034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number D3.7

Report on the results of the middleware experiments during the final testbed experiments

Due date of deliverable: 2008-09-30
Actual submission date: 2008-09-30
Document code: Phosphorus-WP3-D3.7

Start date of project:
October 1, 2006

Duration:
30 Months

Organisation name of lead contractor for this deliverable:
Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract

This report will summarize the results of the testbed experiments of the middleware, including in particular the modifications of the middleware made after the first testbed phase, the experiments with the resource selection service and the final experiments with interfacing the G^{2}MPLS implementation of WP2. These experiments also served as a preparation of the demonstration and training events planned during the Supercomputing Conference in Austin and the ICT Conference in Lyon in November 2008. In addition experiments with the INCA middleware in WP3 will be presented.

Table of Contents

Table of Figures

0 Executive Summary

This report will summarize the results of the testbed experiments of the middleware, including in particular the modifications of the middleware made after the first testbed phase, the experiments with the resource selection service and the final experiments with interfacing the GMPLS implementation of WP2. These experiments also served as a preparation of the demonstration and training events planned during the Supercomputing Conference in Austin and the ICT Conference in Lyon in November 2008. In addition experiments with the INCA middleware in WP3 will be presented.

.

1 Overview

This report summarizes the modifications of the middleware provided by WP3 and the final experiments after the first phase of testbed experiments. The main focus in the reported period is on integration with the G^{2}MPLS developed in WP2, experiments with the resource selection service (RSS), final adaptation of the middleware towards the HARMONY system provided by WP1, and last modifications in UNICORE driven by the integration of RSS, G^{2}MPLS and HARMONY. Finally, INCA activities and results conclude the report.

During the reporting period the main tasks of WP3 were:

- Middleware development and adaptation (MetaScheduling Service (MSS), INCA)
- Interfacing between G2MPLS and Resource Management System via G-OUNI
- Integration and Evaluation of Middleware

The report builds on this separation of tasks, thus section 2 reports on the adaption and developments of the MSS to interface with G^{2}MPLS, section 3 on developments and experiments with the resource selection service (RSS), and section 4 describes the experiments done so far with the INCA middleware. Besides the mentioned modifications smaller modifications of the MSS were made to support workflows including pre-staging and post-staging of application data. The entire report is summarized in section 5, which also draws some conclusions.

1.1 Work package objectives

The work-package objectives during the first 24 months timeframe were:

- Formulation of user requirements to contribute to the definition of the functions and services to be implemented by the overall project
- Development and provisioning of the middleware services to orchestrate requested for an application, i.e. network, compute resources, visualisation devices, etc.

- Integration with network layer developments (Control Plane and Grid-NPRS) with Grid middleware to enable high performance applications running efficiently in the test-bed
- Integration with the network layer by means of the G^{2}MPLS developed by workpackage 2
- Adaptation of selected Applications to showcase the benefit from the integrated test-bed environment developed in the project
- Definition of the properties that network resources (services) should have in order to integrate seamlessly into an environment with other grid services
- Enabling user and applications to automatically get access to the resources that match their requirements e.g. in terms of performance and QoS.
- Enabling users and application to plan resource usage in advance and to dynamically adapt the resource usage to the changing requirements of the application thus avoiding expensive waste of resources.
- Managing all resources required to run an application in an integrated manner with a single service based interface towards to user or the application respectively.
- Testing the provided networking functionality based on application use cases defined in the first project stage and deriving requirements for extensions and enhancements to be carried out in the second 12 month period from the test-bed experiments

1.2 Integrated Approach

The experiments conducted in the reporting period aim at presenting the successful integration of the four layers the PHOSPHORUS testbeds are built upon:

- Physical layer
- Operating systems layer
- Middleware layer
- Application layer

The physical layer is now additionally accessible via the G^{2}MPLS and the G.O-UNI interface between the middleware and the G^{2}MPLS. More details on G^{2}MPLS can be found in the deliverable “Preliminary Grid-GMPLS Control Plane prototype” [D2.5].

The approach for the integration between the middleware and G^{2}MPLS is described in this document in section 2

The interface between the application layer and the middleware layer, in other words how the applications may access middleware services and through these middleware services may benefit from all layers of the PHOSPHORUS service stack is described in several deliverables with “Report on the Results of the Application Experiments During the Final Testbed Experiments” [D3.6] being the latest one.

2 Development of and experiments with an interface to the G^{2}MPLS

G^{2}MPLS is a conceived as a powerful Network Control Plane solution that enhances the standard GMPLS architecture providing single-step resource reservation, co-allocation and maintenance of both network and Grid resources [G2MPLS].

The G^{2}MPLS Control Plane can flood Grid resource information through the G^{2}MPLS domains and receives requests for Grid connections setup via G.O-UNI.

The G^{2}MPLS integration into the middleware layer is aiming on enabling high performance application running efficiently in the testbed.

2.1 Testbed description

For these experiments resources of two PHOSPORUS testbeds have been used: Resources of the VIOLA testbed (see Figure 1) and resources of the testbed of the University of Essex (see Figure 6) were used to carry out the experiments. The two testbeds are connected by a dedicated 1 Gbit link.

