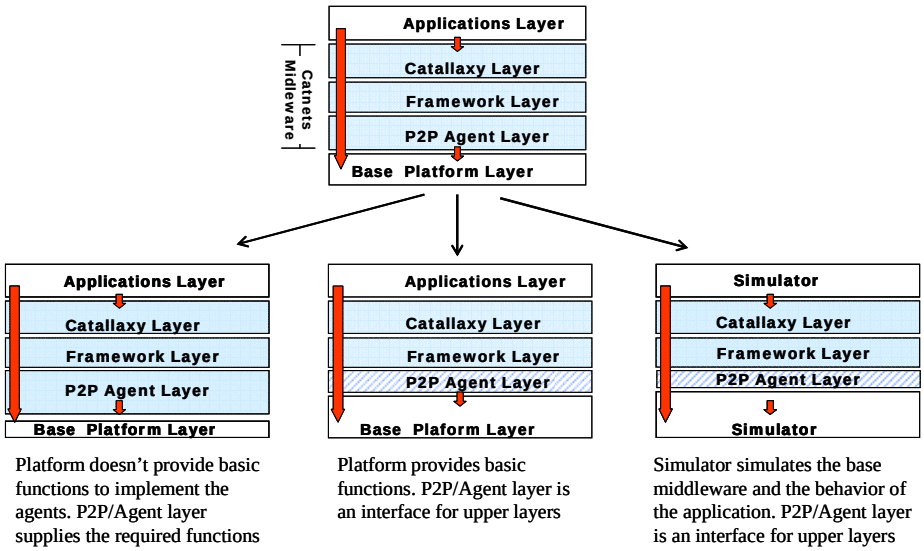


Figure 2.4 - Implementation scenarios for P2P Agent Layer.

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.

The table 2.3 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	● 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	• 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

	management	
Resource Management		
Generic interface to base platform's local resource management	● Resource discovery and query ◖ Resource allocation and deallocation ◖ Resource reservation (future allocation) ◖ Resource monitoring ○ Resource usage accounting ○ Resource related alarms (e.g. malfunctioning)	• Extensible representation of resource properties • Support both direct queries and Publish/Subscribe models for resource information actualization • Support monitoring of frequently changing attributes (e.g CPU workload or network bandwidth)
Communication		
Abstracts the basic communication mechanisms and isolates agents form the complexities of the communication protocols.	● Communication primitives (send, receive, multicast) ○ Logical addressing (global naming) of agents ○ Failure management	• Best effort message delivery • Easy coordination of many parallel conversations by a single agent • Synchronous and asynchronous communication primitives • Support for mobile nodes (location independent addressing) • Support efficient group and system wide multi and any-casts
Overlay Network		
Logical communication topology	● Overlay network construction and maintenance ● Key based routing ● Peer grouping	• Location awareness • Enable both local and system wide information and request dissemination • Support very frequently changing topologies (node membership and communication paths)
Object Discovery		
Localization of catallactic middleware's objects based on attributes	● Resource advertising and location ◖ Resource query and matchmaking ◖ Information cache management ○ Publication/subscription of information changes	• Decentralized; do not requires global repositories • Independent from the semantic of the resource description • Complex queries (multiple resources, multiple attributes, partial matches, range matches) • Scalable to the millions of objects and very frequent updates
Security & Trust		
Assurance interacting agents' identities and rights	● 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	• Compliant with standards • Decentralized/Federated to work in multi-domain environments • Extensible to allow protection of new kind of objects • Auditable

Table 2.3. 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 2.4 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 actives 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 2.4. Performance requirements for P2P Agent Layer.

2.5 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)

[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

[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.

[Bers96] P. Berstain (1996), "Middleware: A Model for Distributed System Services", Communications of the ACM, 39(2):86-98

[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.

[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

[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. Kubiatowicz, 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

[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. Reinickke, 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

[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

[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.

[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

[GHJV95] Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995), “Design Patterns - Elements of Reusable Object-Oriented Software”, Addison-Wesley

[Gon01] L. Gong (2001), “Project JXTA: A Technology Overview”, Sun Microsystems Whitepaper

[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.

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

[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.

[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

[KaBa99] R. Kazman, L. Bass, (1999), “Architecture Based Development”, Software Engineering Institute, Carnegie Mellon Univertity, 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

[Kaz01] R. Kazman (2001), "Software Architecture", in Handbook of Software Engineering and Knowledge Engineering, S-K Chang (ed.), World Scientific Publishing

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

[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

[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. Skarmeas, 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. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja1, J. Pruyne, B. Richard, S. Rollins ,Z. Xu (2002), “Peer-to-Peer Computing”, HP Laboratories Palo Alto, Reseach 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

[P2p02a] P2P Architect Project (2002), “Deliverable D1 - Comprehensive Survey of contemporary P2P technology”, IST-2001-32708

[P2p02b] P2P Architect Project (2002), “Deliverable D4 - P2P applications development process overview”, IST-2001-32708

[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.

