Grid application meta-repository system.

Alexandru Cristian Tudose

School of Electronics and Computer Science

GRID APPLICATION

META-REPOSITORY SYSTEM

Alexandru Cristian Tudose

A thesis submitted in partial fulfilment of the requirements of the University of Westminster for the degree of Doctor of Philosophy

August 2010

To Isabella, Sorana, Aritina and Valentin

Acknowledgements

I would like to express my gratitude to all the people who encouraged and helped me throughout my PhD studies, from its very start and up to the writing up of this thesis.

Special thanks to Dr. Gabor Terstyanszky for the close guidance, advice and mentorship he provided for the past four years. I am indebted to him for the numerous research meetings we had and for his excellent suggestions and corrections during the writing up of various research papers, reports and of this thesis.

Furthermore, I would like to express my gratitude towards Prof. Stephen Winter, my director of studies, for admitting me to the PhD programme and for granting me the research scholarship. I would also like to thank him for his patience, guidance and support throughout my research.

Special thanks are due to Prof. Peter Kacsuk for his invaluable questions, ideas and advice, as well as for his words of encouragement and for helping me understand that no PhD research can hold the answer to all the unsolved issues in computing science.

I am also indebted to my research colleagues for all the debates and discussions held during my years of research, which helped broaden my view of this research area and ultimately improved the quality of my research.

Last, but not least, I wish to express my gratitude towards my dear wife, Sorana – for her emotional support and understanding, and for the countless editorial suggestions she made to this thesis; to my wonderful baby Isabella – for being patient and waiting to be born only after I had finished the first draft of this thesis; to my parents, Aritina and Valentin, who have inspired me to become who I am today; and to all my friends and family – for their uplifting words of encouragement throughout these past four years.

Abstract

As one of the most popular forms of distributed computing technology, Grid brings together different scientific communities that are able to deploy, access, and run complex applications with the help of the enormous computational and storage power offered by the Grid infrastructure.

However as the number of Grid applications has been growing steadily in recent years, they are now stored on a multitude of different repositories, which remain specific to each Grid. At the time this research was carried out there were no two well-known Grid application repositories sharing the same structure, same implementation, same access technology and methods, same communication protocols, same security system or same application description language used for application descriptions. This remained a great limitation for Grid users, who were bound to work on only one specific repository, and also presented a significant limitation in terms of interoperability and inter-repository access. The research presented in this thesis provides a solution to this problem, as well as to several other related issues that have been identified while investigating these areas of Grid.

Following a comprehensive review of existing Grid repository capabilities, I defined the main challenges that need to be addressed in order to make Grid repositories more versatile and I proposed a solution that addresses these challenges. To this end, I designed a new Grid repository (GAMRS – Grid Application Meta-Repository System), which includes a novel model and architecture, an improved application description language and a matchmaking system. After implementing and testing this solution, I have proved that GAMRS marks an improvement in Grid repository systems. Its new features allow for the inter-connection of different Grid repositories; make applications stored on these repositories visible on the web; allow for the discovery of similar or identical applications stored in different Grid repositories; permit the exchange and re-usage of application and application-related objects between different repositories; and extend the use of applications stored on Grid repositories to other distributed environments, such as virtualized cluster-on-demand and cloud computing.

Contents

List of Figures

List of Tables

List of Acronyms

ACL = Access Control List

ADL = Application Description Language

API = Application Programming Interface

BDII = Berkeley Database Information Index

BPEL = Business Process Execution Language

BPEL4WS = Business Process Execution Language for Web Services

CPU = Computing Processing Unit

CRUD = Create Retrieve Update Delete

DAML-S/ OWL-S = DARPA Agent Markup Language - Semantic/ Ontology Web Language - Semantic

DL = Deductive Logic

DNS = Domain Name Server

EDGeS = Enabling Desktop Grids for e-Science

EGEE = Enabling Grids for E-sciencE

FOXML = Fedora Object Extended Markup Language

FQDN = Fully Qualified Domain Name

ftp = file transfer protocol

GAMRS = Grid Application Meta-Repository Service

GEMLCA = Grid Execution Management for Legacy Code Architecture

GGF = Global Grid Forum

gLite = Lightweight Middleware for Grid Computing

gridftp = Grid file transfer protocol

GSI = Grid Security Infrastructure

gsiftp = Grid security infrastructure file transfer protocol

GT2 = Globus Toolkit 2

GT4 = Globus Toolkit 4

GUI = Graphical User Interface

http = hypertext transfer protocol

I/O = Input/Output

IR = Information Retrieval

ITL = Information Terminological Language

JDL = Job Description Language

JSDL = Job Submission Description Language

JSR-168 = Java Specification Requests - 168

LARKS = Language for Advertisement and Request for Knowledge Sharing

LCG = LHC Computing Grid

LCID = Legacy Code Interface Description

LDAP = Lightweight Directory Access Protocol

LDL++ = Logical Data Language

LSI = Latent Semantic Index

MAS = Multi-Agent Systems

MPI = Message Passing Interface

MRDL = Meta-Repository application Description Language

NGS = National Grid Service

OAI-PMH = Open Archive Initiative – Protocol for Metadata Harvesting

OAI-ORE = Open Archive Initiative – Object Reuse and Exchange

OGSA-WG = Open Grid Services Architecture – Working Group

OGSI = Open Grid Services Infrastructure

OS = Operating System

OVF = Open Virtualisation Format

OWLS-MX = Ontology Web Language Semantic Matchmaker

P-GRADE = Parallel Grid Run-time and Application Development Environment

PKI = Public Key Infrastructure

RAM = Random Access Memory

RBS = Resource Brokering System

REST = Representational State Transfer

rfiod = remote file input output daemon

RSL = Resource Specification Language

Scufl = Simple conceptual unified language

SMTP = Simple Mail Transfer Protocol

SOAP = Simple Object Access Protocol

SRB = Storage Resource Broker

srm = storage resource management

SWSL = Semantic Web Services Language

TFIDF = Term Frequency Inverse Document Frequency

UDDI = Universal Description Discovery and Integration

UI = User Interface

URI = Universal Resource Identifier

VNC = Virtual Network Computing

VO = Virtual Organisation

VM = Virtual Machine

WfMC = Workflow Management Coalition

WFML = Web Form Markup Language

WS = Web Services

WSDL = Web Services Description Language

WSMO = Web Services Modeling Ontology

WS-PGRADE/gUSE = Web Service P-GRADE/ grid User Support Environment

WSRF = Web Services Resource Framework

WS-RSL = Web Service – Resource Specification Language

X Scufl = extended Simple conceptual unified language

XML = Extended Markup Language

xRSL = Extended Resource Specification Language

Chapter 1

Introduction

Extensive research is being undertaken these days in the field of Grid computing. On one hand, scientists try to adapt old technologies and old concepts to Grid platforms. On the other hand, as new concepts appear, they are already designed to be compatible with Grid technologies. The impact of Grid on scientific communities all over the world has often being compared to the tremendous impact that the discovery of WWW has had on the worldwide IT community [1].

Grid is a wide network of distributed resources, in which groups of people with common *computationally demanding* or *data-intensive* goals have chosen to share their resources (e.g. computers, storage components, software and applications, data, firmware implementations, sensors, networks, networking services etc.) in a controlled, secure and flexible way.[2] Within Grid, users can gain access to a multitude of applications and resources, with the help of which they can solve their problems more effectively.

Given that the number of Grid applications has been growing steadily in recent years, they are now stored in repositories that offer better options for their management. However, there are many repository frameworks on the market and these vary in terms of access interface, security system, implementation technology, communication protocols and transfer protocols. Moreover, administrators are free to choose among them and also have free choice in defining a specific repository model. At the same time, different Grid applications

are described using one of more than ten application descriptions languages (ADLs), which are either standard-specific or proprietary-specific.

As a result of this diversity, at the time this research was carried out there were no two well-known Grid application repositories having the same structure, same implementation, same access technology and methods, same communication protocols, same security system or same application description language used for application descriptions. This remained a great limitation for Grid users, who were bound to work on only one specific repository, and also presented a significant limitation in terms of interoperability and inter-repository access. The research presented in this thesis provides a solution to this problem, as well as to several other related issues that have been identified while investigating these areas of Grid.

This first chapter introduces and defines the main concepts related to the area of research and offers an outline of the structure of this thesis.

1.1. Short History of Grid Application Repositories

Grid technologies emerged in the mid 1990s as a solution for the optimization of resource sharing in computer networks. Initially, Grid computing research focused on the areas of computing resources, data access, and storage resources. However, the definition of Grid computing resource sharing has evolved in time and now includes any resources made available by a Grid participant, such as computing resources, data, hardware, software and applications, firmware implementations, networking services, and any other forms of computing resource attainment. By using the Grid infrastructure and Grid technologies users today can solve problems related to software capability (e.g. models, simulations, etc.); hardware availability or computing capacity shortage (e.g. CPUs, data storage, etc.); as well as address the need for immediate circuit provisioning of a network or a security event and many more types of critical environmental needs.

GRID APPLICATIONS

One of the aims for which the entire Grid infrastructure and Grid middleware were developed was to enable users to use applications more effectively. On Grid, users can gain access to a large number of applications, as well as to a multitude of resources, with the help of which they can solve their problems quicker and more efficiently.

Grid applications are pieces of software exposed to users through a Grid user interface. Many of these applications are well-known to users and were used in the past as stand-alone software, commonly installed on one PC or on small clusters. With the arrival of Grid technologies they were ported to interact with the Grid middleware and to run on the Grid infrastructure. Provided they acquire adequate rights to access them, many scientific communities can use these applications remotely and obtain the results they needed using the Grid. However, in certain cases, users need to solve problems with a higher degree of complexity, which require the help of not one, but several applications - for example, results obtained from running one application need to be passed as input to another application, which in turn should return the final results to the user. In order to make this process automatic and eliminate the need for intervention from the user, scientific communities have adapted the *workflow paradigm* to Grid.

WORKFLOWS

The Workflow Management Coalition (WfMC) defines *workflow* as "The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules".[3] The Open Grid Services Architecture – Working Group (OGSA-WG) expanded the WfMC statement, and defined the *workflow* simply as a pattern of business process interaction.[4] Such interactions may take place between services residing within a single data centre, or across a range of different platforms located anywhere in the world.

In the context of Grid computing the term *workflow* usually concerns the automation of distributed IT processes. However, there are several differences between Grid workflows and regular (i.e. non-Grid) business workflows, mainly in terms of reliability and performance. Since Grid resources involved in the execution of the workflow may fail during the runtime, Grid workflows rely on advanced workflow fault management techniques (such as workflow checkpointing, recovery and monitoring) to ensure a high level of reliability of the service. In terms of performance, one of the objectives of Grid computing is to provide high performance computing power and Grid workflows therefore have to deal with resource brokerage, scheduling (load balancing), and distributed applications (parallel computing) – services which are usually not needed by regular business workflows.

Given that one intrinsic characteristic of the Grid is its distributed infrastructure – which makes it an ideal candidate for supporting parallel computational tasks, Grid is regarded by researchers as an excellent workspace for workflows. Consequently, scientists and researchers have lately placed increasingly more effort in adapting the workflow paradigm to Grid concepts. As a result, Grid was enriched with several different workflow description languages (e.g. BPEL [5], BPEL4WS [6], Scufl [7], xScufl [8], etc.) as well as with several different workflow engines, such as P-GRADE [9], Taverna [10, 11], Kepler [12, 13], Triana [14], WS-PGRADE/gUSE [15], etc.

WEB SERVICES

At present, the vast majority of Grid software resources can be accessed in a standard client-server way. The server program is usually running in the main memory of the computer as a *daemon*, listening to a certain port on the machine. The client program connects to that port and communication occurs by means of pre-known requests and responses. This means that both client and server know the API (Application Programming Interface) of their requests and responses from the outset. However, with the growth of the Grid infrastructure, services became increasingly numerous and this raised new security issues (i.e. the increased

number of ports became a problem for firewalls and network administrators); development issues (i.e. developers needed to know every particular API for servers in order to develop new clients, sometimes for the same type of service); as well as usage issues (i.e. users became lost in complicated command-line syntaxes).

All these issues can be addressed by using the web services (WS) technology. First, WS uses well-known communication transport protocols (e.g. http, ftp, SMTP), which usually reside on well-known ports (ports that are opened and monitored on almost every network), so the network administrator's effort is reduced drastically. Second, the server publishes a set of communication information into a *registry*. The client therefore does not need to have prior knowledge of the server's API, it can simply query the registry and will find the location of the server, the location of a WSDL (Web Service Description Language) document describing how to initiate the communication, and the patterns of all requests and responses.

GRID SERVICES

Given the benefits of web service technology, and after years of carefully conducted studies and research on this topic, the Global Grid Forum (GGF) extended the web service concept to the concept of *Grid service*.

A *Grid service*, as defined in the Open Grid Services Infrastructure (OGSI), is a web service that conforms to a set of conventions (interfaces and behaviours) that define how a client interacts with that service [16]. Since it is created through the extension of the web service concept, the Grid service receives an ample heritage from its predecessor; for example, Grid services are defined in terms of standard WSDL with minor extensions, and rely on standard web service technologies such as SOAP and WS-Security.

However, the Grid service conventions specified in the OGSI document add further elements by addressing fundamental issues in distributed computing such as how to name, create, discover, monitor and manage the lifetime of stateful services [17].

The OGSI conventions support explicitly stateful services with lifetime management and a base set of service capabilities, including rich discovery facilities (introspection/reflection). In addition to this, they also support a two-level naming scheme similar to the DNS (Domain Name System), which is more user-friendly than a traditional WS address scheme.

REPOSITORIES

As Grid scientists began to port more and more applications to the Grid, managing the growing number of applications gradually became a burden to Grid site administrators. They therefore turned to repository technologies to structure, store and reference – i.e. better manage – Grid applications.

A *repository*, as its name implies, is used to store objects in a structured manner, following a model defined by the repository administrator. The repository provides functions for classification, storage, management and retrieval of the components stored inside it. [18]

In most cases repository frameworks differ from one another in terms of access interfaces, security modules, communication protocols and transfer protocols. Moreover, these repositories vary in terms of the repository model employed because administrators enjoy free choice in defining the repository model. Furthermore, in the case of Grid application repositories each of these repositories employs a different application description language to describe the applications stored in it.

As a result of this diversity, the field of Grid application repositories contains many open questions in the areas of interoperability between repositories, application discovery, as well as cross-operability with Grid services and services outside Grid.

For example, as Grid application repositories start to accumulate applications, users need to be able to find these applications and access the repositories which store them. However, Grid users are not interested in the particularities of Grid middleware, or repository technologies; or in differences between a Web/Grid

service invocation and a standard Grid job submission. Users just want to gain easy access to as many applications as possible and be able to use them to solve their problems. Consequently, Grid application repositories need an architecture that allows for application storage and enables interaction with other application repositories and Grid services, permitting users to discover and utilize such applications. However, current repository architectures are very restrictive and none of the Grid repository solutions used today allows for inter-connectivity with other repositories.

Another example of the restrictive design and functionality of current Grid application repositories is related to the fact that the same application can be found stored in different repositories. In theory, if one repository is unavailable users should have the opportunity to run their application from another repository to solve their problem. However, current repository solutions are not linked in any manner and there is no matchmaking service that can identify similar or identical Grid applications stored in different repositories.

1.2. Outline of the Thesis

The first stage of my research consisted of an in-depth critical analysis of the Grid application repository solutions currently used in Grid infrastructures, based on which I identified a series of shortcomings associated with these solutions. For the rest of my research I aimed to design a Grid application repository that would address these shortcomings. My aim was also to provide a modular, easily extendible solution, based on functional principles that can be followed not only by application repositories usable on Grid, but also by generic application repositories that reside in collaborating environments other than Grid.

In relation to these aims, my research was focused on pursuing four major objectives. The first objective of this research was to design a service able to connect different types of Grid application repositories, but which would still function as a Grid application repository in its own right. The second objective of this

research was to propose a new model for application repositories, which would achieve uniformity in Grid application presentation and would extend the functionality of these repositories beyond Grid. The third objective was to find (or create) an application description language, which would provide uniformity in the presentation of Grid application descriptions; while, at the same time, would allow the use of Grid application repositories and of applications stored by them in scenarios other than Grid, such as virtualisation, source code staging and compilation, or automatic application deployment. The fourth objective was to design a matchmaking methodology and an algorithm able to process information about applications stored in repositories and identify similar or identical applications.

After the specification of the four objectives, the next step in my research was to present the theoretical design and specification of a Grid application repository solution able to meet these objectives. The design phase was followed by the implementation phase, which started with the identification and careful analysis of a series of constraints that could be put in place in order to simplify the development of the solution without restricting its core functionality or its ability to meet the four objectives set out in this research.

Further on I identified the necessary technologies needed for the development of my solution and used these to implement a pilot-solution compliant with the theoretical specifications described in the design phase. After successfully completing of the implementation phase, I moved on to test the solution and analyse test results (analysis phase). As part of this phase, I selected five use-case scenarios, which were representative to prove the functionality of the new Grid application repository solution (GAMRS) and to show that this solution met the research objectives. I successful ran these scenarios and analysed the results. Following the interpretation of these results, I summarized the findings in a series of conclusions, which prove that all requirements and objectives of this research were met, as well as present a critical analysis of the limitations of this research.

The research was finalised with a summary of the contributions brought by this thesis to scientific knowledge, alongside several recommendations on how this solution can be extended through future research.

The remainder of this thesis is organized as follows:

Chapter 2 provides an overview of the main area of research – it describes the major concepts behind Grid application repositories and then offers a comprehensive review of related work from the specialty literature to describe the current state of the art in this field. Due to the complexity of the issues identified during research investigations, the area of research is divided into four topics: Grid repository architecture; Grid repository model; Grid application description language; and Grid application matchmaking system. The concepts related to each of these four topics are discussed in separate sections. The chapter continues with an overview of the challenges identified in the Grid application repository area and concludes with a statement of the objectives of this research.

Chapter 3 describes the design principles behind the solution proposed by this research in order to solve some of the challenges identified in the previous chapter. The chapter describes the Grid Application Meta-Repository System (GAMRS) – i.e. the solution-candidate proposed in this research as a new generation of Grid application repository able to meet the objectives identified in Chapter 2 – following the same division into four separate topics as the one in Chapter 2: GAMRS architecture; GAMRS repository model; GAMRS application description language; and GAMRS matchmaking service.

Chapter 4 describes the testbed implemented in order to test and prove the functionality of the Grid Application Meta-Repository System. The chapter starts with a description of the main constraints and limitations imposed to the pilot-solution (mainly by the time-constraints to which each PhD research is bound). It then moves on to describe the testbed architecture and its implementation, as well as specify five test scenarios. The chapter concludes with an analysis of the results obtained after the implementation of the test scenarios.

Chapter 5 presents the conclusions of the analysis of the testbed results, highlighting the main capabilities of the proposed solution, but also analyzing the limitations of GAMRS. The chapter concludes with an analytical overview of how the requirements and objectives set out at the beginning of this research were met.

Chapter 6 summarizes the contributions of this thesis to scientific knowledge and concludes with several suggestions on how this research could be extended.

Chapter 2

Related Work – Grid Application Repository

This chapter describes the Grid application repository solutions that are currently used in Grid infrastructures and projects in an attempt to analyse the current state of the art in this field and highlight those areas which can still be improved. This overview covers four repository aspects important to any Grid application repository: the repository architecture, the repository model, the application description language and Grid application matchmaking methods.

The investigation is focused primarily on the **architecture** of the most widely used Grid application repositories and on the **repository model** designed by peer research teams for their repository solutions. These two properties are the most important aspects of a repository and they decide most of the functional capabilities of a Grid application repository, such as: the scientific area of usage, the ability to inter-connect with similar solutions, the ability to be easily extendible, the ability to interoperate across different scientific domains, and the ability to be easily accessed by different technologies under various usage scenarios.

The applications stored in Grid application repositories are formally described by **application description languages**. Every application description language is capable of describing a set of properties associated with a Grid application. Most description languages usually refer to a common subset of application features - even though the naming scheme of the formal attributes may differ. However, apart from this common set, each application description language is capable of

modelling particular application properties that will not to be formalized by the other description languages. This chapter presents an overview of the most important application description languages currently used in Grid solutions. Different subsets of Grid application properties decide which usage scenario the application can be employed in and implicitly, they have an impact on those scenarios in which the Grid application repository storing the application can be involved. Therefore, the relation between the description capabilities of an application description language used to describe a certain Grid application and the repository that holds that application can restrict the repository's areas of usage.

Since Grid application repositories store applications and application-related objects, they represent the first choice for end-users and services in the discovery and usage of applications. Furthermore, the same application can be found stored in different repositories. In this case, although different repository models may associate different metadata and different application-related objects to the application, in principle they model the same application. Therefore, Grid application repositories can be subject to matchmaking systems that are looking for similar applications. However, these matchmaking systems are dependent on the amount of information they find in the repository, as well as on the quality and the formal structure of such information regarding the application and application-related objects. This chapter presents a short description of existing **matchmaking methods and solutions** that can be applied to the content of Grid application repositories in order to find similar Grid applications.

The remaining of this chapter continues with the investigation of existing Grid application repository solutions, covering the four repository aspects mentioned above: repository architecture, repository model, application description language and Grid application matchmaking methods. Each aspect is covered in its own section which includes a critical analysis of the related work and conclusions. Following the analysis of the current Grid application repository solutions, as well as taking in consideration the evolution of Grid and other distributed computing paradigms, the chapter continues with a list of requirements **(R1-R4)** that should be met by solutions such as the one suggested in the remainder of this thesis. Next,

this chapter presents a summary of current Grid repository solutions' capabilities and how these comply with the requirements identified by this research.

The chapter concludes with a list of four objectives **(O1-O4)** aimed to deliver a new application repository solution that would provide the functionality defined by requirements **R1-R4**.

2.1. Architecture

This section presents the particularities of the architecture of the Grid application repository. It starts with the description of the traditional repository architecture and highlights its functional modules by explaining step-by-step the most common usage scenario of the repository on Grid. The section continues with the description of the most widely-used application repository solutions that exist in Grid infrastructures today. The final part of the section outlines those questions and issues that remain unresolved in the area of Grid application repository architectures.

2.1.1 General Overview of Grid Application Repository Architectures

As Grid application repositories start to accumulate applications, users become more and more interested in how to find these repositories and access the applications stored there. However, one needs to keep in mind that the users in discussion are no Grid specialists and do not hold an extensive knowledge of computer science. In most cases, they come from other areas of science such as bio-sciences, medicine, physics, mathematics, etc. Furthermore, they are not interested in the underlying Grid middleware, in repository technologies, or in differences between a Web/Grid service invocation and a standard Grid job submission (i.e. accessing a Grid application through the standard client-server

way). Users just want to have access to as many applications as possible and be able to use them to solve their problems.

Consequently, Grid application repositories need an architecture that allows for application storage and enables interaction with other services, enabling users to discover and utilize such applications.

The following figure (Figure 2-1) offers an overview of the architecture common to all major Grid systems that expose Grid application repository services to users.

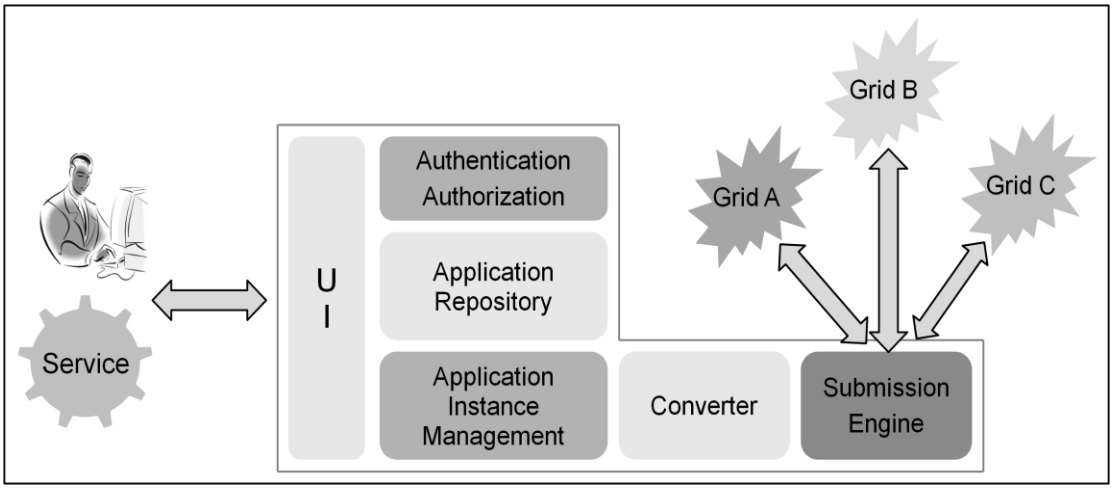


Figure 2-1: Traditional architecture of a Grid system that includes the application repository

The system exposes a *User Interface* (UI) module to users in order to allow them to interact with the system. Next, the *Authentication and Authorization* module authenticates Grid users onto the system and allows/denies them different interactions with the other modules, according to the specific security policy in force.

The *Application Repository* module is usually an implementation of repository software with all the functionalities provided by such a technology (e.g. data classification according to the captured metadata, storage/data management, indexing capabilities, access to the data stored on it, search functions etc.) The repository model implemented in the Application Repository usually captures metadata, such as *user*, *identifier*, *version*, *date of creation*, *application description*, *security*. However, each repository stores the application description in an Application Description Language different from other repositories. For example,

GEMLCA [19, 20] repository uses LCID [20], NGS application repository [21] uses JSDL [22], and myExperiment repository [23, 24] uses Scufl [7].

The architecture description continues with the *Application Instance Management* module. This module handles the set of actions needed by the system to: create a Grid application instance; format the application description into a language supported by the *Submission engine*; submit and monitor the run of the application instance; and retrieve the results of the run. All technology-specific actions are hidden to the user and come under the management of this module.

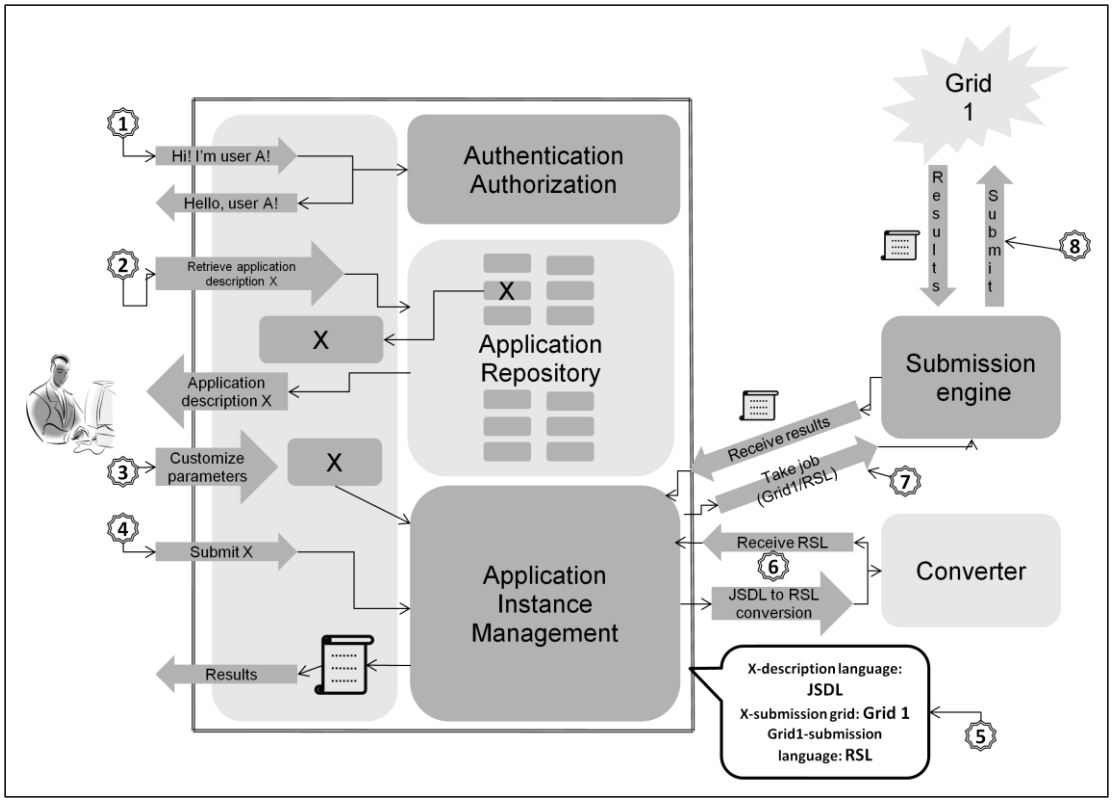


Figure 2-2: Traditional Grid application repository: usage scenario

A common user scenario (Figure 2-2) finds the user authenticating in the system (action 1), choosing an application from the *Application Repository* (action 2), customizing the necessary parameters in the application description document (action 3) and submitting the application (action 4). At this point, either the user waits for the results or, in some cases, monitoring data is returned to the user to update him/her on the state of that application instance. Next, the *Converter* module converts the application description document (actions 5 and 6) from the

language used to describe the application at the higher level (such as JSDL – Job Submission Description Language, or LCID – Legacy Code Interface Description) into an application submission language that can be processed by various Grid submission engines (such as JDL – Job Description Language [25, 26], RSL – Resource Specification Language [27] or xRSL – Extended Resource Specification Language [28]).

The *Application Instance Management* module manages the actions required to authenticate the user on the Grid; it delegates credentials to other services involved in the process; and monitors the entire process. The *Submission Engine* is then responsible for Grid resource/service identification and actual job submission/result retrieval from the underlying Grid infrastructure. Once the conversion from description into submission language is performed, the Application Instance Management module passes the document to the Submission Engine (action 7), which connects to the Grid infrastructure, chooses a suitable resource that matches the job requirements, submits the job and retrieves the results (action 8). The results are finally passed back to the user.

2.1.2 Review of Existing Solutions

BERKELEY DATABASE INFORMATION INDEX (BDII)

BDII [29] is employed as a central information system in gLite/lcg-based Grids. It stores information about resources commissioned to Grid by each site (i.e. a Grid site refers to a number of Grid resources grouped together under the same administration), as well as about the Grid applications installed on them. BDII uses the GLUE schema [30] as description model for these objects.

Communication with the clients occurs via LDAP (Lightweight Directory Access Protocol) commands [31, 32], whether they come from command line or more user-friendly interfaces. The BDII is highly available since it exposes a standard

communication API (i.e. LDAP-query), which means that any LDAP client can be used to connect to the BDII. However, users are required to know the GLUE schema in order to perform relevant searches and understand the results of these searches. Moreover, the way to retrieve relevant and usable information from the BDII is not straightforward to the average user, who needs specialist training and knowledge of GLUE in order to make any search process productive. BDII also lacks any intuitive user interface for publishing applications – the publishing process is usually done by administrators via command-line.

By exposing only the LDAP interface, BDII does not interact either with popular web search engines, or with OGSI/WSRF-compliant Grid Services. In order to access the BDII, a service usually needs to embed an LDAP-client into its code. However, neither the service nor the BDII can make use of the OGSI/WSRF protocols stack – this would prove very useful, as actions performed on application entries in the BDII, such as addition, removal, and modification could be automatically advertised to other Grid services. Furthermore, BDII has no connection to any other Grid application repository – neither in gLite/lcg-based Grids infrastructures nor in any other Grids. BDII is also not able to find similar applications and only provides users with a basic search engine that can perform queries on application metadata values.

Furthermore, BDII was not built on a repository technology able to exchange and reuse objects and employs no communication protocol that could do that. BDII was created and is used exclusively on gLite/lcg-based Grids. The GLUE schema lacks the capability to describe application-related objects and BDII repository implementations are not able to store any application-related object.

BDII is one of the oldest examples of an application repository that can be found on Grid. As such, it was meant to provide only storage for information about an object according to a model (i.e. GLUE schema), the retrieval of that information, and the possibility of searching through metadata. However it lacks any further capabilities, as detailed above.

CHARON EXTENSION LAYER / INTERACTIVE SOFTWARE REPOSITORY (CHARON/ISOFTREPO)

CHARON Extension Layer (simply CHARON or CEL) [33, 34] provides uniform and simple tools for job submission and management in various computer environments such as clusters of computers or Grid environments. The CHARON system is currently available at the National Centre for Biomolecular Research CZ (TROLLCluster and WOLFCluster), at the METACentrum (Czech national Grid project) and at VOCE-UI (Virtual Organization for Central Europe). One of the modules comprised in the CHARON architecture is a software repository called iSoftrepo (Interactive Software Repository), which is used to store information about Grid sites and the applications that run on them.

The capabilities of CHARON/iSoftrepo are similar to those exhibited by BDII, with the exception that the former provides further means of categorizing applications, such as breaking them down into categories following areas of applicability – e.g. Molecular Mechanics and Dynamics, Conversion and Analysis, Visualization, Nuclear Magnetic Resonance, etc. The CHARON/iSoftrepo software package also comes with a collection of static web pages where administrators can manually enter the information about the applications stored in the repository. This makes the application visible to web search engines through page-links, but with the obvious drawback that any change in the application description or any new application added to the repository requires the manual intervention of administrators to change the CHARON/iSoftrepo web pages.

With regard to application-related objects stored on the repository, CHARON/iSoftrepo only captures information about Grid applications on its proprietary repository model but cannot store any application-related objects. Furthermore, like BDII, it is used exclusively on gLite/lcg-based Grids and has no means to interact with other CHARON/iSoftrepo instances.

GRID EXECUTION MANAGEMENT FOR LEGACY CODE ARCHITECTURE (GEMLCA)

The GEMLCA system enables deployment of legacy code applications as Grid services without the need for code re-engineering or access to the source files. With GEMLCA, running a legacy application from a standard Grid service client only requires a user-level understanding of the system. The legacy code runs in its native environment and uses the GEMLCA resource layer to communicate with the Grid client, thus hiding the legacy nature of the application and presenting it as a Grid service. [35]

GEMLCA is capable to store legacy codes descriptions into a repository. While BDII and CHARON/iSoftrepo structured and stored minimal information about the application itself in their repository models, GEMLCA is the first example of a Grid application repository that can store application-related objects. GEMLCA employs an application description language (i.e. LCID) and application description documents written in LCID can be stored in GEMLCA repositories. Moreover, GEMLCA benefits from a very friendly graphical user interface (GUI) which has been developed as a JSR-168-compliant portlet [36] in the P-GRADE portal. From there users and administrators can easily publish new applications because the portlet layout and the LCID metadata naming scheme are intuitive enough not to require a thorough knowledge of LCID on behalf of the user.

The GEMLCA system is built as an OGSI/WSRF-compliant Grid service, which makes it fully interoperable with other Grid services. However, GEMLCA was built as a Grid service and exposes only a Grid service interface; it does not publish information about the applications contained in its repository on any web page, it has no HTTP/REST [115, 116] API that could be used by web crawlers and no support for OAI-PMH [37] protocol that could be used by harvesters and popular search engines. Further on, GEMLCA has no connection to any other Grid application repository. Surprisingly, at this stage there is no communication even between two GEMLCA systems, although this can be done effortlessly by making use of their already-implemented Grid service interfaces.

GEMLCA is the first application repository from the list of repositories under discussion which stores application-related objects. Moreover, beside LCID application description documents, GEMLCA is also capable to store the application binary. This is the first step towards making Grid application repositories more versatile, so that they allow the application to be used in scenarios other than only in conjunction with traditional Grid architectures.

NATIONAL GRID SERVICE APPLICATION REPOSITORY (NGS AR)

The NGS Applications Repository is a Grid portal solution developed in accordance with the JSR-168 standard, which employs a repository module for storing Grid applications. NGS AR provides users with a list of applications available on the NGS Grid [38, 39]; it allows users to parameterize instances of those applications and to run these instances on NGS resources. Applications found on the NGS repository are described using JSDL documents and users can select and save JSDL application descriptions into their own personal space on the repository for subsequent modification and personal configuration.

While the NGS Applications repository provides a considerably richer user interface and a larger set of functionalities than BDII or the CHARON/iSoftrepo systems, in many cases its JSR-168 based implementation proves to be a limitation. Although this approach allows the portal to be distributed and hosted in project-specific portal containers, it limits the access interface to a graphical user interface that is intended exclusively for humans. Therefore, the NGS Applications repository is neither compliant with the OGSI/WSRF-standards stack, nor does it provide any HTTP/REST API. As a consequence, this system can only be accessed by human users who know the location of the portal. Furthermore, while the NGS Application Repository model does allow application-related objects to be stored on or referenced by the repository, these are all stored under the attribute *Application Associated Files / Links,* which makes them indistinguishable for automatic retrieval. At the same time, even human users can have trouble distinguishing

between user documentation, source code or binary, given that the system lacks a structured naming scheme that could be followed by publishers.

GRIMOIRES FRAMEWORK

One of the methods of exposing Grid applications to users (as mentioned in Section 1.1) is with the help of web services technology. Using open standards such as XML, SOAP, WDSL and UDDI, WS can help achieve resource sharing and service sharing in the Grid environment in the form of web based services. References to these services are usually kept into a *registry*. Clients can query the registry and find the location of the web service, as well as the location of the service description. The description of the web service (in WSDL format) holds the patterns of all requests and responses necessary to communicate with the web service. GRIMOIRES is an example of such registry which is used to refer to applications exposed as services on Grid infrastructures.

GRIMOIRES is an UDDIv2-compliant registry for web services that has the ability to augment interfaces with metadata such as functionality, semantic information about their inputs and outputs, or various metrics (e.g. perceived quality of service, trust). [40] In addition to the UDDIv2 interface, the GRIMOIRES framework also provides some other interfaces, such as a metadata interface and a WSDL interface, which allow clients to publish and inquire over metadata and WSDL-related data, respectively.

As this framework was designed to be used on Grid, it employs a GSI (Grid Security Infrastructure)-based [41] authentication system as well as a fine-grained access control for each published entity that is based on the X509 Distinguished Name (X509DN) extracted from the certificate corresponding to the signature. When deployed in an environment supporting OGSI/WSRF [42], GRIMOIRES is able to expose registry entities (such as businesses and services) as WS-resources. Consequently, WSRF standard operations (e.g. using XPath [43] to query resource properties or subscriptions for notifications written according to the WS-Notification [44]) can be used to operate on registry entities. Furthermore, the

lifetime of registry entities are managed according to WS-ResourceLifetime [45] and GRIMOIRES can be implemented as a Grid OGSI/WSRF service. However, GRIMOIRES is a registry technology; it cannot accommodate any application-related objects, except references to applications exposed as web services or Grid services.

MYEXPERIMENT REPOSITORY

myExperiment is a collaborative environment where scientists can publish and share their workflows notwithstanding the scientific areas they belong to. The *myexperiment.org* social web site is used by "thousands of users ranging from life sciences and chemistry to social statistics and information retrieval". [46]

Although myExperiment repository can in principle accommodate workflows designed for various workflow engines (such as Taverna [11], Kepler [13], Triana [14], Trident [47] etc.), at the time this research was carried out, the repository only had the capability of running and analyzing Taverna workflows. Currently, out of all major Grid application repositories integrated in Grid, the myExperiment repository stores the largest number of Grid applications (i.e. approx. 635 in July 2009). This can be regarded as a direct consequence of the HTTP/REST interface, which makes the repository and the applications stored on it visible on the Web. However, myExperiment has no means to access other types of repositories or the applications stored on them. Moreover, it exposes no OGSI/WSRF-compliant interface, which makes it unusable by other standard Grid services.

Similar to GEMLCA and NGS AR, myExperiment benefits from a rich and user-friendly access interface that allows users and administrators to easily publish and find applications stored in the repository. Currently myExperiment does not support the exchange of repository objects in a standard way. However, the myExperiment development team plans to migrate to an Open repository solution able to support the OAI-ORE standard.

Like the NGS Application Repository, the myExperiment model allows application-related objects to be stored or referenced inside the repository. However, it follows

the same pattern as NGS AR and uses only two attributes (i.e. *Files* and *Packs*) for their reference. Again, this approach limits drastically the ability to distinguish between application-related objects, and impedes automatic retrieval of specific objects.

2.1.3 Conclusions

Other Grid application repository architectures exist, such as EGEE Application Repository [48], EDGeS Application Repository [49], g-Eclipse Workflow Builder [50], gUSE Repository [15], as well as Grid Service registries: Lattice registry [51], D-Grid registry [52] and ARCS registry [53]. However these are very similar to the solutions already described in the previous section (2.1.2). Therefore, they were omitted from this discussion, since their architecture does not bring any particular novelty elements in terms of repository architectures compared to the ones already presented above.