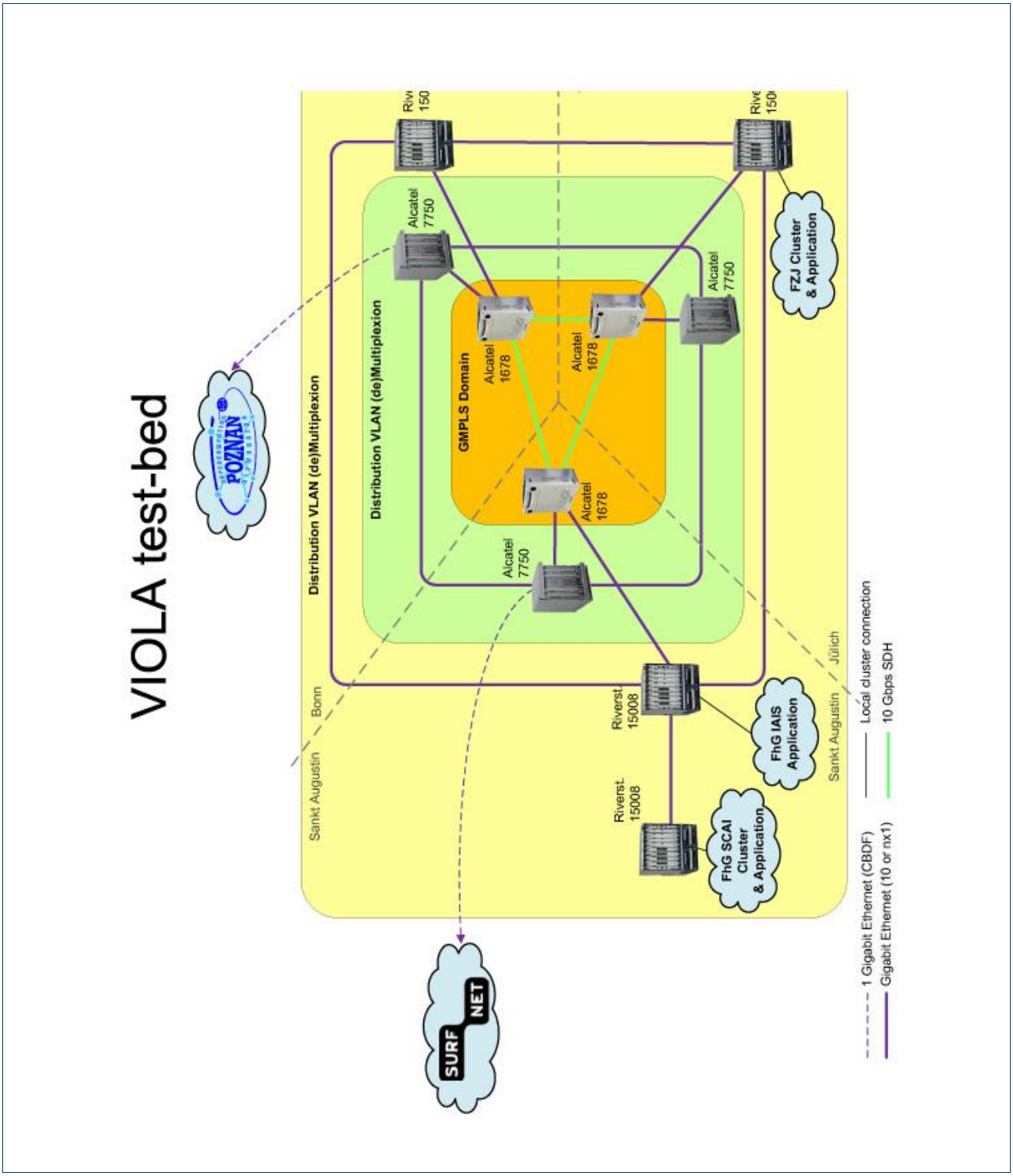

Figure 1: Topology of the VIOLA testbed

One standard being developed within the Open Grid Services Architecture (OGSA) is the Basic Execution Service (BES). BES provides functionality where client can send requests to initiate, monitor and manage computational activities. Clients define activities using the Job Submission Description Language (JSDL).

The integration of MSS and G^{2}MPLS is based two web services: BESFactory and GRRService.

BESFactory is a Basic Execution Service (BES) based service.

Grid Resource Registry (GRR) service allows clients to retrieve and store data describing Grid resources. GRR uses GLUE [GLUE] schema to store and exchange grid resource data

Both web services are written in Java. Webservice functionality is based on Axis2/XmlBeans framework. Axis2 allows generation of service skeletons from known WSDL interfaces. Develop and testing environment takes advantage of Maven. Maven is a software project management tool with possibility of automated testing of software (Junit, plugins for integration tests). BESFactory and GRRService consist of at least two parts: web stack part and service implementation part. The web stack module contains pure Axis2/XMLbeans skeleton for platform independent communication. The implementation part inherits web stack functionality and implements actual service logic.

Basic local functionality was tested with help of Maven's build & test environment. Extended local tests were accomplished manually. For the actual G^{2}MPLS integration remote testing was performed between the FHG site and the site in Essex (UESSEX). A test installation of the services realising the integration was done on the Pack cluster at FHG SCAI . Incoming test requests from Essex were received and directly processed on Pack cluster. Because of UESSEX's strict security regulations, connection was maintained over a ssh tunnel.

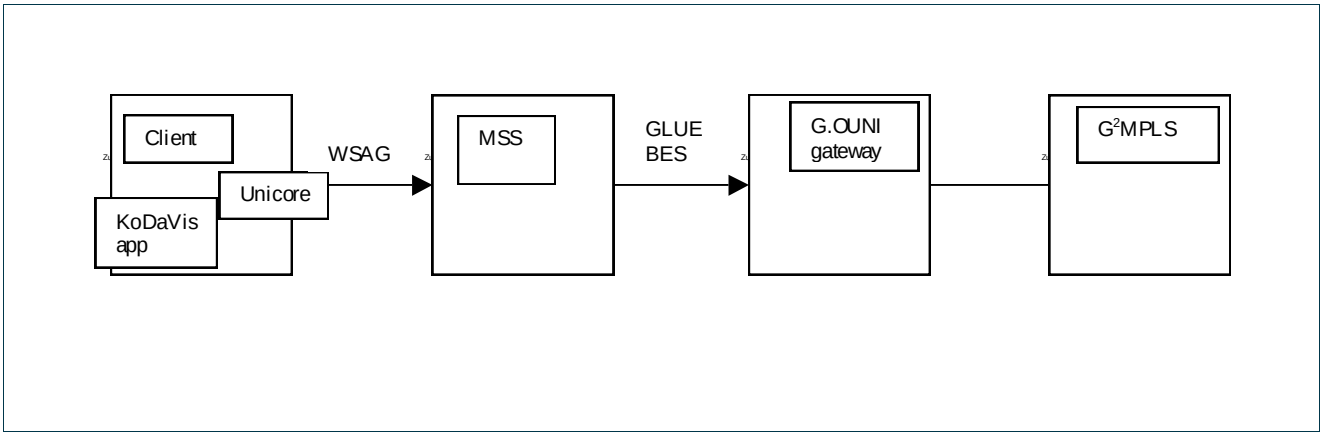

Figure 2: WP3 Middleware G2MPLS workflow, an overwiev

Figure 2 shows a typical workflow of a KoDaVis test case. Our workflow consists of:

1. Aim: start of a KoDaVis application. Network resources are required.

2. Request is submitted to MSS using WS-Agreement.

3. MSS can interact with G.OUNI gateway. One of three scenarios is possible:

    a. The KoDaVis client is manually choosing target resource

    b. Proper target resources are chosen by GRRService

    c. Target resources are chosen by G^{2}MPLS

4. Returns an address of a KoDaVis service instance.

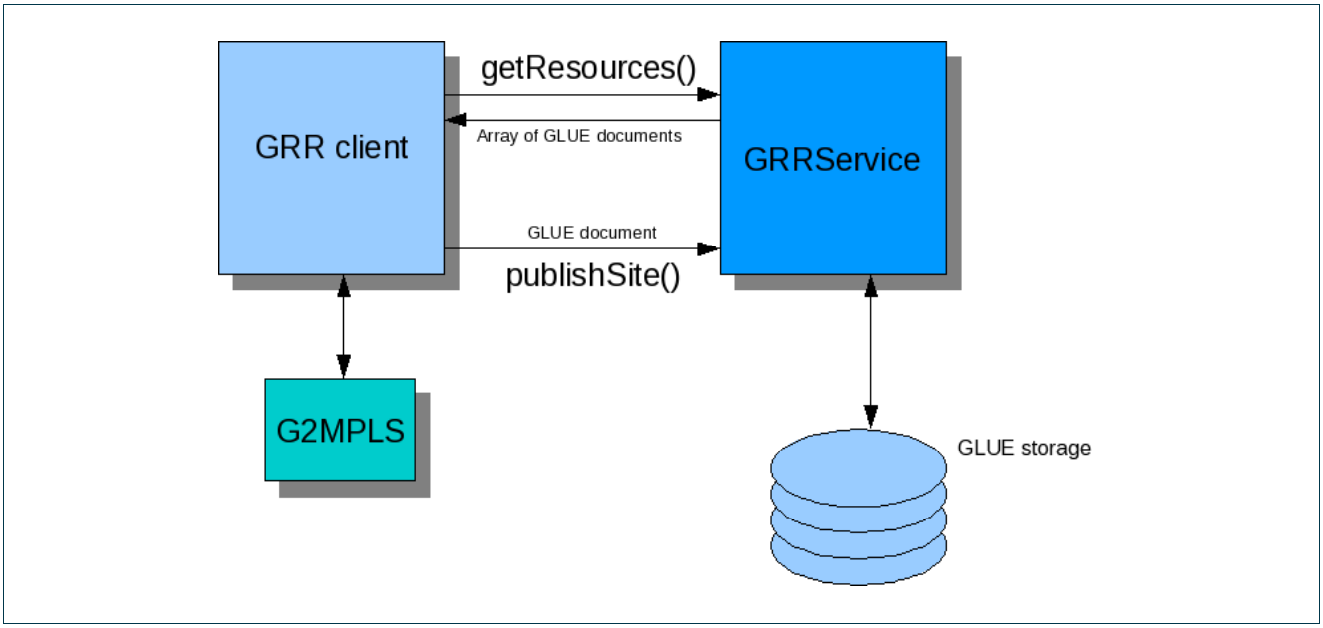

Figure 3: G^{2}MPLS integration: G^{2}MPLS plane connects to middleware to get/receive GLUE documents describing network resources.

G^{2}MPLS and MSS can utilize Grid Resource Registry (GRR) to store and/or retrieve data describing Grid resources. The GRR interface consists of two functions:

- getResources() retrieves an array of already stored GLUE documents,
- publishSite() sends a GLUE document to GRRService to store it.

In a typical scenario G^{2}MPLS registers a Grid resource in the GRR storage, then MSS can retrieve all registered documents for further selection (e.g. slot based).

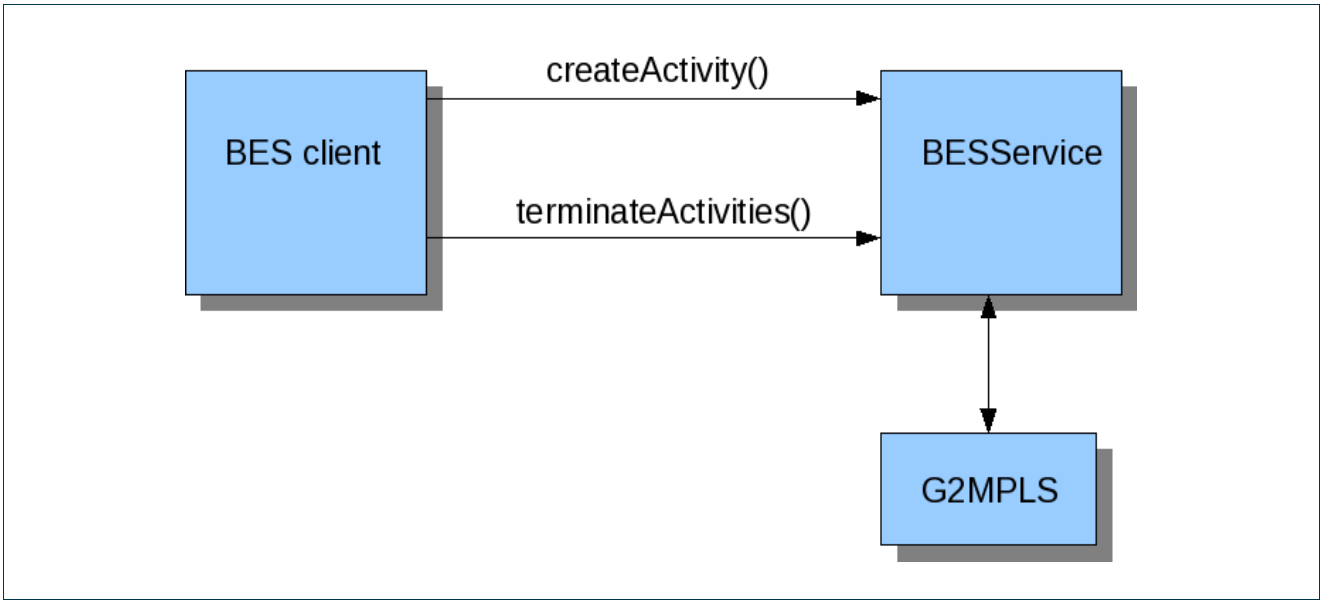

Figure 4: G^{2}MPLS integration: Middleware client (BES client) connects to G^{2}MPLS plane to create/terminate/monitor an activity.