[Pepi03a] Pepito Project (2003a), “Deliverable D1.1 –Required foundations for peer-to-peer systems”, IST-2001-33234

[Pepi03b] Pepito Project (2003b), “Deliverable D4.2 - Report on the basic distribution subsystem”, IST-2001-33234

[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

[RoDr01] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.

[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.

[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

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

3 Middleware toolkits evaluation

3.1 Identification of candidate middleware toolkits and evaluation process

We start this section recalling passages from the 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 sections 1.4 and section 2 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 P2P2 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

3.2 Presentation of the candidates

We first introduce the tools and then describe the analysis, testing done and evaluation.

3.2.1 Introduction

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 is implements WSRF, it is also involved in the same unstable process, moving from OGSI to WSFR 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.

3.2.2 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 3.1 presents a basic Web Services Stack. Annex B to this section 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 3.1: 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”)

3.2.3 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. Annex 3.3 shows the more relevant specifications from WSRF.

The soon-to-be-released Globus Toolkit 4 (GT4) [GT405] (figure 3.2), 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.

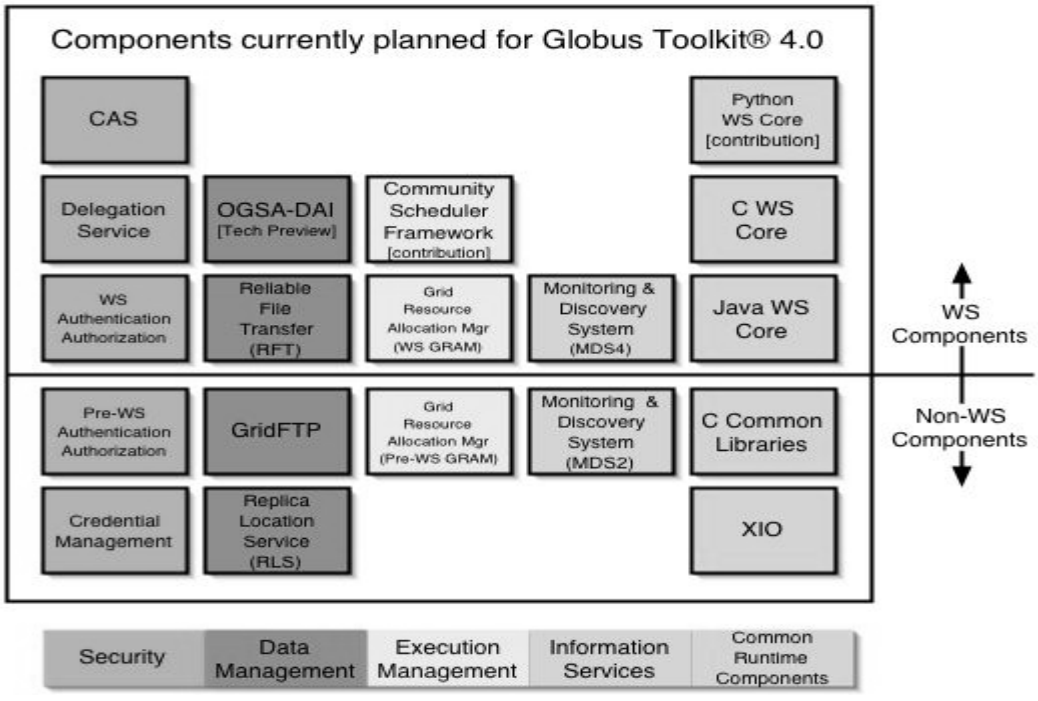


Figure 3.2 GT4 Architecture (from [GT4])

3.2.4 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.

3.2.5 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 3.3 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.