This critical analysis started with an overview of the oldest Grid application repository solution - BDII, which simply stores information about resources commissioned to Grid by each site and about the Grid applications installed on them. However, BDII is not able to store any application-related object and lacks an intuitive user interface for publishing applications. BDII interacts with neither popular web search engines nor OGSI/WSRF Grid Services and has no connection to any other repository – neither in gLite/lcg-based Grids infrastructures nor on any other Grids.

CHARON/iSoftrepo marks an improvement in this respect, as it comes with a collection of static web pages where administrators can manually enter the information about the applications stored in the repository. This makes the application visible to web search engines but with the obvious drawback that any change in the application description or any new application added to the repository requires manual intervention from administrators.

GEMLCA, NGS AR and myExperiment expose user-friendly graphical interfaces, which makes them easily accessible both by Grid-knowledgeable users and by non-Grid users. Notably, GEMLCA highlights the importance of interoperability with OGSI/WSRF Grid Services.

An excellent example in highlighting the importance of application exposure to web is the myExperiment repository which employs a HTTP/REST interface. The ability to find information about Grid applications straight through popular search engines, combined with myExperiment's intuitive and user-friendly interface made this system very popular – as shown by the increasing number of myExperiment users (e.g. 1000 registered users in July 2008, 2478 registered users in July 2009) and the growing number of applications (e.g. 321 registered applications in July 2008, 635 registered applications in July 2009) stored in the repository.

Also, in terms of the ability of exchanging repository objects, myExperiment is the only repository which intends to migrate to a technology, which comes with support for protocols that permit the exchange of objects between repositories. However, even myExperiment lacks support for metadata harvesting clients (i.e. services that collect the metadata descriptions of the objects in the repository so that other services can be built using such metadata), thus limiting the visibility of the objects stored in the repository and reducing the number of scenarios in which myExperiment can be involved.

In conclusion, Grid application repositories are currently not connected in any structured manner as a straightforward consequence of the different repository frameworks they are built on. Moreover, at this stage there is no service that users or other services can inquire to find whether a desired Grid application is stored on any of these repositories.

2.2. Grid Application Repository Models

Repository models are a formal way to structure information and to describe relations between the objects stored in a repository. When applied to Grid application repositories, these models usually refer to the following entities and the relations between them: users; applications and application-related objects; and security policies.

2.2.1 General Overview of Grid Application Repository Models

Since in most cases the technology used for repository implementations is generic and imposes little or no restrictions on what administrators define in their models, these models vary significantly from one repository to another, even though they may refer to similar objects. Besides the metadata associated with users and security policies, little information can be found in these models about the actual application. Only the following information is usually common to these: *application name; application version; the creator of the application description document; a free-text description of the application; creation date; last modification date;* and particular fields used internally by the system such as *universal identification, modification history, usage statistics etc*.

Moreover, Grid application repository models refer to application description documents as objects stored in the repository. The exceptions which allow additional objects to be stored besides application descriptions (i.e. myExperiment, NGS AR, GEMLCA) make no formal distinction between the types of objects that can be added to the repository. This is a consequence of the fact that such repositories are usually tightly coupled with one particular Grid and they are running on platforms well-known to their administrators. Therefore, apart from the case when the application is exposed as a service, the applications exist only in their binary form and are either already-deployed on the execution sites or they can be

staged there from storage facilities *other* than the repository itself. This implies that scenarios involving prerequisite application-related actions such as getting the source code, compiling the code, solving licensing issues, solving software dependencies and application deployment are done by Grid site administrators without any help from what is stored in the repository.

2.2.2 Review of Existing Solutions

NGS AR REPOSITORY MODEL

The NGS AR stores application descriptions written in JSDL – therefore the repository model employed in NGS AR is designed in accordance with the description capabilities of JSDL. Figure 2-3 depicts the repository structure responsible with modelling information about a given application.

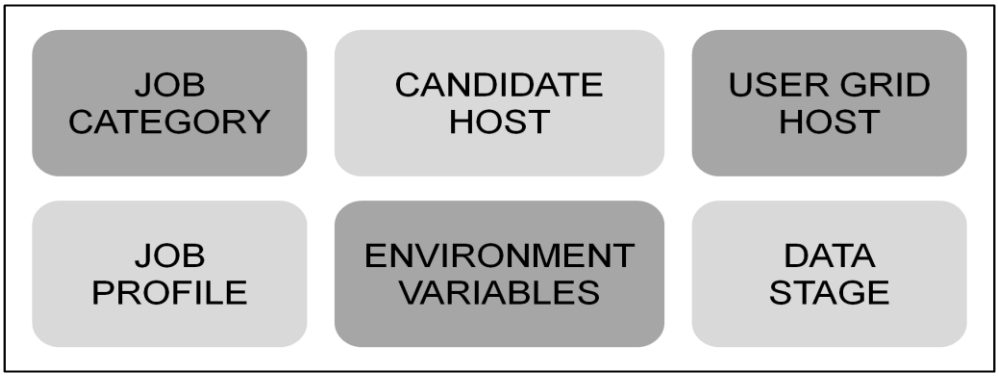


Figure 2-3: NGS AR repository model

The application properties and application-related entities modelled by the NGS AR are derived from the formal structure of the JSDL. For example, the *candidate host* entity, along with the *environment variables* and *data stage* entities, describe the information retrieved from the following JSDL sections: *Resource/CandidateHosts* element; *POSIXApplication/Environment* element; *DataStaging/Source* and *DataStaging/Target* elements. *Candidate host* entities describe the information about the computational resources on which the application can run; the *environment variables* entities refer to the operating system variables that need to be used or to be set to particular values in order to allow the application to run in that environment; the *data stage* entity describes the input and output files that

need to be retrieved or uploaded from/to various storage resources available in NGS Grid.

The NGS AR is part of a system, which not only stores application descriptions, but is also able to instantiate those applications and run them on Grid. This system is coupled with a GridSAM submission engine that is able to provide job submission interfaces ”for submitting computational jobs to many commonly used distributed resource management systems (Condor, PBS, SGE, etc.)”.[54] Theoretically, GridSAM allows the system to be connected to different Grids, hence the *user Grid host* entity in the repository model, which allows users to choose resources located in any of the Grids connected to GridSAM. However, the NGS Grid is currently the only one connected to the system, so the running of any application is effectively restricted to NGS Grid.

The NGS repository model allows applications to be classified in different categories under the attribute *job category*. The model also allows the user to define personal user categories apart from the ones pre-defined by the NGS repository administrators, which are: Tutorials/Examples, Engineering, Bioinformatics, Analysis/Stats, Biomedical, Chemistry, Astrophysics, Image Analysis and CCPb Workshop.

The security policy of the NGS AR is based on Public Key Infrastructure (PKI) X509 certificates and the identity of repository users is established on the *Distinguished Name* attribute of the certificate. The repository stores this information in relation with a *user* entity described by its model. Security policies are basic and they model particular actions associated with two roles: the *administrator* and the *regular user*. As opposed to the regular user, administrators can create/delete/modify new users, new NGS categories and new NGS applications.

The *Job Profile* entity from the NGS AR model gathers all information about an application under one reference and functions as a container for the other entities in the model.

The NGS AR is able to store application-related objects in its repository. However, all these objects are stored under the generic entity *Files/Links*, which makes these objects indistinguishable for automatic retrieval. The repository model is not able to differentiate between different types of application-related objects and furthermore, the formal description document of the application (written in JSDL) is not stored as a whole document in the repository. The next repository model takes a different approach to this matter.

GEMLCA REPOSITORY MODEL

As opposed to the NGS AR, in which JSDL played a central role in the design of the repository model, GEMLCA is not dependent on the formal structure of LCID – the application language used for application descriptions stored in the repository. The repository model consists of five entities necessary for the GEMLCA system to operate (see Figure 2-4): *the user profile; the job information; legacy code environment variables; Grid site information;* and *the Grid Backend profile.*

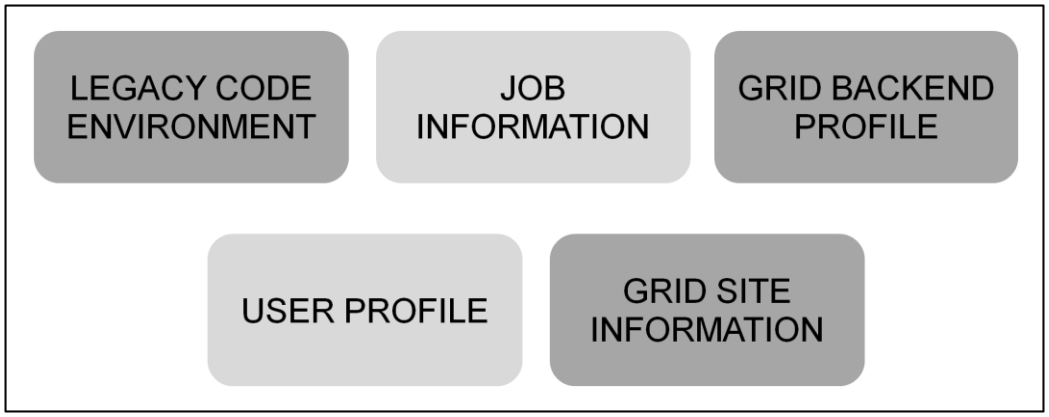


Figure 2-4: GEMLCA repository model

Similar to NGS AR, the security policy of the GEMLCA system is based on PKI (Public Key Infrastructure) X509 certificates and the *Distinguished Name* attribute is used to establish the identity of users. This information is stored in the *user profile* entity along with personal information such as name, affiliation, address and email. The *job information* entity refers to its LCID description, including the state in which a submitted application instance (i.e. job) finds itself in, such as: *submitted, queued, running, done, failed* or *cancelled.* The *legacy code environment* specifies the

operating system variables that are needed in order to ensure a correct run of the application instance.

While the NGS repository was part of a system which used GridSAM as submission engine, GEMLCA has its own *submission backends* that can submit jobs to different Grid infrastructures (e.g. GT2, GT4, gLite/lcg). As production GEMLCA systems do connect to two or more Grids, the backend specifics are described by the repository model in *Grid backend profile* entities. Finally, the *Grid site information* entity is equivalent with the *candidate host* entity from the NGS AR model and describes the computational resource where the application is set to run.

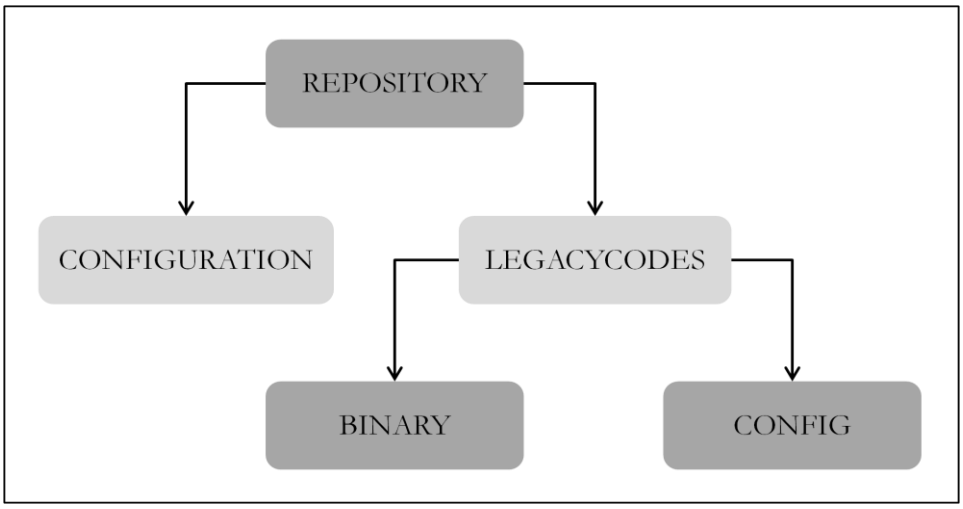


Figure 2-5: GEMLCA storage structure

The GEMLCA storage structure (see Figure 2-5), allows for application *binaries* and application descriptions files (*code config*) to be stored in the repository. By doing this, GEMLCA allows these objects to be used independently in different scenarios, which do not necessarily involve running the application on Grid infrastructures.

CHARON/ISOFTREPO REPOSITORY MODEL

It was mentioned in section 2.1.2 of this chapter that the CHARON/iSoftrepo repository is used on gLite/lcg based Grids (e.g. EGEE, SEE-Grid, EELA Grid, EUMedGrid, EU-India Grid, EUChinaGrid, Baltic-Grid II). Particular to these Grids is

the fact that application binaries cannot be staged and installed on demand on the infrastructure – they are already deployed and can only be run on the sites which expose them. However, CHARON/iSoftrepo administrators understood that a Grid application repository might be used in other scenarios (i.e. not only in connection to Grid) and designed a model, which actually exceeds the requirements of a gLite/lcg Grid.

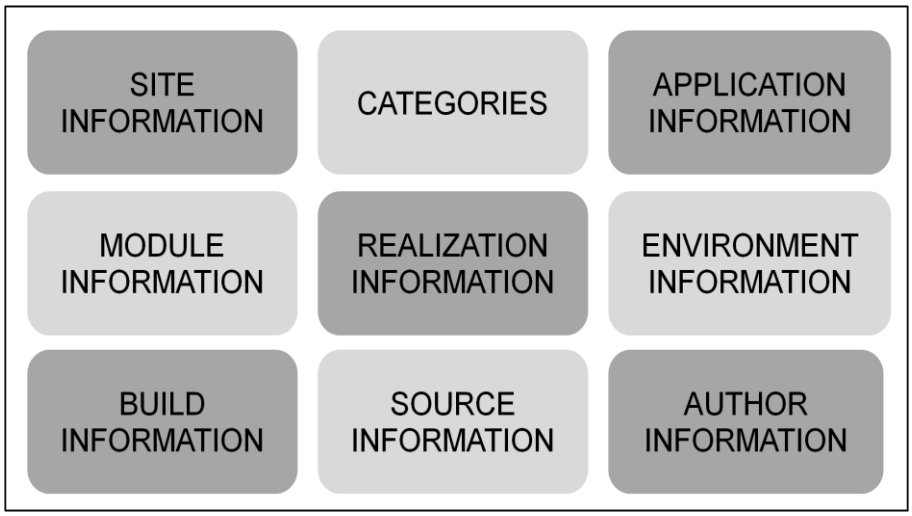


Figure 2-6: CHARON/iSoftrepo repository model

Similar to NGS AR and GEMLCA, CHARON/iSoftrepo model (see Figure 2-6) contains entities, which describe users (*author information*); categories of applications (*categories*); the computational resources used to run the application instances (*site information*); the application description (*application information*); and the operating system environment variables required to run the application (*environment information*). In addition to that, the repository can store the source code in a *tar.gz* archive (*source information*), as well as information about the compiler distributions used to build the runtime executables of the application (*build information*).

Grid infrastructures that support gLite/lcg middleware are Linux-based and therefore a common technique for the dynamic modification of a user's environment is via *modulefiles*. Typically each application needs one or more environment variables to get specific values, while the *modulefiles* approach is to instruct the Linux *module* command to modify or set shell environment variables (such as *path,*

ld_library_path, cc, manpath, etc.) with the necessary values. The CHARON/iSoftrepo repository model captures the information about the *modulefile* associated with a Grid application in the *module information* entity. In cases when different versions of the same application exist on the same system, they are usually deployed as different modules. However, it is customary in such cases that the *modulefile* associated with one version of the application contains references to the *modulefile* associated with the other version of the application, since they use the same values for a common subset of environment variables. This is true not only in the case of different versions of the same application, but also when different applications depend on each other. The CHARON/iSoftrepo model captures the information about the version of an application along with the possible dependencies of that application on other modules in the *realization information* entity.

MYEXPERIMENT REPOSITORY MODEL

As stated in section 2.1.2, myExperiment represents a collaborative environment built around a repository of scientific workflows. Consequently, in order to emphasize the idea of collaboration, the repository model was designed to describe not only relations between users and applications or between applications and their related objects, but also user-to-user relations.

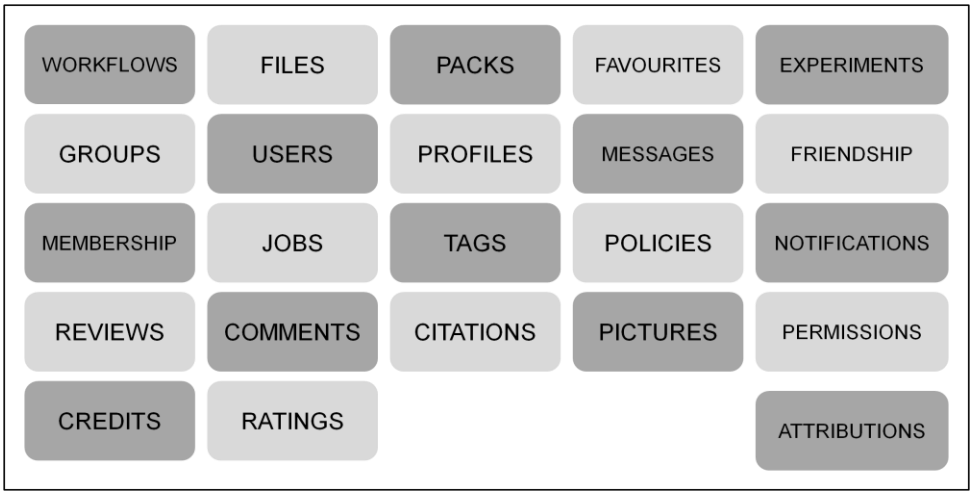


Figure 2-7: myExperiment repository model

As seen in Figure 2-7, myExperiment's repository model focuses extensively on interactions between different users of the system. Entities like *friendship, messages, reviews, comments, citations, pictures, credits* and *ratings* make myExperiment not just an application repository but also a social networking system.

The system keeps the list of users in a *users* entity and each user's description is modelled in a *profile* entity. Users can be grouped together in *groups*, while security policies, access policies and user roles are described via the following entities: *membership*, *policies, permissions and attributions.* Moreover, users can define access policies and can enforce permissions on the applications they own. However, these policies cannot be more permissive than the policy assigned by the system administrator to that particular class of entities.

The myExperiment model describes applications as *workflows* suitable for various submission engines (such as Taverna, Triana, Kepler, etc.). Multiple workflows can be grouped together under the same *experiment* and these workflows can have additional application-related objects stored in the repository. However, the model does not have the capability to distinguish between different application-related objects as it categorizes them in only two classes: *files* and *packs.*

myExperiment is integrated with a Taverna submission engine; therefore, users can run applications (i.e. as Grid *jobs*) on Grid and the system will use *notifications* to update users on the progress of the application running process. The myExperiment repository model allows users to mark their *favourite* applications and permits assignation of application *tags* that help the search process by giving a better categorization of applications.

The myExperiment repository model is much more complex than all the models currently used in production Grid application repositories. This comes as a consequence of the fact that this model was built to satisfy the design requirements of an entire social network system, not only those of an application repository. Nonetheless, the ability to model user-to-user relations as well as the ability to support fine-grained user policies permit application sharing and encourage

collaboration between users. These elements should represent an inspiration for designing future Grid application repository models, especially since the entire Grid is based on the idea of collaboration and sharing.

2.2.3 Conclusions

Other Grid application repository models do exist – GLUE, gUSE, EDGeS, GRIMOIRES, EGEE – but they are similar to at least one of the solutions already described in the previous section (2.2.2). They were therefore omitted from this discussion as they do not bring any further innovation in terms of repository models.

When describing Grid applications, the repository model usually contains information such as the application name, the version, a free-text description of the application, the environment variables necessary to the application to run correctly and the computing resource(s) where the application is set to run. Occasionally, the applications are grouped in categories depending on their scientific domain of applicability.

In cases such as GEMLCA and NGS AR, in which the repositories are part of systems that employ heterogeneous or generic submission engines (i.e. able to use Grid infrastructures based on different middleware), the models were designed to accommodate information about such submission engines. Moreover, such models can capture certain particularities of Grid infrastructures (such as the underlying technology used as middleware, or the application description language used for submission), which are used at submission stage to ensure a correct processing of the application instances.

myExperiment gives a very good solution for exposing information about users, security policies, access permissions, as well as relations between different objects captured in the repository model. This is mainly because the entire myExperiment system is designed to support a collaborative environment and its primary focus is therefore on modelling user-to-user relations.

In terms of making Grid application repositories more versatile and expanding their domain of usage, GEMLCA, NGS AR and myExperiment can store not only application descriptions, but also other application-related objects. However, the attributes used for their reference make the application objects indistinguishable for automatic retrieval. Without a structured naming scheme to be followed by publishers, even human users can have difficulties distinguishing between different application-related objects stored in repository. This approach limits the possibilities to discover specific application-related objects and impedes automatic retrieval of these objects.

2.3. Grid Application Repository Application Description Languages

Application description languages are formal ways to describe applications. These languages are the result of various investigations into the relationship between the *Application* and the *Grid infrastructure/services* and they address a list of features needed to make the application visible and usable on Grid. Unfortunately, there is no single Grid application description language, but several of them, since different scientific research teams have put effort into finding their own way to describe Grid applications.

2.3.1 General Overview of Application Description Languages

Following the timeline from the beginnings of Grid application description languages up to present days, Grid application description requirements can be divided in two categories: *basic requirements* and *advanced requirements*.

BASIC REQUIREMENTS

In order to make the application usable on Grid these description languages needed to be able to describe several basic features of the application (*basic requirements*). Given that Grid is a distributed environment, applications reside on different resources scattered throughout the Grid participating sites. In order to run the application they need, users have to know the *name*, the *version* and the *executable path* of the software they want to use, and these key attributes are a must for every language. Furthermore, they need to describe the remote environment (such as the *working directory* or the *environment variables* needed for the application to run), as well as the location of the *default input files* and the *arguments* that need to be passed on to the application.

Another consequence of the fact that Grid is a distributed environment is that the application description language needs to be able to address remote *file staging* because the executable and/or default input files may reside on resources other than the one chosen for execution.

Given that the Grid architecture enables users to run parallel applications, this issue needs to be addressed as well. Therefore, at the basic requirements level, any description language needs to be able to specify at least the *number of processors* required for the application to run.

Although Grid is quite resourceful in terms of computing and storage power, it is comprised of various sites, which offer Grid users a very diverse panel of resources (e.g. different machine architectures, different operating systems, different usage policies, different CPU/memory/disk limitations, etc.). Consequently, an application description language is also used to specify the *memory/disk/CPU/etc.* requirements needed for the application to run. These requirements are written in a document that is passed on to the execution site. If the execution site policies are met, the execution will be allowed on the site's resources.

As Grid evolved, users began to have more expectations from the way Grid handled the applications and consequently, the capabilities of application

description languages evolved as well. On one hand, newer technologies emerged and were adopted by Grid (Grid/Web services) and on the other hand, applications grew in complexity, demanding finer grained descriptions from description languages.

ADVANCED REQUIREMENTS

To cope with the new requirements, description languages had to be improved with newer capabilities (*advanced requirements*). First, resource requirements were fine-tuned to allow specific memory/disk/CPU/network requirements to be specified (for example: *core dump size limits*, *virtual memory requirements*, *pipe size limits*, *minimum network bandwidth requirements,* or *open file descriptors limit*). Moreover, at the beginnings of Grid, file staging usually occurred with the help of two transfer protocols, namely *gridftp* [55] and *rfiod* (UNIX remote file access daemon). As more file *transfer protocols* were adopted by Grid (such as SRB [56] or srm [57]), description languages had to make room for them in their schema as well. Later on, as *service technologies* (i.e. Web services and OGSI/WSRF Grid services) were adopted by Grid, the application description language schema had to be modified to accommodate descriptions of such technologies. Also, as applications grew in complexity, the default configuration of such applications required increasingly more knowledge from users. Consequently, administrators had to create a *template* for the application: in some cases by supplying values (unchangeable by users) for certain parameters; or, in other cases, by specifically asking users for mandatory input of certain application arguments. The idea of a *template* was to provide users with the set of parameter values necessary for a correct run of the application (for example, in order to run accurately, applications may demand a certain machine architecture, a certain OS, a minimum amount of memory, or a specific value for an environmental variable). Implicitly, it reduces the complexity and the amount of knowledge a regular user is required to have about the Grid infrastructure or about the application set up in a particular Grid environment. As the notion of application *template* began to gain terrain in the past few years (e.g. GEMLCA, gUSE) description languages also needed to reflect this trend accordingly. Finally, while at the beginnings of Grid parallelism was addressed only in terms of number of

processors and Grid parallel computing referred mainly to cluster jobs such as MPI jobs, as time went on, description languages had to be extended to be able to express other forms of parallelism, such as *parameter sweep* applications and *multi-process* applications.

To summarize, a minimal set of functionalities that any application description language is expected to implement in order to comply with current Grid requirements today includes:

- **Legacy compatibility** – Any new application description language should implement all the basic application description requirements to ensure full backwards compatibility with all previous solutions.
- **Advanced features** – On top of the basic features, the description language should be able to describe complex application/resource specification as well as complex data-staging. Furthermore, this language must be able to accommodate Web and Grid service technologies; therefore its schema must permit service-like entries in it (e.g. service endpoints).
- **Advanced parallel behaviour description** – Any new application description language should be able to express advanced parallel capabilities such as parameter sweep and multi-process applications.

2.3.2 Review of Existing Solutions

This section brings into discussion the most important application description languages used on existent production Grid middleware, including: RSL, JDL, xRSL, WS-GRAM RSL, LCID and JSDL.

RESOURCE SPECIFICATION LANGUAGE (RSL)

RSL [27] is one of the first application description languages that emerged for Grid. It was developed by the Globus team [58] along with the GT2 Grid middleware [59]

and other Grid specific tools. Since it came packaged with one of the widely spread Grid middleware like GT2, RSL became the first choice of description language for many researchers and users. As such, it can be regarded as a landmark for the evolution of application description languages, since it set up the basic application description requirements for languages to come.

With regard to the first requirement (**Legacy compatibility**), RSL represents the legacy in discussion and defines and implements the basic application description requirements. However, since it is so old, **Advanced features** are almost non-existent. Users cannot specify features like: file size limit, open descriptors limit, process count, machine architecture, OS requirements, CPU requirements, virtual memory requirements, or file system description etc. In terms of its **Advanced parallel behaviour description** capabilities, RSL allows users to describe parallel jobs and MPI [60] jobs through the following parameters: *jobType* and *hostCount*, but has no support for parameter sweep jobs or multi-process jobs (e.g. no thread limit or number of processes limit).

JOB DESCRIPTION LANGUAGE (JDL)

Another legacy application description language, JDL [25, 26] is used on gLite/lcg middleware-based Grids. Similar to RSL, JDL implements all the basic application description requirements (**Legacy compatibility)**, but does not natively implement the **Advanced features**. However, it supports BDII constraints into its schema. The BDII schema improved over time and nowadays allows site administrators to add important metadata to their resources (such as machine architecture, OS type, number of CPUs, memory, disk size etc). JDL therefore indirectly supports a large part of the **Advanced features** through its BDII constraints. However, some fine grained features cannot be expressed via JDL, such as locked memory limit, open descriptors limit, network bandwidth, thread count limit etc.

The JDL schema is able to accommodate and describe the parallel behaviour of an application with the same limiations as in the RSL case; therefore JDL is only partly able to satisfy the **Advanced parallel behaviour description** requirements.

EXTENDED RESOURCE SPECIFICATION LANGUAGE (XRSL)

xRSL [28] is a Grid application description language used in Grids based on ARC [61] middleware (e.g. NorduGrid [62], Swegrid [63], KnowARC [64], NDGF [65]). As its name implies, xRSL was created through the extension of the RSL schema by several new features.

Many of xRSL extensions over RSL capabilities are related to the ARC middleware (e.g. *notifications*, *ACLs (Access Control Lists)*, *ftp threads*). However, several improvements from RSL can be noticed with reference to the **Advanced features** requirement: at the resource level xRSL displays some extra disk-related parameters and at the data staging level it accommodates URIs. However, the **Advanced features** requirement is not entirely.

(WS-) GRID RESOURCE ALLOCATION AND MANAGEMENT (GRAM4) RSL

As its prefix WS suggests, GRAM4 RSL [66] was mainly built to address services. Once Grid began to adopt the Web service technology (which was later transformed and adapted to Grid, finally emerging as the *OGSI/WSRF-Grid Service* technology), a new description language was needed to support service descriptions and invocations. The Globus team modified its GT2 Grid middleware and adapted it to the new Grid service paradigm and so they created the GT4. As in the case of GT2, GT4 came packaged with an application description language, only that this time, the new application description language was oriented towards the new service technology. This became known as WS-GRAM RSL or GRAM4 RSL.

The GRAM4 RSL remains quite similar to RSL and is not a marked improvement from that in terms of its **Legacy compatibility** and **Advanced features** requirements. However, the WS-GRAM schema supports file staging using different credentials. WS-GRAM RSL can express a *myproxy* service [67] invocation for credential retrieval used for file staging (myproxy is a Grid service that holds user PKI X509 credential-delegates [68]), thus describing a scenario

when files can be transferred using different credentials (e.g. *jobCredentialEndpoint, stagingCredentialEndpoint*).

However, apart from this particular case, and as its forerunners RSL and xRSL, the WS-GRAM RSL schema still lacks some fine granularity such as multi-process description capability or parameter sweep description capability to meet the **Advanced parallel behaviour description** requirement.

LEGACY CODE INTERFACE DESCRIPTION (LCID)

LCID [20] is an application description language used by GEMLCA to describe the applications stored in its repository. From the point of view of meeting the *basic requirements* and the *advanced requirements* this solution has fewer description capabilities than the other languages brought in discussion. However, LCID needs to be mentioned because, historically, it is the first application description language to implement the idea of a *template*. LCID permits administrators to *fix* values for certain application description parameters (i.e. through the attribute *fixed*) and can also *demand* users to input values for a specific description field (i.e. through the attribute *mandatory*).

However, apart from this ability, LCID meets only the **Legacy compatibility** requirement and provides partial support for the **Advanced features** and **Advanced parallel behaviour description** demands.

JOB SUBMISSION DESCRIPTION LANGUAGE (JSDL)

The JSDL's schema [22] marked a decisive step forward for the Grid application description language standards. Not only that JSDL meets the **Legacy compatibility** requirements, is also meets all the **Advanced features** requirements identified in Section 2.3.1. Furthermore, its schema has the advantage of being extendible (i.e. through the use of *other* attribute).

The JSDL schema was able to accommodate multi-process application descriptions from the outset and while the original schema could not express

parameter sweep behaviour, through the use of JSDL's native extension capability researchers effortlessly added the parameter sweep extensions [69] making JSDL able to fully meet the **Advanced parallel behaviour description** requirement.

2.3.3 Conclusions

The previous section (2.3.2) described the most important application description languages used in current Grid infrastructures. Almost all of these languages implement the basic requirements. However, with the notable exception of JSDL, these languages implement only parts of the advanced requirements and they cannot be regarded as a complete solution that meets both basic and advanced requirements.

This analysis also found that a substantial subset of application description attributes is common to all application description languages. Therefore, a future description language might consider the idea of extending an old ADL rather than create a new one. An excellent example in this matter is given by JSDL, which exhibits native extension capabilities that allow it to accustom new parameters, types and attributes while still remaining compliant with the original language schema.

2.4. Grid Application Matchmaking Systems

In computer science, *matching* (or *matchmaking*) can be defined as the process of evaluating the degree of similarity between two objects.[70] Objects are characterized by properties and a matchmaking system would run an algorithm which compares these properties, analyses the results of the comparison and returns a *matching degree*. In most cases the matching degree is a real number with values in the interval [0, 1], where usually 1 denotes *identical* or *equal* objects, while 0 denotes *opposed* or *completely different* objects (Note: there are some

cases, such as the edit-distance metrics, where 0 means *equal objects*). A value in the interval (0, 1) – i.e. with both endpoints excluded from the interval – can be translated as *similar to a certain extent*.

Traditionally, the properties of an object are formally encoded in a description document following a predefined model of the object. However, in many cases, one property of the object can be encoded in the model via a set of two or more description attributes. Usually, the description documents contain tuples like (*name, value*), where *name* refers to the description attribute, while in most cases *value* is a number, a string of characters or a Boolean value.

2.4.1 General Overview of Grid Application Matchmaking Systems

Grid application description documents follow the same concept as the one described above and therefore can be processed by matchmaking systems. However, there are different approaches to matchmaking and there is no evidence in related research that they were applied to Grid applications until now. Examples of matchmaking techniques that can process Grid application description documents include syntactic methods, string-distance metrics and semantic techniques, which are discussed below.

SYNTACTIC MATCHMAKING

Syntactic matchmaking uses the structure or the format of an object description in order to perform the matching process. Syntactical matching systems do not take into consideration either the meaning of the attribute name or the meaning of the attribute value found in a description; they process the fields without knowledge of their semantic value.

Applications are described using formal languages and structures that can be either proprietary or standard-based. In most cases, syntactic matchmaking is performed

on homogenous descriptions, but they have been extended to heterogeneous descriptions as well.

In the case of homogenous descriptions, the structure of the description documents of the two applications under comparison is the same. The matchmaking system usually employs very strict mathematical and logical functions to compare the values of each pair of attributes. The partial results are then combined according to an aggregation model, which supplies the final result. The aggregation model usually employs a weighing system and assigns scores to each partial result in relation to the importance of each attribute within the overall description of the application. Next, the system then matches the score against a threshold and provides the final decision to the comparison process.

In the case of heterogeneous descriptions, a syntactic structure matchmaking is first performed in order to find the correspondence between those attributes that encode the same property of the application.

Information about an application property can be formally encoded using a set of *description attributes*.

Table 2-1: Example of two formalisms encoding the same application property

Application	**Formalism no. 1**		**Formalism no. 2**	
Property	**Attribute(s)**	**Value(s)**	**Attribute(s)**	**Value(s)**
Execution Site	*SiteLocation*	ngs2.rl.ac.uk:2119/ jobmanager-pbs-short	*SiteName*	ngs2.rl.ac.uk
			Port	2119
			Jobmanager &Queue	jobmanager-pbs-short

When encountering heterogeneous structures, it is common to find the same information about an application property encoded in different sets of description attributes, which means different formalisms may have no syntactical symmetry when describing the same application property. The example given in Table 2-1 shows two different approaches to modelling the *execution site* where the application can run: one formalism uses only one attribute (i.e. Formalism no. 1 - *SiteLocation*) to encode this property, while the second uses a set of three

attributes (i.e. Formalism no. 2 – *Site, Port and Jobmanager&Queue*) to encode the same property.

Consequently, matchmaking systems used in these cases have to perform an *alignment* of the information about the application property by recomposing the original set of descriptions attributes. Once the alignment has been performed, the process follows exactly the same steps as in the case of homogenous descriptions, applying comparison functions and combining partial results to generate a final answer to the matching case.

STRING DISTANCE METRICS

String distance metrics are "a class of textual based metrics resulting in a similarity or dissimilarity (distance) score between two text strings for approximate matching or comparison" [71].

Nowadays string-distance metrics are used in a multitude of areas such as fraud detection, plagiarism detection, ontology comparisons, DNA sequencing and analysis, data mining, evidence based machine learning or Web interfaces (e.g. word suggestions as you type, typing error detection, etc.)

On Grid, every application description language has a field called *[Application] Description*. The value of this field usually contains a paragraph of free-text in which application administrators describe the application history, functionality and purpose of the application using natural language, with no formal constraints. This text contains information about the application, but syntactic matchmaking systems leave this information unprocessed due to the complexity of the functions needed for such a comparison, as well as due to the level of uncertainty induced by their outcomes, which would require a far more complex result-processing system. However, these paragraphs of text can be processed by algorithms that use string-distance metrics to find similarities between these texts and to eventually give users an indication of how similar two Grid applications are.

String-distance metrics can be divided in three categories: *edit-distance metrics, token-based metrics* and *hybrid metrics*.

Edit distances compute the dissimilarity between strings as the cost of the best sequence of edit operations that convert the first string to the second string. Typical edit operations are character *insertion, deletion* and *substitution*. Nowadays, many methods also use character *transposition* in their functions. Examples of techniques that use edit distances are Damerau-Levenshtein [72], Smith-Waterman [73], Jaro-Winkler [74], Needleman-Wunsch [75] and Monge-Elkan [76].

Token-based distance functions assume that strings are sets of words (or tokens). Functions associated with token-based metrics usually compare the similarity and diversity of token sets. More advanced methods compute token frequency statistics from the complete corpus of documents to be matched and use vector analysis functions and probabilistic approaches to compute the matching score between two strings. Some approaches see the token sets as samples from an unknown distribution of tokens and compute the distance between two sets of tokens based on similarity/divergence scores of such distributions. Notable methods in token-based string matching are the TFIDF/Cosine similarity function [77], Jaccard distance [78], Tanimono coefficient [78], Dice coefficient [79], Jensen-Shannon Divergence [80] and Jelinek-Mercer mixture model [81].

The **hybrid methods** are usually combinations between edit and token based distances (e.g. SoftTFDIF [82]), but sometimes they can combine functions from the same class (e.g. the two-level edit-distance algorithm proposed by Monge and Elkan [83]).

SEMANTIC MATCHMAKING

While syntactic matchmaking processes descriptions based solely on their structure and format, semantic matchmaking systems also look at the meaning of the description attributes name and value.

Semantic matchmaking systems are not new. Extensive research has been undertaken in recent years in the world of semantic web services, ontology matchmaking, natural language processing, and document analysis - many of them with outstanding results. Currently Grid does not hold a specific example of semantic matchmaking; however, semantic analysis methods employed in semantic web services and document-processing areas can be applied to Grid application descriptions documents in order to find similarities between applications.

As in all approaches that involve a structured description of an object, the formal semantics of the object are specified with the help of a language, such as LARKS, LDL++, DAML-S/OWL-S, WSMO, OWLS-MX, SWSL etc. These languages provide a core set of markup language constructs for describing both the properties and the capabilities of an object in unambiguous, computer-interpretable form. [84, 85] However, as they are limited to the lexicons and terminologies used in different ontologies, these approaches do not address cases of implicit semantics that can be found in patterns or relative frequencies of terms in object descriptions.

Semantic matchmaking can also analyze the linguistic semantic associations between words such as *synonyms* (i.e. words that have similar meaning and can often be used interchangeably), *antonyms* (i.e. words that have opposite meanings), *hypernyms* (i.e. words that have a more general meaning), *hyponyms* (i.e. words that have a more specific meaning), *meronyms* (i.e. words that represent a *part-of* relation) and *holonyms* (i.e. words that represent a *whole* relation). For example, ‘leaf’ is a meronym of ‘tree’ and ‘tree’ is a holonym of ‘leaf’. [86]

2.4.2 Review of Existing Solutions

While matchmaking techniques have not been used for application matching in Grid until now, they have successfully been used in matching Grid resources. [70, 87, 88, 89] The following analysis presents four such solutions used for resource-

matching in Grid. The analysis is complemented with that of three general matchmaking systems, which describes how string-distance and semantic techniques can be used to match objects based on their description.

CONDOR

Condor is a software system designed to manage a dedicated cluster of workstations. Its advantage resides in the ability to effectively harness non-dedicated, pre-existing resources under distributed ownership. [90] One of the modules provided by the Condor system is the matchmaking framework that matches resource owners with resource consumers.

The framework is based on a semi-structured data model [91, 92] called classified advertisements (*classads*) which are used to describe resources and requests. There are two types of classads: *resource offer ads* and *resource request ads*. A *resource offer ad* is submitted by resource providers and represents a formal mapping between resource properties (such as machine architecture, operating system, available disk space, available RAM memory, CPU type, CPU speed, virtual memory size, physical location, current load average, etc.) and a *value* expression. A *resource request ad* is specified by the users when submitting a job and follows the same mapping as the resource offer ads.

Condor matchmaking takes two classads (i.e. a resource offer ad and a resource request ad) and evaluates each of these against the other. A strong requirement of Condor is that the provider and the requester know each other's classad structure. A classad has a special attribute named *Requirements* and two classads *match* only when the values of the field *Requirements* of both classads under comparison are evaluated to be true. If users want a finer match to their requests after finding the machines that met the requirements, they can express their preference via the *Rank* attribute. For example, a resource requirement ad specifying

```
Requirements = Memory >= 1024 && OpSys="SOLARIS10" && Arch="SUN"

Rank = Memory >= 2048
```

asks Condor to choose all SUN machines running the operating system SOLARIS 10 with more than 1GB of memory and expresses a preference to run the program on machines with more than 2GB of physical memory provided such machines are available.

BERKELEY DATABASE INSFORMATION INDEX - BDII

The central information system in gLite/lcg-based Grids (i.e. BDII) stores information about Grid resources and uses the GLUE schema as a description model for the referenced objects. The GLUE schema provides support for two types of gLite/lcg Grid resources: *computing elements (CEs)* and *storage elements (SEs)*.