The Basic Execution Service (BES) provides G.OUNI connectivity for MSS (see Figure 2). The MSS connects to G.OUNI via BES interface (client rectangle in Figure 4) to manage a given activity. BES provides functions like createActivity() or terminateActitivities() towards the MSS. Activity requests are forwarded to G.OUNI via BESservice. G.OUNI responds with a BES compliant message to the MSS.

2.2 Test description

The first series of tests focussed on functionality of the interface. No tests of the non-functional properties, e.g. temporal behaviour, have been made in this phase. This will be done during the lfollowing phase of the project.

The first remote tests in both directions were aiming on basic connectivity between the testbed sites University of Essex and Fraunhofer SCAI. Several predefined dummy requests and responses were created to test general web service connectivity between the testbed sites. We accomplished those tests without problems.

In second stage of the test phase, submitting and processing of meaningful data between Fraunhofer SCAI and University of Essex was tested.

As for GRR, functionality for requesting information about sites worked fine in both directions. Requests using getResourcesOperation()[3] returned correct response (Figure 3). This function returns a array of GLUE documents. GLUE documents are stored on server. Sending and storing GLUE documents did also work without problems. publishOperation()was correctly transmitting and storing GLUE documents in our test bed. This function transmits a GLUE document to GRRService to store it. As for BESFactory, there was more

functionality to test. After connectivity tests, we concentrated on basic functionality of functions like: createActivity() [4] and terminateActivities() (Figure 4).

Missing BES functionality was added through extensions. Extensions have been used for adding features like start-, end time or network bandwidth. Functions for creating and terminating jobs were correctly transmitting and parsing proper values.

After those tests , we had to replace our web stack for BES and GRR because of merging with a different piece of code. Basic connectivity is restored now. Our previous tests are also running smooth. We still missing BES interface tests besides createActivity()and terminateActivities() .

3 Resource Selection Service - developments and experiments

The Resource Selection Service (RSS) has been designed in order to allow the MetaScheduling Service (MSS) to place a job request automatically on an appropriate computing system taking into account the specific requirements of the job as imposed by the application to be executed.

To fulfil this requirement, the RSS framework has bee designed and implemented in WP3. This framework consists mainly of two different ontologies, which in turn will deliver on enquiry

- a list of cluster names suitable for the job out of which the MSS selects one by negotiating, or
- an empty list in case there is non available

3.1 Testbed description

These experiments were carried out using resources of the VIOLA testbed (see Figure 1) where the MetaScheduling Service of the PHOSPORUS testbeds is hosted. Additionally, resources of the PSNC testbed (see Figure 5) and resources of the testbed of the University of Essex (see Figure 6) were used as targets of the resource selection. The VIOLA testbed and the PSNC testbed are connected through a dedicated cross-border dark-fibre, the VIOLA testbed and the testbed of the University of Essex are connected by a dedicated 1 Gbit link.