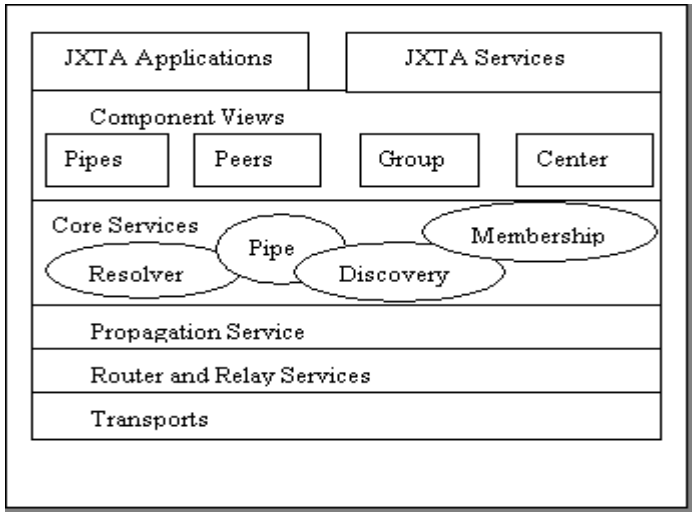


Figure 3.3 JXTA Architecture (from [Li01]).

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.

3.2.6 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 distincted 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.

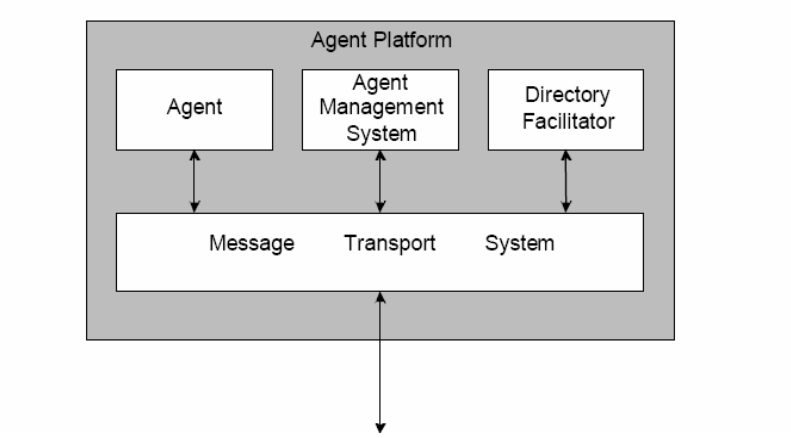


Figure 4.4. FIPA/JADE architecture (from [JadA04])

3.2.7 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 4.5.

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 4.5. DIET Agents Architecture (from [DietA04]).

3.3 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 supported, degree of modularity, persistence, communication functions and security. A detailed evaluation concerning each of the identified functional requirements is presented in Table A.1 of annex A at the end of section 3.

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. Detailed explanation of those requirements is available in section 2.4

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 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 managements 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.

3.4 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 Table A.2 in the Annex A at the end of section 3. 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 x RMI time.

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

Scalability Nº 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 whenneeded.

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 developpers expects from GT4 information Services (MDS4) to be able to scale up to 10000 nodes without performance degradation. JXTA super-peers network with the rendevous 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 rendevous 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 Nº 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 rendevous 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].

3.5 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 Table A.3 in Annex A at the end session 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 3.2.3 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 3.2.5 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 honours 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.

3.6 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 is complementarily, 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

3.7 Conclusions

3.7.1 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 3.6 we graphically summarize the scoring of each middleware toolkit regarding the set of functional, technical and development views properties.