BDII allows communication with clients via LDAP (Lightweight Directory Access Protocol) queries. Matchmaking in BDII is similar to the matching system in Condor. Namely, the client specifies a set of needed requirements in a field called *Requirement*; only when these *Requirements* are met the two GLUE entries are considered to be a match. Like Condor, clients using BDII are required to know the formal schema used by the system (i.e. in this case, the GLUE schema).

The following example describes a client querying one of the BDIIs located at CERN (i.e. lcg-bdii.cern.ch:2170):

```
ldapsearch -x -H ldap://lcg-bdii.cern.ch:2170  -b Mds-Vo-name=local,
o=grid'(&(GlueHostArchitectureSMPSize=4)
(GlueHostMainMemoryRAMSize>=1024)
(GlueHostArchitecturePlatformType=i686))'
```

The result will be a list of machines with i686 architecture, a symmetric multiprocessing power of 4 and more than 1GB of RAM. Furthermore, similar to Condor, BDII uses a *Rank* attribute to select the best resource from the list of candidates.

RESOURCE BROKERING SYSTEM (RBS)

Similar to the two systems described above (i.e. Condor and BDII), RBS is a matchmaking algorithm developed for Grid resource discovery and matching. RBS implementation is based on the Condor classad matchmaking algorithm, but it also provides a Latent Semantic Indexing (LSI) based algorithm using concepts commonly applied to IR (i.e. information retrieval) and internet search engines.

The brokering service interprets client requests and correlates them with virtual organization policies regarding resource access. The RBS enables users to query a VO (i.e. Virtual Organization) index service for a specific resource and acts as a mediator between the resource consumer and the resource producer. Producers register their resources inside the VO index using Condor classads, which will be later queried by resource consumers.

RBS' greatest achievement rests nevertheless in its framework for automatic publishing, collecting and classifying of resource properties in the VO index service. For example, RBS is able to realize automatic mapping between GT4 resource properties and Condor classads.

With regard to the matchmaking algorithm proposed by RBS, the new LSI approach is effective and efficient [88]. However, in Grid resource description and matchmaking, the Condor classad notation is regarded as *lingua franca;* therefore, the LSI algorithm was integrated with the Condor matchmaking algorithm. A comparison between the RBS matchmaking algorithm and the Condor matching system described in [88] showed that the combined approach (LSI + Condor) exhibited better results in terms of effectiveness than the simple Condor matching system; however, the complexity of LSI calculus is a drawback in terms of performance, which places the RBS system in second place behind Condor.

NAREGI RESOURCE MATCHMAKER (NAREGI-RMM)

As opposed to the first three matchmaking systems, which are examples of syntactic matchmaking algorithms, NAREGI-RMM is the first system that uses a

description based on semantic web technologies - in particular ontologies [93]. NAREGI-RMM enables complex and diverse descriptions of user requirements and resources in production Grid (e.g. complex workflow and co-allocation) which are mapped on ontologies developed using Protégé 2 [94] and the OWL standard.[95]

Given that the NAREGI Grid middleware uses JSDL to express job requirements, the subsequent ontology is able to describe all information that can be found in a JSDL description (such as candidate hosts, i.e. a list of hosts that can be allocated; operating system; CPU architecture; individual CPU count; total CPU count; total physical memory; total virtual memory; executable; data staging; etc.). Furthermore, the NAREGI-RMM ontology can also be used to describe network bandwidth; network latency; time reservation-related attributes (i.e. the date after which the job must start, the date before which the job must end, the time the resource is reserved); cluster reservations; access policies; list of applications installed on the resource; user policies; and group policies.

The resource matchmaking system is integrated with the NAREGI submission engine. Once a job is submitted, the request is translated into an XML document compliant with the NAREGI-WFML schema.[96] Because each system has a timetable of resource reservations, the next step is to check the time constraints of the request. If the time constraints are matched, the next step is to validate the user/group constraints against the policies enforced on the system (e.g. in a university, a student may not have the same permissions as a professor; or, the group of Grid site administrators may have different permissions and constraints than the group of Grid application users). Next, the matchmaker checks whether the user application is installed on the system and if not, the system does not match the request and will not be part of the solution. If the application is deployed and available, the NAREGI-RMM checks whether the resource requirements are satisfied and returns the first from a list of candidate resources. In the final step of the process the submission engine submits the job to the suggested resource and then delegates the job monitoring process and the result retrieval aspects to other Grid services.

OWLS-MX

OWLS-MX is a hybrid semantic matchmaker used in Web service matching. The OWLS-MX matchmaker is based on LARKS [97, 98, 99, 100], but it differs from it in terms of the description language and description language logic (DL) used in the matching process. While LARKS uses a proprietary capability description language and logic, OWLS-MX uses the standard OWL-S and OWL-DL for service capability description and description logic. OWL-S is an OWL-based Web service ontology with three main components: the service profile for advertising and discovering services; the process model, which gives a detailed description of a service operation; and the grounding, which provides details on how to interoperate with a service via messages.[101]

The service matching performed by the OWLS-MX matchmaker exploits both logic-based reasoning and content-based IR techniques for OWL-S service profile I/O matching. OWLS-MX takes in consideration pairs of service advertisements and service requests and for each of them computes the matching degree by successively applying seven different filters: *exact, plug-in, subsumes, subsumed-by, logic-based fail, nearest-neighbour* and *fail*.

The OWLS-MX matchmaker takes any OWL-S-compliant request as a query, runs the filters against the service descriptions publicized by the providers and returns an ordered set of relevant services that match the query. Each service returned by the matchmaker is annotated with the individual degree of matching with the initial request, as well as the syntactic similarity value between the service and the request. Furthermore, the client can specify a preferred matching degree as well as a syntactic similarity threshold. In particular, OWLS-MX also determines the syntactic similarity between the conjunctive I/O concept expressions (described in OWLLite [102]). OWLS-MX recursively unfolds each query and service I/O concept and includes the primitive components of a basic shared vocabulary in the local matchmaker ontology. Furthermore, if the degree of syntactic similarity between the respective unfolded service and request concept expressions exceeds a given

similarity threshold, OWLS-MX will tolerate any failure of logical concept subsumption produced by the integrated deductive language.

INFOSLEUTH

InfoSleuth is a generic agent-based information discovery and retrieval system, which bases its syntactic and semantic matching process on so-called *broker agents.*[103, 104, 105, 106, 107, 108, 109] The InfoSleuth matchmaking architecture defines three types of agents: *operational agents, querying agents* and *broker agents*. The *operational agent* is equivalent to the concept of service provider; the *querying agent* is equivalent to the service requester; and the *broker agent* matches querying agents against operational agents.

The broker maintains a repository about the operational agents and their services and enables the querying agent to locate all available agents that can provide services that meet their interests.

The InfoSleuth system can match requests to agents on the basis of the syntax of incoming messages used to wrap the requests (i.e. syntactic matchmaking). In addition to this, InfoSleuth can also match requests to agents on the basis of the requested agent capabilities or services (i.e. semantic matchmaking).

The agent capabilities and services are described in a common shared ontology of attributes and constraints which uses a specific vocabulary. All operational and querying agents can use this vocabulary to specify advertisements and requests to the broker. The service capability information is written in LDL++ [110] and broker agents use a set of LDL++ deductive rules to support inferences about whether an expression of requirements matches a set of advertised capabilities.

REUSABLE TASK STRUCTURE-BASED INTELLIGENT NETWORK AGENTS (RETSINA)

RETSINA is a multi-agent infrastructure proposed and developed at Carnegie Mellon University in Pittsburgh, Pennsylvania (USA). Following the same structure

of agents as InfoSleuth, RETSINA proposes three general agent categories: *the service provider, the service requester,* and *the middle agent*. Mediation in this system also relies on service matchmaking, although in order to describe agent capabilities in the matching process they have defined a new agent capability description language called LARKS (Language for Advertisement and Request for Knowledge Sharing). LARKS offers the option to use application domain knowledge in any advertisement or request by using a local ontology, provided that the local ontology is written in ITL (Information Terminological Language). In that case, the ontology can automatically be incorporated in LARKS and can be processed by the RETSINA systems.

The RETSINA matchmaking system allows service providers to register their capabilities in an *advertisement*, which provides a short description of the agent, a sample query, input and output parameter declarations, and other constraints. When the matchmaker agent receives a query from a service requester it searches its dynamic database of *advertisements* for providers that can fulfil the incoming request. Although RETSINA was developed for generic MAS (Multi-Agent Systems), the most known and used implementation deals with Web service discovery and matchmaking. In this case, each service provider advertisement gives a semantically-based view of the web service, including the abstract description of the capabilities of the service, the specification of the service interaction protocol, and the actual messages that it exchanges with other web services. The matchmaking system employs techniques from information retrieval, artificial intelligence and software engineering to compute the syntactical, string-distance, and semantic similarity degree between different service capability descriptions. The matching engine of the matchmaking system contains five different filters for namespace comparison, word frequency comparison, ontology similarity matching, ontology subsumption matching, and constraint matching [111]. Users can select which filters to use and can also apply custom thresholds to these filters in order to improve the accuracy of the matchmaking and balance between the performance and the quality of the matching system.

2.4.3 Conclusions

Other matchmaking systems exist which may be used to find similar applications in Grid repositories. Examples such as PromptDiff [112], Minersoft [113] or GridLET [114] employ matchmaking methods that can be adapted for Grid applications (e.g. token-based solutions with indexing and categorization methods based on term frequency), but they are similar to those already described. Therefore, their description was omitted, as they do not present any new method over the ones already mentioned in section 2.4.2.

In Grid, syntactic matchmaking methods were successfully used in resource matching and therefore represent the first option when trying to match objects based on their formal descriptions. However, in cases like gLite/lcg-based Grids – a type of infrastructure widely used throughout the world – the formal description document of the application is usually missing from the repository. Furthermore, the metadata associated with the application does not contain the basic and advanced requirements of the application (i.e. as discussed in Section 2.3.1). Hence, the accuracy of syntactic matchmaking techniques is limited, as the most important application properties are not formally contained in a document that could be processed by such matchmaking systems.

In such cases, the repository needs to rely on other forms of matchmaking, for example techniques which use string-distance or semantic methods that can process the free-text field *Description* from the repository model, as well as matching techniques which process application-related objects other than the application description document itself (such as binaries or source code).

Within the time constraints imposed by the lengths of a PhD, this research managed to analyze four matching methods – i.e. syntactic, string-distance, application running and binary matching – and proposed several others which may help identify similar applications stored in Grid repositories.

2.5. Challenges

Applications play one of the most important roles in Grid, regardless of the methods used to describe such an application or to expose it to Grid users.

Initially, Grid application repositories were regarded by Grid scientists as of moderate importance and were used mainly by Grid administrators for improved storage and management of applications. Moreover, since the information about applications was structured following a repository model, they were also used by administrators to present users with a uniform view of application metadata.

However, as the number of Grid applications grew year by year, Grid application repositories became not only important, but a necessity. Between July 2008 and July 2009 the number of applications found on the University of Westminster application repository (represented by a GEMLCA resource) increased by 94%, from 18 to 35 Grid Applications; the number of applications found on the NGS Application repository increased by 51%, from 33 to 50 applications; and the number of Grid applications stored in the myExperiment repository increased by 98%, from 321 to 635 applications. Meantime, repository technologies and distributed computing technologies evolved and nowadays Grid application repositories face a far more complex set of challenges.

For a better understanding of the Grid application repository solutions and of the functionality and capabilities that these repositories must exhibit in order to interoperate with current and future distributed computing environments, the following requirements have to be met:

(R1) APPLICATION PUBLISHING: The first and foremost requirement of any Grid application repository is to store and manage Grid applications. The repository model needs to be able to describe the application along with descriptions of related entities (such as author, access policies, provider, application-objects – binaries, documentation and licenses), as well as relations between them. Furthermore, any application repository needs an intuitive, user-friendly interface,

such as graphical clients or web-based solutions, which would make application publishing easy for both Grid administrators and users. Command-line solutions demand a much more thorough understanding of the underlying concepts and technologies, which users should not be required to have.

(R2) APPLICATION DISCOVERY: Currently, each production Grid application repository stores tens or hundreds of Grid applications. It is obvious that for users or services interested in such applications, these repositories should be the first place to look for them. Therefore, a Grid application repository needs to provide means to discover the Grid applications it stores and this process should be made easy to both users and services. A Grid application repository should permit the discovery of applications it stores following several principles:

- **(R2.1) Expose their application to the Web:** Grid application repositories should be built using technologies that interact with popular web search engines and web metadata harvesters. It is therefore desired that application repositories expose interfaces such as HTTP/REST [115, 116] or OAI-PMH, which would make the application discovery process easier to both human users and services. Such protocols would permit services to retrieve Grid application metadata and Grid application objects from repositories using the ubiquitous HTTP clients and simple HTTP queries.
- **(R2.2) Interoperability with any OGSI/WSRF Grid service:** Grid application repositories should expose an OGSI/WSRF Grid service interface, which would make them able to interact seamlessly with any other OGSI/WSRF Grid service in a standard, serviceable manner. In 2005 Globus implemented the first Grid service-based middleware, based on OGSI/WSRF standards stack, and since then many Grid projects have been developing and using OGSI/WSRF Grid services (such as The Lattice Project, GEMLCA, D-Grid Projects, WS-PGRADE/gUSE, and ARCS). From the application discovery point of view, by fulfilling **R2.1** and **R2.2,** any application stored in such repository could be easily discovered by human users, Grid services and any other

service equipped with a simple HTTP client. However, having an OGSI/WSRF service interface also offers the possibility to manage the repository and the objects stored inside through a service interface. Moreover, by implementing OGSI/WSRF WS-Notifications providers, clients can be notified automatically when certain actions are performed, such as the addition/deletion/modification of a new Grid application in the repository.

- **(R2.3) Connection with other repositories:** While various Grid application repositories are currently not connected in any manner, a service able to connect different repositories and to find whether a desired Grid application is stored in any of the connected repositories would prove extremely valuable to both human users and services. The lack of connectivity between Grid application repositories comes as a straightforward consequence of the different repository frameworks, which vary in terms of access interface, security system, implementation technology, communication protocols and transfer protocols. A service able to connect such repositories would simplify the application discovery process.
- **(R2.4) Ability to find similar applications:** The set of applications found on two or more Grid application repositories might overlap (for example, no less than 29% of the Grid applications found on the NGS AR and CHARON/iSoftrepo are similar – July 2009). A service able to find the same application or similar applications on different repositories would be very helpful to users and services in cases when certain repositories become inaccessible or when the Grid infrastructure behind one given repository becomes unavailable. Such a service would also give users the choice between application description languages (for example if a user wants to use application AUTODOCK [117] and knows JSDL better than LCID, s/he would prefer to use the NGS AR, where this application is available in JSDL, rather than the Westminster GEMLCA repository, where the same application is described using LCID).

(R3) OBJECT EXCHANGEABILITY & REUSEABILITY: Grid application repositories should permit the exchange and re-usage of Grid application objects and application-related objects with other repositories.

Repository technologies evolved rapidly during recent years and scientists put a lot of effort into trying to standardize communication between them to make them interoperable and permit automatic exchange of objects between them. In December 2007 Open Archive Initiative [118] publicly released the first version of such a standard (i.e. OAI-ORE [119]) and by December 2008 (the date of the last version) OAI-ORE was already adopted as a standard by two of the most important Open repository technologies (i.e. FEDORA [120], ePrints [121]). The third one, DSpace [122], is in the process of adopting it as well.[123] Should Grid application repositories allow object reuse and exchange (for example, by implementing the OAI-ORE standard), this would also allow administrators to easily find and relocate objects in fresh repositories in an automatic manner and would permit services to simultaneously extract metadata and objects from repositories.

(R4) VERSATILITY IN USAGE: Grid application repositories should extend the scope of the distributed environments so that applications stored by them can be used to include computing concepts similar to Grid such as application-on-demand, cluster-on-demand and cloud computing. Traditionally, Grid application repositories used in current production environments are *hard-wired* to a Grid submission engine and can only be used on Grid architectures. In most cases these repositories store only descriptions of the applications written in an application description language processed by the underlying Grid, which effectively makes them usable only in that particular framework. However, emerging distributed computing concepts such as *cluster-on-demand*, virtual *computer-on-demand* and cloud computing are very similar to Grid. As applications stored in Grid application repositories are able to run on any of these architectures, these repositories should not limit themselves to Grid, but should try to extend the scope of the distributed environments where these applications can be used.

Application repositories could store application-related objects such as the executable, the source code, library dependencies, the documentation, licenses, etc., all in the same place, giving users and administrators countless possibilities to use them in different scenarios and on different distributed computing architectures.

2.5.1 Architecture

Based on the discussion in Section 2, the following table summarises existing Grid application repository solutions and how they comply with requirements **R1-R4**:

Table 2-2: Current Grid application repository solutions vs. Requirements **R1-R4**

	R1: PUBLISHING	R2: DISCOVERY	R3: EXCHANGE & REUSE	R4: VERSATILITY
BDII	- Publishing done by automated services via scripts that contain suites of console commands; - No graphical/web interface for human users.	- Console commands containing LDAP queries; - No OGSI/WSRF Grid service interface; - No Web visibility; - No HTTP/REST interface; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO	- GLUE attributes: name, version, location; - Used only to list EGEE sites where the application resides and can be run.
CHARON	- Command-line only for human users; - No access support for services;	- Collection of static Web pages; - No OGSI/WSRF Grid service interface - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO	- Metadata for *Application* object only; - Used only to list EGEE sites where the application resides and can be run.
GEMLCA	- Graphical interface for human users; - OGSI/WSRF Grid service interface for services.	- OGSI/WSRF Grid service interface; - Human users can find application information through PGRADE portals or using a GEMLCA Service Client; - No Web visibility; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO	- Can store the application binary and application description documents only; - Permits the usage of different Grid submission systems.
NGS AR	- Graphical interface for human users - No access support for services	- JSR-168 web application interface – for human users; - No OGSI/WSRF Grid service interface; - No HTTP/REST interface; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO	- Stores application description documents; - No support for distinctive application-related objects; -Can be used only in traditional Grid frameworks in conjunction with its submission system.

GRIMOIRES	- Human users and services can register web services via UDDI clients.	- Visible to UDDI clients; - Visible to human users through a collection of static web pages.	N/A	N/A
myExperiment	- User-friendly web interface for human users; - HTTP/REST interface for services.	- Intuitive web interface for human users; - Exposes HTTP/REST interface; - No OGSI/WSRF Grid Service interface; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO	- No support for distinctive application-related objects.

Unfortunately, none of the current Grid application repositories that were discussed in this chapter can be seen as a comprehensive solution that meets all **R1-R4** requirements. Notably, the GEMLCA system highlights the importance of interoperability with Grid WSRF/OGSI Services and the myExperiment repository accentuates the value of application visibility on Web. Currently, there is no connection between application repositories and there is no service which users or other services can inquire to find whether a desired Grid application is stored in one of the Grid repositories. In conclusion, at the time this research was carried out existing Grid application repository solutions only partly met requirements **R1 – R4**.

2.5.2 Repository Model

Traditional Grid application repository models revolve around two entities: the *user* who described the application and the *application* itself. While more advanced solutions do allow for the storage of *application-related* objects, they store them without any categorization or name-pattern tagging, which makes them indistinguishable for searches or meaningful automatic retrieval.

According to requirement **R2.4** an application repository solution able to connect multiple Grid application repositories to its system requires a new entity to be described in the model – the *Provider*.

In addition to user, application and provider, any repository model is required to contain descriptions of different *Access Policies*, which deal with security, visibility

and permission attributes for each of the object they relate to. Moreover, in order to expand the area of usage of Grid application repositories, the repository model should not limit itself to the description of application-related objects, but should permit the modelling of user-related objects and provider-related objects.

The *Provider Related Objects* refer to those objects that may have to be stored in Grid application repositories in order to give a comprehensive description of a provider and to ensure proper access and connectivity to it (e.g. PKI public certificate, software client, etc.).

The *User Related Objects* refer to those objects that can be stored in Grid application repositories and can be used for identification, application running or data staging (e.g. X509 proxies, PKI public keys, username/password sets, etc.).

Figure 2-8 shows the entities that need to be present in a Grid application repository model, in line with requirements **R1-R4**:

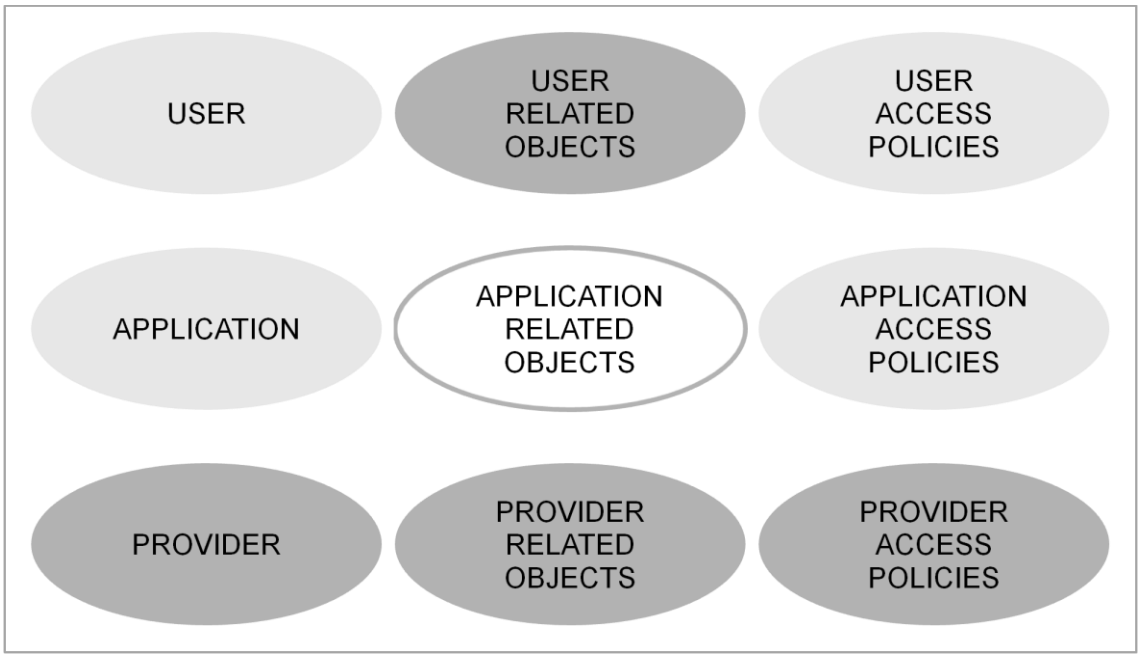


Figure 2-8: Grid application repository model entities

Table 2-3 below summarizes the critical analysis of the five repository models described in Section 2.2.2 (MyExperiment, NGS, GEMLCA, GUSE and CHARON/iSoftrepo) assessing their ability to describe the entities specified above:

user, user policies and user related objects; application, application policies and application-related objects; provider, provider policies and provider-related objects.

Table 2-3: Traditional Grid application repository models and repository entities

	myExperiment	NGS AR	GEMLCA	GUSE	CHARON/ iSoftrepo
User	**YES**	**YES**	**YES**	**YES**	**YES**
User-related objects	no	no	no	no	no
User access policies	**YES**	**YES**	**YES**	**YES**	**YES**
Application	**YES**	**YES**	**YES**	**YES**	**YES**
Application related objects	**YES***	**YES***	**YES***	**YES***	**YES***
Application access policies	**YES**	**YES**	**YES**	**YES**	**YES**
Provider	no	no	no	no	no
Provider-related objects	no	no	no	no	no
Provider access policies	no	no	no	no	no

Note: **YES*** from Table 2-3 – current solutions store only a few types of *application-related* objects; furthermore, these are stored without any meaningful categorization, which makes them indistinguishable for searches or automatic retrieval.

In conclusion, none of the repository models currently in production is able to fully satisfy the requirements set out above.

2.5.3 Application Description Language

Based on the critical analysis of Grid application repositories from their early stages (e.g. BDII, CHARON/iSoftrepo) to the most recent available solutions (e.g. myExperiment, NGS Application Repository, GEMLCA), and in conjunction with the repository requirements **R1-R4** (i.e. publishing; discovery; reuse and exchange;

and versatility), this research identifies the following life-cycle for a Grid application that resides in a Grid application repository:

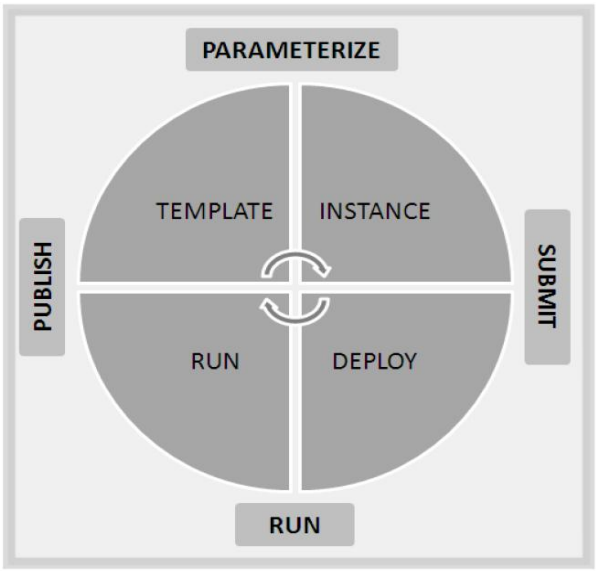


Figure 2-9: The life-cycle of a Grid application stored in an application repository

Once the application is published in the repository, the application description document contains all the necessary values for the configuration parameters, which make it usable on Grid (e.g. machine architecture, minimum amount of RAM, default input files, required command parameters, etc.). The parameters are predefined by the application publisher that is responsible for providing the necessary values for those attributes. These attributes will have to remain unchanged and free from users' interference, otherwise the application may become unusable. When such a formal description skeleton is provided, the application is said to be in the *Template state*.

The user can retrieve such an application template and can provide values for the rest of the parameters necessary for a particular run. At this stage the application enters the *Instance state*. In this state, the application is parameterized and ready to run. (Note: the parameterization can be specific to each run or to each user.) The application can then be submitted via a submission engine to run on Grid. However, before the actual run of the application, application *deployment* and data staging may occur.

In the *Deploy state* the application binaries and the application input files are retrieved from different locations (including the repository where the application resides), possibly using different authentication methods. Furthermore, these application objects could be staged from different Grids, using different X509 certificates and possible different transport protocols. Moreover, it is advisable that the staged data benefit from data protection (e.g. hash sums) which would insure that the transferred data is genuine and would protect against data corruption that may occur during transfers. Data are staged to the Grid resource where the application instance will run (*stage in*) but output can also be staged from the execution resource to storage servers (*stage out*).

Next, once the staging is completed, the application starts *running* on the Grid infrastructure resource that has been commissioned for it. Currently, there are only two types of application running set up on Grid infrastructures: first, when the application binaries are already present on the resource and only the input files are staged on the resource; and second, when both the binaries and the input files are staged on the Grid resource. However, Grid applications stored on a Grid repository should benefit from the fact that many other application objects could be available, which would permit other types of application running such as: virtual machine-embedded applications that can be staged and run on the resource; source code staging (plus, possibly, software dependencies and libraries) and compilation directly on the resource; as well as running licensed software which would require license staging and acceptance prior to the actual application running. Nevertheless, whichever the type of application run, the results can be retrieved upon completion – and depending on their relevance to the description of the application, some of them can be published in the repository (for example, the publishing of a *test suite* – input set, running script, and expected output set – which would later help with testing the applications or with application matchmaking).

Grid application description languages must be able to describe the states from the life-cycle described above, as well as the **Legacy compatibility**, **Advanced**

features, and **Advanced parallel behaviour description** requirements (see Section 2.3.1).

The following table summarizes the critical analysis of the six application description languages repository models described in Section 2.3.2 (RSL, JDL, xRSL, WS-GRAM, LCID and JSDL) versus the requirements mentioned above:

Table 2-4: Traditional Grid application description languages vs. requirements

	RSL	JDL	xRSL	WS-GRAM	LCID	JSDL
Legacy compatibility	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
Advanced features	partly	partly	partly	partly	partly	**YES**
Advanced parallel behaviour	partly	partly	partly	partly	partly	**YES**
Multi-Grid/ multi-certificate secure data access	no	no	no	**YES (service only)**	no	no
Multiple transfer protocols supported as URI definitions	no	no	**YES**	no	no	**YES**
Data protection	no	no	no	no	no	no
Template	no	no	no	no	**YES**	no
Application run types	Binaries already deployed on the resource	Binaries already deployed on the resource	Binaries already deployed on the resource	Binaries already deployed on the resource	Binaries already deployed OR Binaries staging	Binaries already deployed on the resource
Native extension	no	no	no	no	no	**YES**

The conclusion of this analysis is that the languages currently used in Grid only partly meet the requirements identified above. From the table above it becomes obvious that JSDL is the language which comes closest to meeting the

requirements set before. JSDL has been lately gaining popularity on all major Grids and a lot of effort has been put in adopting JSDL as the default job description language on Grid (e.g. GridWay [125], GridSAM [54], NAREGI [126], GRIA [127] and Genesis II [128]). Moreover, the JSDL schema has the advantage of being extendible. Consequently, instead of creating a new language, a better solution would be to extend JSDL to add the missing parts.

2.5.4 Matchmaking Systems

Syntactic matchmaking methods were successfully used in resource matching in Grid and therefore represent the first option when trying to match objects based on their formal descriptions. However, in cases like gLite/lcg-based Grids – a type of infrastructure widely used throughout the world, such as in EGEE [129], SEE-Grid [130], EELA Grid [131], EUMedGrid [132], EU-India Grid [133], EUChinaGrid [134], Baltic-Grid II [135] – the formal description document of the application is usually missing from the repository. Hence, the accuracy of syntactic matchmaking techniques is limited. In such cases, the matching system needs to rely on other forms of matchmaking, for example techniques which use string-distance or semantic methods that can process the free-text field *Description* from the repository model, or matching techniques that process application-related objects other than the application description document itself (such as binaries or source code).

In order to use any of the algorithms described in the critical analysis section (i.e. Condor, BDII, RBS, NAREGI-RMM, OWLS-MX, InfoSleuth and RETSINA) for grid application matchmaking, the following requirements need to be met:

- *Ability to process the ADL used for application description*: Every matchmaking algorithm that searches for similarities between objects by processing their description documents relies on a formal description language. These description languages are used to express objects' properties and capabilities in a structured way. Although subsets of

matching methods and matching techniques may be common to many matchmaking systems, usually each matchmaking system is bound to only one description language and can process documents written in that particular formalism. Each of the algorithms described in the critical analysis section (see Section 2.4.2) is bound to such formalism. For example, Condor and RBS processes classads; BDII is linked to GLUE; NAREGI-RMM uses NAREGI-WFML and TRIPLE [136]; OWLS-MX and other similar algorithms such as those described in [137, 138] use DAML-S/OWL-S; InfoSleuth understands LDL++; and RETSINA processes LARKS.

In the case of Grid applications, the application description documents are written in a Grid ADL. The first and foremost challenge of any Grid application matchmaking system with the ability to process Grid application description documents is to be able to understand the language the documents are written in. Unfortunately, none of the existing matchmaking algorithms identified by this research (in Grid or similar areas, such as WS, semantic web, ontology matchmaking, etc.) can process any Grid ADL document.

- *Ability to process the application descriptions written in free-text:* As mentioned before, every description language has a field called *[Application] Description,* which usually contains a free-text description of the application history, application functionality and its purpose. The text contains information about the application but many matchmaking systems leave this information unprocessed, as it is written in natural language, with no formal constraints. Syntactic matchmaking systems usually omit this type of fields automatically. Among semantic matchmakers, only those with linguistic semantics capabilities (e.g. RETSINA, InfoSleuth) are able to process such paragraphs of free text.
- *[Ability to process sources of information other than the description document:* Apart from the application description document, Grid application repositories can also store application-related objects such as binaries, source code, hash sums and test files. These objects can help

with the identification of similar applications and a Grid application matchmaking system should be able to process such objects.

Table 2-5 below summarizes how the matchmaking solutions described in Section 2.4.2 meet the requirements mentioned above:

Table 2-5: Matchmaking systems vs. requirements

	Process ADL	Syntactic and free-text processing		Processing of application-related objects
CONDOR	No (class-ads)	**YES**	no	no
BDII	no (GLUE)	**YES**	no	no
RBS	no (class-ads)	**YES**	pilot-LSI	no
NAREGI-MM	No (NAREGI-WFML)	**YES (ontology approach)**	no	no
OWLS-MX	No (DAML-S/OWL-S)	**YES**	no	no
		(semantic ontology approach)		
InfoSleuth	no (LDL++)	**YES**	**YES**	no
		(IR functions)		
RETSINA	no (LARKS)	**YES**	**YES**	no
		(IR functions)		

Based on this analysis, we can conclude that each matchmaking system is bound to its own language. Therefore none of the current solutions is able to interpret the Grid application description languages described in Section 2.3.2. Furthermore, the matchmaking systems under discussion were designed specifically for document processing – therefore these solutions are not capable to use any other Grid application-related objects besides the application description document. However, as the string-distance functions used by some of these algorithms can be applied to any free-text paragraph, these functions could be applied to Grid application descriptions as well.

2.6. Objectives

Since none of the repositories currently used on Grid can be regarded as a comprehensive solution to the requirements identified in Chapter 2 section 2.5 **(R1-R4)**, this research aims to design a Grid application repository able to meet all those requirements. The ultimate aim is to create a modular, easily extendible solution which is based on functional principles that can be followed not only by application repositories usable on Grid, but also by generic application repositories that reside in collaborating environments other than Grid. In order to achieve that, this research was aimed to fulfil de following objectives:

OBJECTIVE O1: The first objective of this research was to design a service able to connect different types of Grid application repositories, but which would still function as a Grid application repository in its own right. By meeting this objective, the service would connect Grid application repositories notwithstanding their different implementation technologies, methods of access and authentication, communication protocols and transport protocols **(R2.3),** while at the same time human users and services could use an access interface to store and retrieve Grid applications directly from the service's repository **(R1)**.The design of this service was also meant to provide a solution to the *application discovery* problem **(R2)** and make applications stored on Grid repositories accessible to other Grid services **(R2.2)** and also visible to the Web **(R2.1)**. Therefore, achieving this objective would not only solve the current interoperability issue, but also expose the applications to Web search engines and through them, to a much larger community interested in Grid applications. Furthermore, the service would also employ methods and protocols for the exchange of objects, thus fulfilling the requirement **(R3)**.

OBJECTIVE O2: The second objective of this research was to propose a new model for application repositories, which would achieve uniformity in Grid application presentation and would extend the functionality of these repositories beyond Grid **(R4)**. After reviewing all the major repository models used on Grid, results have shown that these models imposed limitations on the applicability

domains of Grid application repositories. The objective was therefore to design a new repository model that would provide a comprehensive description of an application along with a suggestion for a new categorization of application-related objects. This would allow Grid application repositories to be compatible and ready-to-use in conjunction with newly emerging technologies such as virtualization, automatic virtual machine creation, cloud computing and automatic service deployments, as well as to be ready-to-use in future distributed computing designs. The final goal related to the repository model was to investigate how the objects stored in a repository following such a model can help with the identification of similar Grid applications **(R2.4)**.

OBJECTIVE O3: While the architecture described in objective **O1** would be capable to connect multiple repositories with different types of application description languages **(R2.3)**, the third objective of this research was to find an application description language, which would provide uniformity in the presentation of Grid application descriptions. Furthermore, the aim was to find (or create) an application description language which would allow for Grid application repositories and the applications stored by them to be used in scenarios other than Grid, such as virtualisation; source code staging and compilation; or automatic application deployment. (**R4)** Moreover, the proposed solution should also provide answers to several Grid interoperability problems, such as multi-Grid data staging and using different security certificates for job-submission and data staging. A secondary aim related to application description languages was to investigate and define a structured life-cycle for any Grid application that resides in a Grid repository, including the different states in which the application can be found (i.e. template, instance, deployment, running) and how these states can be accommodated in an application description language schema. The final goal related to the description language was to investigate how the information about a Grid application modelled by such a language can help with the identification of similar Grid applications **(R2.4)**.

OBJECTIVE O4: The fourth objective was to design a matchmaking methodology and an algorithm able to process information about applications stored in

repositories and identify similar or identical applications (**R2.4**). When one repository becomes unavailable, such a matchmaking service would help users and services to find a subset of the same applications on other repositories connected to the architecture proposed in **O1**. The aim was to identify or create matchmaking techniques that can process various application-related objects stored in repository and can be applied in different scenarios. The final goal was to analyze the performance of a subset of such matchmaking techniques when applied to real-case scenarios, using data found on production Grid repositories as well as objects stored following the repository model described in objective **O2**.

Figure 2-10 shows the relations between the research objectives and the challenges identified in this research:

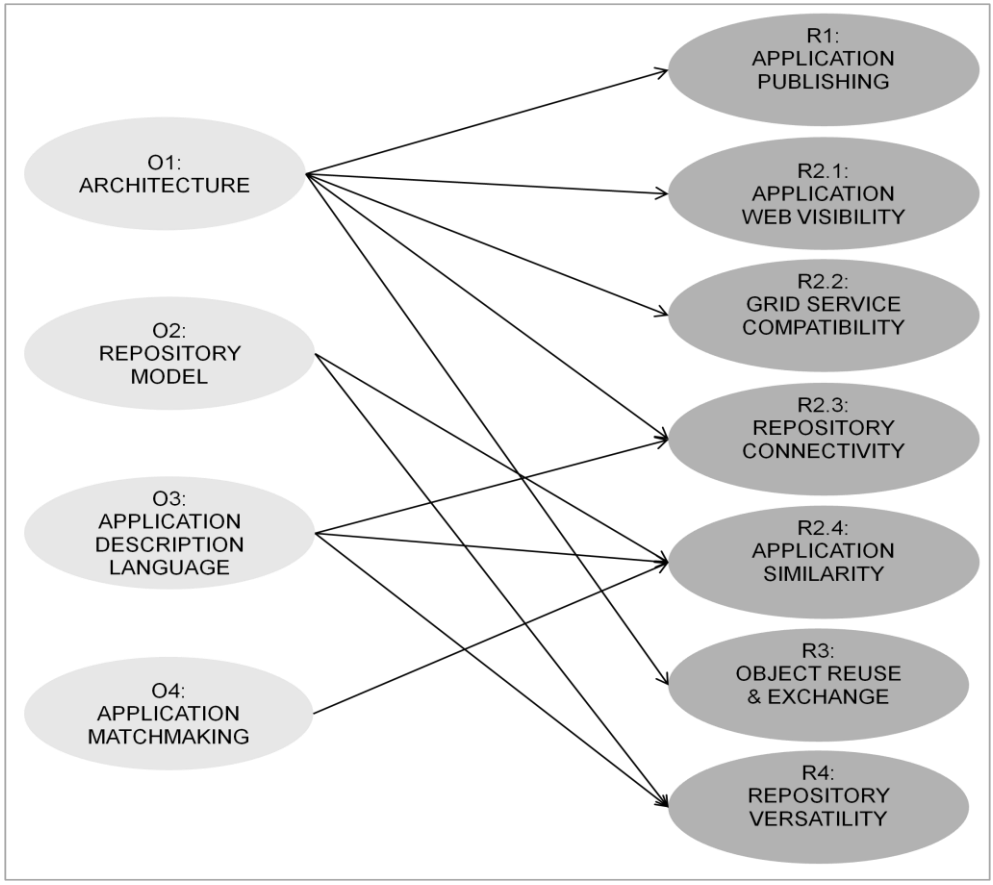


Figure 2-10: Research objectives and their relations to requirements R1-R4

Chapter 3

The Grid Application Meta-Repository System

This chapter describes the Grid Application Meta-Repository System (GAMRS) as the solution proposed by this research to respond to the challenges identified in the previous chapter. Chapter 2 highlighted the diversity of repository implementations currently found on Grid, as well as the problems posed by this heterogeneity in the area of Grid application repositories. The chapter concluded with the specification of a set of requirements (i.e. **R1-R4**) and four research objectives (i.e. **O1-O4**) essential for the design and functionality of a repository that would address the problem of heterogeneity.

In order to meet the research objectives **O1-O4**, this research proposes a new Grid application repository called **The Grid Application Meta-Repository System (GAMRS)**. In line with the four research objectives, this chapter covers the four repository aspects identified in Chapter 2: repository architecture, repository model, application description language, and Grid application matchmaking methods. Each aspect is described in a separate section, which presents a short overview, the design principles, the solution, and the functionality of that module. The chapter

concludes with a summary of the novel capabilities of GAMRS, which allow it to meet requirements **R1-R4**.