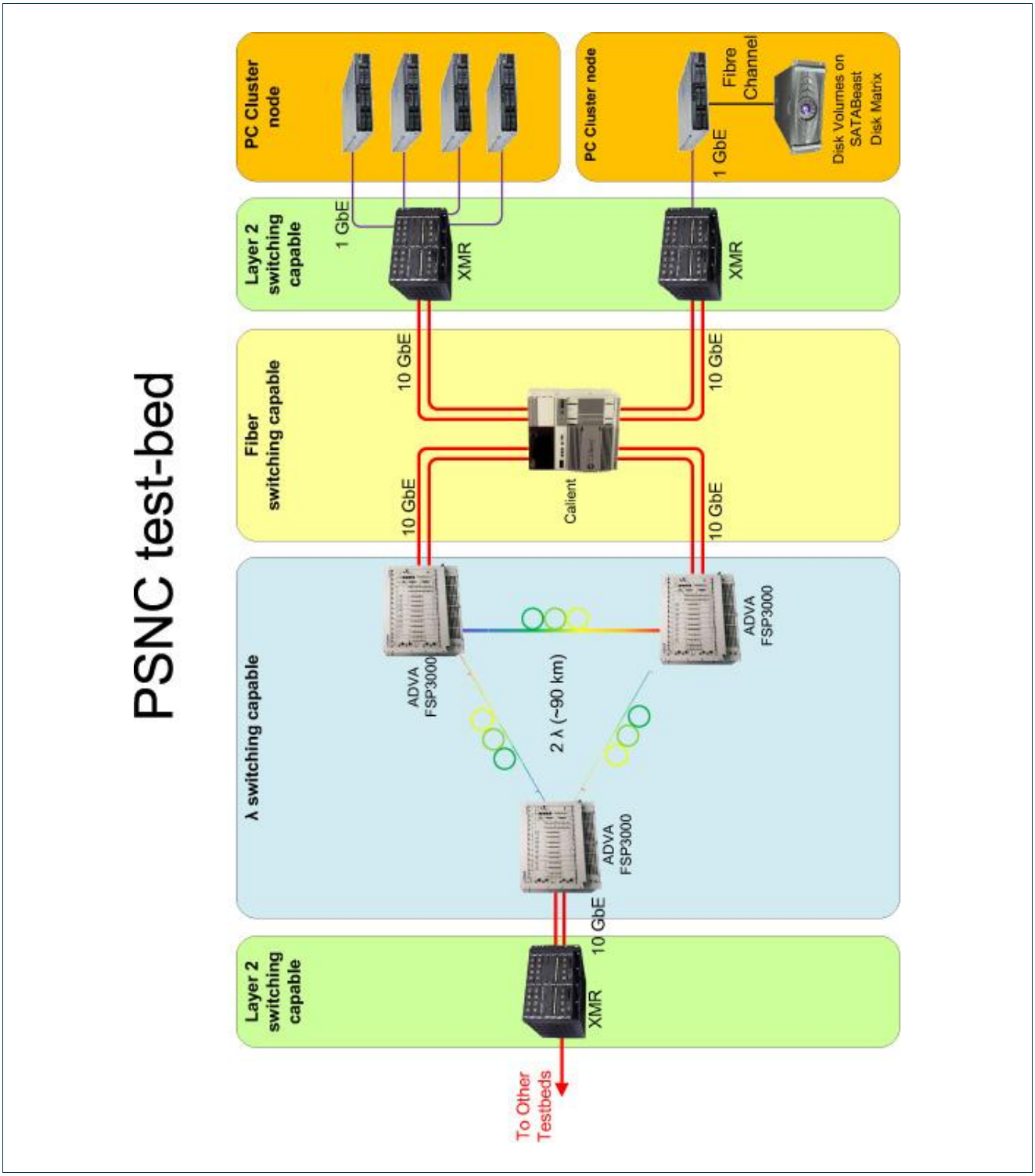

Figure 5: Topology of the PSNC testbed

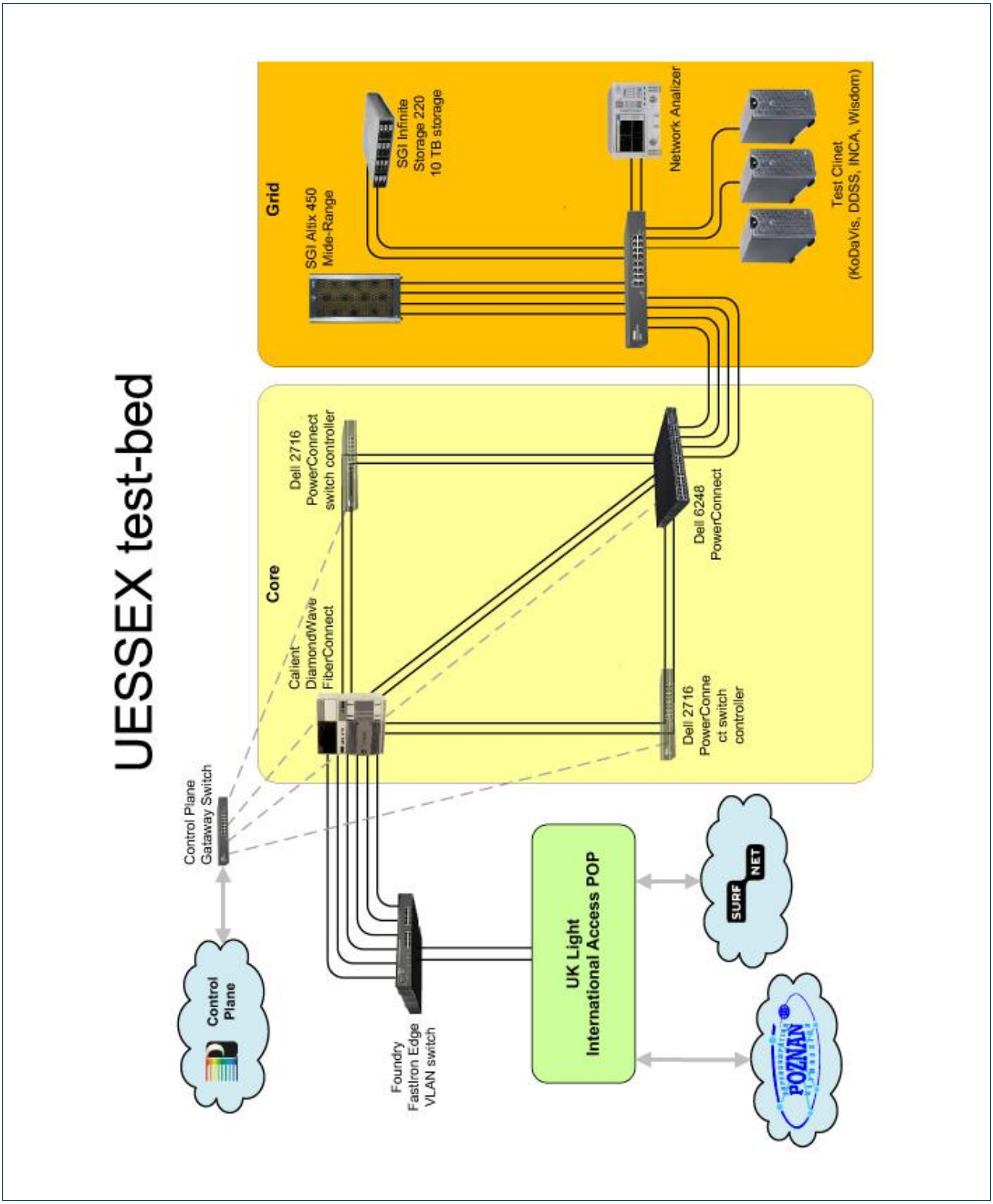

Figure 6: Topology of the University of Essex testbed

3.2 Ontology design

We have designed two different ontologies (OWL-Files: one for the requirements of the applications and one for the offered resources of the clusters in the testbed). The classes and its instances are annotated with different data-types and the designated values of the application´ requirements and the properties of the computing systems which are provided in the testbed (Fig 1).

The Application Ontology consists of 27 classes and 8 instances with its specific value pairs (e.g. needOfMemory : 512, as an Integer-Type defined as MB).

The Resource Ontology consists of 14 classes and 3 instances with its specific value pairs (e.g. offersMemory : 2000, as an Integer-Type defined as MB).