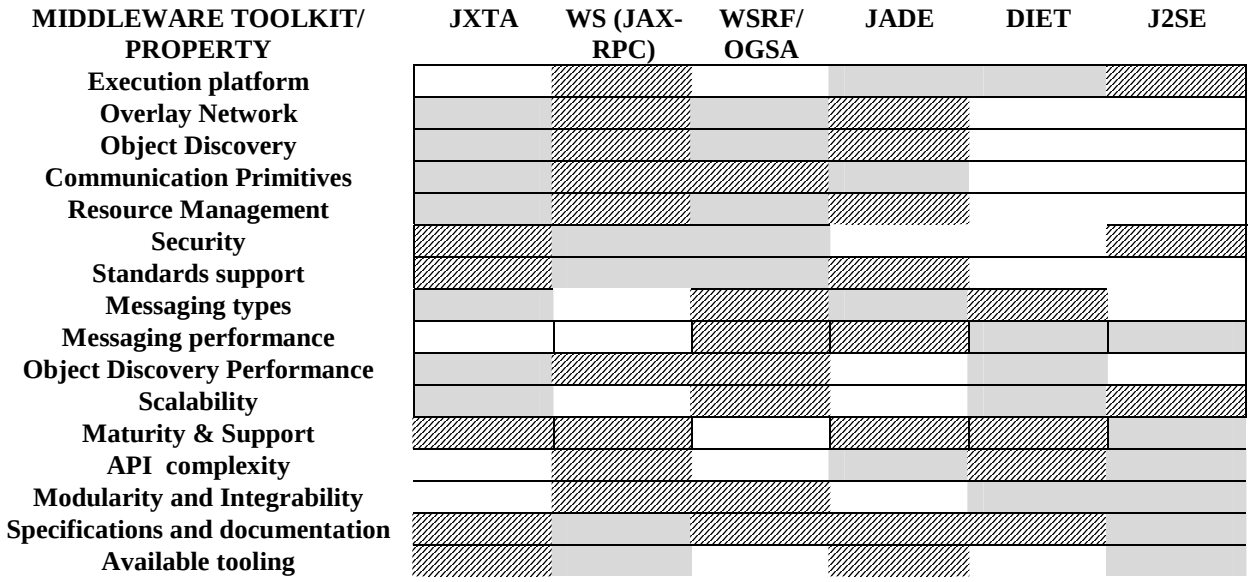


Figure 3.6: Middleware toolkit scoring on the 3 views

3.7.2 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.

3.8 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.cbdiforum.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 Agenticies, 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://vsis-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, Worshop on Web Services and Agent-based software engeenering - 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 3.1- Middleware toolkits Evaluation Tables:

Table 1 - Functional view

Table 2 - Technical view

Table 3 - Developers view

TABLE 1 CATNETS middleware tool / requirement	**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
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

TABLE 1 CATNETS middleware tool / requirement	**JXTA**	**WS (JAX-RPC Imp)**	**WSRF /OGSA**	**JADE**	**DIET**	**J2SE**
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-IIOP 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
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

TABLE 1 CATNETS middleware tool / requirement	**JXTA**	**WS (JAX-RPC Imp)**	**WSRF /OGSA**	**JADE**	**DIET**	**J2SE**
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
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

TABLE 1 CATNETS middleware tool / requirement	**JXTA**	**WS (JAX-RPC Imp)**	**WSRF /OGSA**	**JADE**	**DIET**	**J2SE**
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
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

TABLE 2 TECHNICAL PROPERTY /PLATFORM	**JXTA**	**Web Services (JAX-RPC Imp)**	**WSRF / OGSA**	**JADE**	**DIET**	**J2SE**
- **Standards** - **& Protocols**	- peers, groups - JXTA pipes - XML advertisements	- SOAP - WSDL - UDDI - XML	- Web Services - Index Service - GridFTP - Grid Security	- FIPA ACL - AMS & DF - Behaviours - Interaction - Protocols	- bottom-up design - decentralized - standard agnostic	- JVM - RMI - JNDI
- **Messaging Channels**	- JXTA pipes, various protocols - Firewall and NAT friendly	• Any (normally HTPP) - Firewall friendly	- Same as WS (Axis)	- RMI, ORB - HTTP, JMS by existent plugging	- UDP and TCP - Mobile agents - Any other can be plugged	- RMI - Sockets
- **Messaging** - **Types**	- Asynchronous - P2P /unicast or multicast	- Synchronous	- Synchronous	- Asynchronous	- Asynchronous - P2P / mechanism need to be plugged	- Synchronous
- **Messaging** - **Perform.** - **(ms)**	- 10 x RMI	- 9 x RMI	- Relies on WS invocation (Axis) - GT4 improves 4 x GT3 performance	- 2xRMI	- Fast for UDP or TCP sockets - Rest depends on transport plugged	- Round Trip Time (RTT) average : - 0.25ms
- **Resource Discovery Performance**	- Average, XML processing penalizes - Good for key based routing	- Average (jUDDI) - Degradation for concurrent publications	- Expected to be Average - Index Service refactored in GT4 (with JNDI)	- Good for small and medium sized MAS - Average/Bad for federated DFs	- Good for known hosted agents - Discovery depends on plugged mechanism	- Depends on plugged service - (e.g. JNDI over LDAP)

TABLE 2 TECHNICAL PROPERTY /PLATFORM	**JXTA**	**Web Services (JAX-RPC Imp)**	**WSRF / OGSA**	**JADE**	**DIET**	**J2SE**
- Scalability - Nº 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)

TABLE 3 **DEVELOPER VIEW** **/PLATFORM**	**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

TABLE 3 **DEVELOPER VIEW** **/PLATFORM**	**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

ANNEX 3.2: 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 3.2.3, or available links:

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

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

  http://docs.oasis-open.org/wsrf/2004/11/wsrf-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/wsrm/2004/06/WS-Reliability-CD1.086.pdf

ANNEX 3.3: 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.

4 Conclusions

4.1 Conclusions on the architecture design process

We consider that the proposed architecture brigs a set of important characteristics to Catnets, namely a appropriated separation of concerns that will facilitate the implementation process, a great deal of flexibility and a strong "agnosticism" regarding the underlying platforms and the application scenario, which will make more adaptable to changing environments.