3.1. GAMRS Architecture

3.1.1 Overview

In line with the first objective (**O1**) of this research, GAMRS was designed to function as a Grid application repository with the additional capability to connect different types of Grid application repositories notwithstanding their different implementation technologies, methods of access and authentication, communication protocols, and transport protocols (**R2.3**).

The GAMRS architecture specifies abstract interfaces that permit the connection, authentication, and retrieval of information about applications stored in repositories connected to GAMRS. The architecture specifies human- and service-friendly interfaces (i.e. GUI/web, HTTP/REST, OGSI/WSRF) that can be used to publish Grid applications directly in the service's repository **(R1)**. These interfaces can also be used to retrieve information about Grid applications stored or referenced by GAMRS.

Furthermore, the design of this service provides a solution to the *application discovery* problem **(R2)**: Grid services can access information about applications via the standard OGSI/WSRF Grid interface, while through the HTTP/REST interface all the applications stored on GAMRS or referenced by it become visible to the Web and, therefore, available to a much larger community interested in Grid applications.

The GAMRS architecture was also designed to be compatible with OAI standards, with support for OAI-PMH and OAI-ORE protocols, which facilitates communications with other OAI-compatible repositories and permits easy discovery of objects through metadata harvesting. Furthermore, the GAMRS architecture

suggests that objects stored in the repository are described in a language capable to embed (or refer to) metadata and associated datastreams (i.e. FOXML, OAI-ORE implementations) – digital assets related to an object , i.e. files – which would allow for the straightforward relocation and exchange of repository objects (**R3**).

The design of GAMRS had to take in consideration all the challenges identified by this research, including the system of identification of similar Grid applications. Consequently, the architecture design also contains the Matchmaking Service, which will be discussed in detail in Section 3.4.

3.1.2 Design

As none of the application repository architectures currently used in Grid is able to answer to the challenges identified by this research, a new architecture had to be designed for GAMRS. Figure 3-1 depicts the GAMRS architecture:

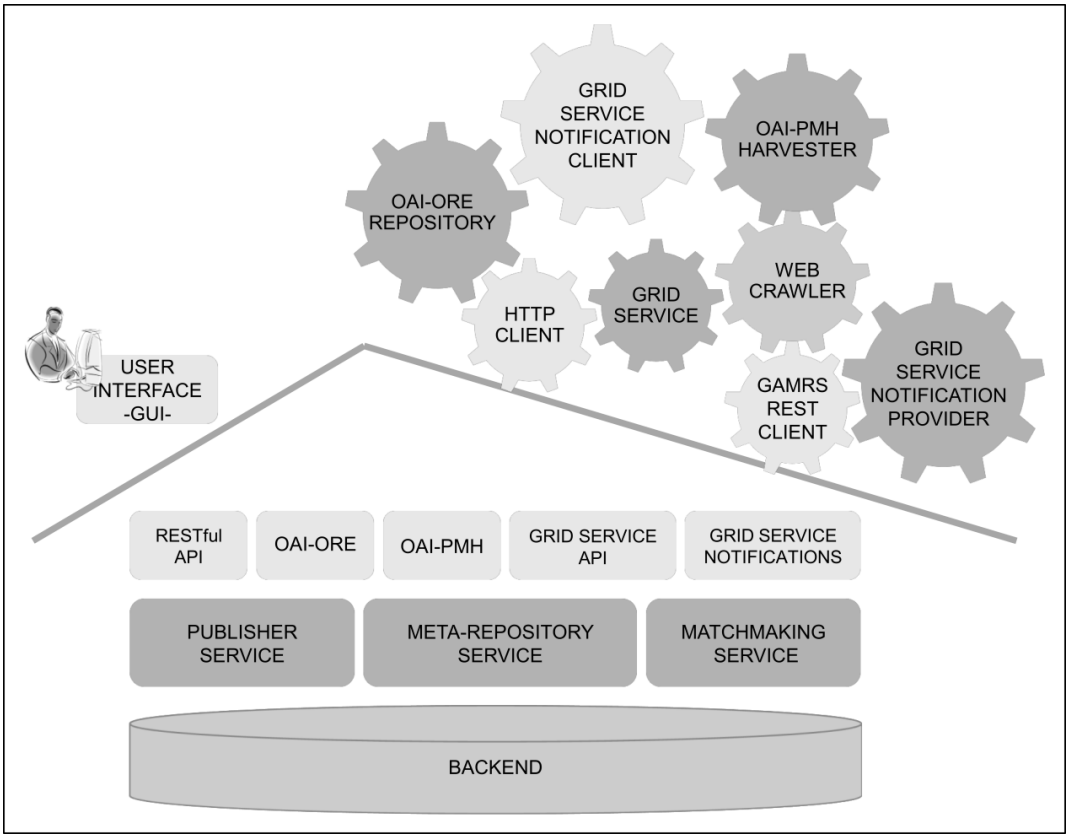


Figure 3-1: GAMRS Architecture

The system consists of three core services: **Publisher, Meta-Repository** and **Matchmaking**. These core services are designed and implemented separately so that they can also be used independently and can easily be extended with new methods in the future.

- Should new repository standards for publishing emerge in the future, they could always be added to the **Publisher** service.
- Should new application repositories be added to Grid, the **Meta-Repository** service would already have a connection interface and new adapters could be written and integrated to it.
- The **Matchmaking** service finds similar applications in repositories connected to GAMRS and is also implemented as a separate service in order to allow new matchmaking methods to be added to it.

The three core services are backed up by the GAMRS **Backend** module, which is responsible for the actual storage of repository objects.

GAMRS PUBLISHER SERVICE

This service is responsible for the publishing of Grid application information for various clients. It exposes all the necessary interfaces through which objects stored in the repository can be accessed or published: HTTP/REST, OAI protocols (i.e. PMH and ORE), and an OGSI/WSRF Grid Service interface. Examples of services that can access GAMRS are: HTTP clients, other OAI-ORE compliant repositories, any OGSI/WSRF Grid service, web crawlers, OAI-PMH harvesters, and WS-Notification subscribers. Human users can use the graphical interface (GUI) to search, publish, and retrieve Grid application information and application-related objects from the repository.

The following figure (i.e. Figure 3-2) offers an overview of the Publisher service architecture, highlighting the various modules that provide the GAMRS access interfaces:

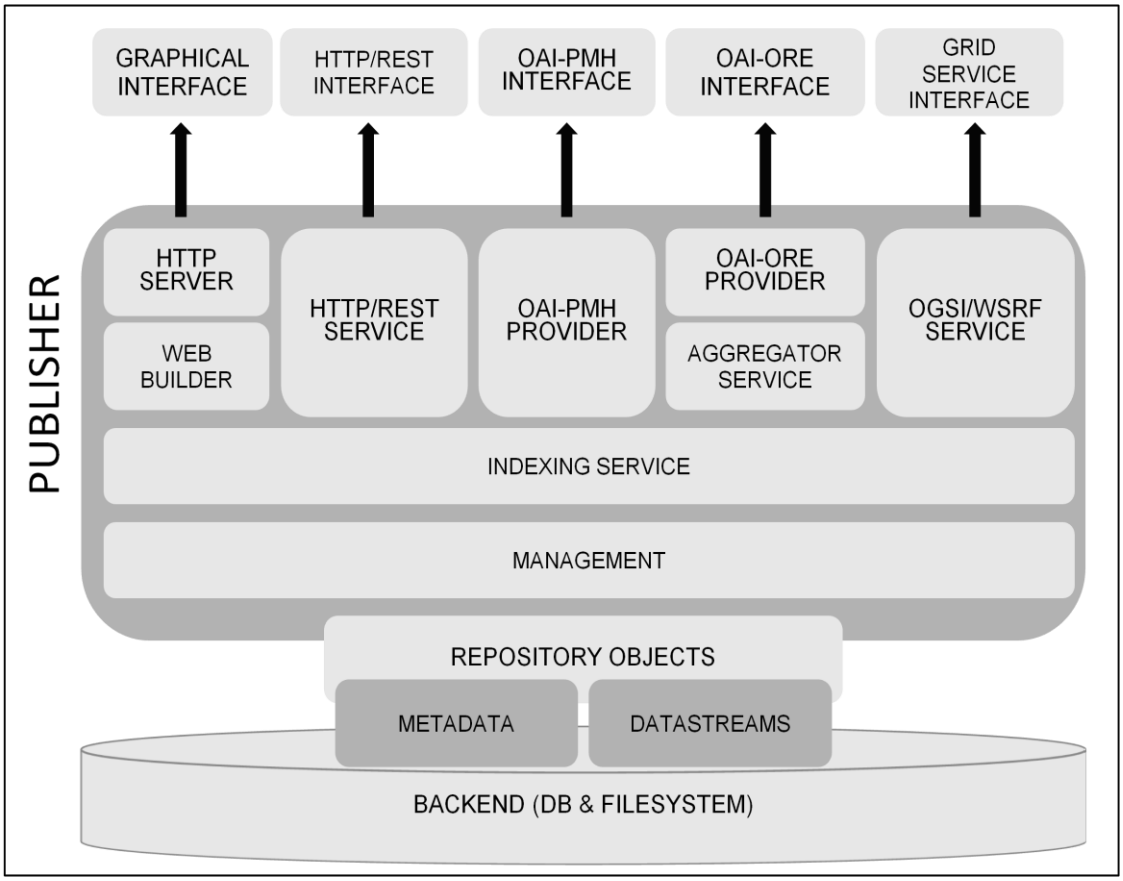


Figure 3-2: GAMRS Publisher service architecture

The *GUI* interface permits users to create, retrieve, update, and delete objects stored in repository and metadata associated with them. GAMRS differentiates between the *metadata* associated with an object and the actual file (also called *datastream* in literature), which represents the digital representation of the object. These are stored on the backend, with the help of a relational *database* – which maintains a coherent representation of the relations between objects, metadata, and datastreams – and a *filesystem,* which is used for file storage. In order to allow fast searches over the metadata associated with the objects, the Publisher service employs an *Indexing Service*, which indexes and sorts the information about the objects stored in the repository and which, ultimately, is used by all the other modules for querying and listing of such information.

With the help of a *Web builder* module and a *HTTP server* users are provided with a suite of web pages in which they can perform in a graphical, human-friendly

manner the CRUD (create/retrieve/update/delete) actions mentioned above. In addition, the graphical interface offers search facilities to human users. These facilities are used to perform queries (with the help of the *Indexing Service*) on the metadata associated to applications stored or referred by GAMRS.

The low-level transactions with the backend and the error handling system, as well as the authentication and policy assessment (i.e. permissions) systems are implemented in the *Management* module.

The *HTTP/REST service* is built around the transfer of *representations of resources*. The *resource* in this case is a repository object, while the *representation* of a resource is represented by a document that captures the current state of a resource. The methods used in a HTTP/REST interface are the well-known HTTP GET, PUT, POST, and DELETE; and they correspond roughly to the CRUD verbs READ, UPDATE, CREATE, and DELETE. [139] A GAMRS object can be created with the help of a POST method and a document (usually written in XML) which contains together the datastream and the metadata associated with it. An object can be *read* with the help of a GET method, which retrieves the object's description document (i.e. metadata and datastream) from the repository. An *update* can be done via a PUT method plus an object description document, which will replace the one already stored in the repository; while an object purge can be done via a DELETE command. However, more complex actions can be performed with the help of HTTP/REST interface such as, for example, searching for a particular object. The HTTP/REST service used in GAMRS permits searching for repository objects by querying the Indexing Service.

The following figure (i.e. Figure 3-3) gives the implementation details for an "application search" scenario. It shows the sequence of actions, the GAMRS Publisher's modules which participate to the scenario, and the message exchange between these modules.

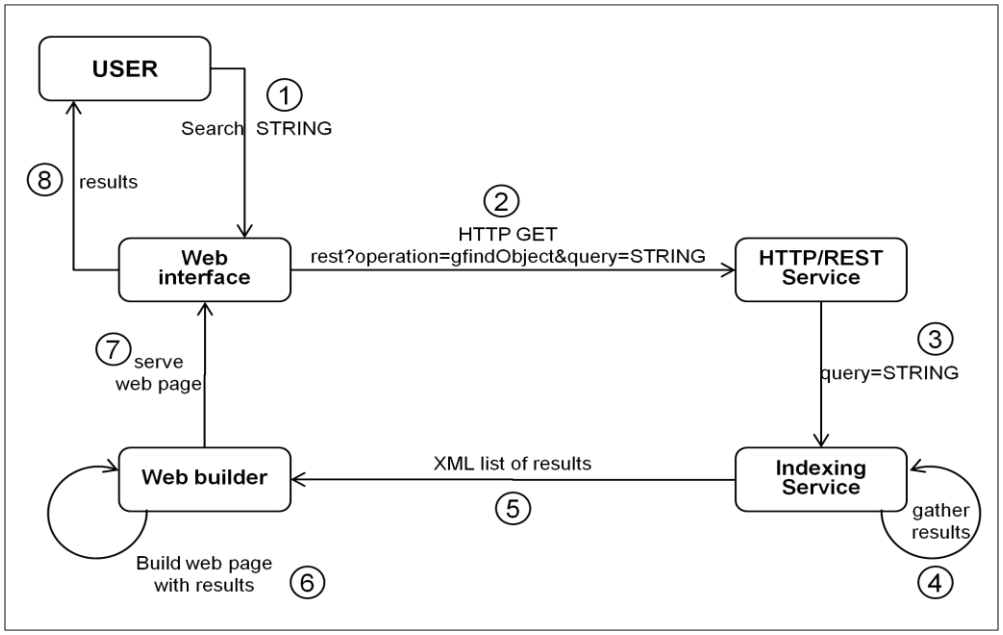


Figure 3-3: Using the Publisher's HTTP/REST service to search for applications in GAMRS

The *OAI-PMH Provider* is used for metadata harvesting and its functionality is very similar to that exhibited by the HTTP/REST Service. However, there are two major differences between the two services: first, as opposed to the HTTP/REST operations – where the names of actions can be user-defined (e.g. *findObjects, ingestObject*), the OAI-PMH uses standard names for its actions as defined in [37] (e.g. *ListIdentifiers, ListRecords, GetRecord*); second, while in a REST implementation one can use all the four CRUD actions, OAI-PMH limits itself only to the READ operation. The following figure (i.e. Figure 3-4) gives the implementation details for a "list all records in the repository" scenario using the OAI-PMH standard.

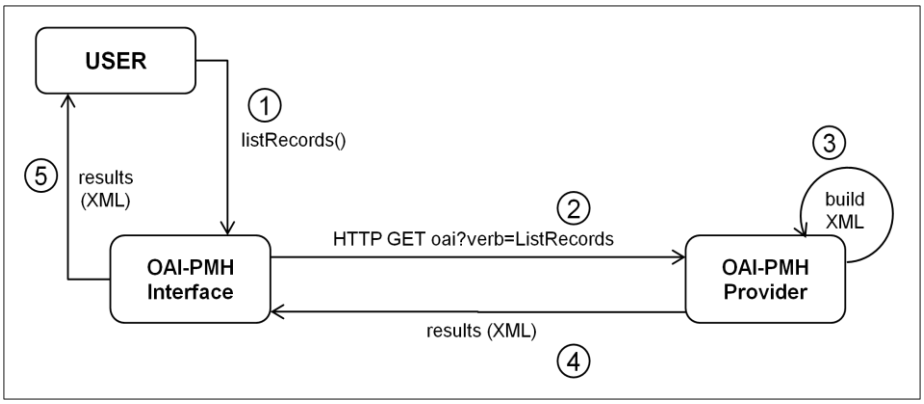


Figure 3-4: Using the Publisher's OAI-PMH provider to list all the records stored in GAMRS

Both the OAI-PMH and HTTP/REST services help with application discovery for potential users, since these services are usually used by web crawlers and harvesters working behind web search engines. Ultimately, with the help of OAI-PMH and HTTP/REST services, GAMRS helps improve the application's visibility on the Web. The following figure (i.e. Figure 3-5) gives the implementation details for a "search for a specific application" scenario using the OAI-PMH standard.

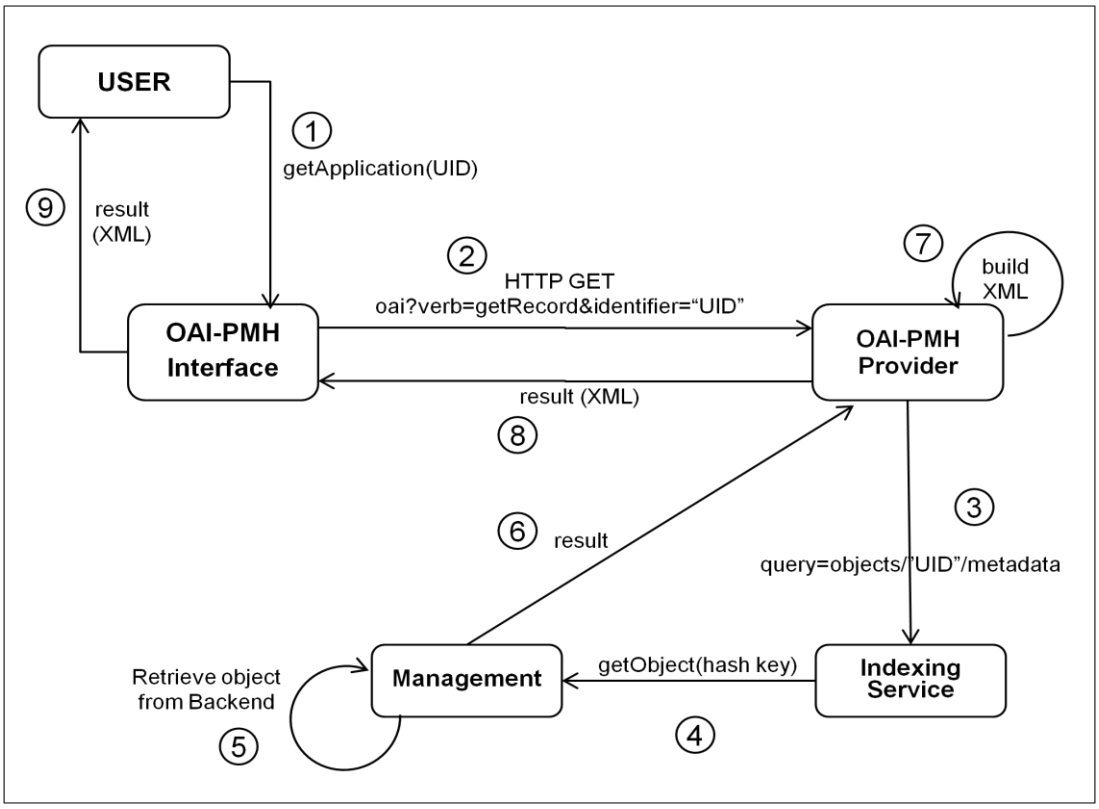


Figure 3-5: Using the Publisher's OAI-PMH provider to retrieve a specific application (metadata only) from GAMRS

The *OAI-ORE Provider*'s functionality is very similar to GET actions implemented in the HTTP/REST interface; namely, it retrieves the object stored in the repository in a serialized form including metadata and datastreams. If the datastream (i.e. file) is represented by an XML-like structured document, the datastream will be automatically embedded in the serialized form of the object. Otherwise, the datastream is only referenced with its full URI in the serialized form of the object.

The OAI-ORE standard introduces the concept of Aggregations and Aggregated Resources (i.e. an Aggregation is simply a set of Aggregated Resources, all of

which are represented by URIs). [140] The main difference from the document described in the HTTP/REST service is that the OAI-ORE provider works by building an XML document according to the OAI-ORE standard specifications, which maps the predicate *ore:aggregates* to metadata and datastreams.

The OAI-ORE interface exposed by the GAMRS Publisher Service can be used by other OAI-ORE compliant repositories to easily exchange and reuse repository objects. Moreover, by handling the serialized form of the complete object (metadata and datastream), administrators can easily implement fail-over and backup solutions for their repository objects. The following figure (i.e. Figure 3-6) gives the implementation details for a “retrieve a specific application” scenario using the OAI-ORE standard.

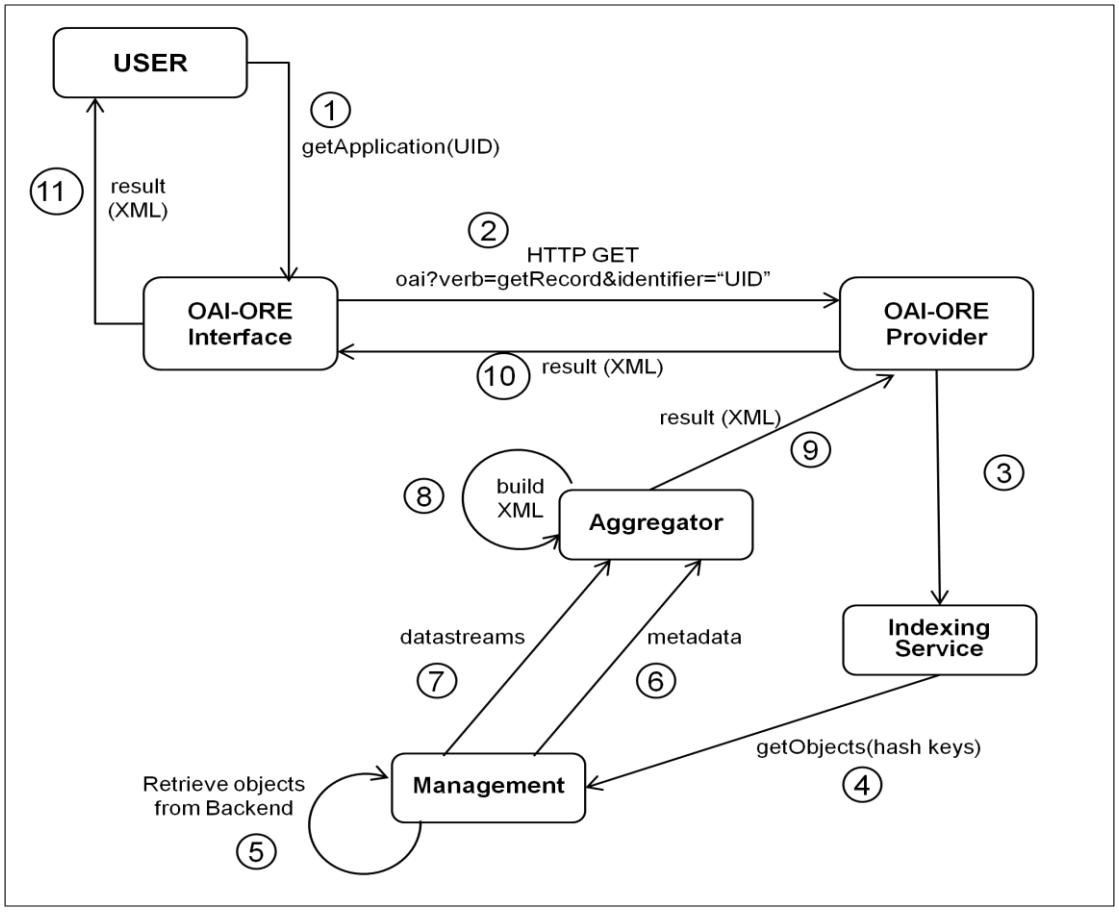


Figure 3-6: Using the Publisher's OAI-ORE provider to retrieve a specific application (metadata and datastreams) from GAMRS

The *OGSI/WSRF* service is present in the Publisher Service's architecture in order to permit Grid services to interact with GAMRS in a serviceable manner in line with the OGSI/WSRF standard. This service allows the same actions as those described in the HTTP/REST service, only that their implementation is consistent with the specifications found in the WSRF standard suite. The serialized form of each repository object is seen as a WS-Resource and CRUD operations can be performed on these resources.

While the majority of current Grid repositories cannot be queried directly by OGSI/WSRF Grid services, the applications stored in them become visible through the Publisher service's OGSI/WSRF interface if these repositories are connected to GAMRS.

GAMRS META-REPOSITORY SERVICE

The Meta-Repository Service is responsible for connecting together various types of Grid application repositories, independent of the underlying technology these are built on, because it can negotiate through their different methods of access and authentication, communication and transport protocols. In this way the service grants access to any application stored in connected Grid repositories, notwithstanding the differences between various methods.

Figure 3-7 shows the Meta-Repository service connecting four repositories, each of them employing different security methods (i.e. GSI, username/password, public access), different access interfaces (i.e. Grid Service access interface, JSR-168 access interface, HTTP/REST interface, Web Service interface) and each of them storing different types of Grid applications (i.e. stand-alone jobs, workflows, web services), which in turn are described using different application description languages (JSDL, LCID, SCUFL, and WSDL).

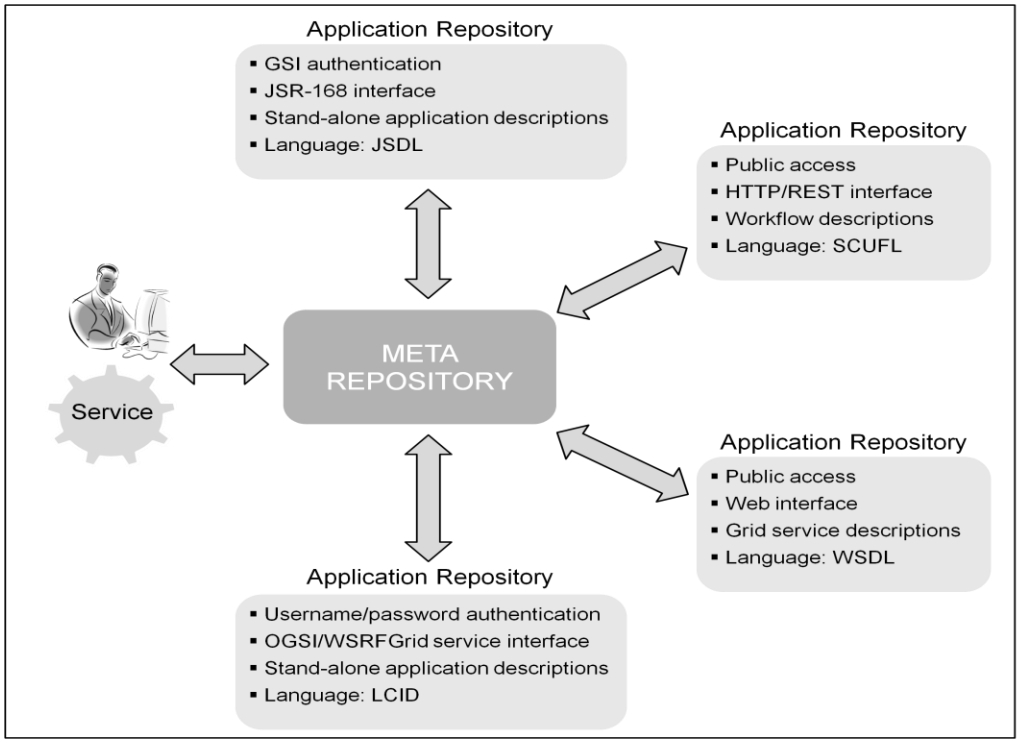


Figure 3-7: Connecting different types of Grid application repositories

Figure 3-8 shows the three-layered architecture of the Meta-Repository Service – the *Access* layer, the *Management* layer and the *Storage* layer.

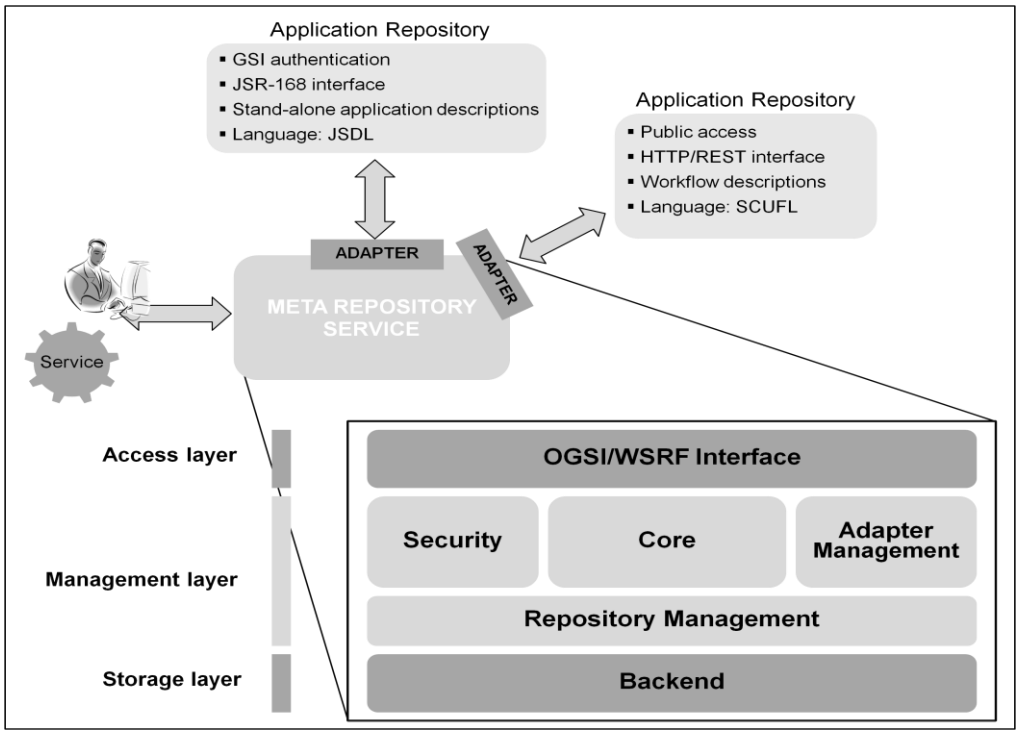


Figure 3-8: Meta-Repository Service architecture

The *Access layer* exposes an OGSI/WSRF interface through which users and client services can interact with the Meta-Repository service. Through it, users can authenticate within the system and can retrieve application description documents and application metadata from repositories connected to the Meta-Repository Service.

The logic and the functionality associated with these actions are addressed by the *Management layer*:

The first module in the management layer is *Security*. This module deals with authentication and authorization methods both on the Meta-Repository service and on connected repositories. The Meta-Repository service uses GSI to achieve full-compatibility with other Grid services; however, authentication on repositories with security infrastructures other than GSI is achieved with an orchestrated effort of the *Security* module, the *Core* module and the *Adapter* responsible with communication with the corresponding repository. The Meta-Repository service is deployed in a web application container able to understand X509 certificate/proxy authentication. Users require a valid Grid certificate issued by well-known Certification Authorities in order to use the service. The security module maintains a list of the public certificates of these Authorities, as well as a white-list of X509 *distinguished names* taken from the certificates belonging to users, which are allowed to use the service. When a user requests access to the service, s/he would be required to present his/her certificate. The *Security* module compares the distinguished name of the certificate with those maintained in the white-list. If a match is found (i.e. the user is allowed to use the service) the *Security* module compares the signature of the Certification Authority found on the user certificate against the signature of the Certification Authority already recorded in the list of public certificates. If the signatures match (i.e. the certificate is genuine) and the certificate is still valid (i.e. did not pass the expiration date), the user is permitted to use the service.

The *Core* module is responsible for the integration and supervision of all the other Meta-Repository service modules. It specifies the interfaces of communications

between different modules; it translates the commands received from the access interfaces; it specifies the formal interfaces for repository queries and answers; it distinguishes between queries directed to the Meta-Repository service and queries intended for application repositories connected to the service; and it relays the query to the appropriate module. The *Core* module is therefore the centre point of the system and it coordinates all the activity between different modules of the Meta-Repository service. The Meta-Repository service implements commands (e.g. COMM_ADD_APPLICATION, COMM_GET_APPLICATION, etc.) in order to provide access and allow CRUD actions on the repository objects, as well as to connect to other providers and update the list of applications they expose.

The *Adapter Management* module is responsible for instantiating adapters, as well as for supervising and accounting for their actions.

The *Repository management* module is responsible for communication with the Backend. This module provides access to the CRUD operations (i.e. Create/Retrieve/Update/Delete) exposed by the backend and also supervises the actions associated with these operations. This module contains the implementation of the Meta-Repository commands described by the *Core* module in conjunction with an implementation of the Backend API (i.e. driver implementation) necessary to run the database transactions and the file storage transfers of repository objects to and from the Backend.

The *Storage layer (Backend)* is responsible for the actual storage of repository objects. It consists of a relational database and a filesystem (local or remote via a storage server). The database is used to store the metadata associated with each repository object, as well as the relations between such objects. The filesystem is used to store serialized forms of the repository objects and the datastreams such as binaries, source code, licenses, virtual machines etc.

Users access the Meta-Repository service through its access interface, and before any command is issued, the *Security* module performs GSI authentication procedures on the user's credentials. If access is granted, users can interact with the application repository via Meta-Repository service commands. Once a

command is issued, the *Core* module will interact with the storage layer through the *Repository Management* module, which implements the necessary drivers to communicate with the *Backend*. In this fashion, users can store, retrieve, modify, and delete objects from the repository. One particular command – COMM_UPDATE_PROVIDER – updates the list of applications from the repositories connected to the Meta-Repository service.

The Meta-Repository service allows different types of repositories to be connected to it through the use of *Adapters*. From the service point of view, an adapter provides a uniform connection to other repositories. In order to accomplish that, the adapter implements the following four modules (see Figure 3-9 below):

- The *Communication protocol* module is responsible for accessing the connected repository. This requires knowledge of the repository API.
- The *Authentication and authorization* module helps the Meta-Repository service to mediate the authentication process on connected repositories. Usually, the authentication process differs from one Grid application repository to another (e.g. GSI, username/password, free access). This module is therefore required to implement the authentication process appropriate to each of the repository connected to the Meta-Repository service.
- The *Transport protocol* module helps the service retrieve the application metadata and the application-related objects stored on connected repositories. Every repository can implement a different transport protocol (e.g. HTTP, SOAP, GridFTP). Subsequently, the adapter must implement the transport protocol that is appropriate to the respective repository technology.
- The *Meta-Repository interface* is the module responsible for the management of commands received from the Meta-Repository service, as well as for formatting the responses according to those particular interface specifications accepted by the service.

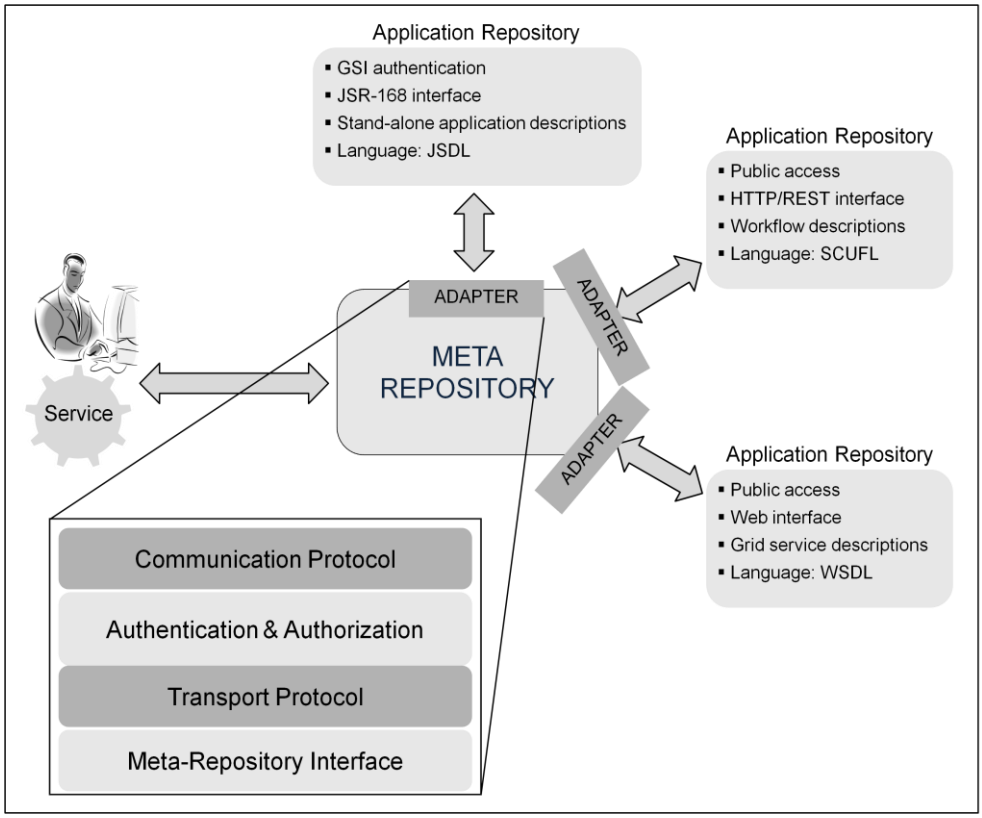


Figure 3-9: Meta-Repository service Adapter architecture

The Meta-Repository service defines the high-level command COMM_UPDATE_PROVIDER for interaction with connected repositories. This command updates the list of applications exposed by the repositories connected to the Meta-Repository service. The command accepts as arguments universal identifiers of the providers needed to be contacted for updates or, if provided with no arguments, will contact all the repositories connected to the service at that time. The adapter translates this command into five different subcommands defined in its interface:

```
CONNECT
DISCONNECT
GET_APPLICATIONS
GET_APPLICATION_DESCRIPTION
RUN_QUERY
```

The implementation of the adapter requires knowledge of the communication protocols, authentication methods, and transport protocols used in repositories connected to the Meta-Repository service. Using this knowledge, the commands enumerated above will be implemented to provide the following functionality:

- CONNECT – This command is used to connect to the remote system. Its implementation requires knowledge of the transport protocol and communication API employed on the remote system. This stage may require authentication from the remote repository service. The implementation of this function should provide all the remote retrieval locations (e.g. myproxy servers, voms servers) or files (i.e. PKI certificates/ keys; X509 proxies; username/password tuples) necessary to the authentication process.

  Note: Some systems delay the authentication process until a formal query is passed to the system – for example, to list applications or search for applications. In this case, the authentication phase is not implemented in command CONNECT, but is moved to the implementation of another command such as GET_APPLICATIONS or RUN_QUERY. The authentication system implemented on the remote service infrastructure needs to be known in advance.

- DISCONNECT – This command is used to perform a *clean* disconnection from the remote system. In many cases the break of or the simple non-utilization of the communication channel for a certain period of time eventually breaks down the connection with the remote server. However, most servers would implement a *disconnect* option, which gives them a better management over their resources, such as de-allocation of memory, closure of file descriptors, or cleanup of temporary files. If the disconnect feature is present in the remote server's API, then it should be used in the implementation of the Adapter.

- GET_APPLICATIONS – This command is used to retrieve the list of applications from the remote server. The implementation of this command requires knowledge of the remote server's API in terms of the formal enquiry and formal response that the Meta-Repository sends to and receives from the remote server. The list contains tuples like the following: (the name of the application; the unique reference point that can be used to retrieve application metadata and datastreams from the

remote server).

- GET_APPLICATION_DESCRIPTION – This command is used to retrieve the formal description document written in ADL, which is assigned to each application by the remote repository. Each application description document is added to the list created by the command GET_APPLICATIONS to an entry corresponding to the application it describes. This list is kept by the Adapter for further reference. After performing the next update the Adapter will compare for each application the description document received from the server with the ones kept in its list. If any differences exist, it will replace the old description document with the new one, at the same time instructing the Core module to modify the object stored in GAMRS accordingly. If the application does not exist anymore on the remote repository, the Adapter will remove it from its list and will instruct the Core module to delete the object stored in the GAMRS repository. Similarly, if the remote repository exposes new applications, the Adapter's list will be updated with the new entries and new objects will be added to the GAMRS repository.
- RUN_QUERY – This command performs various personalized queries on the remote repositories. Under current requirements, the Meta-Repository service only needs the list of applications and their formal descriptions from the connected repositories in order to make these applications visible in GAMRS. However, as repositories evolve, they begin to store more types of application-related objects and more diverse metadata. The RUN_QUERY command can give GAMRS the means to retrieve such objects and collect more information about an application, provided that such information is needed by future extensions or meta-repository designs.

Upon receiving the COMM_UPDATE_PROVIDER command, the Adapter will use the command CONNECT to establish a communication channel with the remote repository. Next, it will issue the GET_APPLICATIONS command and will create a list of applications, each entry containing the name of the application and its remote

access point reference. For each application reference point, the Adapter issues the GET_APPLICATION_DESCRIPTION command and retrieves the formal description document of the application, which is then used to update its list of applications. Next, the Adapter uses a transformation method to convert the formal description document retrieved from the remote repository into the application language used by GAMRS. After that, the GAMRS repository object document is created by adding metadata and two datastreams: the formal application description document written in the native language of the remote repository; and the formal application description document written in the language employed on GAMRS.