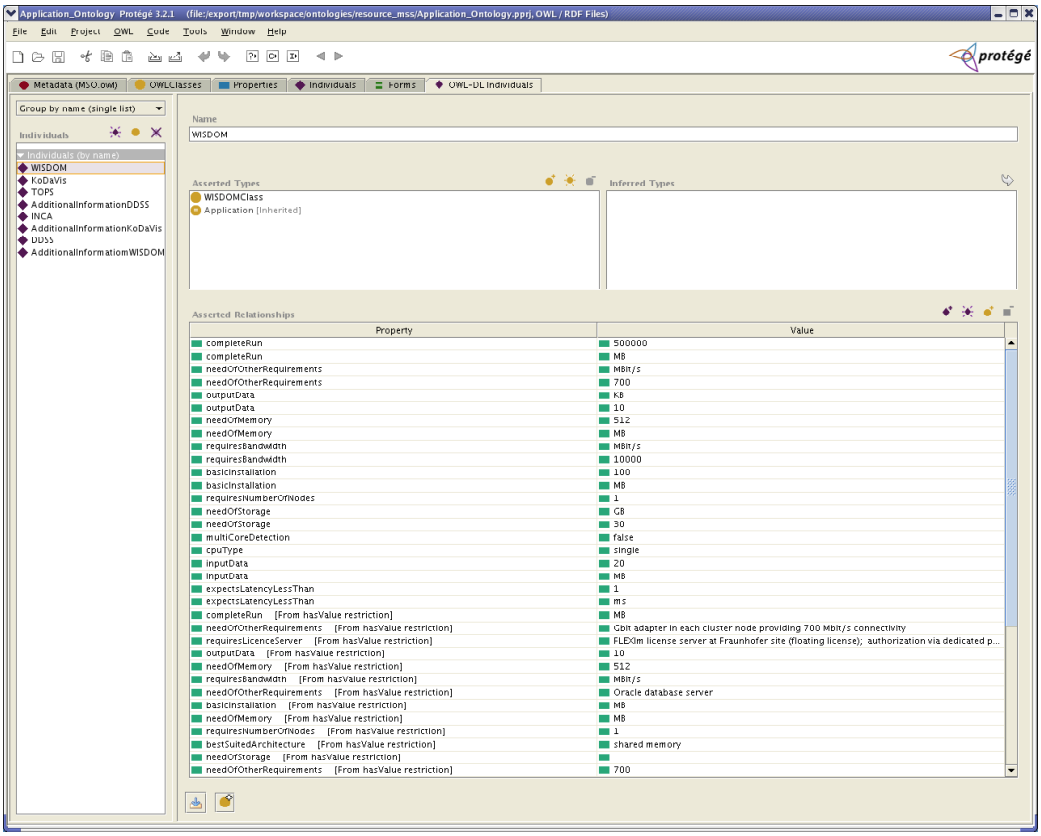

Figure 7: Detail of the Application Ontology

The RSS is modelled as a Webservice within the Jena framework, which allows running SPARQL queries against the ontologies in a Java environment. The matching process of the results from the ontologies is conducted in this Jena framework.

First of all we tested if the ontologies can be used in order to retrieve the specific entries (e.g. How much memory does WISDOM need or How many nodes need to be used):

```
REFIX owl: <http://www.w3.org/2002/07/owl#>
      PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

      SELECT ?Subject ?Predicate ?Object
      WHERE
       { ?Subject ?Predicate ?Object .
              FILTER regex(str(?Subject), "WISDOM")
```

As the result of this query, you will get a complete list of entries, which are all addilated to WISDOM:

```
    <result>
      <binding name="Subject">
        <uri>http://www.owl-ontologies.com/MSO.owl#WISDOM</uri>
```

```
      </binding>
      <binding name="Predicate">
        <uri>http://www.owl-ontologies.com/MSO.owl#needOfMemory</uri>
      </binding>
      <binding name="Object">
        <literal datatype="http://www.w3.org/2001/XMLSchema#integer">512</literal>
      </binding>
    </result>
```

To retrieve entries from an OWL/RDF-File, we need a query language – in our case SPARQL, due to its compatibility to the programming language JAVA.

The enquiry itself consist of a Simple Protocol and Query Language for RDF-Query (SPARQL):

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?Subject ?Predicate ?Object
WHERE {
      ?Subject ?Predicate ?Object .
      FILTER regex(str(?Subject), \"" + object + "\") .
      FILTER (datatype(?Object) = xsd:integer)
}
```

With these kinds of queries it is possible to retrieve the favoured entity from the ontology.

3.3 Tests within the Jena Environment

Only local experiments in the VIOLA testbed have been conducted so far to ensure that the RSS framework runs properly. Since in the current design of the PHOSPHORUS testbed only one instance of the MSS running at one of the sites (actually it is running at a node at FHG) is needed to provide the Grid Scheduler functionality to all testbed sites the resource selection capabilities are automatically available at all testbed sites.

In fact, we tested a) the framework with each application (WISDOM, KoDaVis, and DDSS) which will be available in the testbed, and got the following results:

Wisdom:

```
-------- TESTING ---------
possible candidates for WISDOM :
FZJ , FHG , PSNC ,
-------- TESTING ---------
```

KoDaVis:

-------- TESTING ---------
possible candidates for KoDaVis:
FZJ , FHG ,
-------- TESTING ---------

DDSS:

-------- TESTING ---------
possible candidates for DDSS:
FZJ ,

As expected, the results indicate for each application the maximum number (and names) of sites, which are suitable to run the application based on the description of the sites' capabilities and the applications' requirements as collected through a questionnaire beforehand. The data of the returned questionnaires were used to create the two ontologies, which are part of the RSS framework.

Getting the results of a query takes approximately 2390msec. with the aforementioned number of classes and instances.

As a secondary test, we extended the number of instances (+50/ontology) of the ontologies to enhance the complexity in order to stress the algorithm. Though, the algorithm is designed to extract and compare designated value pairs (e.g. `MEMORY : VALUE`) which are useful to estimate the proper cluster. Those kinds of values (see Figure 7) are compared. Due to to those specific values, the algorithm can handle many instances without running into an error.

However, the time to retrieve results increases up to approx. 3450msec. Thus, it increases approx. 1000msec. per +50-instances.

As an extraordinary test, we have wilfully written b) some wrong entities in the ontologies (e.g. a wrong instance name, less amount of memory) to see what happens with the algorithm. And as expected, the result returned is an empty list, since there is no matching resource:

-------- TESTING ---------
possible candidates for WISDOM :

-------- TESTING ---------

It is obvious that extensions of the ontologies and also those of the matchmaking algorithm must made in compliance with the specified framework.

3.4 Example

We would like to illustrate the aforementioned example (a) (for details of the framework see Figure 8):

The question which is sent by the MSS to the RSS, would sound in natural language like ”On which cluster may I run the application called WISDOM?”

The first step is that the Jena framework sends a SPARQL query with WISDOM as the 'object' in its formulae to the Ontology of Application in order to retrieve the requirements of the application.

Next, another query is sent to the Ontology of Resources with the different clusters (PSNC, FHG, FZJ) as the 'object'-part of the query.

These two results will be matched as follows:

The results contain strings, which have a XMLSchema datatype and a corresponding value. And exactly these values will be compared if and only if the datatypes are identical. The appropriate cluster will then be chosen accordingly to the values, and will be forwarded as a string to the MSS, which submits the job to the chosen cluster finallly.

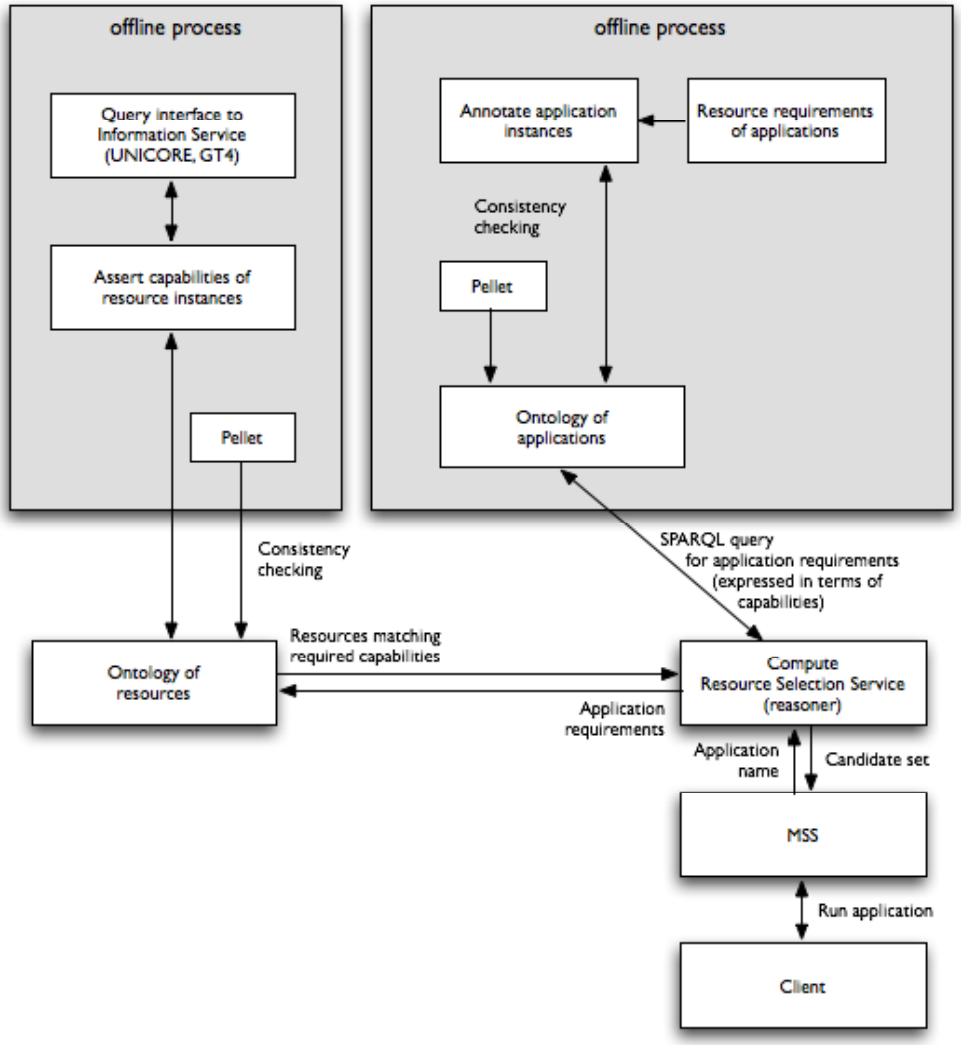

Figure 8: Resource Selection Service Framework

4 Experiments with the INCA middlewarel

In contrast to the previous phases INCA used simulations to evaluate the enhanced functionality in different simulated network topologies.

The INCA system has been evaluated over extensive simulations. Instead of a custom made simulator the OPNET Modeller was used to improve the reliability of the simulations. In this section the results of the simulations are presented for various numbers of nodes and various underlying network topologies.

As an objective during this phase was to perform the simulations based on a realistic network model where even the triangle inequality is violated, real round trip time measurements have been used derived from Meridian King data set which provides RTT measurements among 2500 nodes.

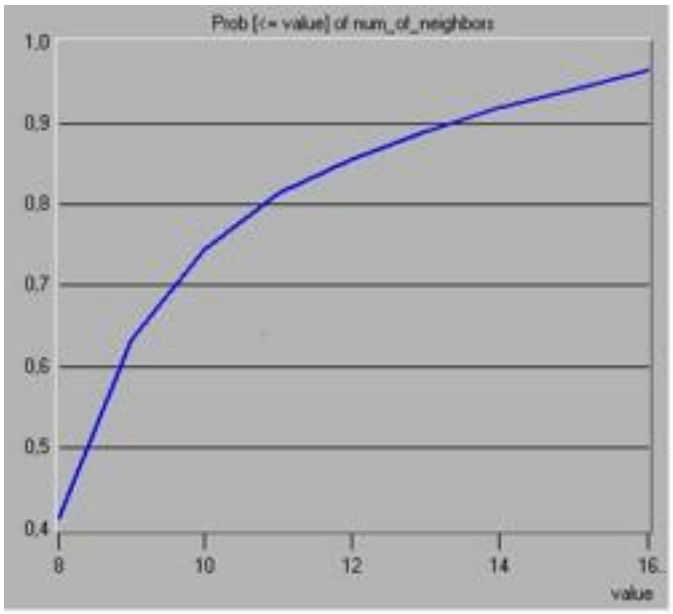

Figure 9: Cumulative density functions of the routing table size of each node in INCA with 2500 nodes and 4 dimensions

As a proof for achieving high levels of balanced routing tables we have used the cumulative density functions of the size of routing tables in INCA As Figure-1 demonstrates our system achieves a good level of balance in routing tables, in contrast to other overlays. In a system with 2500 nodes almost every node has between 7 and 16 neighbours. In Table 1 we can also see the routing table sizes as system scales. Through this table it is obvious that the routing table sizes remain almost constant thus balanced as system scales.

	500	1000	1500	2000
0.1	7	7	7	7
0.5	8	8	9	9
0.9	11	12	13	14

Table 1: The 10th the 50th and the 90th percentile with the routing table size which has each node in a 4-dimentional overlay for various number (500,1000,1500,2000) of nodes.

To evaluate the performance of our system when routing is carried out in INCA we compare the network stretch with that of the CAN. We have simulated our system with 4-dimentions and found that our system's network stretch outperforms a typical CAN system's stretch for 2000 nodes (Figure 2) by around 50%. In table 3 we can see the 50th percentile of stretch values for a different number of nodes.