However, there are some open issues that should be addressed during the detailed middleware design:

- Lack of standards for APIs: there are no standard application interfaces for some critical functions like P2P overlay management or for communication primitives. This could limit the experimentation with different middleware toolkits. For example, changing the overlay management from JXTA to FreePastry might require an intensive re-work of the code that depends on this functionality. One possible approach to overcome this is to develop a set of abstract APIs and map them to each implementation, but the risk is to find discrepancies in the semantic that might result impossible to unify under a single model or to end up with functions with a semantic so generic that results unintelligible.
- Need for a flexible framework for resource management. The ultimate function of the Catallactic middleware is to offer a platform for implementing resource allocation mechanisms. Therefore, the integration with the resource managers offered by the diverse base platforms is a critical feature. However, each base platform will use a different resource management mechanism for resource allocation and resource monitoring. This lead to the need for a flexible framework that allows a consistent view and management of resources using a uniform set of mechanisms.
- Complexity of the design of systems as a group of simple interacting agents with emergent properties. There are no proven methodologies to systematically address the design and implementation problems derived from such a radical change in software engineering, where basic systems properties are emergent, instead of being a product of carefully designed and predictable mechanism

4.2 Conclusions on the middleware toolkit selection process

The middleware toolkit selection process was carried out taking into account three different but related aspects, which are potential applicaction 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 Catallactic 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 toolkit should provide be mature, have a rich set of development tools and good documentation.

Figure 4.1 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.

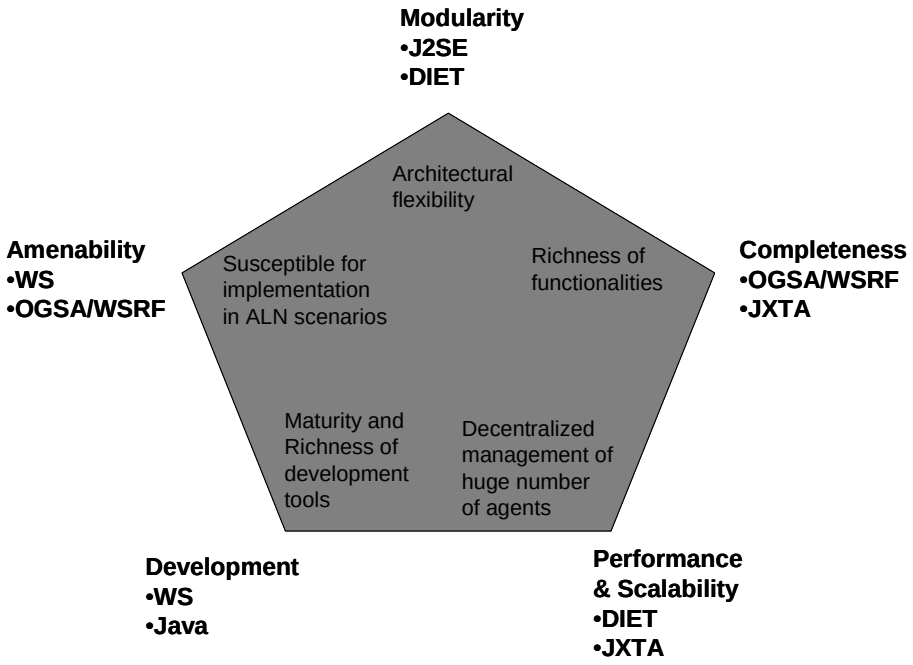


Figure 4.1. Condensed view of the key middleware toolkit evaluation criteria

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.

4.3 Future steps

Currently we work on a prototype to validate the architecture presented in this document. This prototype is oriented towards Grid scenario using the with DIET, JXTA and Globus toolkits. The prototype will provide additional insight into technical and functional properties which should be useful to confirm the feasibility of the software architecture. The prototype should have a balance between efficiency in execution and the flexibility to experiment with different implementation approaches or tools. Efficiency can be achieved by using simple and direct designs approaches that takes advantage of features and mechanisms optimized for the specific implementation

platform, whereas the flexibility requires generic mechanism and more complex designs patterns.

Experimentation will be important for to test critical features which will have a significant impact on the architecture, specially the overlay network and the object discovery. In this area, we will consider implementations like FreePastry, P2PSimple and P2P-DIET.

We expect that the results form experiments with this prototype will benefit the specification of the components made in deliverable D3.2. Secondly, it will reveal implementation issues and provide a framework to evaluate different design alternatives.