The document of the repository object document is then passed on to the Repository Management module to be added to the GAMRS (or to modify an existing object). Upon a new addition, the unique GAMRS identifier of the object is passed back to the Adapter to be stored in its list of applications for further update actions. The following figure (i.e. Figure 3-10) gives the implementation details for the COMM_UPDATE_PROVIDER command. It shows the sequence of actions, the GAMRS Meta-Repository's modules which participate to the scenario, and the message exchange between these modules.

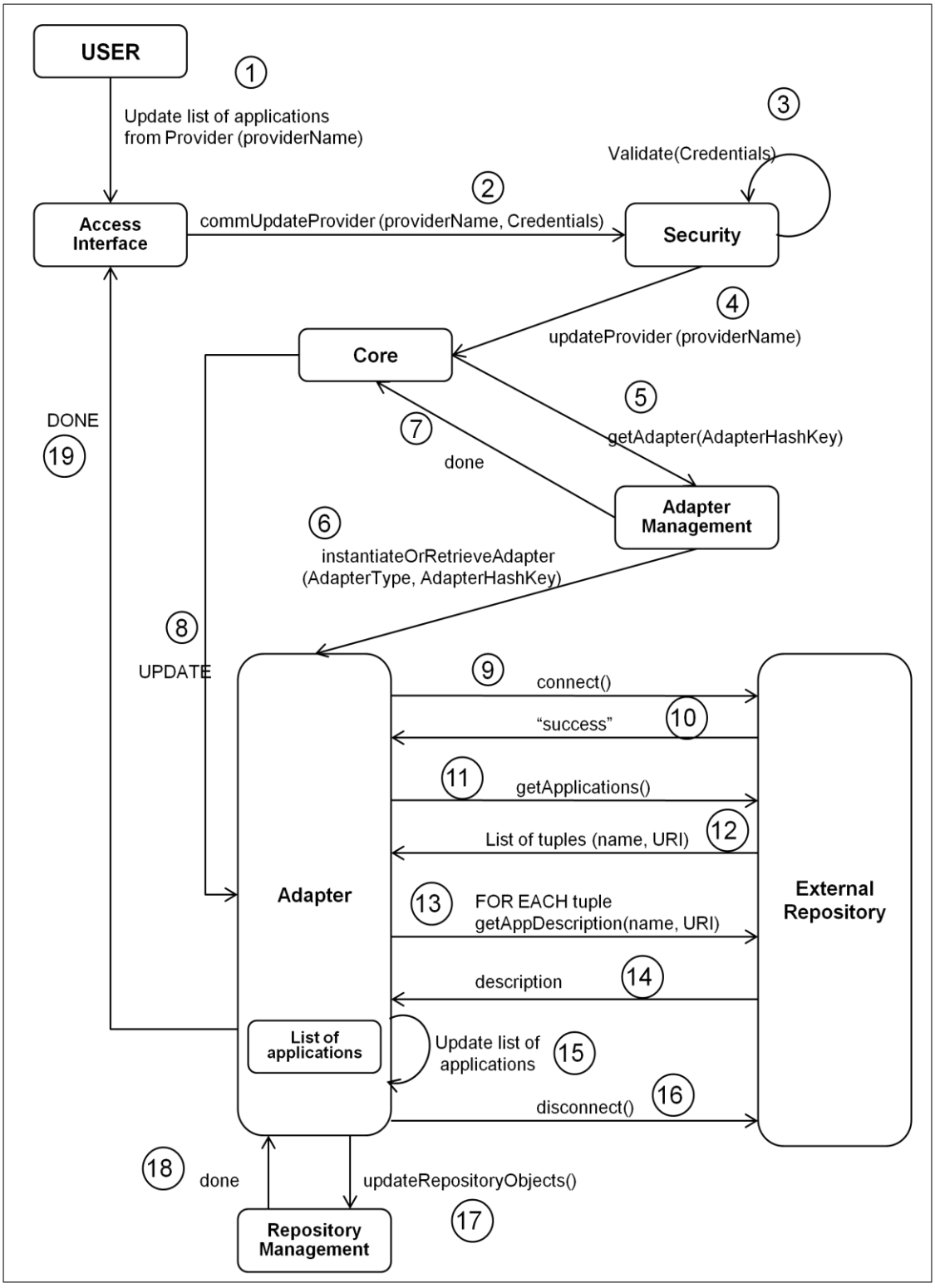


Figure 3-10: Meta-Repository service – Implementation of the COMM_UPDATE_PROVIDER command

GAMRS MATCHMAKING SERVICE

The Matchmaking service is a service used to find similar or identical applications stored on different repositories connected to GAMRS. This service has the ability to process application description documents written in the application language used by GAMRS and can also use application-related objects stored in GAMRS. The Matchmaking service uses different types of matchmaking techniques to analyze the description documents and its service architecture allows developers to extend it easily with further matchmaking methods. This service can also implement methods that are able to process not only description documents, but also other application-related objects such as hash sums, binaries, test files, source code, and list of dependencies.

The limited timeline of a PhD research only allowed for the implementation and performance analysis of four matchmaking methods – syntactic, string-distance, application-running, and binary matching. The *syntactic* algorithm processes description documents and relies on the application information contained in them. The *string-distance* methods can help with the identification of similar applications on the basis of the information contained in the free-text description of the application. This research proposes a new technique of improving the accuracy of string-distance metrics by using *entropy-generated stop-lists* and test results have showed that this can increase the performance of string-distance methods. The *application-running* method compares two applications by running two application binaries with a common set of input files (retrieved from one of the application test suites) and compares their output set. The *binary matching* method uses application binaries stored in GAMRS and hash sums to identify identical applications.

The architecture proposed for the Matchmaking service and the algorithm used for that are extendable, hence future research can implement and analyze the performance of other matchmaking methods when applied to the objects stored in the GAMRS repository.

Section 3.4 in this chapter offers an in-depth overview of the GAMRS Matchmaking service.

GAMRS BACKEND

The backend is responsible for the actual storage of repository objects and the backend structure follows the layout of the GAMRS repository model. The backend consists of a relational database and a filesystem (i.e. local or remote via a storage server). The database is used to store the metadata associated with each repository object, as well as the relations between such objects. The filesystem is used to store serialized forms of the repository objects consisting of metadata and datastreams. As mentioned before, if the datastream is not represented by an XML-like structured document, it will be only referenced with its full URI in the serialized form of the object. Therefore, the filesystem can also be used to store non-XML datastreams, such as binaries, source code, licenses, or virtual machines.

All services may access the backend directly; however, each module can also access the backend by using the interfaces provided by the Publisher service. In conclusion, if GAMRS is extended in the future with other services, there will be no need for these services to implement a new communication module with the backend because they could use existing interfaces provided by the Publisher service.

3.1.3 Summary

The following table (i.e. Table 3-1) is a reiteration of Table 2-2 from Chapter 2, Section 2.5.1, with the addition of GAMRS repository capabilities. This table summarizes the capabilities of existing Grid application repository architectures and those of the GAMRS architecture against requirements **R1-R3**, which correspond to objective **O1** (see Chapter 2, Figure 2-10):

Table 3-1: Current Grid application repository architectures and GAMRS architecture vs. **R1-R3**

	R1: PUBLISHING	R2: DISCOVERY	R3: EXCHANGE & REUSE
BDII	- Publishing done by automated services via scripts that contain suites of console commands; - No graphical/web interface for human users.	- Console commands containing LDAP queries; - No OGSI/WSRF Grid service interface; - No Web visibility; - No HTTP/REST interface; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO
CHARON	- Command line only for human users; - No access support for services;	- Collection of static Web pages; - No OGSI/WSRF Grid service interface - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO
GEMLCA	- Graphical interface for human users; - OGSI/WSRF Grid service interface for services.	- OGSI/WSRF Grid service interface; - Human users can find application information through PGRADE portals or using a GEMLCA Service Client; - No Web visibility; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO
NGS AR	- Graphical interface for human users - No access support for services	- JSR-168 web application interface for human users; - No OGSI/WSRF Grid service interface; - No HTTP/REST interface; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO
GRIMOIRES	- Human users and services can register web services via UDDI clients.	- Visible to UDDI clients; - Visible to human users through a collection of static web pages.	N/A

myExperiment	- User-friendly web interface for human users; - HTTP/REST interface for services.	- Intuitive web interface for human users; - Exposes HTTP/REST interface; - No OGSI/WSRF Grid Service interface; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO
GAMRS	- Graphical interface for human users; - HTTP/REST interface for services.	- Intuitive web interface for human users; - Exposes HTTP/REST interface; - Exposes OGSI/WSRF Grid Service interface; - Supports connections to other repositories; - Exposes a system of identification of similar Grid applications; - Support for OAI-PMH protocol.	- Support for OAI-ORE protocol;

3.2. GAMRS Repository Model

3.2.1 Overview

The second objective of this research **(O2)** was to propose a new model for application repositories, which would achieve uniformity in Grid application presentation and would extend the functionality of these repositories beyond Grid **(R4)**. Current major repository models used on Grid impose limitations on the applicability domains of Grid application repositories. Therefore, the objective was to design a new repository model that would provide a comprehensive description of an application along with a suggestion for a new categorization of application-related objects. This will allow Grid application repositories to be compatible and ready-to-use in conjunction with newly emerging technologies such as virtualization, automatic virtual machine creation, cloud computing, and automatic service deployments, as well as to be ready-to-use in future distributed computing designs.

Following the conclusions of Section 2.5.2, I decided to extend the traditional repository model with the necessary entities that would make it able to describe user-related objects; the provider and its associated policies; and provider-related objects (entities in dark grey in Figure 3-11).

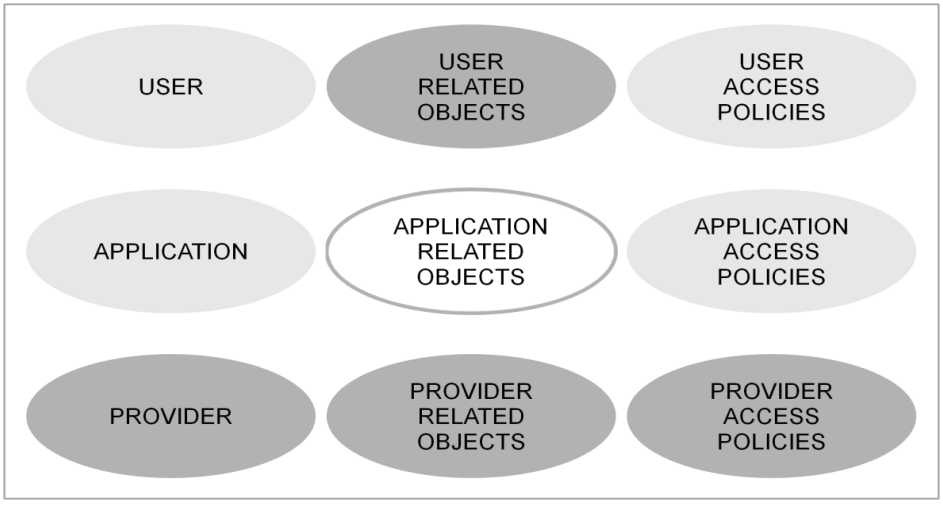


Figure 3-11: Grid application repository model entities

The *User* entity is required by the model to describe a repository user. Users can have *user related-objects* associated with them (e.g. PKI signature certificates, authentication certificates or username/password suites), which can be modelled and stored in the repository and would be available on demand for various scenarios. The *user access policies* describe the actions each repository user is permitted to perform on user entities (e.g. create new users, delete users, etc.). The *Application* entity is required by the model to describe a Grid application stored or referred by a repository. An application entity contains a reference to its *application-related objects* (i.e. description document, binaries, source code, etc.) as well as a reference to the user who created it. The *application access policies* are used to describe those actions that repository users are permitted to perform on application entities and their related objects. The *Provider* entity is required by the model in cases when the repository is connected to other repositories. In a network of inter-connected repositories, each of these repositories is described by a *Provider* entity. These providers can have *provider related-objects* associated with them (e.g. PKI signature certificates, software clients to facilitate access to them, etc.) which can be modelled and stored in the repository as well. A *Provider* entity contains a reference to its applications as well as a reference to the user who created it. The *provider access policies* are used to describe the actions repository users are permitted to perform on provider entities and their related objects.

Moreover, the *Application Related Objects* had to be extended with new types of objects in order to help the GAMRS solution to meet the objectives set out in this research in Section 2.6 .

The suggested set of application-related objects needed to be modelled in a Grid application repository includes:

- *Application description documents:* These should be stored in Grid application repositories since they provide the link to submission systems that allow users to run the application on Grid. Administrators can create preconfigured description files (i.e. templates) in which resource attributes and other Grid specific attributes have already been filled (such as the

minimum CPU frequency, the minimum amount of memory, or the minimum number of file descriptors required for the application to run).

- *Application binaries:* These are referenced in the application description document, but in most cases they are already deployed on a Grid site, and cannot be retrieved, only accessed. Once the application binary is stored in the repository, services which deal with automatic deployment or automatic virtual machine generation can make use of the binary. Furthermore, if the application deployed on a Grid site becomes unavailable but the user insists to use that particular site for personal reasons, the executable could still be staged and run on-demand.
- *Application source code:* This can provide an additional information source for application matchmaking systems and can help with the automatic deployment of applications on heterogeneous machine architectures and operating systems, as it can be compiled directly on the target system.
- *Application libraries:* Libraries should be stored on application repositories as they can help in those cases where compilation is needed. They can also help with automatic application deployment and automatic virtual machine generation.
- *Application dependency software:* As with *Application libraries*, the dependency software is necessary in cases that require automatic application deployment.
- *Application documentation:* This is a useful set of objects for users and administrators who want to know more about the application than the short summary available in the application description document under the attribute *Description*.
- *Application test files:* These can help with matchmaking – matchmaking systems can run a candidate-match application with the set of test input files. If the output files prove to be the same as the test output files there is a high possibility that the applications are the same.

- *Virtual Machine-embedded application*: In order to make the application easily deployable in virtualized environments, one solution is to embed it from the outset in a virtual machine. That allows the application to run in its native environment, overcoming potential architecture- or operating system-related issues.
- *Hash sums*: These were added to the model to help check the integrity of records after the completion of a data transfer. Hash functions are also used in matchmaking algorithms to speed up the matching process.
- *Application licenses*: Since this repository model can also be used to accommodate commercial applications, the application license acceptance is a necessary prerequisite to any automatic deployment process or job submission involving commercial software.

3.2.2 Design

In order to provide a coherent structure for the description of the GAMRS repository a new entity has been added (i.e. the *MetaRepository*), which acts as a container for all the objects described in GAMRS. This entity also helps with the exchange and reuse requirement identified by this research (**R.3**), as it permits the serialization of all GAMRS entities under one root element. In this way, the whole repository can be effortlessly relocated to another repository framework.

The user policy, provider policy and application policy entities have been unified in a single entity called *Policy.* Moreover, in order to permit the better management of users, the GAMRS repository model also employs a *Group* entity.

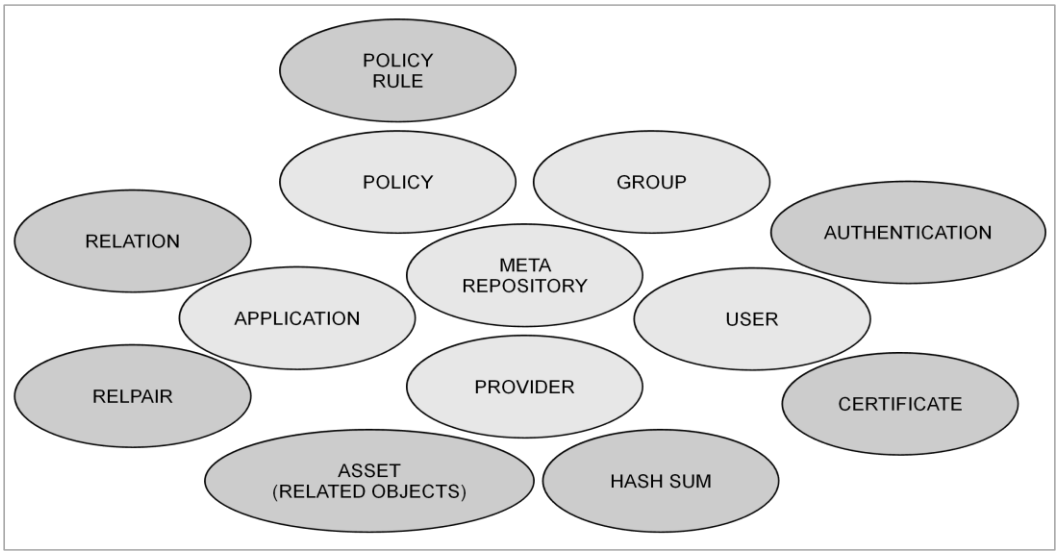


Figure 3-12: Entities in the GAMRS repository model

Apart from the six main entities (i.e. MetaRepository, User, Application, Provider, Policy and Group) the repository model required the addition of seven new entities: PolicyRule, Authentication, Certificate, Asset, Hash sum, Relation, and RelPair. These describe the following aspects:

- Describe and differentiate between policies related to users, groups, applications, providers or other private policies (i.e. PolicyRule).
- Describe and differentiate between various types of authentication methods needed by users or requested by various providers (i.e. Authentication and Certificate).
- Describe and differentiate between various types of objects related to applications, providers and users (i.e. Asset).
- Describe and differentiate between various types of hash sums, which can be used to strengthen data integrity on GAMRS (i.e. Hash sum).
- Describe the property of an application to be identical or similar to a certain extent to another application stored or referred by the repository (i.e. Relation, RelPair).

Figure 3-12 above gives a general overview of the entities in the GAMRS repository model. Each of the entities in the GAMRS repository model will be explained separately below.

Figure 3-13 provides a more in-depth view of the GAMRS model with its six main entities and the relations between them: MetaRepository, User, Application, Provider, Policy and Group.

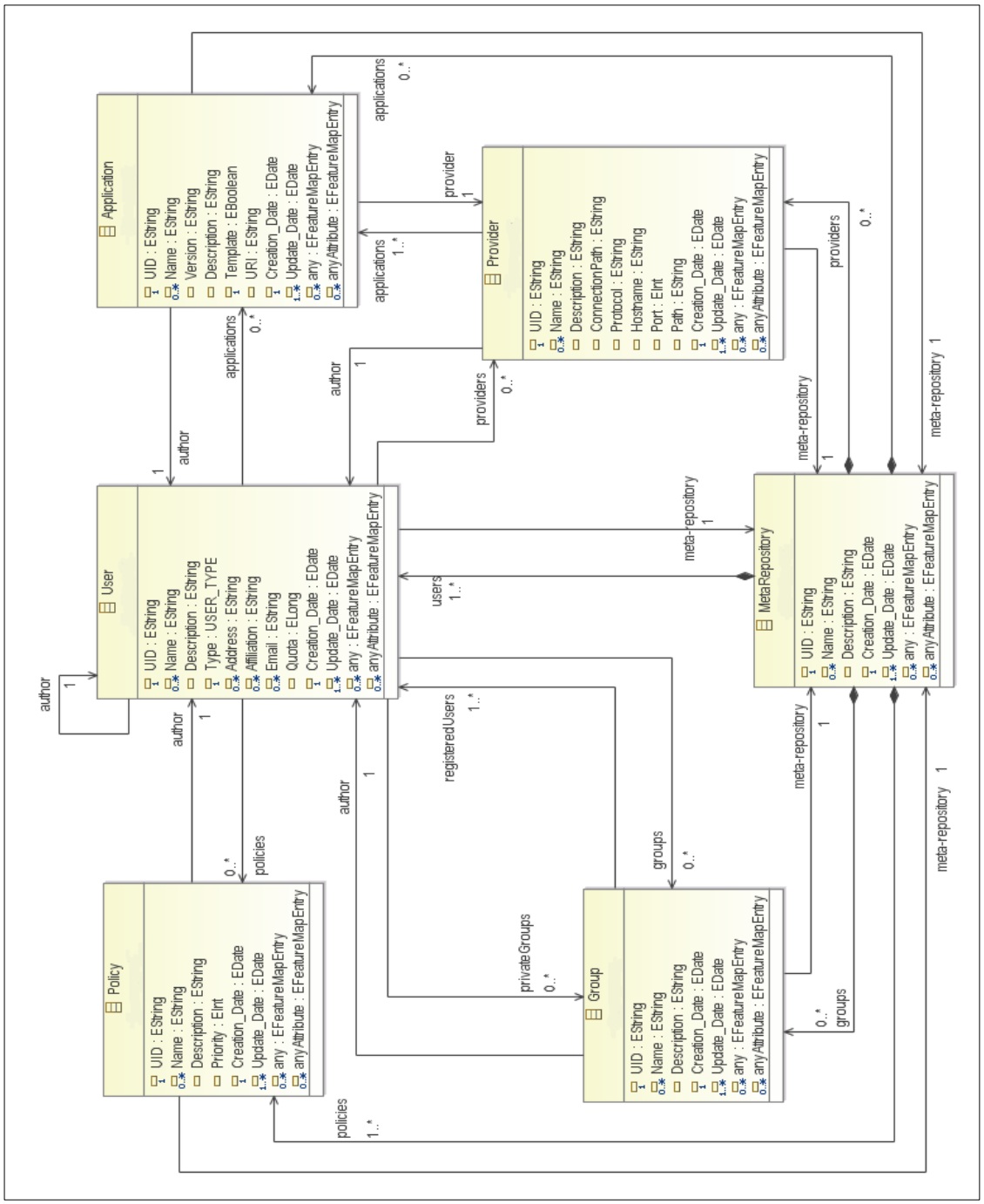


Figure 3-13: The six main entities of the GAMRS repository model

Every entity in the GAMRS repository model contains five attributes and an association to the *User* entity, all of which are used for management purposes, i.e. to uniquely identify the objects, track changes and log their modification history:

- *UID (universal identifier)*: This attribute is needed for the unique identification of the object within the GAMRS. The UID is composed of two parts: The first part represents the fully qualified domain name (i.e. FQDN) of the server where GAMRS is hosted. The second part represents a string, unique to each object within one meta-repository system. The FQDN part of the UID helps with the identification of objects in a federated architecture of GAMRS. Since the architecture can be cascaded, one GAMRS system can be connected to another GAMRS system. Constructing the UID in this way ensures that the objects can be uniquely identified within the federated architecture. This attribute is mandatory and the attribute multiplicity is one.
- *Name*: This attribute is used to model the name of the object. Its main purpose is to allow a friendly identification of objects by human users. As the unique identification of objects can be done via UID, this attribute is not mandatory. The multiplicity of *Name* is one.
- *Description*: Each object may contain a paragraph of free-text, which represents a short description of the object properties and capabilities. This attribute is used to capture such descriptions of objects. The attribute is not mandatory and its multiplicity is 0 - 1.
- *Creation date*: This attribute is used to model the date when the object was created. The attribute is mandatory and the multiplicity is one.
- *Update dates*: This attribute represents the set of dates when the object was accessed and/or modified. It is used to help logging/audit systems to keep track of changes of the object. The attribute is non-mandatory and the multiplicity is 0 - multiple.
- *Author:* With the exception of the *MetaRepository* entity, all other entities in the repository contain a relation to the user who created them. This is

represented by an association to the *User* entity called *author*. This attribute is mandatory and the multiplicity is one.

The GAMRS model is extendible and GAMRS entities can be extended with other attributes and relations to other objects. This is done via two attributes – *anyAttribute* and *any* – that are present in each entity description contained in the model. The attribute *anyAttribute* may be used to add new attributes to an entity description, while the attribute *any* may be used to add new associations between GAMRS entities.

THE METAREPOSITORY ENTITY

The MetaRepository is the central entity of the GAMRS repository model. The MetaRepository represents the container for the main entities of the repository: User, Group, Policy, Application and Provider.

Figure 3-13 shows that the association between the *MetaRepository* entity and the *User* entity, as well as the association between the *MetaRepository entity* and the *Policy* entity, is a 1..* multiplicity, meaning that the meta-repository is bound to have at least one user (i.e. the default user) and one policy (i.e. the default policy). The default user represents the meta-repository administrator and the default policy represents the default administrator policy, which grants him/her permissions to create, modify, and delete other objects. From here on the default user is responsible for the creation and management of users, groups, and policies, as well as for the assignment of such policies to different entities.

THE USER ENTITY

The *User* entity captures the following information: *name, address, affiliation* and *email.* This can be often associated with human users. However, the entity can be extended to include further information about a user (such as *telephone, fax*, *etc.)* via the attribute *anyAttribute.* A quota is associated to each user on the repository storage system (modelled through the attribute *Quota*), which offers a better control over the repository backend and prevents a user from (intentionally or un-

intentionally) filling up the storage, which would in turn make the system unavailable.

A user can be part of one or more groups and can create his/her private group(s) of users (friends). A user can add applications, application-related objects, and provider objects to the system and can also create his/her private policies and apply them to the objects s/he owns.

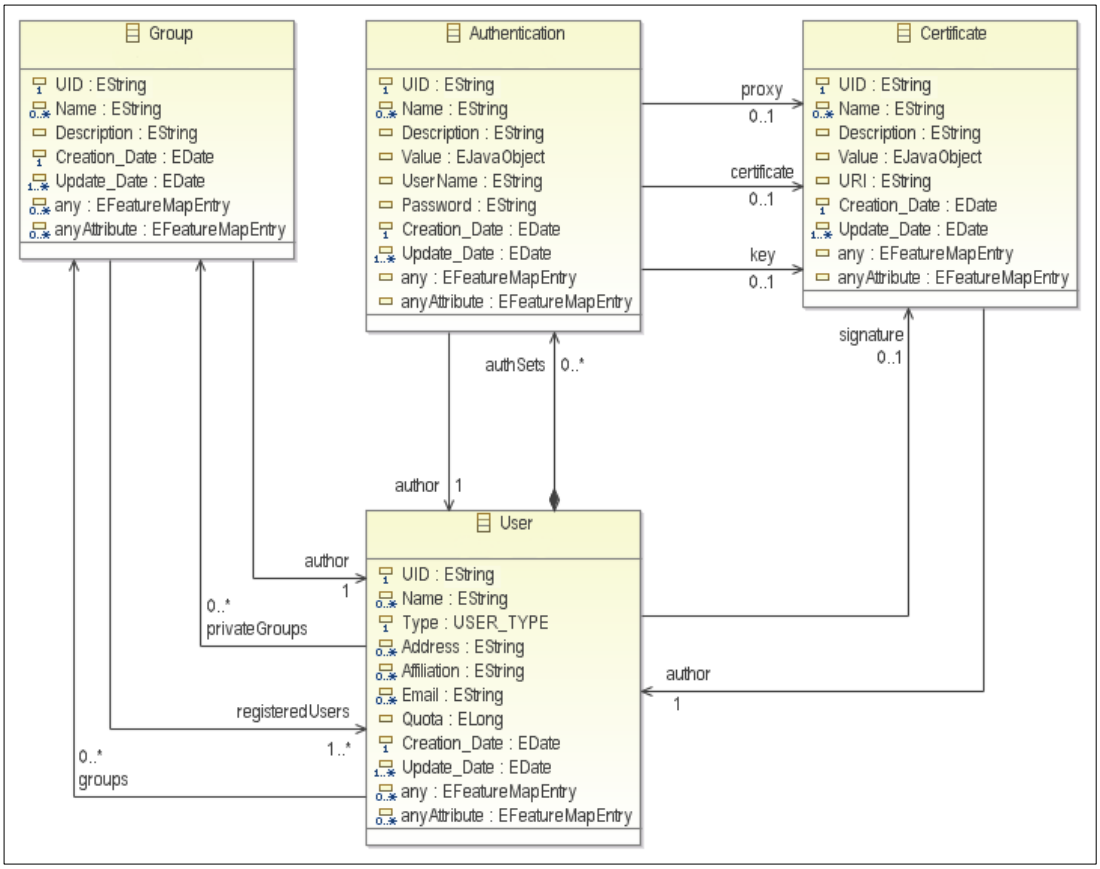


Figure 3-14: The *User* entity and its relations to the *Group*, *Authentication,* and *Certificate* entities

As shown in Figure 3-14, the user object is associated with different types of authentication. Grids normally use GSI (Grid Security Infrastructure) for authentication – this is a type of Public Key Infrastructure implementation based on X509 certificates. GSI is captured in the GAMRS model on two objects: *Authentication* and *Certificate*. The model permits users to own multiple authentication sets and therefore allows them to access various resources located in different Grid infrastructures. Furthermore, the authentication object also permits

the specification of *(username, password)* tuples that can be used with resources that accept such type of authentication.

The *Certificate* entity is used to model all three types of X509 certificates: *certificate, private key* and *delegated proxy*. This entity contains an attribute called *URI (Uniform Resource Identifier)*, which is used to indicate that the certificate is stored on a remote resource (i.e. usually a *myproxy* server). However, the GAMRS repository can be used to store the certificate as well. Therefore, the *Certificate* entity also contains the attribute *Value,* which points to the actual digital file stored in the repository. The GAMRS model also allows each user to choose one certificate and use it as digital signature.

THE GROUP ENTITY

As mentioned before, the GAMRS model allows users to be organized in groups. The *Group* object ensures a better management of users and allows for a simpler and easily understandable design and association of policies. For example, a policy may specify that provider objects can be read, but are not open to modifications by users with non-administrative rights. Instead of creating all the associations between this policy and each one of the users with non-administrative rights, these users can be organized in one group and only one association is needed between this group and the policy.

THE POLICY ENTITY

The GAMRS model uses the *Policy* entity to model interactions between various objects in the repository. The GAMRS model defines four operations that can be applied to any object in the model, with the exception of the *MetaRepository* object, the default user and the default policy: create, read, modify and delete. These operations are defined within the structure of an *enumeration* entity, which also contains the literal *other* that makes the model extendible to permit further operations such as *deploy*, *execute*, etc.

The *Policy* entity acts as a container for *PolicyRule* objects. Each *PolicyRule* represents a rule, which, conceptually, follows the following pattern: "the *Actor* is allowed to perform the following *operation(s)* upon the *Object.*"

In the GAMRS model the Actor is represented by a *User* object or by a *Group* object. The operations are represented by the four actions mentioned above (*create, read, modify and delete*) and the Object can be represented by any of the following entities: *User, Group, Provider, Application, Asset, ApplicationRelation* and *Policy*.

Each of these entities contains an association with the *PolicyRule* entity called *ruleObject (i..e ruleObjectUser, ruleObjectGroup, ruleObjectProvider, ruleObjectApplication, etc)*. However, in the case of the *User* and *Group* entities, these two can function both as *Actor* and as *Object* of the policy rule. Therefore, the *User* entity and the *Group* entity each contain two associations to the *PolicyRule* entity – *ruleActorUser* and *ruleObjectUser; ruleActorGroup* and *ruleObjectGroup*.

Figure 3-15 shows the two entities *Policy* and *PolicyRule* along with the enumeration entity *Permissions* used to model the four operations defined in GAMRS.

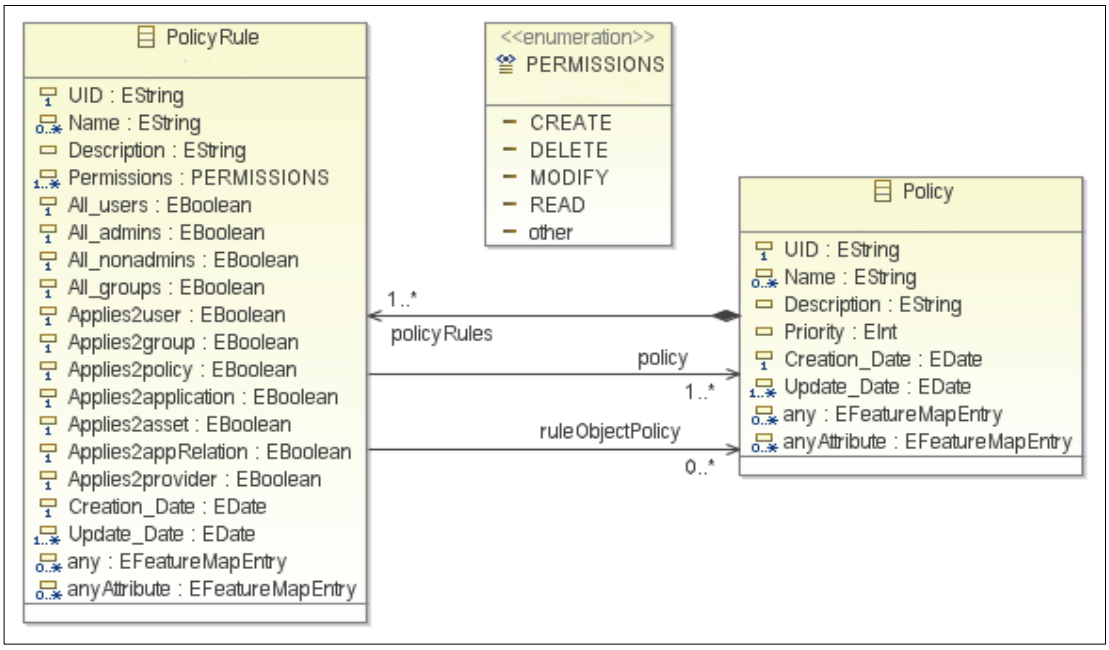


Figure 3-15: The *Policy* entity and its associations to the *PolicyRule* entity

The *PolicyRule* entity permits the specification of Actors in the following ways:

- *all users*: through the Boolean attribute *All_users*;
- *all groups:* through the Boolean attribute *All_groups*;
- *all users with administrative rights*: through the Boolean attribute *All_admins*;
- *all users with non-*administrative rights: through the Boolean attribute *All_nonadmins*;
- *individual users:* through *ruleActor* relations to *User* objects;
- *individual groups:* through *ruleActor* relations to *Group* objects;
- *combined:* by making combinations between c, d, e and f.

The *PolicyRule* entity permits the specification of operations through the attribute *Permissions* and the specification of Objects through a set of *applies-to* Boolean attributes (*Applies2user, Applies2group, Applies2provider, Applies2application, Applies2asset, Applies2appRelation, Applies2policy*) and corresponding *ruleObject* relations to the entities *User, Group, Provider, Application, Asset, ApplicationRelation* and *Policy.*

If one of the *applies-to* attributes is marked *true* and no *ruleObject* relations are specified to corresponding objects, then the rule will apply to all instances of that entity. Moreover, the use of an *applies-to* attribute does not imply the automatic exclusion of the other *applies-to* attributes. Like in the case of Actors, where the specification of different combinations of Actors is possible, the GAMRS *PolicyRule* object permits combinations between different Objects.

The example below shows a possible GAMRS rule, which can specify both complex sets of Actors and complex sets of Objects:

Example

Rule: “The users Alex, Steve and Gabor, plus the group CPC, are allowed to add, retrieve and modify applications, application-related objects and providers to GAMRS.”

GAMRS Rule:
“The Actors: Alex (entity *User*), Steve (entity *User*), Gabor (entity *User*) and CPC (entity *Group*)
Operations: *Create, Read, Modify*
Objects: *Provider* (all instances), *Application* (all instances) and *ApplicationAsset* (all instances).”

GAMRS model XML excerpt:

```
<MetaRepository, UID="https://161.74.69.171:4561/1", Description=
"University of Westminster GAMRS" (...)>
<Name>GAMRS ONE</Name>
<users UID="https://161.74.69.171:4561/1001" (...)>
        <Name>Alex</Name> (...)  </users>
<users UID="https://161.74.69.171:4561/1002" (...)>
        <Name>Steve</Name> (...) </users>
<users UID="https://161.74.69.171:4561/1003" (...)>
        <Name>Gabor</Name> (...) </users>
(...)
<groups UID="https://161.74.69.171:4561/101" Description="CPC Group",
registeredUsers="//@users.3 //@users.4 //@users.5" (...)>
        <Name>CPC</Name></groups>
 (...)
  <policies UID=https://161.74.69.171:4561/10000, Description= "CPC policy"
(...)>
        <Name>Example</Name>
        <policyRules UID=https://161.74.69.171:4561/10001,
 Applies2application="true", Applies2asset="true", Applies2provider="true",
 ruleActorUser=//@users.0 //@users.1 //@users.2
ruleActorGroup="//@groups.0"          (...)>
                <Name>First CPC rule</Name>
                <Permission>CREATE</Permission>
                <Permission>READ</Permission>
                <Permission>MODIFY</Permission>
        </policyRules> (...)
 </policies> (...)
</MetaRepository>
```

The *Policy* entity also contains the attribute *Priority,* which is used internally by GAMRS to prioritize different policies that can be applied to an object. Users can define their own private policies to help regulate access to their own objects. However, these policies will always be superseded by any administrator-defined policy in force for that particular entity. For example, for security reasons an administrator-defined policy may permit a particular group of users to *read*, *modify* and *delete* any application asset in the repository (e.g. to remove virus-infected files). A user may create a private policy and try to restrict these permissions to *read* for this particular group in relation with his/her application assets. However, this private policy will never be enforced as the rules in the administrator-defined policy take precedence over the rules defined by common users.

THE APPLICATION ENTITY

The *Application* entity is used to capture information about the Grid application stored in GAMRS or in one of the application repositories connected to the system.

The *Application* entity contains the five general attributes: UID, name, description, creation date, update date, and three specific attributes: *template, version* and *URI*.

- The *template* attribute is used to provide users with preset values for certain application parameters – such as machine architecture, OS, minimum amount of memory, or environmental variables, which are necessary for a correct run of the application.
- The *version* attribute helps distinguish between different versions of an application as the repository can accommodate several versions of the same application.
- The *URI* refers to the webpage of the company that developed the application, which usually contains the most comprehensive description and documentation about the application.

The *Application* entity contains an association with the *Provider* entity, which describes the repository from where the application originates. Figure 3-16 shows this relation, along with the containment association betweenthe *Application* entity

and its related objects (i.e. *ApplicationAssets*) and other relations between the following entities: *Application, Provider, Asset, AppplicationAsset* and *Hash*.

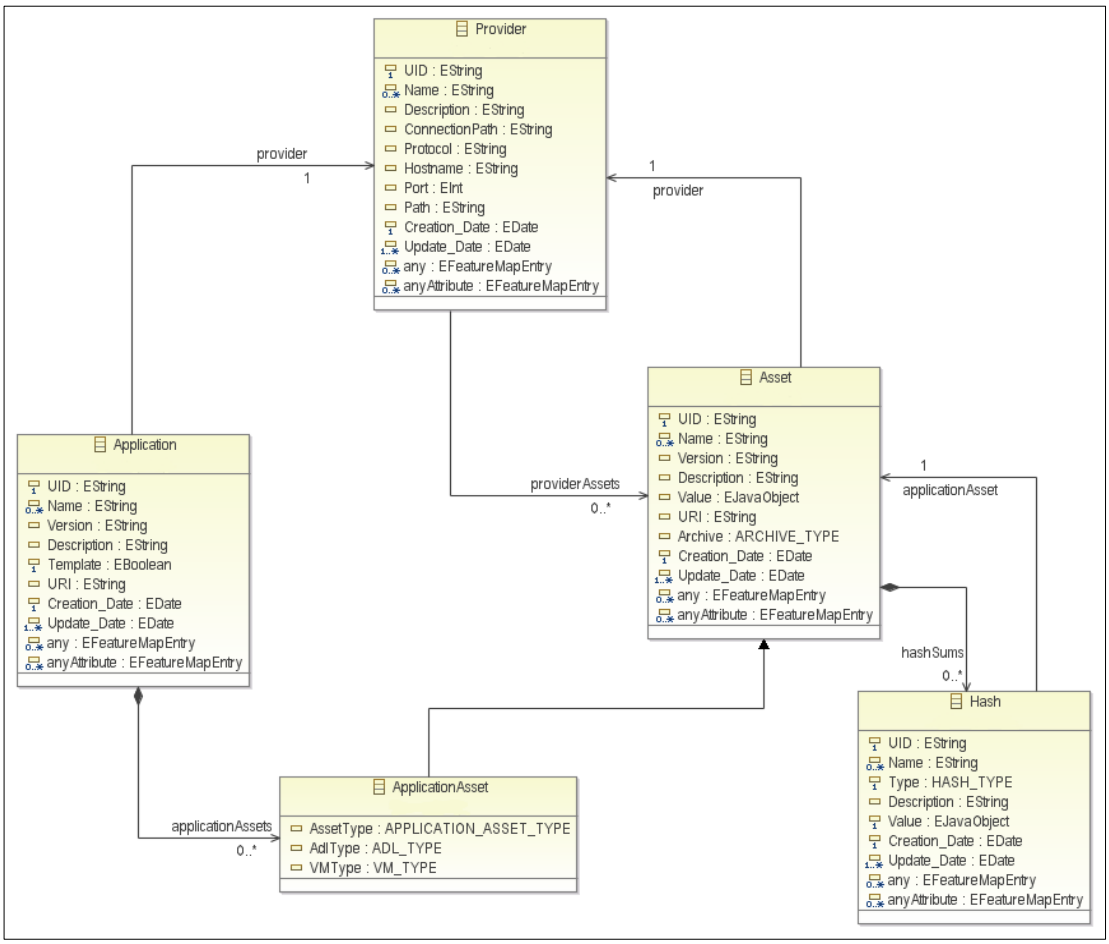


Figure 3-16: The *Application* entity together with the *Provider*, *Asset*, *Hash* and *ApplicationAsset* entities

THE ASSET ENTITY

The *Asset* entity is used in the GAMRS model to describe application-related objects and provider-related objects. Beside the general attributes, an *Asset* object contains the following extra attributes: *Version, Value, URI* and *Archive*. The *Version* and *URI* attributes hold the same meaning as in the case of the *Application* object.