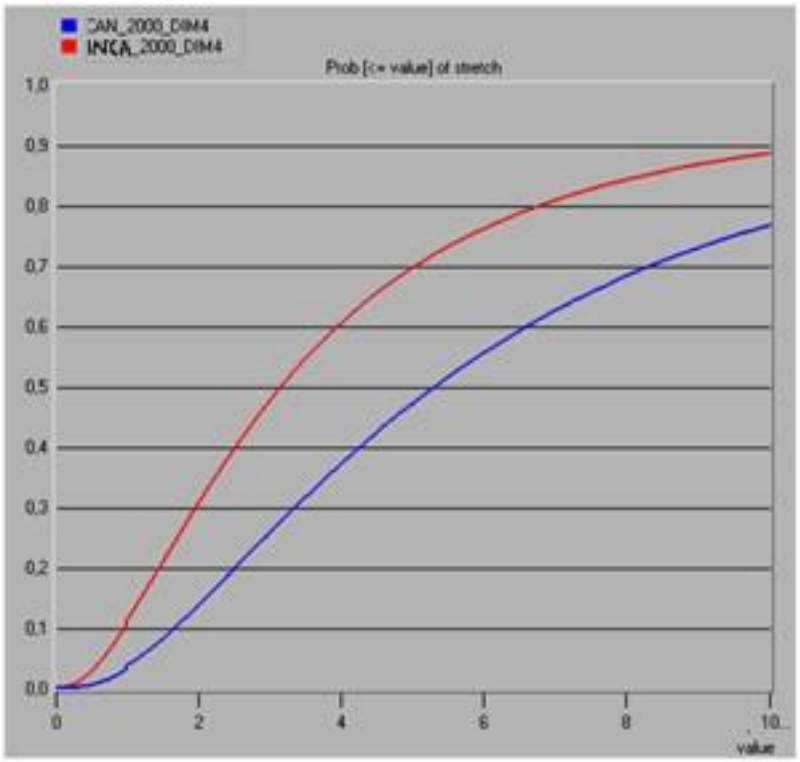

Figure 10: Compares the network stretch of 2000 nodes in a typical CAN, and in our system

	500	1000	1500	2000
CAN	3.5	4.3	4.9	5.3
INCA	2.3	2.6	2.8	3.1

Table 2. The 50th percentile of stretch values that has been observed in CAN and in our system for different number of nodes (4-dimensions).

It is spotted here that differences become larger as system scales. Summarising the evaluation of our system, INCA successfully captures locality information, maintaining balanced routing tables and is orthogonal to any load balance algorithm for the distribution of the keys to the various nodes.

5 Conclusions

This report highlights the results achieved in experiments made with middleware components after modifications based on experience and results of the previous middleware and application tests.

This report summarizes the results of the testbed experiments of the middleware, including in particular the modifications of the middleware made after the first testbed phase, the experiments with the resource selection service and the final experiments with interfacing the G^{2}MPLS implementation of WP2. The focus was on the G^{2}MPLS integration since this was a new development during the reported period.

The results of the experiments show that the improvements of the middleware towards automated resource selection through the Resource Selection Service deliver the expected behaviour. Now a user only needs to specify the application that should be executed and the framework automatically selects the appropriate resources, starts the negotiation on availability, and does the necessary reservations for the user.

With the interface between G^{2}MPLS and the middleware it could be shown that the resource selection may already start at the network layer based on the information on topology, connectivity and link status available there. The relevant information is then passed to the middleware layer and presented to the user for the final decision.

Finally, the INCA experiments showed in simulations the scalability and performance for a network environment with a much larger number of nodes and links than available in the PHOSPHORUS network.

The experiments also served as a preparation of the demonstration and training events planned during the Supercomputing Conference in Austin and the ICT Conference in Lyon in November 2008. In addition experiments with the INCA middleware in WP3 have been presented.

6 References

[G2MPLS] "Deployment and Interoperability of the Phosphorus Grid Enabled GMPLS (G2MPLS)" , DOI 10.1109/CCGRID.2008.120

[D2.5] D2.5 - Preliminary Grid-GMPLS Control Plane prototype, http://www.phosphorus.pl/wiki/images/0/00/Phosphorus-WP2-D2.5v1.0.doc

[D3.6] D3.6 Report on the Results of the Application Experiments During the Final Testbed Experiments, http://www.phosphorus.pl/wiki/images/3/3e/Phosphorus-D3.6.pdf

[GLUE] GLUE Working Group (GLUE), http://forge.gridforum.org/sf/projects/glue-wg

[GRRService] WSDL file, http://packcs-e0.scai.fraunhofer.de/phosphorus-wp2/services/GridResourceRegistryService?wsdl

[OGSA-BES] OGSA-BES Working Group, http://forge.gridforum.org/sf/go/projects.ogsa-bes-wg/ http://www.scai.fraunhofer.de/index.php?id=2384&L=1

7 Acronyms

AAA	Authentication, Authorisation, Accounting
DDSS	Distributed Data Storage Systems
e2e	end to end
EGEE	Enabling Grids for E-sciencE (European Grid Project)
FC	Fibre Channel
FC-SATA	Fibre Channel to SATA technology (mixed technology used in disk matrices: disk matrix have Fibre Channel ports for hosts connectivity, but contains SATA disk drives)
GEANT2	Pan-European Gigabit Research Network
GEANT+	the point-to-point service in GEANT2
GMPLS	Generalized MPLS (MultiProtocol Label Switching)
G^{2}MPLS	Grid-GMPLS (enhancements to GMPLS for Grid support)
G.O-UNI	Grid Optical User Network Interface (integrating optical networks with grid services)
GT4	Globus Toolkit Version 4 (Web-Service based)
INCA	Intelligent Network Caching Architecture
KoDaVis	Tool for Distributed Collaborative Visualisation
MSS	MetaScheduling Service (a Grid Level Scheduler developed at the Fraunhofer Institute SCAI)
NREN	National Research and Education Network
NRMS	Network Resource Management System
NRPS	Network Resource Provisioning System
PoP	Point of Presence
Protégé	Ontology Editor and Knowledge Acquisition System
QoS	Quality of Service
SNMP	Simple Network Management Protocol
TOPS	Technology for Optical Pixel-Streaming
TPD	Tiled Panel Display
TUAM	Tool for Universal Annotation and Mediation
UNI	User to Network Interface
UNICORE	European Grid Middleware (UNIform Access to COmpute REsources)
VLAN	Virtual LAN (as specified in IEEE 802.1p)
VIOLA	A German project funded by the German Federal Ministry of Education and Research (Vertically Integrated Optical Testbed for Large Applications in DFN)
VPN	Virtual Private Network
WISDOM	Wide In Silicio Docking On Malaria