The *Archive* attribute refers to the possibility of an asset object being stored in a compressed form. Since application assets can be quite large in size (for example, a virtual machine can be several GB in size), it is best that such objects are stored

in an archived form in order to save storage space. The *Archive* attribute is of type *ARCHIVE_TYPE* (see Figure 3-17). This type contains compression types most widely used in computer science (such as *zip, rar, jar, arj, tgz*, etc.) and the literal *other,* which allows for the specification of new methods.

The *Asset* entity also contains the attribute *Value,* which is used to model the actual digital file stored in the repository backend (on the filesystem).

The *ApplicationAsset* entity is derived from the entity *Asset* and, in addition to the inherited attributes inherited from *Asset*, contains the following attributes: *AdlType, VMType* and *AssetType.* The *AssetType* attribute is used to differentiate between different types of application-related objects. *AssetType* is of type *APPLICATION_ASSET_TYPE* (see Figure 3-17) and can be one of the following: *default_input*, *document_description, executable, library, license, software_dependency, source_code, test_suite* and *virtual_machine*. The literal *other* was also added to support future extensions, i.e. the specification of new types of application-related objects.

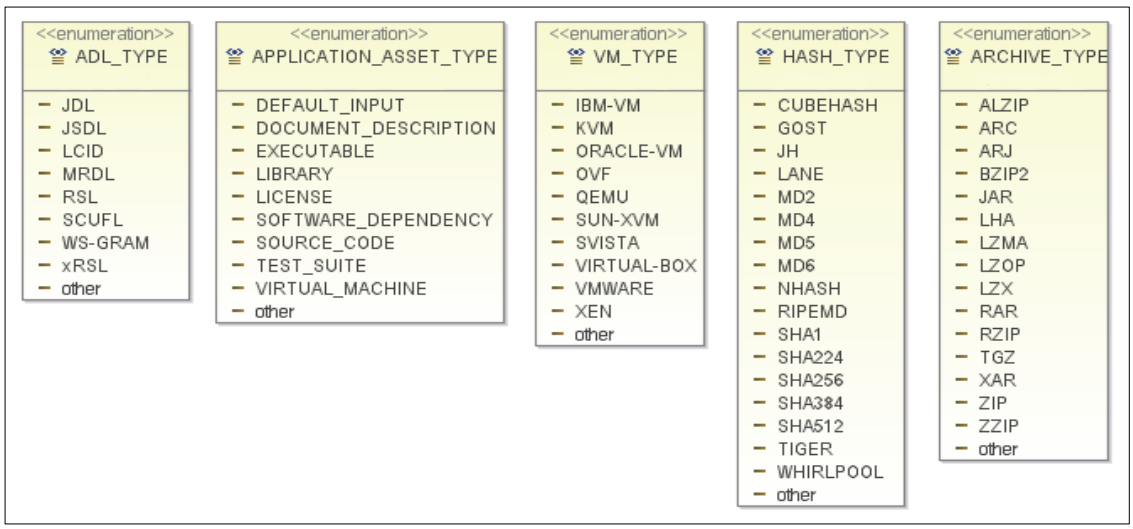


Figure 3-17: Types used in the GAMRS repository model

In cases when the *ApplicationAsset* object refers to the formal description document of the application, *AdlType* is used to designate the application description language in which the document is written. The *AdlType* attribute is of type *ADL_TYPE* (see Figure 3-17), which can be one of the language descriptions

that are extensively used in current Grid infrastructures: JDL, JSDL, LCID, RSL, SCUFL, WS_GRAM and xRSL. *ADL_TYPE* also contains the literal *MRDL (Meta-Repository Description Language),* which designates the application language proposed in this research in Chapter 3, section 3.3. Similar to other types, the ADL_TYPE entity contains the literal *other* to ensure that additional application description languages can be added to the model in the future.

In cases when the *ApplicationAsset* object refers to a virtual machine, the *VMType* attribute is used to identify the type of the virtual machine and, implicitly, of the virtual hypervisors able to run such a virtual machine. The *VM_TYPE* (see Figure 3-17) specifies the virtual machine types that are used in current virtual environments (IBM-VM, KVM, ORACLE-VM, OVF, QEMU, SUN-XVM, SVISTA, VIRTUAL-BOX, VMWARE and XEN). However, because of the rapid evolution of virtualized infrastructures and the ongoing research efforts invested in this area, new virtual machine solutions emerge almost every day. Therefore, *VM_TYPE* contains the literal *other,* which allows for the addition of new virtual machine types in the future.

THE HASH ENTITY

In GAMRS one or more hash sums can be associated to any asset. The *Hash* entity is used to model the value of a hash algorithm when applied to an application asset. For example, an application binary can be stored in the repository as an application asset. One can use a hash function to calculate the hash sum for this binary and then store it as a hash object in the repository as well. The *Hash* entity contains two specific attributes: *Type* and *Value*.

The *Type* attribute is used to identify the hash algorithm used to generate the hash sum. This attribute is of type *HASH_TYPE* (see Figure 3-17), which is used to specify the most widely used hash algorithms – cubehash, gost, jh, lane, md2, md4, md5, md6, nhash, ripemd, sha1, sha224, sha256, sha384, sha512, tiger and whirlpool. As in the case of the other types defined in the GAMRS model,

HASH_TYPE contains the literal *other,* which ensures that future hash types can be added to the model.

The *Value* attribute represents the hash value of the digital object stored in the repository.

THE APPLICATIONRELATION ENTITY

In Chapter 2, section 2.5, requirement **R2.3** specified that a Grid application repository system should allow connectivity with other repositories. As a result of such connectivity, similar applications could be found throughout a network of inter-connected Grid application repositories. To that end, the repository model needs to be able to capture the aspect that an application is identical or similar to a certain extent with another application stored elsewhere on such a network. In order to do that, the GAMRS repository model employs two entities: *ApplicationRelation* and *RelationPair*. Figure 3-18 shows these two entities and the associations between them, along with the *Application* entity and the relationships to it.

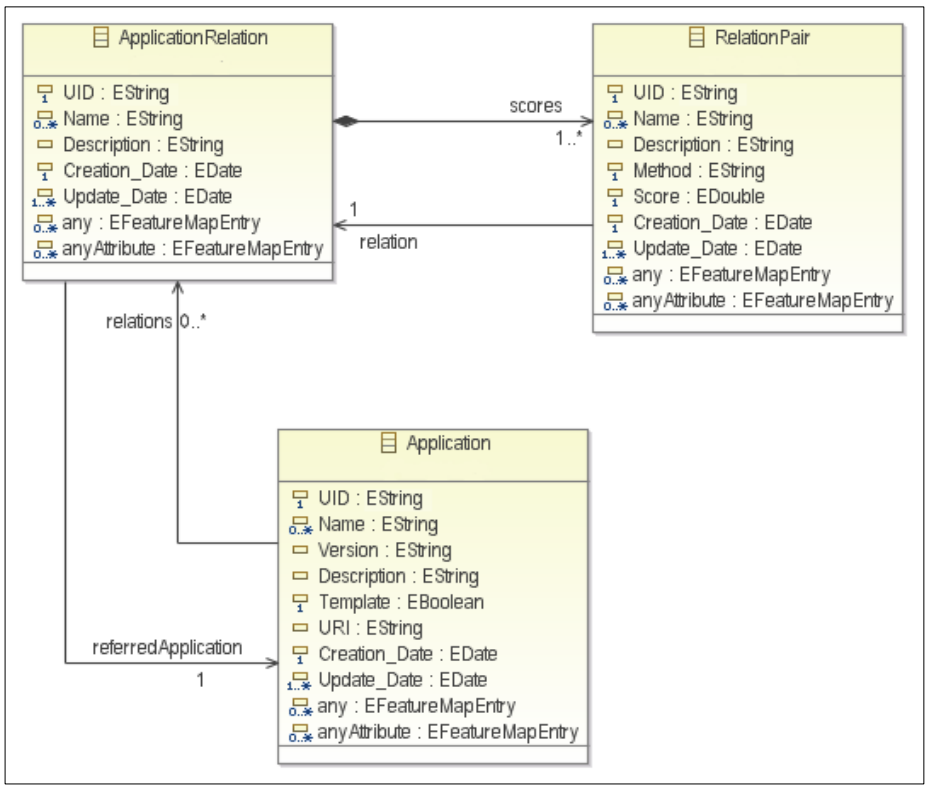


Figure 3-18: The *Application* entity with the *ApplicationRelation* and *RelationPair* entities

The *ApplicationRelation* is used to model the relation of correspondence between two applications that are found to be similar. Each *Application* entity object may contain references to one or more *ApplicationRelation* entities. In return, the *ApplicationRelation* entity holds an association called *referredApplication* (of multiplicity one) to that *Application* entity object, which is considered similar to the first application. Each such *ApplicationRelation* object contains a reference to one or more *RelationPair* objects that record the similarity method used and the similarity score obtained.

Example
If applications A and B were compared using the TFIDF/Cosine string-distance method and the result showed that they were similar, the following objects would be created:
- First, a *RelationPair* object would record the *Name* of the method and the *Score*.

- Second, an *ApplicationRelation* object would be created and a relationship added to the *RelationPair* object.

- Finally, two more relationships would be added, linking the *ApplicationRelation* object to the *Application* objects A and B.

THE PROVIDER ENTITY

In GAMRS the *Provider* entity is used to model the place of origin of an application, an application-asset, or a certificate. In the case of applications, the provider represents a repository where the application is stored; in the case of application assets, the provider can be represented by any type of storage-servers; and in the case of certificates, the provider is usually represented by proxy servers (e.g. myproxy [67], voms [141]). The *Provider* entity may also be used to model pools of virtual machines on virtual servers used in clouds or other types of virtualized infrastructures. Finally, as GAMRS can be used as a repository in its own right, the *Provider* may be used to designate a GAMRS system as a place of origin for some applications. For a seamless integration in a PKI security environment, such as the

one used in Grid, the *Provider* can store its digital certificate and use it as signature for identification within the security infrastructure. The *Provider* entity has an association with the *Asset* entity, which is used to model provider-related objects that can help ensure proper connectivity and access to a provider – for example, a software client.

The full GAMRS repository model containing all the types, entities and the associations between these entities can be found in Appendix A.

3.2.3 Summary

Table 3-2 and Table 3-3 below summarize the critical analysis of the five repository models described in Chapter 2, section 2.2.2 (MyExperiment, NGS, GEMLCA, GUSE and CHARON/iSoftrepo) assessing their ability to describe the entities specified at the beginning of this section (user, user policies and user related objects; application, application policies and application-related objects; provider, provider policies and provider-related objects) by comparison to the GAMRS solution.

Table 3-2: Traditional Grid application repository models vs. proposed GAMRS model (except application-related objects)

	myExperiment	NGS AR	GEMLCA	GUSE	CHARON/ iSoftrepo	GAMRS
User	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
User-related objects	no	no	no	no	no	**YES**
User access policies	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
Application	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
Application access policies	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
Provider	no	no	no	no	no	**YES**
Provider-related objects	no	no	no	no	no	**YES**
Provider access policies	no	no	no	no	no	**YES**

Table 3-3: Traditional Grid repository models vs. proposed GAMRS repository model (application-related objects)

	myExperiment	NGS AR	GEMLCA	GUSE	CHARON/ jSoftrepo	GAMRS
Description document	**YES**	**YES**	**YES**	**YES**	no	**YES**
Binaries	n/a	reference	**YES**	no	reference	**YES**
Source code	n/a	no	no	no	reference	**YES**
Library dependencies	possible (generic tag)	no	no	no	no	**YES**
Software dependencies	possible (generic tag)	no	no	no	no	**YES**
Documentation	reference	no	no	no	reference	**YES**
Test files	possible (generic tag)	no	no	no	no	**YES**
VM embedded	no	no	no	no	no	**YES**
Licenses	**YES**	no	no	no	reference	**YES**
Hash sums	no	no	no	no	no	**YES**

3.3. GAMRS Application Description Language (MRDL)

3.3.1 Overview

The third objective set out in this research was to find an application description language, which would provide uniformity in the presentation of Grid application descriptions. The application description language should allow for Grid application repositories and the applications stored by them to be used in scenarios other than Grid, such as virtualisation, source code staging and compilation or automatic application deployment. At the same time, further research related to current language capabilities revealed other shortcomings, which can lead to Grid interoperability problems, such as their reduced ability to describe the use of different X509 certificates for staging data from different Grid infrastructures and their limited capacity to differentiate between the X509 certificate used for job submission and X509 certificates used for data staging.

The conclusion of the critical analysis conducted in Section 2.5.3 was that, because JSDL passed most GAMRS requirements and its schema supported extensions natively, its further extension with the missing parts would be a better solution than creating a completely new language.

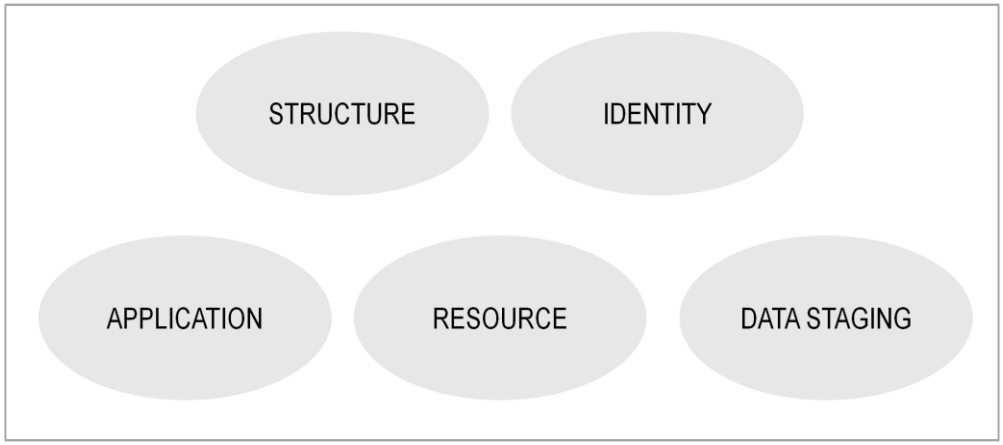


Figure 3-19: JSDL's main entities

JSDL defines five core entities in its model*: Structure, Identity, Application, Resource* and *Data staging* (see Figure 3-19). The *Structure* entity is used only as a container for the other entities and represents the root element of the JSDL schema. The *Structure* entity contains the attribute *Description,* which is used to give a free-text description of the application.

The *Identity* entity contains information about the project in which the application is used, along with identification fields that are used internally by JSDL processing/submission systems for management purposes.

The *Resource* entity is used to describe the computational resources required by the application in order to insure a correct run. Initially, this entity only permitted the specification of basic requirements (as described in Section 2.3.1), such as: machine architecture, operating system, CPU requirements, memory requirements and disk requirements. However, the original JSDL schema [22] was immediately followed by an extension [69], which also permits the specification of advanced resource requirements, such as: core dump size limits, virtual memory requirements, pipe size limits, minimum network bandwidth requirements, open file descriptors limit. In time, the JSDL schema was also extended to describe parameter sweep applications and multi-process applications.

The *Data staging* entity describes the data requirements of the application, namely which files are used as input (stage in) and which files are produced as output (stage out). It specifies the files that should be moved to the execution host before the application enters running stage and the files that should be moved from the execution host after the run. The JSDL schema employs no restrictions on the transfer protocol description used for data staging; therefore, any transfer protocol can be used in the application description document (e.g. *http, ftp, gridftp, rfiod, srb*, etc.). However, the JSDL processing/submission system should contain implementations of transfer clients able to understand such protocols and, subsequently, to perform the staging of required data from storage servers to the execution resource and the other way around.

The *Application* entity contains the attributes necessary to describe the name, version, executable, executable arguments, and environment variables. It also contains associations to the *Data stage* entity, which identify the input and output files used or created during the application execution.

In correlation with the application description language requirements and the life-cycle outlined in Section 2.5.3 , the JSDL model was extended during this research to provide the following capabilities:

- **Multi-Grid/multi certificate secure data access:**
  - *Distinction between the application instance submission certificate and the staging certificate:* Current application description languages cannot describe a situation where an X509 certificate other than the one used for application instance submission needs to be used for data staging. WS-GRAM [66] allows this kind of specification but only in the rare case where the certificate is stored into a MyProxy Grid service [67] and the submission engine knows how to make an OGSI/WSRF Service invocation.
  - *Multi-Grid data staging with different X509 certificates:* Existing description languages are not able to specify different X509 certificates for data staging from different Grids. This comes as a direct derivative of the point above, i.e. the impossibility of current languages to describe the following scenario: a user has two X509 certificates (A and B) and s/he uses two storage servers to keep their files (X and Y); one certificate (i.e. A) is used to authenticate on server X, and the other one (i.e. B) is used as a proof of authentication for server Y; if s/he needs to run an application that requires data to be staged from both servers, this would simply not be possible, because there would be no way to define the location of his/her X509 certificates within the application description document (except for the particular case where the ADL used is WS-GRAM and the certificates are stored on a *myproxy* server).
- **Data protection:**
  - *Hash sums for staged data*: Current application description languages are

not able to specify any hash sums to files, whether this is an application binary or a file that needs to be staged.

- **Template:**
  - *Advanced parameter/attribute descriptions*: Existing description languages have the capacity to describe the functionality of each attribute or parameter of an application and allow users to define values for them. However, in many cases a much more fine-grained usage of parameters and attributes is required. For example, many applications have a set of mandatory parameters that need to be specified at every run and this scenario cannot be described with the help of existing languages (*mandatory*). Moreover, the use of a certain parameter may require another parameter that needs to be specified as well (*requires*). In many cases, the person responsible for the administration of a certain application would want to fix the value of a parameter or attribute; a common user is not supposed to know that a certain application needs 2GB of memory to run, but the application administrator might know it and may want to *fix* that value, so that when a user creates an application instance s/he should not be able to overwrite that parameter and cause unwanted outcomes at runtime.
- **Application type of running:**
  - *Additional information such as location of licenses, libraries, code for compilation***:** One of the main shortcomings of existing description languages is their inability to describe an application not only as a software executable, but as a complex set of application-related objects associated with the formal description of the application. The GAMRS set of application assets contains, but is not restricted to, the following digital objects associated with a Grid application: licenses, source code, executables, test files, libraries/software dependencies, user documentation, and images of virtual machines running the application inside them.
  - *Virtual machine-embedded application***:** This requirement is based on the same inability of existing Grid application description languages to describe the application assets mentioned above. As virtualization gains terrain in

Grid, applications can be run in their native environment on a virtual machine, avoiding the problem raised by machine architecture-incompatibilities, Operating System-incompatibilities, dependency-failings, etc. Hence, the GAMRS description language should be able to describe the case when the application is embedded in a virtual machine and therefore the whole virtual machine needs to be staged, not only the application executable.

- *Application pre-run prerequisites* such as *requires compilation*, *license acceptance*, *staging dependencies* and *VM-embedded*: This requirement refers to the ability of a description language to model actions associated with application-related objects. This requirement needs to be met in order to address the following scenarios:
  - The application can be run by staging the source code and compiling it directly on the Grid resource selected for running the application.
  - The application requires license acceptance prior to submission and running on the Grid infrastructure.
  - The application requires additional dependency software to be staged on the Grid resource to ensure a correct run of the application.
  - The application will run as embedded in a virtual machine.

In order to meet these requirements, JSDL was extended into an application description language called the **Meta Repository application Description Language (MRDL),** which can be used within the GAMRS framework.

Figure 3-20 shows the JSDL extensions proposed in this research, highlighting the new entities added in MRDL, as well as the modifications made to existing (old) JSDL entities. The *Structure* and *Identity* entities remained the same as JSDL. The *Application, Resource* and *Data staging* entities were extended with the *Advanced parameter/attribute descriptions* additions, in order to permit MRDL to describe application *templates*. The *Data staging* entity was extended with data protection capabilities in terms of hash sums and also with the necessary attributes and associations to permit the description of secure data access from storage servers

located in different Grids with the use of different X509 certificates. The *Application type of running* was modelled in a single entity within the MRDL model, which also implements a concept borrowed from the LCID description language – the property of an application to run on different types of Grid *middleware*.

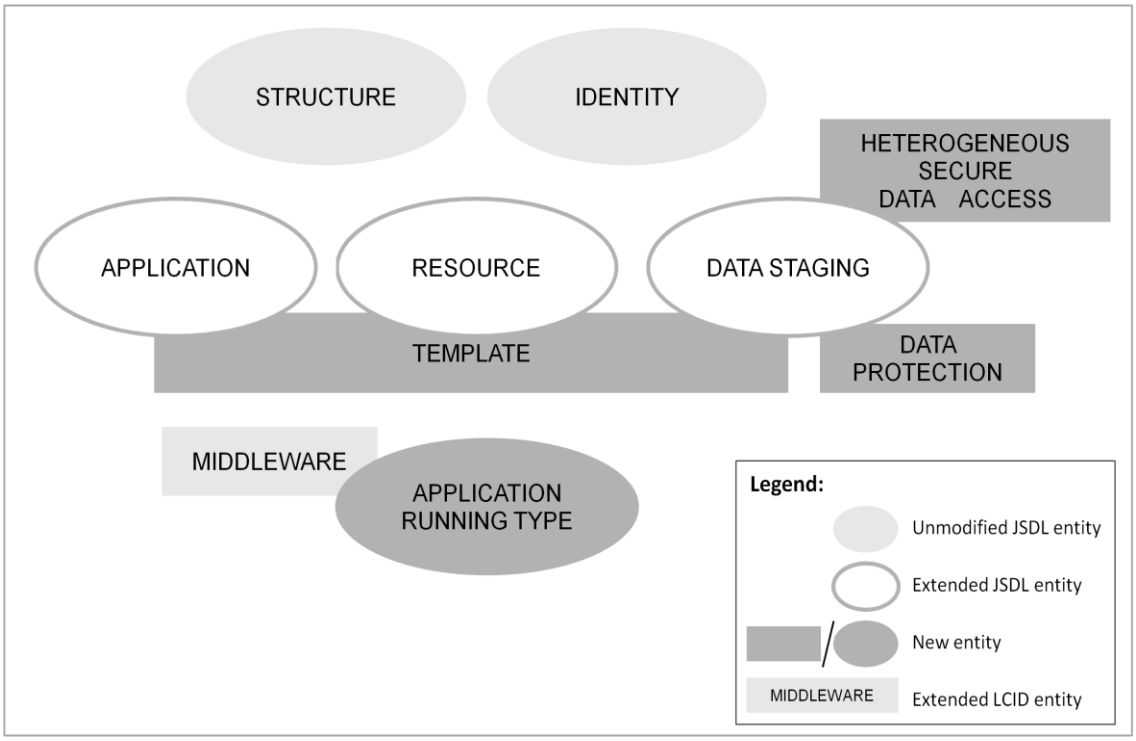


Figure 3-20: JSDL extensions (MRDL entities)

3.3.2 Design

In order to satisfy the requirements related to **Multi-Grid/multi certificate secure data access** (*multi-Grid data staging with different X509 certificates* and the *distinction between the application instance submission certificate and the staging certificate*), JSDL was extended to include two new entities: *Authentication* and *X509Credential*. The two entities in MRDL are similar to the entities *Authentication* and *Certificate* used in the GAMRS model. However, in the GAMRS model these types of objects were focused more on user actions (such as accessing other repositories for application retrieval), while in MRDL they are used to highlight the fact that an application requires different certificates for data staging and/or running. Moreover, in the GAMRS model, these entities were used for the actual storage of

the certificate in the repository (hence the presence of the attribute *Value*), while an application description language does not model such action. Thus, in MRDL, the *X509Credential* entity does not contain the attribute *Value*, only the attribute *URI*, which points to the location where the certificate is stored.

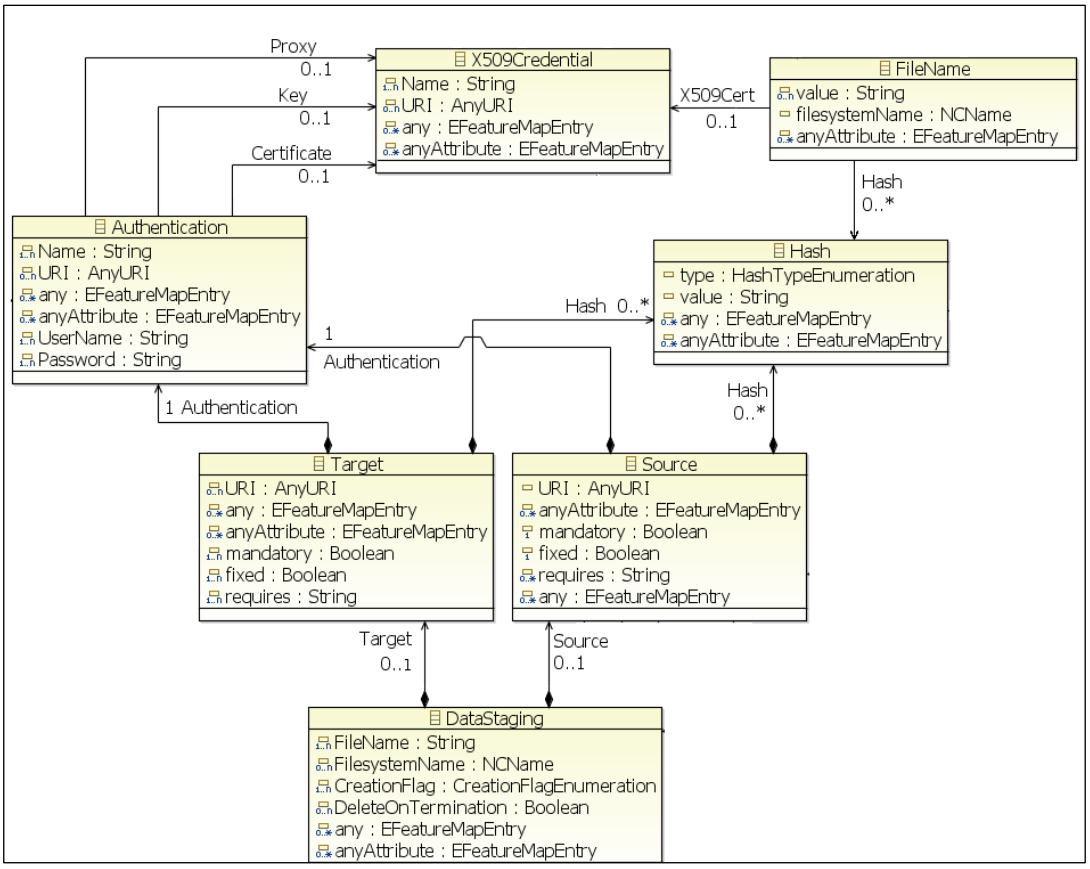


Figure 3-21: Partial view of the *MRDL* model highlighting the *Authentication, X509Credential* and *Hash* entities

With regard to the requirement regarding **Data protection** (*hash sums for staged data*), a new entity – *Hash* – was added to the original JSDL schema (see Figure 3-21). This entity is mainly used to prevent data corruption that may occur during file transfers between the storage server and the resource where the application runs. It also guards against accidental damage that may occur to files while they are stored on servers. In MRDL, *Hash* objects are associated to JSDL's *File* entities (used to model files residing on the filesystem of the Grid resource where the application is submitted to run) and to the *Source* and *Target* entities, which are used to model remote files residing in repositories or dedicated storage servers and are used for data staging.

The **Template** (*advanced parameter/attribute descriptions*) requirement was met through the addition of three attributes (*Mandatory, Fixed* and *Requires*) to the following JSDL entities: *Resources, Filesystem, Operating system, Candidate host, Cpuarchitecture, Target, Source* and *Argument*.

The attribute *Mandatory* indicates whether the presence of that object in the description of an application is compulsory or not. It relates to the fact that specific information about an application may be needed in order to ensure an accurate description and a correct run of such an application on Grid. In structural terms, in a document written in MRDL, the field(s) that describes an object containing the attribute *Mandatory* has to be provided with values.

The attribute *Fixed* denotes whether the value entered for a specific object is non-changeable. This attribute should be used by application administrators who know the specific requirements needed by a particular application to run correctly. This would prevent users from entering wrong values for that field and from causing an erroneous run of the application. Moreover, the attribute *Fixed* can be used for the specification of default input files – usually configuration files that are required for each and every run of an application.

The attribute *Requires* denotes that the presence of a value for a particular object requires the value of another object to be specified as well. For example, some applications which need remote connectivity to a service or server contain the argument *Host,* which points to the location of such server. In most cases, if a user specifies a value for *Host*, s/he is automatically required to introduce a value for a second parameter called *Port*. Therefore, the attribute *Requires* can be used to describe such particular scenarios, where the use of one object might require a follow-up object to be specified as well.

Grid applications can run on various Grid infrastructures and each of these infrastructures may be serviced by a different Grid submission system. Grid description languages should be able to capture in their schema information about such submission systems, since it presents users with a comprehensive picture of the Grid infrastructures where the application can run. Currently, GEMLCA's LCID

is the only application description language able to describe the functional property of a Grid application to run on GT2, GT4 and gLite/lcg Grid infrastructures.

The MRDL language proposed in this research uses this LCID concept to extend the JSDL schema with a new entity called *SubmitterData*. The *SubmitterData* entity along with its associations to other MRDL objects models the **application running type** additions: it can describe the *additional information about a Grid application* by including information such as location of licenses, libraries, code for compilation**;** can describe the staging of *virtual machine-embedded applications*; and can also specify *application pre-run prerequisites,* such as *requires compilation*, *license acceptance* or *staging dependencies*.

In addition to the capabilities of LCID, through these extensions, an application description written in MRDL is able to model the following scenarios that are not currently available in any other application description language used in Grid:

- An application can be presented as embedded in a virtual machine. Such applications can be easily deployed and run on distributed virtualized infrastructures either in Grid or in cloud computing environments.
- With the help of libraries and source code an application can be deployed, compiled and run on Grid resources without needing the application binary. Furthermore, even when the binary is available, this scenario may prove helpful in cases when Grid resources employ machine architectures and operating systems different from the ones that the application binary was compiled for.
- Applications that require license acceptance can be run on Grid infrastructures because MRDL allows the specification of license locations in its schema.

Figure 3-22 shows the *SubmitterData* entity along with its relationships to the following JSDL/MRDL entities: *JobDescription*, *Source* and *Argument*.

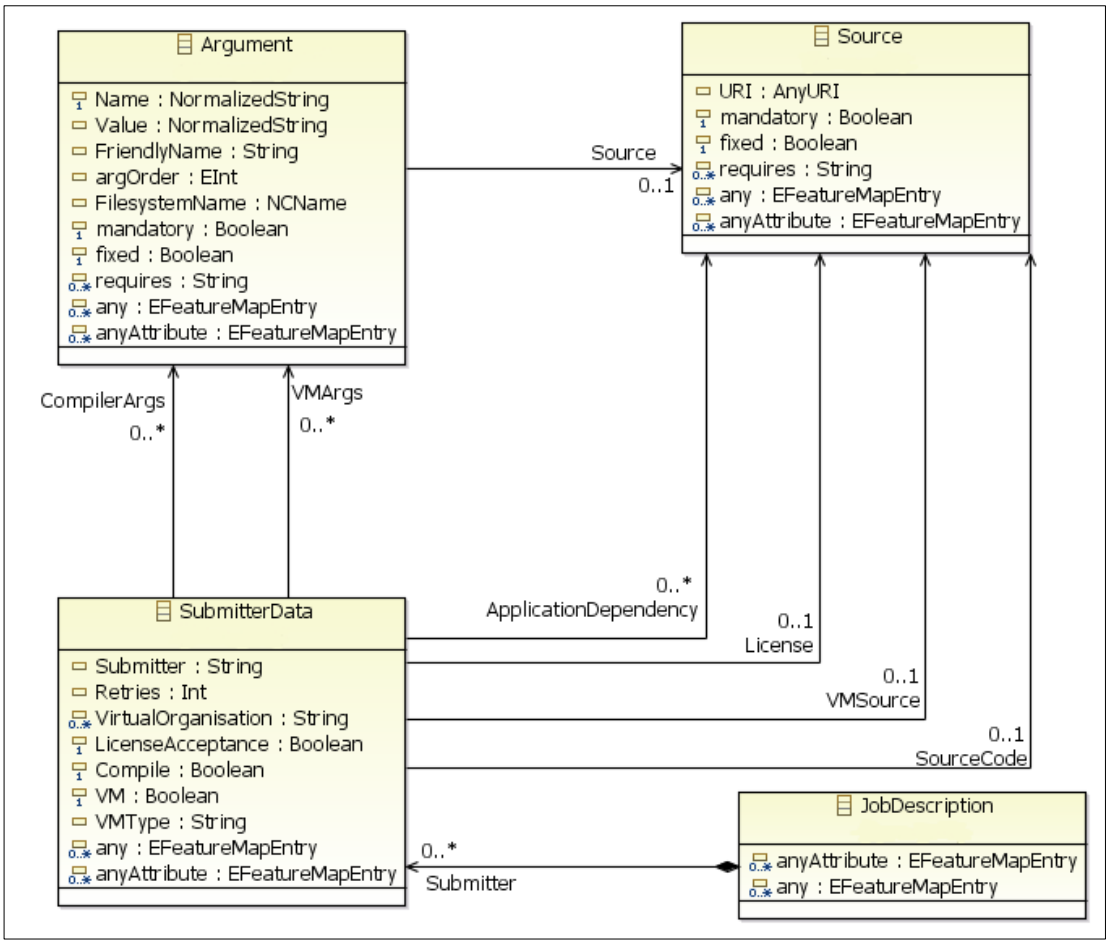


Figure 3-22: The *SubmitterData* entity and its associations to the *Argument*, *JobDescription* and *Source* entities

The *JobDescription* entity represents the parent element in the JSDL/MRDL schema to which the *SubmitterData* entity is related to.

The *SubmitterData* entity has four associations with the *Source* entity – *license, source code, VMsource* and *ApplicationDependecies*, which are used to describe the specific locations of the following application-related objects: the application license, the application source code, the virtual machine in which the application is embedded, and, respectively, the application software dependencies (e.g. libraries, compilers).

The two associations with the *Argument* entity – *CompilerArgs* and *VMArgs* – permit the specification of necessary arguments that are used for source code compilation or for running virtual machine-embedded applications.

In order to be able to describe the new scenarios mentioned above, the *SubmitterData* object contains the following attributes:

Specified in LCID:

- *Submitter* – this attribute specifies the type of submission system used;
- *Retries* – if a submission error occurs, this attribute specifies how many times a submission system should try to re-submit the application on the designated Grid resources before reporting it as failure;
- *VirtualOrganization* – specific to gLite/lcg Grids, this attribute specifies the group of Grid users allowed to run the application).

New attributes:

- *LicenceAcceptance* – this attribute is used to specify that a license acceptance is required before the actual submission and run of the application;
- *Compile* – this attribute is used to specify whether the application requires to be compiled before the actual run of the application;
- *VM* and *VMType* – the VM attribute is used to specify whether the application is embedded in a virtual machine; (the type of the virtual machine is specified by the attribute *VMType*)
- *any* and *anyAttribute* – these attributes allow for future extensions to be added to the *SubmittedData* entity;

The *SubmitterData* entity is linked to the other JSDL main entities through an association to the JSDL's *JobDescription* entity. The full MRDL application description language model can be found in Appendix B.

3.3.3 Summary

Table 3-4 summarizes the critical analysis of the application description languages discussed in Chapter 2, section 2.3.2 (RSL, JDL, xRSL, WS-GRAM, LCID and

JSDL), and assesses their ability to meet the application description language challenges identified in this research (legacy compatibility; advanced features; different submission certificate and staging certificate; multi-Grid data staging; hash sums; advanced parameter/attribute description; multiple transfer protocols supported as URI definitions; additional information – licenses, libraries, code for compilation; application pre-run prerequisites; virtual machine staging; advanced parallel behaviour; and native extension) by comparison to MRDL.

Table 3-4: Traditional application description language capabilities vs. MRDL

	RSL	JDL	xRSL	WS-GRAM	LCID	JSDL	MRDL
Legacy compatibility	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES/ inherited**
Advanced features	partly	partly	partly	partly	partly	**YES**	**YES/ Inherited**
Different submission certificate and staging certificate	no	no	no	**YES (service only)**	no	no	**YES**
Multi-Grid data staging	no	no	no	**YES (service only)**	no	no	**YES**
Hash sums	no	no	no	no	no	no	**YES**
Advanced parameter/attribute descriptions	no	no	no	no	partly	no	**YES**
Multiple transfer protocols supported as URI definitions	no	no	**YES**	no	no	**YES**	**YES/ inherited**
Additional information (licenses, libraries, code for compilation)	no	no	no	no	no	no	**YES**
Application pre-run prerequisites	no	no	no	no	no	no	**YES**
Virtual machine staging	no	no	no	no	no	no	**YES**
Advanced parallel behaviour	partly	partly	partly	partly	partly	**YES**	**YES/ inherited**
Native extension	no	no	no	no	no	**YES**	**YES/ inherited**

3.4. GAMRS Matchmaking Service

3.4.1 Overview

The fourth objective set out in this research was to design a matchmaking methodology and an algorithm able to identify similar or identical applications stored in Grid repositories connected to GAMRS. The aim was to identify or create matchmaking techniques that could: compare two applications stored or referenced by GAMRS; process their application-related objects found in the repository; and decide whether two applications are similar or not. The time contraints of a PhD research permitted the implementation and performance analysis of only a subset of the methods identified. Nevertheless, the architecture proposed for the matchmaking system and the algorithm used here are extendable, and future research can implement and analyze the performance of other matchmaking methods when applied to the objects stored in GAMRS repository.

Following the critical analysis described in Section 2.5.4, I started with the design and testing of the most widely used approach to matchmaking: *the syntactic algorithm*. The syntactic algorithm was designed to process MRDL description documents and to extract the application information contained in them. Moreover, the syntactic approach also takes into consideration the new additions included by MRDL, such as *data protection* (i.e. hash sums) and *template* (i.e. advanced parameter description), in order to provide a more accurate answer to the Grid application matching problem.

Due to the rigid logic of the syntactical approach and to the scarceness of information present in real-case application description documents, this method may often return inconclusive results. Moreover, in gLite/lcg-based Grids the formal description document of the application is usually missing from the repository. Hence, a scenario involving application information gathered from gLite/lcg-based Grid application repositories cannot use the syntactic matchmaking technique as there would be no formal description document to process.

However, all these applications come from a repository and they contain a metadata field called *Description,* in which information about the application is stored as a paragraph of free text. Consequently, since the syntactic method was not a suitable solution in such cases, I moved on to investigating how *string-distance* methods could help with the identification of similar applications based on the information contained in the free-text description of the application. Based on reviews and successful case studies published in the specialised literature I selected eleven of the most widely-used string-distance techniques and applied them to Grid application descriptions. At the same time, this research proposed a new technique of improving the accuracy of string-distance metrics by using *entropy-generated stop-lists*. Following this investigation, I identified at least two string-distance methods that could be used to identify similar applications stored in Grid repositories and also showed that the new entropy-generated stop-list method proposed by this research can increase the performance of the string-distance methods. Furthermore, the entropy-generated stop list technique is generic and can be applied to other scenarios involving the usage of string-distance metrics, besides the identification of similar Grid applications.

However, the string-distance approach has its limitations as well, especially because it bases its matching mechanisms on paragraphs of free-texts. Such paragraphs are always affected by the subjectivism of the author, as there are no formal constraints on what needs to be written in a *Description* field.

Therefore, in an attempt to provide a more objective matchmaking result, I considered a different approach, which does not rely on the information found in the application description document or the metadata *Description* field, but on the ability of GAMRS to store application binaries and application test suites.

The GAMRS repository permits the storage of application binaries and test suites. A test suite in this case is comprised of a complete set of input files (i.e. the *testIN* set); the complete set of output files corresponding to an application run with the input contained in *testIN* (i.e. the *testOUT* set); and a *script* providing the values for environment variables and the suite of commands necessary to run the application

with the input files contained in testIN (i.e. the *run* script). The method proposed (i.e. *application-running*) compares two applications by running two application binaries with a common set of input files (retrieved from one of the application test suites) and compares the output set.

Another method of identifying identical Grid applications based on application-related objects other than the description document is the *binary matching* method. This technique relies on the availability of application binaries stored in Grid repositories. The method implements a strong collision resistant hash algorithm and computes the hash sum for each application binary. It then compares these hash sums with each other and decides whether any two application binaries are the same (i.e. the applications are identical) or not. Whichever the result, the method also updates the application objects stored or referred in GAMRS with the newly computed hash sums.

This research also suggests other matchmaking methods that can be used to identify similar or identical Grid applications stored in Grid repositories. However, due to existing time constraints, these could not be implemented and tested within the timeframe of this PhD research.

3.4.2 Design

Based on the available sources of information about an application stored in GAMRS, matchmaking techniques can be grouped in three categories:

- techniques that extract and process application information contained in the formal application description document;
- techniques that can extract information by processing application-related objects stored in GAMRS other than the description document (e.g. binaries, source code, dependency software);
- hybrid techniques, which use combinations of methods from the first two categories.

Figure 3-23 shows the architecture of the GAMRS Matchmaking service, along with suggested matchmaking methods from the three categories listed above that can be used for finding similar applications stored in Grid repositories.

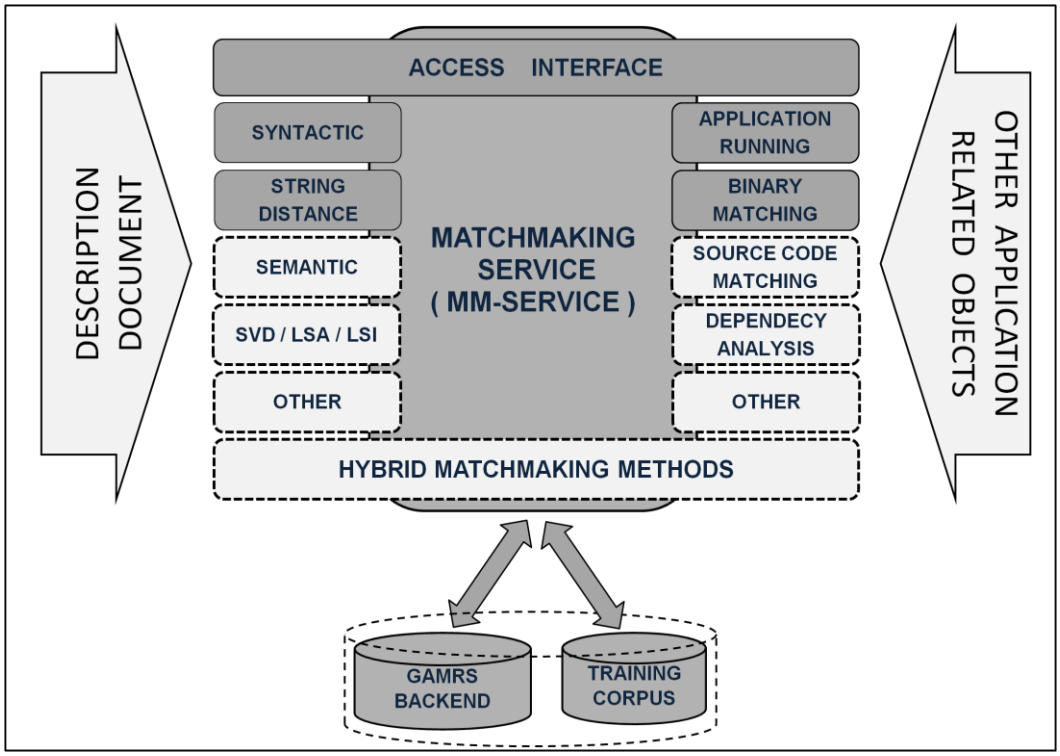


Figure 3-23: GAMRS Matchmaking service architecture

The architecture of the proposed Matchmaking service includes the following elements:

- **ACCESS INTERFACE**: This interface allows communication between the Matchmaking service and users.
- Modules that process application description documents:
  - **SYNTACTIC MODULE**: This module returns the degree of similarity between two applications by applying syntactical similarity functions to two application description documents written in MRDL.
  - **STRING-DISTANCE MODULE**: This module returns the degree of similarity between two applications by applying string-distance similarity functions to two application descriptions written in free-text (i.e. retrieved from the attribute *Description* associated with the applications in the GAMRS model).

This module can implement different types of string-distance method (i.e. edit-based, token-based and hybrid). The token-based functions that need *training* can use the **TRAINING CORPUS** for that. This module also contains the proposed method of *entropy-generated stop-list*, which helps to increase the accuracy of the string-distance methods.

- **SEMANTIC MODULE**: This module returns the degree of similarity between two applications by applying semantic similarity functions to two application description documents written in MRDL or to application descriptions written in free-text.
- **SVD/LSA/LSI MODULE**: This module returns the degree of similarity between two applications by applying latent semantic analysis methods to two application description documents written in MRDL or to application descriptions written in free-text.
- **TRAINING CORPUS**: The training corpus is needed for token-based string similarity functions, which use statistics and probabilities in their computation. It can come as a separate entity or, because the Matchmaking service is connected to the **GAMRS BACKEND**, a training corpus can be created by retrieving the application descriptions stored there.

- Modules that process application-related objects other than the application description document:
  - **APPLICATION-RUNNING MODULE**: The *application-running* module can find similar applications by executing jobs with test input files, retrieving the output files and checking for differences between output files.
  - **BINARY MATCHING MODULE**: This module can be used to optimize the accuracy of matching systems by comparing and analyzing two application binaries, provided these are stored in the repository.
  - **SOURCE CODE MODULE**: This module can be used to optimize the accuracy of matching systems by comparing and analyzing two application source codes, provided these are stored in the repository.
  - **DEPENDENCY ANALYSIS MODULE**: This module can be used to

optimize the accuracy of matching systems by comparing and analyzing two sets of application dependencies, provided these are stored in the repository.

- **HYBRID MATCHMAKING METHODS**: This module contains an aggregation model able to combine together the scores returned by different matching modules with the aim of increasing the accuracy of the matchmaking system.
- **OTHER**: The GAMRS Matchmaking service is extendible with other matchmaking modules, which could help with finding similar applications stored in Grid repositories.
- **GAMRS BACKEND**: The backend stores the GAMRS repository objects.

This investigation focused on the analysis of only four of the modules mentioned above: the **syntactic module** (because it is the most widely-used technique and the first choice of matchmaking method for documents formatted according to a formal structure); the **string-distance module** (because it is the first time this technique is used to identify similar applications stored in Grid repositories); the **application-running module** and the **binary matching module** (because the application binary is the most common type of application-related object likely to be stored in a repository – apart from the application description document). At the same time, a new method of **entropy-generated stop-list** was proposed in conjunction with the string-distance module in order to analyse to what extent the accuracy of string-distance methods could be improved through the use of such stop-lists.

The rest of the methods suggested and their applicability in Grid environments could make the subject of further research – this could test the performance and accuracy of such techniques when applied to Grid applications. Should the results of such tests prove successful, these methods could be easily implemented at a later stage as modules in the GAMRS Matchmaking service.

THE SYNTACTIC MODULE

The syntactic module tries to identify similar or identical applications by comparing their description documents, which are written in MRDL – the GAMRS application description language. MRDL was chosen as the best candidate for the syntactic module for two reasons: first, it provides a uniform description of applications, bridging the discrepancies between the other ADLs currently used in Grid repositories, while it does not lose information in the translation process; and second, for applications published directly in the GAMRS repository, its new extensions provide more sources of information about the application than traditional Grid ADLs.

The syntactic module uses a language parser to parse trough the structure of MRDL. The following attributes are processed by the syntactic function: *name; version; all resource attributes* (e.g. *architecture, OS, RAM, disk, virtual memory …); default input files (name and path); mandatory attributes (i.e. resources and application arguments); designated running sites; binary paths; application arguments;* and *hash sums for binaries, for default input files and for test files.*

At the same time, the syntactic module defines *fast-track* subsets of attributes, which can help identify identical applications quicker. In this case, if the comparison of each of the following subsets of attributes returns the Boolean *true*, then the Grid applications are the same: *{hashsums of binaries}* and *{designated running site, binary location, mandatory application arguments}.*

The following algorithm is proposed for the syntactic module:

SYNTACTIC APPLICATION MATCHMAKER (UID_1 , UID_2)
1: retrieve description documents $MRDL_1$ and $MRDL_2$ from GAMRS using UID_1 and UID_2
2: IF $MRDL_1$ or $MRDL_2$ is missing
3: RETURN decision: "*syntactic matching not possible in this case*"
4: END_IF
5: IF $MRDL_1$ contains the hash sum of the application binary
6: YES: $HashType_1 \leftarrow$ retrieve the type of hash sum from $MRDL_1$
7: $Hash_1 \leftarrow$ retrieve the value of hash sum from $MRDL_1$

8: END_IF
9: IF $MRDL_2$ contains the hash sum of the application binary
10: YES: $HashType_2$ ← retrieve the type of hash sum from $MRDL_2$
11: $Hash_2$ ← retrieve the value of hash sum from $MRDL_2$
12: END_IF
13: IF ($HashType_1$, $HashType_2$, $Hash_1$, $Hash_2$ *exist*) AND ($HashType_1$ EQUALS $HashType_2$) AND ($Hash_1$ EQUALS $Hash_2$)
14: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are identical"*
15: END_IF
16: DRS_1 ← retrieve designated running site from $MRDL_1$
17: DRS_2 ← retrieve designated running site from $MRDL_2$
18: BLP_1 ← retrieve binary location path from $MRDL_1$
19: BLP_2 ← retrieve binary location path from $MRDL_2$
20: AL_1 ← retrieve the list of mandatory arguments and fixed value arguments from $MRDL_1$
21: AL_2 ← retrieve the list of mandatory arguments and fixed value arguments from $MRDL_2$
22: IF (DRS_1 EQUALS DRS_2) AND (BLP_1 EQUALS BLP_2) AND (AL_1 EQUALS AL_2)
23: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are identical"*
24: END_IF
25: compare $MRDL_1$ and $MRDL_2$ over the following fields: application name, version, resource elements, list of application arguments and binary names
26: IF the comparison (step 25) returned the Boolean *true*
27: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are similar, but based on the information contained in their description documents the syntactic matchmaker cannot decide on their identicalness"*
28: END_IF

28a: IF the test suite for the application identified by UID_1 in GAMRS exists stored in the repository
28b: YES: retrieve the test suite from GAMRS using $UID_1 \rightarrow TestSuite_1$
28c: END_IF
28d: IF the test suite for the application identified by UID_2 in GAMRS exists stored in the repository
28e: YES: retrieve the test suite from GAMRS using $UID_2 \rightarrow TestSuite_2$
28f: END_IF
28g: D_1 ← retrieve the value of attribute *DeterministicType* from $MRDL_1$
28h: D_2 ← retrieve the value of attribute *DeterministicType* from $MRDL_2$
28i: IF ($TestSuite_1$ EQUALS $TestSuite_2$) AND (D_1 EQUALS D_2 EQUALS 'Boolean *true*')

28j: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are identical"*
28k: END_IF

The notations used in the algorithm are as follows:

- *UID_1, UID_2*: The two GAMRS *universal identifiers* (UIDs), which refer to the applications under comparison;
- *$MRDL_1$, $MRDL_2$*: The application description documents corresponding to the two applications identified in GAMRS by UID_1 and UID_2;
- *$Hash_1$, $Hash_2$*: Hash sums of binaries corresponding to the two applications identified in GAMRS by UID_1 and UID_2;
- *$HashType_1$, $HashType_2$*: The type of hash sums $Hash_1$, $Hash_2$;
- *DRS_1, DRS_2*: Designated running sites of the two applications identified in GAMRS by UID_1 and UID_2;
- *BLP_1, BLP_2*: Full path to the location of application binaries of the two applications identified in GAMRS by UID_1 and UID_2;
- *AL_1, AL_2*: List of executable arguments of the two applications identified in GAMRS by UID_1 and UID_2;
- *$TestSuite_1$, $TestSuite_2$*: Application test suites (each consisting of a set of input files, a set of output files and a running script) associated with the two applications identified in GAMRS by UID_1 and UID_2;
- *D_1, D_2*: *DeterministicType* – new attribute of *Boolean* type added by the MRDL to the original JSDL *POSIXApplication* entity, which has the value TRUE if the application exhibits a deterministic behavior (i.e. *"Given a particular input, it will always produce the same output, and the underlying machine will always pass through the same sequence of states"* [142]) and FALSE if the application behaves in a non-deterministic manner. D_1 and D_2 retrieve the values of this field from $MRDL_1$ and $MRDL_2$.

The algorithm starts by retrieving the application descriptions documents $MRDL_1$ and $MRDL_2$ from GAMRS with the help of universal identifiers UID_1 and UID_2 (step 1). If $MRDL_1$ or $MRDL_2$ is missing, then the syntactical technique cannot be applied (steps 2-4). The algorithm continues with the tests of fast-track subsets of attributes: first, it retrieves the hash sum types and the hash sums of binaries for the two applications under comparison (steps 5-12); second, it tests if the hash sums are of the same type and if they have the same value (step 13). If there is a match, then the binaries of the two applications are the same, meaning that the applications are identical (step 14). If the comparison returned Boolean *FALSE*, the algorithm continues with the retrieval of the designated running sites (steps 16-17), the binary location paths (steps 18-19), and the list of mandatory arguments and fixed value arguments (steps 20-21) from the description documents $MRDL_1$ and $MRDL_2$. If the designated running sites are the same, it means that the two applications are meant to run on the same Grid resource; if the binary paths are the same, it means that the applications point to the same location on a filesystem or a storage server. In combination, the designated running site and the binary location path could point to the same application that was probably exposed in two different Grid repositories connected to GAMRS. However, there is an additional test to be performed in order to decide whether the applications are truly the same: the list of mandatory and fixed value arguments. This test is required to pinpoint a specific application from groups of applications: in some cases, multiple Grid applications are exposed in batches as one single software package (i.e. BSoft, AMBER). The call to run one particular application from such packages includes the name of the package binary followed by the name of the application given as a fixed-value argument and followed by the list of mandatory arguments required by that particular application to run. Consequently, if the designated running sites, the binary location paths and the list of arguments (mandatory and fixed) match (step 22), then the two MRDL documents actually describe the same application (step 23).

If the two fast-track subsets were unsuccessful in finding a match, the algorithm continues with the comparison of the following MRDL fields: *application name,*

version, resource elements, list of application arguments and binary names (step 25). If the comparison returns a match (step 26), then the applications are similar; however, the algorithm cannot decide on their identicalness (step 27).

In this form (steps 1-27, without steps 28a-28k), the syntactic algorithm has been designed to process information from MRDL description documents only. However, steps 28a-28k provide an addition to the algorithm, by adding a new fast-track decision set.

One of the conclusions drawn after testing the *application-running* module (which will be discussed later in this section) was that the method cannot be applied to applications that exhibit non-deterministic behaviour. Moreover, none of the application description languages currently used in Grid contains such information about the deterministic nature of the application. Consequently, a new attribute, *DeterministicType,* was added in MRDL, and has the Boolean value TRUE if the application exhibits a deterministic behaviour. This addition helped enhance the syntactic algorithm with a new fast-track decision set: *{test Suite files (input, output, running script), 'DeterministicType =TRUE'}*. Namely, if two applications are deterministic and their test input files, test output files and running scripts are identical, then the applications are identical.

The algorithm provides this test in steps 28a-28k: first, it retrieves the test suites from GAMRS for the two applications under comparison (steps 28a-28f); next, it retrieves the value of the *DeterministicType* attribute from the description documents of the two applications (steps 28g-28h); finally, it performs the test (step 28i) and, if the test suites are identical and the applications are deterministic, it returns the decision that the *applications are identical* (step 28j).

If used only with steps 1-28, the algorithm can be identified as the *Syntactic module* in the GAMRS Matchmaking service architecture shown at the beginning of Section 3.4.2 in Figure 3-23. When the algorithm is used with the additional steps 28a-28k, it is categorized as a *hybrid matchmaking method* in the GAMRS Matchmaking service architecture because it processes not only the application description

document but also other application-related objects stored in repository (i.e. binaries and test suites).

THE STRING-DISTANCE MODULE

This research investigated eleven string-distance methods – four edit-based functions (Damerau-Levenshtein, Jaro-Winkler, Case and Fixed Weight), six token-based functions (TFIDF/Cosine, Jaccard, Dice, Jensen-Shannon Divergence (JSD), Dirichlet JSD, Jelinek-Mercer JSD) and one hybrid method (Jaro-Winkler/TFIDF). All these methods have been used successfully in matching paragraphs of free text in other contexts. [70, 81, 82, 83, 86, 97, 98, 103, 105] These eleven string-distance methods were implemented in the string-distance module of the GAMRS Matchmaking service and their performance in matching Grid application descriptions was analysed subsequently. The mathematical background and explanations of each of these methods can be found in Appendix C. The string-distance methods were used to identify similar applications by comparing the information contained in the free-text description of the application. This module was proposed as an alternative to other methods, in cases where the application description document or other application-related objects such as binaries or test suites were missing from the repository. This research also proposed a new technique for improving the accuracy of such string-distance methods by using entropy-generated stop-lists.

A *stop-list* (sometimes called a block-list) is, by definition, a list of terms which are filtered out prior to (or sometimes after) the processing of natural language texts. A stop-list includes the most common parts of speech, which usually occur very frequently in any text. In most cases a stop-list includes punctuation marks, prepositions (e.g. *of, to, in, for, with*), conjunctions (e.g. *and, but, or, nor*), and articles (e.g. *the, a, an*).

In this research I decided to expand the concept of stop-list and not to limit its contents to the elements enumerated above. The purpose of this extension was to optimize the matching abilities of the string-distance metrics used here by

generating a stop-list containing the terms with the lowest entropy levels in the whole corpus.

The underlying assumption was that token-based methods, which depend on term frequency (such as TFIDF/Cosine, Jaro-Winkler/TFIDF and Jensen-Shannon variants), could improve their matching accuracy if the terms that occur most frequently in the corpus (but not only prepositions, conjunctions, articles and punctuation marks) were filtered out beforehand.

In order to test this optimization solution via an entropy-generated stop-list, I proposed the following function to generate the stop-list:

GENERATE_STOP-LIST (*C, thd, S*)

1: FOR each distinctive t_i in *C*

2: $p(t_i) \leftarrow \frac{c(t_i)_C}{|C|_t}$

3. $H(t_i) \leftarrow -p(t_i)\log(p(t_i)$

4: IF $H(t_i) \leq thd$

5: add t_i to *S*

6: END_IF

7: END_LOOP

8: RETURN S

The following notations were used:

- *t = term:* basic entity in token-based analysis; the equivalent of a word in natural language;
- *d = document:* a set of terms; in our case, a paragraph of text written in English, which represents a free-text description of a Grid application;
- *C = corpus:* a collection of documents;
- *S:* the *stop-list* returned by the function;
- *thd:* the *entropy threshold* used for the generation of the stop-list;
- *H: Shannon entropy* function*;*

- $c(t)_C$: the number of occurrences of term t in the collection of documents C;
- $|C|_t = cardinality\ of\ C\ in\ relation\ to\ t$: the number of terms in corpus C.

The function above computes the probability of each term t_i (step 2) as its frequency of occurrence in corpus *C* (i.e. maximum likelihood), and uses this probability to compute Shannon's entropy for term t_i (step 3). If the entropy is lower than a threshold *thd* (step 4), it adds the term to the stop-list *S* (step 5). The entropy threshold *thd* can take any real value in the interval (0, 1). Each corpus *C* may have its own optimal values/interval for *thd*. Finding such values/interval takes into account the following two aspects:

- First, if threshold *thd* is too low, then too many non-important terms would be processed by the matching methods;
- Second, if threshold *thd* is too high, then too many important terms would not be processed by the matching methods.

The stop-list *S* is then used in the process of matching two application descriptions. First, using the following function, the terms contained in the stop-list are removed from each description:

```
TRIM_DESCRIPTION (d, S, δ )
1:      FOR each distinctive term t_i contained in the description d
2:          IF t_i not contained in the stop-list S
3:              add the term t_i to the description δ
4:          END_IF
5:      END_LOOP
6:      RETURN δ
```

Second, the following function is used to match two application descriptions:

```
MATCH_DESCRIPTIONS (d_1, d_2, M, S, result)
1:      IF use the stop-list S
2:          NO: result ← apply M (d_1, d_2)
3:          YES: δ_1 ← TRIM_DESCRIPTION (d_1, S, δ_1)
```

4: $\delta_2 \leftarrow$ TRIM_DESCRIPTION (d_2, S, δ_2)

5: *result* $\leftarrow$ apply M (δ_1, δ_2)

6: END_IF

7: RETURN *result*

The following notations were used:

- d_1, d_2: the original, untrimmed descriptions of two applications;
- S: the stop-list;
- δ_1, δ_2: the descriptions d_1, d_2 without the low-entropy terms;
- M: an implementation of a string-distance metric;
- *result*: similarity returned by the string-distance method M.

The MATCH_DESCRIPTIONS function can be used with any of the string-distance metrics considered in this research. Moreover, the function can be used both in cases were a stop-list is required, and in cases were matching is done without a stop-list.

Finally, the string-distance algorithm applies the MATCH_DESCRIPTION function for each of the string-distance methods implemented:

STRING DISTANCE APPLICATION MATCHMAKER (UID_1 , UID_2 , MM_k , C, T, $thd_l^{MM_l}$])

1: retrieve from GAMRS the free-text application description for $UID_1 \rightarrow d_1$

2: retrieve from GAMRS the free-text application description for $UID_2 \rightarrow d_2$

3: IF d_1 OR d_2 does NOT EXIST

4: YES: RETURN decision: *"string-distance algorithm cannot be applied in this case"*

5: END_IF

6: FOR each string-distance method MM_k

7: IF MM_k needs training

8: YES: train MM_k on T

9: END_IF

10: IF MM_k method uses a threshold for stop-list

11: YES: $S_k \leftarrow$ GENERATE_STOP_LIST(C, $thd_k^{MM_k}$, S_k)

12: END_IF

13: MATCH_DESCRIPTIONS (d_1, d_2, MM_k , S_k, *result*) $\rightarrow result_k$

14: END_LOOP

15: RETURN the list of tuples $\{ (MM_i, \text{result}_i) : i = 1, \dots, k \}$

The notations used in the algorithm are as follows:

- UID_1, UID_2 : The two input GAMRS application *universal identifiers* (UIDs), which refer to the applications under comparison;
- MM_k : the string-distance method *k;*
- *C* : the collection of application descriptions;
- *T* : the collection of application descriptions used as training corpus;
- $thd_k^{MM_k}$: the entropy threshold used for the generation of the stop-list required by the string-distance method MM_k .
- d_1, d_2 : The two free-text description of the applications identified in GAMRS by UID_1 and UID_2;
- S_k: the stop-list used by the string-distance method MM_k .
- $result_k$: the similarity score returned by the string-distance method MM_k;

The algorithm starts by retrieving the free-text description of the applications identified as UID_1 and UID_2 in GAMRS from the metadata associated to them in the repository (steps 1-2). If one of these application descriptions is missing (steps 3-5), the string-distance methods cannot be applied and the algorithm stops. Next, the algorithm takes each string-distance method MM_k (step 6) and executes the following steps: if the method needs training, then it uses the training corpus *T* to train it (steps 7-9); next, if the method can be applied with the entropy-generated stop-list (step 10), it generates the stop-list S_k (step 11) using the threshold $thd_k^{MM_k}$ given as argument for that particular string-distance method; finally, it runs the method using the MATCH_DESCRIPTIONS function (step 13) described above and records the result. The algorithm finishes by returning the list of methods and the score of the comparison for each of them (step 15).

APPLICATION-RUNNING MODULE

The GAMRS repository permits the storage of application binaries and test suites. The *application-running module* compares two applications by running their

application binaries with a common set of input files (retrieved from one of application test suites) and comparing their output sets. The proposed algorithm is presented below.

APPLICATION RUNNING MATCHMAKER (UID_1 , UID_2)

1a: retrieve description documents $MRDL_1$ and $MRDL_2$ from GAMRS using UID_1 and UID_2

1b: $D_1 \leftarrow$ retrieve the value of attribute *DeterministicType* from $MRDL_1$

1c: $D_2 \leftarrow$ retrieve the value of attribute *DeterministicType* from $MRDL_2$

1d: IF D1 OR D2 EQUALS 'Boolean *false*'

1e: YES: RETURN decision: *"one or both applications identified in GAMRS by UID_1 and UID_2 is non-deterministic; the application running module cannot be applied in this case"*

1f: END_IF

1: IF the test suite for the application identified by UID_1 in GAMRS exists stored in the repository

2: YES: retrieve the test suite from GAMRS using $UID_1 \rightarrow TestSuite_1$

3: $IN_1 \leftarrow$ retrieve the set of input files from $TestSuite_1$

4: $OUT_1 \leftarrow$ retrieve the set of output files from $TestSuite_1$

5: $RUN_1 \leftarrow$ retrieve the running script from $TestSuite_1$

6: END_IF

7: IF the test suite for the application identified by UID_2 in GAMRS exists stored in the repository

8: YES: retrieve the test suite from GAMRS using $UID_2 \rightarrow TestSuite_2$

9: $IN_2 \leftarrow$ retrieve the set of input files from $TestSuite_2$

10: $OUT_2 \leftarrow$ retrieve the set of output files from $TestSuite_2$

11: $RUN_2 \leftarrow$ retrieve the running script from $TestSuite_2$

12: END_IF

13: IF (IN_1 EQUALS IN_2) AND (OUT_1 EQUALS OUT_2) AND (RUN_1 EQUALS RUN_2)

14: YES: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are identical"*

15: END_IF

16: IF (IN_1 EXISTS) AND (OUT_1 EXISTS)

17: run application identified in GAMRS by UID_2 with $IN_1 \rightarrow OUT$

18: save IN_1 and OUT in $TestSuite_2$

19: update the application identified in GAMRS by UID_2 with $TestSuite_2$

20: IF (OUT EQUALS OUT_1)

21: YES: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are identical"*

22: NO: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are NOT identical"*
23: END_IF
24: END_IF
25: IF (IN_2 EXISTS) AND (OUT_2 EXISTS)
26: run application identified in GAMRS by UID_1 with $IN_2 \rightarrow OUT$
27: save IN_2 and OUT in $TestSuite_1$
28: update the application identified in GAMRS by UID_1 with $TestSuite_1$
29: IF (OUT EQUALS OUT_2)
30: YES: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are identical"*
31: NO: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are NOT identical"*
32: END_IF
33: END_IF
34: IF (IN_1 EXISTS)
35: run application identified in GAMRS by UID_1 with $IN_1 \rightarrow OUT_1$
36: run application identified in GAMRS by UID_2 with $IN_1 \rightarrow OUT_2$
37: save IN_1 and OUT_1 in $TestSuite_1$; and save IN_1 and OUT_2 in $TestSuite_2$
38: update the application identified in GAMRS by UID_1 with $TestSuite_1$; and update the application identified in GAMRS by UID_2 with $TestSuite_2$
39: IF (OUT_1 EQUALS OUT_2)
40: YES: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are identical"*
41: NO: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are NOT identical"*
42: END_IF
43: END_IF
44: IF (IN_2 EXISTS)
45: run application identified in GAMRS by UID_1 with $IN_2 \rightarrow OUT_1$
46: run application identified in GAMRS by UID_2 with $IN_2 \rightarrow OUT_2$
47: save IN_2 and OUT_1 in $TestSuite_1$; and save IN_2 and OUT_2 in $TestSuite_2$
48: update the application identified in GAMRS by UID_1 with $TestSuite_1$; and update the application identified in GAMRS by UID_2 with $TestSuite_2$
49: IF (OUT_1 EQUALS OUT_2)
50: YES: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are identical"*
51: NO: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are NOT identical"*
52: END_IF
53: END_IF

54: RETURN decision: "*The application-running module could not reach a decision based on the information contained in the test suites of the applications identified in GAMRS by UID_1 and UID_2*"

The notations used in the algorithm are as follows:

- *UID_1, UID_2*: The two input GAMRS application *universal identifiers* (UIDs), which refer to the applications under comparison;
- *$MRDL_1$, $MRDL_2$*: The application description documents corresponding to the two applications identified in GAMRS by UID_1 and UID_2;
- *D_1, D_2*: The values of the field *DeterministicType* from $MRDL_1$ and $MRDL_2$;
- *$TestSuite_1$, $TestSuite_2$*: Application test suites (each consisting of set of input files, set of output files and running script) associated with the two applications identified in GAMRS by UID_1 and UID_2;
- *IN_1, IN_2*: Sets of input files retrieved from the test suites *$TestSuite_1$, $TestSuite_2$*;
- *OUT_1, OUT_2*: Sets of output files retrieved from the test suites *$TestSuite_1$, $TestSuite_2$*;
- *RUN_1, RUN_2*: Running scripts retrieved from the test suites *$TestSuite_1$, $TestSuite_2$*;

Initially, the application-running algorithm consisted only of steps 1-50. However, after testing the algorithm with real applications published in Grid repositories, the results showed that the method could not be applied to applications that exhibited non-deterministic behaviour. In conclusion, the algorithm was improved with steps 1a-1f, according to which it retrieves the values of field *DeterministicType* from the two description documents of the applications identified in GAMRS by UID_1 and UID_2 and continues only if the two applications are deterministic. The algorithm continues with the retrieval from GAMRS of the two test suites corresponding to the two applications identified by UID_1 and UID_2. At the same time, from each of the test suites the algorithm extracts the set of test inputs, the set of test outputs and

the running script (steps 1-12). Next, it tests whether the two sets of inputs, the two sets of outputs and the running scripts are identical (step 13). Provided these are identical, it returns the decision that the applications are the same (step 14). Next, if one application has both the set of inputs and the set of outputs present (step 16 /25), it runs the other application with the set of inputs and records the new set of outputs in OUT (step 17 /26). It then records the set of inputs used to run the second application in its test suite along with the OUT set of outputs (step 18 /27) and it updates the test suite in GAMRS (step 19 /28). The algorithm continues with the comparison of the set of outputs belonging to the first application (step 20 /29) against the OUT set of outputs belonging to the second one (i.e. the application which has been run). If there is a match, this means that both applications are deterministic and they produced the same set of output files after being run with the same set of input files; the algorithm therefore decides that the two applications are identical (step 21 /30). If the comparison failed and the two set of output files are not identical, the algorithm decides that the two applications are not the same (step step 22 /31). If only the set of input files is present (step 34 /44), then both applications are run with this set of input files and their set of output files are recorded in OUT_1 (step 35 /45) and OUT_2 (step 36 /46). Similar to the functionality explained above (steps 18-19), the test suite of each application is updated with the set of input files used for the run and the corresponding set of output files (step 37 /47); and GAMRS is updated with the new test suites (step 38 /48). Next, the algorithm tests whether OUT_1 is identical to OUT_2 (step 39 /49), and if they are, it marks the two applications as identical (step 40 /50). Otherwise, the algorithm decides that the two applications are not the same (step step 41 /51). Finally, if all of the above tests fail (e.g. both two test suites are missing the set of input files), the algorithm acknowledges that it cannot reach a decision based on the information contained in the test suites of the applications identified in GAMRS by UID_1 and UID_2 (step 54).

BINARY MATCHING MODULE

The binary matching technique is used to identify identical applications based on the comparison of their binaries. Consequently, it can be applied only when both application binaries are available in GAMRS. The proposed algorithm is presented below.

APPLICATION BINARY MATCHMAKER (UID_1 , UID_2)
1: IF the binary for the application identified by UID_1 in GAMRS exists stored in the repository
2: YES: retrieve the binary from GAMRS using $UID_1 \rightarrow Binary_1$
3: compute hash sum of type *SHA-512* of $Binary_1 \rightarrow Hash_1$
4: $HashType_1 \leftarrow$ *SHA-512*
5: update GAMRS application identified by UID_1 with $Hash_1$ and $HashType_1$
6: NO: RETURN *"The binary matching method cannot be applied in this case."*
7: END_IF
8: IF the binary for the application identified by UID_2 in GAMRS exists stored in the repository
9: YES: retrieve the binary from GAMRS using $UID_2 \rightarrow Binary_2$
10: compute hash sum of type *SHA-512* of $Binary_2 \rightarrow Hash_2$
11: $HashType_2 \leftarrow$ *SHA-512*
12: update GAMRS application identified by UID_2 with $Hash_2$ and $HashType_2$
13: NO: RETURN *"The binary matching method cannot be applied in this case."*
14: END_IF
15: IF $Hash_1$ EQUALS $Hash_2$
16: RETURN decision: *"the applications identified in GAMRS by UID_1 and UID_2 are identical"*
17: END_IF
18: RETURN decision: *"the binaries of the applications identified in GAMRS by UID_1 and UID_2 are not identical"*

The notations used in the algorithm are as follows:

- *UID_1, UID_2*: The two input GAMRS application *universal identifiers* (UIDs), which refer to the applications under comparison;
- *$Hash_1$, $Hash_2$*: Hash sums of binaries corresponding to the two applications identified in GAMRS by UID_1 and UID_2;
- *$HashType_1$, $HashType_2$*: The type of hash sums $Hash_1$, $Hash_2$;

- *Binary$_1$, Binary$_2$*: Binaries corresponding to the two applications identified in GAMRS by UID$_1$ and UID$_2$;

The algorithm commences with the retrieval from GAMRS of the application binary for the first application with the help of the universal identifiers UID$_1$ (step 1). If the application binary is missing, then the binary matching technique cannot be applied in this case (step 6). If the application binary exists, it is retrieved from GAMRS (step 2) and then the algorithm computes a hash sum of this binary (Note: in this description the algorithm uses the SecureHashAlgorithm-512 to compute the hash sum; however, there is no restriction on the type of hash algorithm that can be used in implementation) – step 3. Next, the algorithm records the type of hash algorithm (step 4) used to generate the hash sum. It then updates the GAMRS application binary entity (step 5) with the hash type and the hash sum of the application binary.

The algorithm repeats the same set of actions for the second application, which is identified in GAMRS by the universal identifier UID$_2$ (steps 8-14). Next, it continues by comparing the newly computed hash sums of the two application binaries (step 15). If these hash sums are the same, then the applications are identical (step 16). If the hash sums are not the same (step 18), the algorithm cannot decide over the similarity or the identicalness of the two applications (e.g. the application binary is dependent on the operating system of the target machine; hence, even though two binaries are different they might represent the same application but compiled to run under two different operating systems).

THE GAMRS MATCHMAKER

All the matchmaking techniques proposed as GAMRS Matchmaking modules in the architecture process information about applications stored in GAMRS and can help identify similar applications. The following matchmaking algorithm is proposed to be used in conjunction with the GAMRS Matchmaking service:

GRID APPLICATION MATCHMAKER (UID_1 , UID_2 , MM_k , cnf_k)
1: FOR each matchmaking method MM_k implemented
2: IF application-related objects necessary for MM_k are not retrieved yet
3. YES:retrieve objects from GAMRS repository using UID_1 and UID_2

4: IF the objects are missing
5: YES: mark MM_k as not-available in this case
6: CONTINUE from 1:
7: END_IF
8: END_IF
9: apply $MM_k \rightarrow result$
10: record $result$ as the partial score PS_k
11: END_LOOP
12: RETURN the set of tuples $\{ (PS_i, \ suggested\ decision_{PS_i}) : i = 1, \dots, k \}$

12a: calculate the final score $FS(PS_k, cnf_k)$
12b: RETURN the tuple $(FS, \ suggested\ decision_{FS})$

The notations used in the algorithm are as follows:

- UID_1, UID_2 : The two input GAMRS application *universal identifiers* (UIDs), which refer to the applications under comparison;
- MM_k : the matchmaking method *k;*
- cnf_k : the weight of the matchmaking method *k;* in the case of an aggregation model, it specifies how trustworthy the method *k* is, i.e. the level of confidence of the system in the accuracy of the method *k*;
- PS_k = *partial score*: the score returned by the matchmaking method MM_k
- FS = *final score*: the score returned by the aggregation model after combining the partial scores (PS_k) scaled with their corresponding weights (cnf_k).

The Matchmaking service architecture is extendible; consequently, the algorithm has to be generic enough to permit the usage of all suggested methods – including hybrid methods that use aggregation models.

The algorithm starts by taking each matchmaking method MM_k specified in the matchmaking request document and uses the universal identifiers of the two applications (UID_1, UID_2) to retrieve the application-related objects necessary for MM_k to run (steps 2-8). If any of the objects required for a correct run of MM_k is missing (step 5), the method is marked as *not-available* and will not be used in the

matchmaking process. Next, the algorithm runs the method and records the partial score (step 10). For each partial score PS_k, the algorithm returns the score and the suggested decision regarding the similarity/dissimilarity of the two applications (step 12). If an aggregation model is put in place, the algorithm computes a *final score (FS)* based on the partial scores PS_k and their respective weights cnf_k. (step 12a) Finally, the algorithm returns the score *FS* and the suggested decision with regard to the similarity degree of the two applications (step 12b). The final result (*FS*) and the partial results (PS_k) can be recorded in a GAMRS *RelationPair* object, which can then be used in further matchmaking cases.

3.4.3 Summary

The GAMRS Matchmaking service proposes a matchmaking algorithm which aims to identify similar or identical Grid applications stored in repositories by processing various application-related objects stored in these repositories. This research identified the set of objects that could help in the process of application matchmaking (e.g. application description document, free-text descriptions of applications, application binaries, application binary hash sums, application test-suites, application source code, application dependencies) and suggested several matchmaking approaches based on this set.

Due to existing time constraints, only four matchmaking algorithms were proposed: the syntactic matchmaking, the string-distance matchmaking, the application-running matchmaking and the application binary matching algorithm. The rest of the methods suggested in this research (i.e. semantic, LSI/LSA/SVD, source code matching, dependency analysis, and hybrid matchmaking based on aggregation models) remain to be implemented and analysed in future research. The question about how the latter set of methods can help with the identification of similar Grid applications stored in repositories remains open.

The *syntactic* algorithm proposed in this research can be used in two forms: *purely syntactic* or *hybrid*. In its *purely syntactic* form, the algorithm processes application

description documents only. The choice of language was the GAMRS's MRDL which can help improve the algorithm's accuracy thanks to some of its new additions, such as *hash sums* and *advanced parameter description*. In its *hybrid* form, the algorithm can be extended to other scenarios and its performance can be increased by using application *test-suite* objects found stored in the repository.

The *string-distance* algorithm is meant to be used in scenarios where the repository holds no application-related object that can be used by other application matching methods and the only available information is the free-text description of the application. The algorithm can process such paragraphs of free-text and return the degree of similarity between them. An important addition to the string-distance algorithm is the *entropy-generated stop-list,* which can increase the performance of the string-distance methods. Furthermore, the entropy-generated stop-list technique is not restricted to be applied to Grid descriptions only, but it can be applied to any scenario involving training-based string-distance metrics.

The *application-running* algorithm compares two applications by running two application binaries with a common set of input files and compares the output set. This method is meant to be used in conjunction with two GAMRS application-related objects: *binary* and *test suite*, and shows excellent results when matching deterministic applications.

The *binary matching* algorithm compares two applications by computing hash sums of their binaries and testing whether they are identical or not. This method uses the *binary* and can be applied successfully in scenarios involving the matching of applications meant to run on Grid resources belonging to one Grid (e.g. all gLite Grid resources run Scientific Linux CERN v4/5).

3.5. Conclusions

The architecture of the Grid Application Meta-Repository System specifies a collection of services that work together in a system meant to provide the functional specifications for a new generation of Grid application repositories.

GAMRS provides an intuitive Web user interface, where users can easily publish applications and application-related objects into the repository. Furthermore, GAMRS provides an OGSI/WSRF Grid service interface that can be used by Grid Services to interact with the system, and a HTTP/REST interface that can be used by non-Grid services to publish applications directly onto GAMRS.

GAMRS connects different Grid application repositories and allows users and services to discover the Grid applications stored in them. It provides web visibility for all applications in the repositories connected to GAMRS, even if the majority of connected repositories did not provide it initially. Furthermore, Grid applications stored or referenced by GAMRS can be discovered by any Grid service compliant with the OGSI/WSRF standards stack and the HTTP/REST interface mentioned above allows search engines to discover such applications. Moreover, GAMRS employs an OAI provider, which other services can use for application metadata discovery (through the OAI-PMH protocol). The OAI provider also allows the exchange and reuse of application-related objects between repositories by using the OAI-ORE protocol. Both the object metadata and the actual object can be imported as well as exported automatically using XML-like documents compliant with OAI-ORE specifications.

GAMRS allows applications to be deployed embedded in virtual machines; therefore, the application runs in its native environment and can be used in application-on-demand, cluster-on-demand and cloud architectures. GAMRS can also embed commercial applications that require license acceptance and paid services. It offers a framework for deploying and running commercial applications provided a fee-based model is put in place to that end. Finally, GAMRS'

architecture and model allow Grid administrators to find and use all the required objects for Grid application deployment in one place - the GAMRS repository.

In addition, the proposed GAMRS repository model extends traditional models allowing inter-operability between Grid application repositories and other Grid services. The GAMRS repository model proposes a more detailed description of a Grid application and is also able to function as a mediator between older application repository models. Furthermore, the GAMRS model is able to describe new types of objects (such as the application provider object, user-related objects and provider-related objects) and also suggests a new set of application-related objects that should be modelled by any Grid application repository model. These new categories of application-related objects allow for Grid application repositories to be used in conjunction with newly emerging technologies such as virtualization, automatic virtual machine creation, cloud computing and automatic service deployments. Furthermore, thanks to its novel model, GAMRS can be used in different scenarios, many of them not necessarily involving Grid infrastructures. For example, by exposing applications as virtual machines, Grid administrators and users can easily deploy these applications on virtualized infrastructures without being required to know or to perform any of the following procedures: Operating System installation, application installation, software dependency installation or even application configuration. Finally, these GAMRS application-related objects can also be used to help with the identification of similar or identical applications stored in the repositories connected to GAMRS.

This research identified the structured life-cycle for a Grid application which resides in a Grid repository, taking in consideration the different states in which the application can be found (i.e. template, instance, deployment and running) and how these states can be accommodated in an application description language schema. The GAMRS application description language proposed here – MRDL – extends the list of capabilities of traditional languages in the following areas: *multi-Grid/multi certificate secure data access, data protection, template, and application type of running.* Moreover, MRDL is able to refer application-related objects in its schema, which makes it able to reflect the ability of a Grid application to be used in different

deployment and running scenarios – such as remote compilation of the source code; staging application binaries and software dependencies; as well as deployment and running of virtual machine-embedded application in virtualized infrastructures. Furthermore, the new elements of MRDL also improve the accuracy of application matchmaking methods when trying to identify similar or identical applications stored in the repositories connected to GAMRS.

Finally, GAMRS also contains a matchmaking service able to process information about applications stored in repositories and identify similar or identical applications. The Matchmaking service architecture proposed in this research contains modules able to process not only application description documents, but also other application-related objects that can be found stored in Grid application repositories. The architecture of this service is extendible to other matching modules; the algorithm suggested in this research is generic and can accommodate multiple matching methods. The limited timeframe of this PhD restricted the implementation and testing of all matchmaking solutions proposed for Grid application matchmaking, so this research has focused on only four of them: *syntactic, string-distance, application running and application binary matching.* Furthermore, a new method of generating stop-lists based on term entropy has been proposed in conjunction with the string-distance algorithm. The analysis of test results shows that these methods can be applied successfully to Grid application matchmaking and lists both successful scenarios as well as scenarios where these methods have limited applicability or poor performance.

Chapter 4

Implementation and Tests

This chapter describes the implementation of the Grid Application Meta-Repository System, as well as the suite of tests designed to demonstrate the novel capabilities of GAMRS and to show how this solution could be used to successfully meet some of the current challenges related to Grid application repositories.

The chapter starts by discussing the implementation constraints resulted from the time limitations inherent to a PhD research. Next, it presents the design of the testbed and the implementation of the GAMRS experimental solution. It continues with the specification of the suite of tests used to prove GAMRS' capabilities and the metrics used for performance measurements as part of five different scenarios designed to test the functionality of GAMRS against the requirements set out at the beginning of this research. The chapter ends by presenting the conclusions drawn from the analysis of results obtained from the testbed.

The chapter explains how the proposed GAMRS architecture was implemented in a pilot-solution that contains all three core GAMRS services (*Publisher, Meta-Repository* and *Matchmaking*) along with the *Backend* and the access interfaces. Furthermore, it describes how the Meta-Repository service was implemented using

the OGSI/WSRF Grid service standard **(R2.2)** and how it was used to connect three different Grid application repositories (NGS AR, GEMLCA and myExperiment) to GAMRS. The pilot-solution provides access to these three repositories, as well as enables the retrieval of applications stored by them **(R2.3)**.

The chapter also describes a repository technology suitable to be used as the *Publisher* service for GAMRS. The technology supported the addition of OAI providers (both PMH and ORE) and provides a HTTP/REST API, which helps improve application visibility on the web **(R2.1)**. At the same time, the repository stores objects in XML format, which embeds datastreams and hence allows for the exchange of objects between similar repositories **(R3)**. Furthermore, the repository technology provides a friendly user interface which can be used to publish, search, modify and delete Grid applications **(R1)** and other GAMRS.

The chapter moves on to explain how the GAMRS repository model and the proposed application description language (MRDL) were used to store a Grid application in GAMRS as a virtual machine-embedded application. Staging this scenario helped demonstrate the versatility requirement of GAMRS **(R4)** by deploying and running the virtual machine-embedded application on a virtualized infrastructure.

Furthermore, the repository model and MRDL were used in conjunction with the Matchmaking service to help identify similar Grid applications in connected repositories **(R2.4)**. This chapter includes the analysis of results obtained by using the matchmaking algorithms proposed in Chapter 3, Section 3.4.2, with a focus on string-distance module and the new entropy-generated stop-list, and makes suggestions regarding the methods which proved the most suitable to Grid application matchmaking.

4.1. Constraints

Due to the time constraints inherent to a PhD research several restrictions had to be put in place in order to narrow down the implementation of the solution to a manageable timeframe. However, as discussed further in this chapter, these limitations do not hamper the results and contributions brought by this to scientific knowledge.

First, apart from the six Grid application repositories described in the critical analysis section (BDII, CHARON/iSoftrepo, GEMLCA, NGS AR, GRIMOIRES and myExperiment), which are also the most widely used in Grid, other application repository solutions exist (e.g. EGEE Application Repository, gUSE Repository), which have not been discussed or tested in the course of this research. However, their functionality is usually similar to the functionality of at least one of the repositories analysed here.

Second, with regard to the GAMRS Matchmaking service, one of the limitations of the pilot solution implemented is that it matches only applications described as *standalone-job* applications. However, the matchmaking methods employed here are not necessarily restricted to this type of Grid applications and could therefore be applied to workflows or Web Service-published applications in the future.

Third, although many matchmaking methods were mentioned when designing the Matchmaking service, the GAMRS pilot solution did not test all of these due to the time constraints mentioned above. However, the Matchmaking service does allow for the expansion of its capabilities with any other new method in the future. For now, this research was focused on testing *syntactical*, *string-distance, application-running* and *binary matching* techniques. Furthermore, it proposed and tested a new method of improving string-distance techniques by applying stop-lists based on the entropy of the terms contained in the corpus.

Fourth, the number of string-distance similarity techniques (i.e. Damerau-Levenshtein, Jaro-Winkler, Case, Fixed Weight, TFIDF/Cosine, Jaccard, Dice,

Jensen-Shannon Divergence (JSD), Dirichlet JSD, Jelinek-Mercer JSD and Jaro-Winkler/TFIDF) used in the GAMRS string-distance module is by no means exhaustive. The methods used here were chosen after investigating the most commonly-used techniques described in the literature that can process paragraphs of free-text and, consequently, could be applied in the case of matching Grid application descriptions.

Another limitation regarding the matchmaking system refers to the use of English language only in the descriptions of the applications.

Finally, although at conceptual level both the GAMRS repository model and the GAMRS application description language are able to meet all the challenges described in Chapter 3, Sections 3.2 and 3.3, the limited timeframe of this research did not allow for the full description and implementation of test scenarios for each of the aspects considered in these challenges. Such scenarios and their functionality will instead be discussed in Chapter 5 (*Contributions to Knowledge and Extensions*) as suggested extensions for the future.

4.2. Experimental Architecture

The following GAMRS implementation Architecture was designed in order to test the functionality of GAMRS (see Figure 4-1):

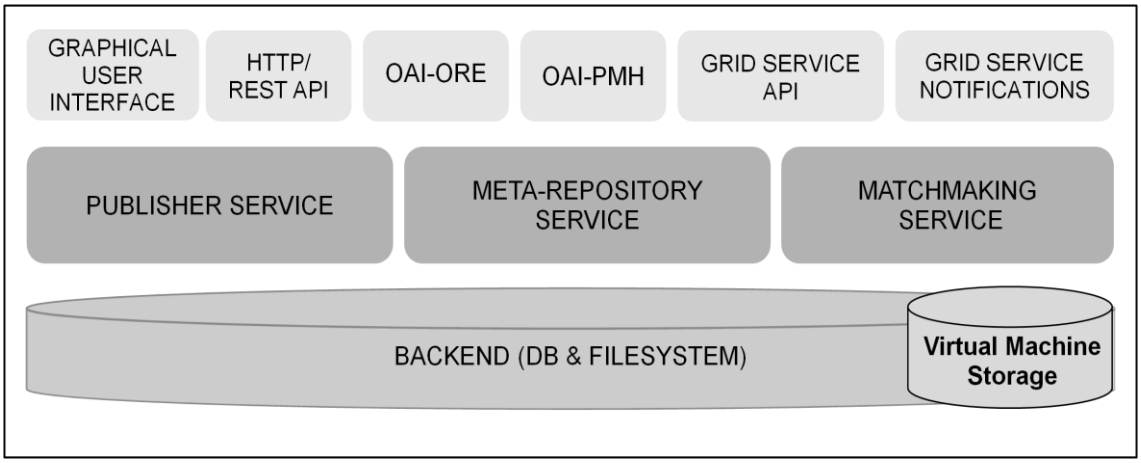


Figure 4-1: GAMRS implementation architecture

The architecture of the pilot implementation follows closely the theoretical GAMRS architecture, with one addition which does not alter the conceptual design and functionality of GAMRS. This modification is related to the inability of current repository technologies to store and handle efficiently files larger than 1GB in size. As the virtual machines can often exceed 1GB in size, a new storage service (*Virtual Machine Storage*) was added to the GAMRS Backend with the purpose of storing and managing large-size files.

The *Publisher* service was built on the most acclaimed open repository technology currently in production, namely Fedora Commons [120]. Fedora was chosen as a suitable repository technology candidate for the GAMRS Publisher service based on a critical analysis performed on seven widely used repository frameworks: Fedora Commons, ePrints [121], Oracle 10g Warehouse Builder [143, 144], IBM WebSphere Service Registry and Repository module [145, 146], ACS [147], WebGRelC [148], and Java COG kit [149]. These repository frameworks were compared with regard to eight properties necessary to meet the GAMRS requirements: their availability for download and their commercial status (free/paid); their ability to comply with GSI or HTTPS, which would make them usable on Grid security infrastructure; their ability to offer the basic CRUD and Search operations; their ability to accustom a user-defined repository model (i.e. the GAMRS repository model); the availability of a user-friendly access interface; their ability to provide a HTTP/REST interface; and their ability to support OAI-PMH/OAI-ORE protocols. The result of this analysis (which can be seen in Appendix D) presented Fedora as the most suitable candidate for the GAMRS *Publisher* service. Fedora was also augmented with two additional modules, *Islandora* [150] and *Drupal* [150], which provide a very user-friendly web interface (GUI) required by GAMRS to satisfy the **R1** challenge. These two software packages provide the web builder module and the web server (see Section 3.1.2, Figure 3-2) necessary to build and expose the Publisher GUI interface to human users.

Fedora, Islandora and Drupal were installed, configured and adapted to GAMRS requirements in terms of security and repository access. The Fedora repository technology also provided the necessary database structure, which functions as the

GAMRS Backend service, as well as the management module able to communicate with the database. Fedora technology supports the addition of an indexing service called gSearch [151]. Following the installation notes found at [152], gSearch was installed and configured to be used by the Fedora repository instance used for this research. The HTTP/REST interface was provided by default by the Fedora technology. Once the GAMRS repository model was mapped onto the repository, objects could be stored, retrieved, modified and deleted using the HTTP/REST interface.

Fedora was also chosen for the implementation of the GAMRS Publisher service because of its capability to support OAI-PMH and OAI-ORE protocols. The OAI-PMH interface was implemented with the help of an OAI-PMH provider compatible with the Fedora technology [153]. The provider was configured following the installation notes found at [154] and the OAI-PMH interface was tested using OAI-PMH queries directed to the GAMRS Provider service. The following snapshot shows the answer returned by the GAMRS Provider service to the OAI-PMH query *ListRecords*. (Note: The GAMRS solution ran on a private network under the IP 192.168.1.68; since the service had no DNS resolve, the OAI provider appended the default *"oai:example.org"* to its answers. Provided the GAMRS runs under a public IP with a full domain name such as *gamrs.cpc.wmin.ac.uk*, the provider answers would look like "*oai:gamrs.cpc.wmin.ac.uk*"). The list shows a partial view of the applications stored in GAMRS, which are identified as *gamrs:applicationXXX*.

http://192.168.1.68:8080/fedora/oai?verb=ListRecords&metadataPrefix=oai_dc
oai:example.org:gamrs:application563 2009-10-27T03:17:52Z EMBOSS
oai:example.org:gamrs:application564 2009-10-27T03:17:52Z EXONERATE
oai:example.org:gamrs:application565 2009-10-27T03:17:52Z FASTA
oai:example.org:gamrs:application567 2009-10-27T03:17:52Z GAMESS-UK
oai:example.org:gamrs:application568 2009-10-27T03:17:52Z GAMESS(US)
oai:example.org:gamrs:application569 2009-10-27T03:17:52Z GATE
oai:example.org:gamrs:application578 2009-10-27T03:17:52Z mpiBLAST
oai:example.org:gamrs:application579 2009-10-27T03:17:52Z NAMD

```
oai:example.org:gamrs:application583 2009-10-27T03:17:52Z Octave
oai:example.org:gamrs:application584 2009-10-27T03:17:52Z PC-GAMESS
oai:example.org:gamrs:application586 2009-10-27T03:17:52Z R
oai:example.org:gamrs:application587 2009-10-27T03:17:52Z Sabre(parallel)
oai:example.org:gamrs:application588 2009-10-27T03:17:52Z Sabre(serial)
.....
```

Appendix E shows the answer of an OAI-PMH *GetRecord* query over the Dublin core [155] metadata associated to one of the applications stored in the GAMRS repository (i.e. AMBER application, identifier *gamrs:application549*).

The OAI-ORE interface was implemented with the help of an OAI-ORE provider compatible with the Fedora technology [156]. The provider was configured following the installation notes found at [157]. This provider implementation follows the OAI-ORE implementation specifications of ResourceMaps as Atom feeds [158]. After successful configuration, the interface was tested using the Fedora administrative interface. Appendix F shows an example of a GAMRS application object serialized following OAI-ORE specification. Furthermore, the document shows the application description document written in MRDL, which was incorporated in the OAI-ORE representation, along with audit data and Dublin core metadata. (Note: The full document contains other information as well – i.e. application description written in original JSDL language, information about the collection of application-related objects, and other GAMRS model additions, but it has been reduced to show only the MRDL, Dublin core and audit metadata for the sake of compactness).

The OGSI/WSRF interface was implemented in Java programming language and uses the REST API provided by Fedora. OGSI/WSRF commands were implemented by embedding a HTTP client and using the HTTP/REST commands to create, retrieve, modify and delete objects stored in GAMRS.

The *GAMRS Meta-Repository service* and *GAMRS Matchmaking service* were developed from scratch, as described in the following sections of this chapter.

4.3. Test Scenarios

Five scenarios were designed to test the functionality of the GRID Application Meta-Repository System. These scenarios were chosen in relation to the research objectives set out in this thesis. By staging these scenarios it can be assessed whether and how the GAMRS solution meets objectives **O1-O4**.

The five scenarios were created to prove that GAMRS fully meets objective **O1,** by showing its ability to:

- connect to different Grid application repository technologies and solutions and retrieve the applications stored in them;
- be accessed by OGSI/WSRF Grid Services;
- expose the applications to the web via the HTTP/REST interface;
- support Search operations on metadata associated to objects stored in the repository;
- be accessed on WWW by any service equipped with a HTTP client;
- present the repository objects in a format that permits their exchange and reuse on other repositories built on technologies similar to that of GAMRS.

The five scenarios also aimed to prove that GAMRS fully meets objective **O2,** by showing GAMRS's ability to:

- function as a Grid application repository in its own right and allow users to publish objects inside GAMRS;
- store application-related objects following the categorization designed in the GAMRS repository model;
- be used on distributed infrastructures other than Grid.

Moreover, the five scenarios were created to prove that GAMRS also fully meets objective **O3,** by showing GAMRS's ability to:

- convert application description documents in MRDL for a uniform presentation of applications;
- facilitate the application matchmaking process;
- be used in new scenarios (i.e. deployment, running, testing) previously unavailable in traditional solutions (such as source code staging and compilation, virtual-machine running and testing a correct application deployment and functioning with the help of test suites).

Finally, the five scenarios were aimed to prove that GAMRS fully meets objective **O4,** by showing that:

- the matchmaking modules proposed in this research are suitable for the identification of similar applications stored in Grid repositories;
- the newly-proposed entropy-generated stop-list can improve the accuracy of string-distance methods when applied to matching applications stored in Grid repositories.

Successful completion of the following five scenarios therefore proves that the GAMRS solution satisfies the objectives set out in this research:

SCENARIO 1: CONNECTING GRID APPLICATION REPOSITORIES TO GAMRS

Connect the following repositories to GAMRS: the NGS application repository; the GEMLCA application repository provided by the University of Westminster; and the myExperiment application repository **(R2.3)**. Next, use the OGSI/WSRF interface and retrieve the application descriptions stored by the three repositories mentioned above **(R2.2)**. Save the application description document (written in the formal language employed by the respective repository) in GAMRS. Finally, convert their application description document into a new document written in MRDL (to provide uniform description formalism suitable for further processing) and save this document in the GAMRS.

The purpose of this scenario is to test GAMRS' ability to connect different Grid application repositories exhibiting different communication protocols, different access protocols and different security protocols and then to retrieve information about the applications stored by them. This scenario also tests GAMRS' OGSI/WSRF interface – which allows any other Grid service built according to OGSI/WSRF specifications to access and use GAMRS through this interface.

SCENARIO 2: SEARCHING IN GAMRS

Connect to GAMRS using the HTTP/REST interface **(R2.1)** and test the search function of GAMRS by introducing different keywords in the search field and directing the search to different metadata fields. The most widely used method for this is to search the *Name* field of applications; however, the search can also be directed to fields like *Description* or *Author*.

The purpose of this scenario is to tests the HTTP/REST interface (can be tested from any web browser), which provides web visibility to applications stored in GAMRS and connected repositories. Furthermore, this scenario can also test the *Search* capability of GAMRS.

SCENARIO 3: STORING APPLICATION-RELATED OBJECTS IN GAMRS

Use the *GAMRS Publisher service* to insert application-related objects into a Grid application stored in GAMRS. Next, test whether these objects can be accessed by users with the help of the graphical user interface (GUI) **(R.1)**. Finally, check whether the formal FOXML documents associated with these objects contain/refer datastreams – this should make such objects easily exchangeable and reusable in other Fedora-based repositories **(R.3).**

The purpose of this scenario is to test the ability of GAMRS to store application related-objects following the repository model. At the same time, this scenario tests the publishing requirement of GAMRS as well as its ability to exchange objects.

SCENARIO 4: USE GAMRS IN DISTRIBUTED INFRASTRUCTURES OTHER THAN GRID

Use a virtual-machine embedded application asset to deploy and use the Grid application on virtualized infrastructures.

The purpose of this scenario is to test GAMRS' ability to be used on infrastructures other than Grid **(R.4)**. This scenario required the deployment and configuration of a virtualized infrastructure at the Centre for Parallel Computing laboratory (University of Westminster) – hypervisors, servers and resource pools; as well as the creation of a new web service needed to deal with actions associated with the deployment and running of a new virtual machine on such infrastructure (e.g. virtual machine transfer, registration, cloning and start-up/shut-down).

SCENARIO 5: IDENTIFY SIMILAR OR IDENTICAL GRID APPLICATIONS USING GAMRS

Using the *GAMRS Matchmaking service,* test the syntactic algorithm on the applications retrieved from the three repositories connected to GAMRS: NGS, GEMLCA and myExperiment.

Create a training corpus of free-text application descriptions from the following three repositories: NGS, CHARON/iSoftrepo and EGEE. GEMLCA could not be used for this scenario as no free-text description was found in its application description documents for any of the applications exposed by this repository. The myExperiment repository stores Grid applications of the type *workflow* – as opposed to NGS and GEMLCA, which store Grid applications of type *standalone* – and no common set of applications was identified either with NGS or with GEMLCA. However, CHARON/iSoftrepo and EGEE both describe standalone Grid applications and a common set of applications was identified between the NGS repository and CHARON/iSoftrepo. Therefore, in order to make the testing of the string-distance method relevant, GEMLCA and myExperiment were replaced in this scenario with the free-text descriptions retrieved from CHARON/iSoftrepo and EGEE. The time constraints prevented the implementation of Meta-Repository

adapters for EGEE and CHARON/iSoftrepo. Consequently, the application descriptions were retrieved from these two repositories with the help of shell-scripts implemented specifically for this purpose.

Next, use a subset of application descriptions retrieved from NGS and CHARON/iSofrepo repositories to build a test corpus. Test the string-distance module by applying the methods on the test corpus and analyze the results to decide which string-distance method is most suitable for finding similar or identical applications stored in Grid repositories. Further, apply the entropy-generated stop-list method with different entropy thresholds and identify the cases when this method optimizes the performance of token-based string-distance techniques. Finally, construct three different training corpora from the application descriptions retrieved from the repositories; induce training of string-distance methods separately on each of these corpora; and analyze how this affects their accuracy in finding similar Grid applications.

Create test suites for the applications found in NGS and GEMLCA and analyze the performance of the application-running technique.

The purpose of this scenario is to test the ability of GAMRS to find similar applications that may reside in connected repositories **(R2.4)** using syntactic, string-distance and application-running methods. (Note: The binary matching technique is explained theoretically.) Furthermore, this scenario tests a new method of generating stop-lists based on the entropy of terms and looks at how this method can optimize the accuracy of string-distance techniques.

* * *

The following table shows the relation between each scenario, the Grid application repository challenge addressed, the research objective it relates to and the final goal of the test:

Table 4-1: Scenarios, objectives and goals

	CHALLENGE	OBJECTIVE	GOALS
SCENARIO 1	R2.2: Be interoperable with any OGSI/WSRF Grid services R2.3: Connect multiple Grid application repositories	O1: Architecture O3: MRDL	- To show GAMRS' ability to connect to different Grid application repository technologies and solutions (O1); retrieve the applications stored in them; and convert their description documents in MRDL for a uniform presentation of applications and to facilitate the application matchmaking process (O3). - To demonstrate GAMRS' ability to be accessed by OGSI/WSRF Grid Services (O1).
SCENARIO 2	R2.1: Expose Grid applications to the Web	O1: Architecture	- To show GAMRS' ability to support *Search* operations (O1) on metadata associated to objects stored in the repository. - To demonstrate GAMRS' ability to expose the applications to web via the HTTP/REST interface (O1). - To show GAMRS' ability to be accessed on WWW by any service equipped with a HTTP client (O1).
SCENARIO 3	R1: Application publishing R.3: Object exchangeability & reusability	O1: Architecture O2: Repository model	- To show GAMRS' ability to function as a Grid application repository in its own right and allow users to publish objects inside GAMRS (O2). - To demonstrate the storage of application-related objects following the categorization designed in the GAMRS repository model (O2). -To show GAMRS' ability to present the repository objects in a format that permits their exchange and reuse on other repositories which are built on technologies similar to that of GAMRS (O1).

SCENARIO 4	R4: Versatility	O2: Repository model	- To demonstrate that GAMRS can be used on distributed infrastructures other than Grid (O2). - To show that the storage of application-related objects following the categorization proposed in the GAMRS repository model (O2) helps expand the area of usage of GAMRS with new scenarios previously unavailable in traditional solutions.
SCENARIO 5	R2.4: Find similar applications	O3: MRDL O4: Matchmaking	- To show that matchmaking modules proposed in this research (O4) are suitable for the identification of similar applications stored in Grid repositories (O3). - To show that the newly-proposed entropy-generated stop-list can improve the accuracy of string-distance methods when applied to matching applications stored in Grid repositories (O4).

4.4. Testbed Results

SCENARIO 1: CONNECTING GRID APPLICATION REPOSITORIES TO GAMRS

	CHALLENGE	OBJECTIVE	GOALS
SCENARIO 1	R2.2: Be interoperable with any OGSI/WSRF Grid services R2.3: Connect multiple Grid application repositories	O1:Architecture O3: MRDL	- To show GAMRS' ability to connect to different Grid application repository technologies and solutions (O1); retrieve the applications stored in them; and convert their description documents in MRDL for a uniform presentation of applications and to facilitate the application matchmaking process (O3). - To demonstrate GAMRS' ability to be accessed by OGSI/WSRF Grid Services (O1).

The implementation of this scenario was done in Java programming language, under the Eclipse Integrated Development Environment (IDE). The deployment of the Meta-Repository service was done in a globus-tomcat container, which was configured to accept GSI certificates for authentication. The implementation of the Meta-Repository service was done following the latest OGSI/WSRF specifications, version 1.2 [42], which permits automatic interoperability with any other Grid service implemented according to this standard.

In this scenario the Meta-Repository Service connected the following three repositories to GAMRS: GEMLCA, NGS AR and myExperiment. Each of these repositories exposes an access interface different from the others: GEMLCA exposes an OGSI/WSRF Grid service interface; NGS is built on the JSR-168 portlet standard and had to be modified with the addition of a web service interface in order to permit services an easy access for its contents; and myExperiment exposes a HTTP/REST interface. Furthermore, the security systems of GEMLCA and NGS AR are based on GSI and HTTPS, while myExperiment is public, therefore no secure access is required to view and retrieve applications stored in it.

The Meta-Repository service implemented three adapters able to access the three repositories mentioned above. With the help of these adapters GAMRS was able to retrieve the application description documents stored by GEMLCA, NGS AR and myExperiment. GEMLCA was accessed with the help of an OGSI/WSRF Grid service client provided by the GEMLCA software package. NGS AR was accessed with the help of a Web service client written specifically for this research. myExperiment was accessed with the help of a HTTP client provided by the org.apache.commons.httpclient jar, which comes with the Apache server distribution [159]. For the secure access of GEMLCA and NGS AR, we used a GSI-compliant X509 certificate, which was issued by the NGS Certification Authority UK [160].

To provide uniformity in the way Grid applications are presented to GAMRS users, the information about all applications was formalized following the GAMRS repository model and the MRDL application description language. To accomplish that, the application description document of each application stored in the three repositories connected to GAMRS was retrieved and processed. (Note: The average size of a Grid application description document is around 4500B, therefore no strain was put on the GAMRS storage system during this process.)

In order to extract the necessary information needed for GAMRS storage and translation into MRDL language, two language converters were implemented to translate from GEMLCA's LCID to MRLD and from myExperiment's Scufl to MRDL. NGS AR's JSDL needed no conversion to MRDL, because MRDL is itself an extension to the JSDL schema and accepts all JSDL objects and relations. The two converters, the GAMRS repository model, and MRDL were developed in Java under the Eclipse Modelling Framework (EMF) – a tool that allows users to combine graphical modelling with Java development. The EMF module is provided as a plug-in for the Eclipse IDE. Documentation and download locations for EMF can be found in [161].

Following the processing of the application description document, all the information about the application (including the two description documents – one in

native language, the other in MRDL) was compiled in a FOXML document, which is the XML standard understood by the Fedora repository technology. Each object stored in a Fedora repository needs to be described in FOXML following a repository model. In this case, the FOXML document was created to describe objects according to the GAMRS repository model proposed by this research. An example of FOXML document describing the application BSoft retrieved from NGS repository is annexed in Appendix G. (Note: The BSoft FOXML document contains the Dublin core metadata; the RDF statements about the repository object (i.e. needed by Fedora internal management system); a thumbnail image (reference to a datastream of type IMAGE/JPEG stored in the repository); four application assets (one reference to a datastream of type pdf stored in the repository – *installation notes*; one reference to a datastream of type TAR_GZ-archive stored in the repository – *source code*; one reference to a datastream of type ZIP-archive stored in the repository – *test suite*; and one reference to a virtual machine stored by the Virtual Machine Storage Service – *virtual machine*); the GAMRS model metadata as a datastream of type XML, which is embedded in the FOXML document; the application description written in native application language (i.e. in this case JSDL) as a datastream of type XML, which is embedded in the FOXML document; and the application description written in MRDL as a datastream of type XML, which is embedded in the FOXML document.)

Due to time constraints, the implementation of the Meta-Repository service was not extended with automatic insertion of the FOXML documents in GAMRS. The FOXML documents were inserted in the repository manually, with the help of a shell script that uses Fedora administrative command lines. After adding the documents to the repository, the applications became visible in GAMRS.

Apart from Application objects, all the other GAMRS objects needed for this scenario (such as users, providers, policies) were created using the Fedora Administrative Web Interface.

Figure 4-2 represents a snapshot of the GAMRS graphical user interface which shows twelve applications out of the 505 retrieved from the three repositories (GEMLCA, NGS AR and myExperiment) connected to GAMRS:

Figure 4-2: GAMRS applications retrieved from NGS, GEMLCA and myExperiment

Successful completion of SCENARIO 1 permitted GAMRS to retrieve and store information about the applications stored in three Grid repositories: GEMLCA, NGS AR and myExperiment.

SCENARIO 2: SEARCHING IN GAMRS

	CHALLENGE	OBJECTIVE	GOALS
SCENARIO 2	R2.1: Expose Grid applications to the Web	O1: Architecture	- To show GAMRS' ability to support Search operations (O1) on metadata associated to objects stored in the repository. - To demonstrate GAMRS' ability to expose the applications to web via the HTTP/REST interface (O1). - To show GAMRS' ability to be accessed on WWW by any service equipped with a HTTP client (O1).

The Fedora repository framework permitted the addition of an *indexing* service, which was configured to index the terms found in the metadata associated to GAMRS objects. The user interface includes graphical *search* fields that can be used to search for keywords contained in the metadata of a GAMRS object. Furthermore, the interface permits search combinations that can be directed concomitantly to up to three metadata fields (e.g. title, author, description) – see Figure 4-3.

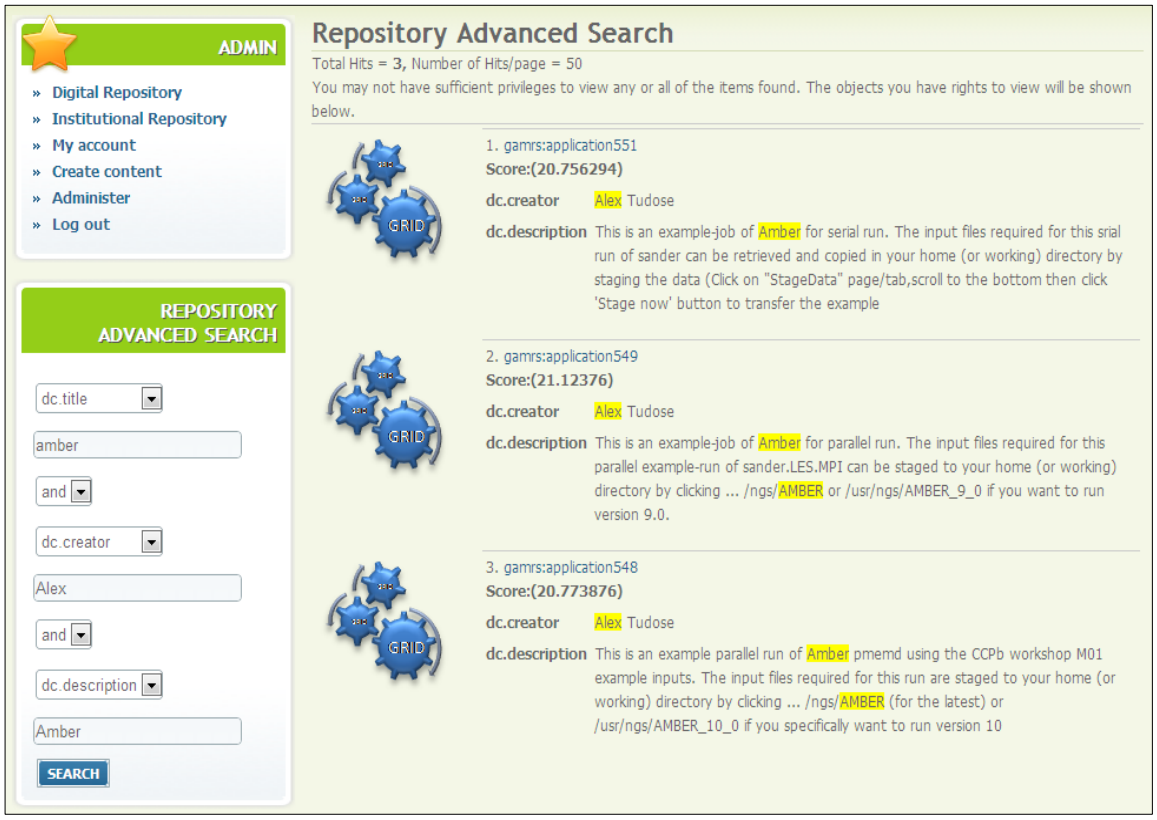


Figure 4-3: Example search for applications created by *"Alex"* that contain the word *"amber"* in title and the word *"amber"* in description

SCENARIO 2 was completed successfully and users can search for Grid applications using the graphical web interface provided by the Islandora/Drupal module, which is part of the *GAMRS Publisher*. Services can also use the HTTP/REST interface and perform searches on the GAMRS. The following snippet is an example of such a query which can be issued by any service equipped with a HTTP client:

Query: ”Search for all applications that contain the word *amber* in their descriptions and were created by any user whose name is *Alex*.”

HTTP:
http://X.X.X.X:8080/fedoragsearch/rest?operation=gfindObjects&query=dc.description"amber"+AND+dc.creator:"Alex"

The results of this query can be seen in Figure 4-4:

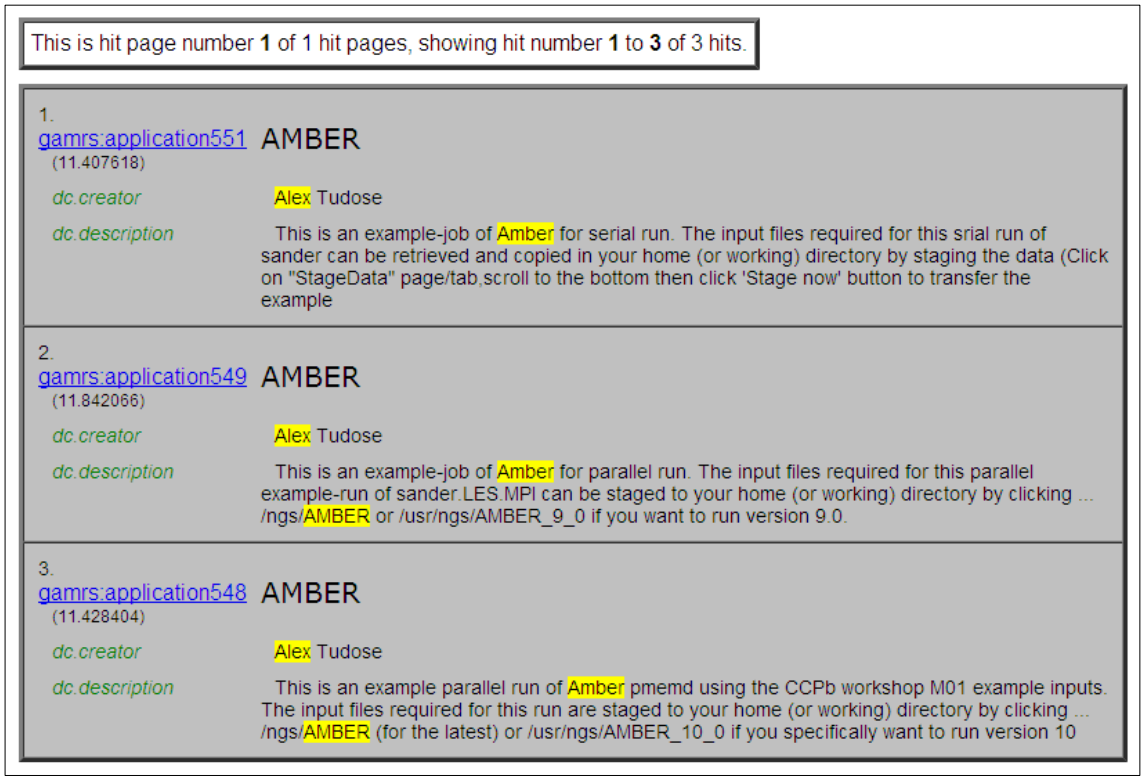


Figure 4-4: Example of a GAMRS HTTP/REST interface test

SCENARIO 3: STORING APPLICATION-RELATED OBJECTS IN GAMRS

	CHALLENGE	OBJECTIVE	GOALS
SCENARIO 3	R1: Application publishing R.3: Object exchangeability & reusability	O1: Architecture O2: Repository model	- To show GAMRS' ability to function as a Grid application repository in its own right and allow users to publish objects inside GAMRS (O2). - To demonstrate the storage of application-related objects following the categorization designed in the GAMRS repository model (O2). -To show GAMRS' ability to present the repository objects in a format that permits their exchange and reuse on other repositories which are built on technologies similar to that of GAMRS (O1).

In this scenario the repository administrative interface exposed by the *GAMRS Publisher service* was used to insert the following application assets to the BSoft Grid application [162]: source code, installation notes, test suite, and an OVF-compliant virtual-machine, which contains the BSoft application deployed inside. Bsoft is a collection of software for image and molecular processing in structural biology, which was adapted and is used on the NGS Grid infrastructure.

Figure 4-5 represents a screenshot of the GAMRS web interface showing the BSoft application and the metadata associated to it.

The full *name* of this application is PFT3DR-Bsoft and a short description of it can be seen in the *Description* field. The field *Owner* points to the GAMRS user who added the application to the system. Further on, the *Template* attribute is true, which means that this application is to be used as template for future instances. The value of *ADL Type* is JSDL, which refers to the application description language in which the application was originally published. The *Reference* provides users and services with the URI where the application information can be retrieved from GAMRS. The *Provider* field points to the GAMRS *Provider* object, which describes the repository where the application was originally retrieved from.

Figure 4-5: BSoft application stored in GAMRS

The *External ref* field contains the URI to the location of the application in its repository of origin (not in GAMRS). In this example, the URI (i.e. *http://portal.ngs.ac.uk/JobProfiles.jsf*) shows that this application was retrieved from NGS AR. The *Assets* field contains the list of references to the application objects related to this application. The *Relations* field contains the set of similar applications: the *Peer* attribute stores the reference to a similar application and the *Score* attribute registers the name of the matching method and the degree of similarity between the two applications.

Once GAMRS connects to a repository, the description documents of all applications contained in that repository are translated into MRDL and stored in GAMRS – both in their native language (*Application description – native ADL*) and in MRDL (*Application description – GAMRS ADL*). The *Detailed list of content* section contains the documents that refer to this repository object and are specific to the Fedora repository technology. Finally, the *Application Assets* icon shown at

the bottom of the figure represents the collection of application related objects linked to this application.

Correct storage and exposure of these application-related objects was tested via the GAMRS graphical web interface and references to the four objects can also be seen in Figure 4-5 under the headline *Assets.*

For this scenario, the virtual machine-embedded BSoft application was created with the help of the GAMRS pilot solution. First, the operating system was installed on the virtual machine. Next, from within the virtual machine, the Bsoft source code (see Figure 4-6) was downloaded via the HTTP/REST interface from GAMRS. Using the same interface, the BSoft installation notes were downloaded on the virtual machine as well. Following the instructions detailed in the installation notes, the source code was compiled and Bsoft was installed inside the virtual machine.

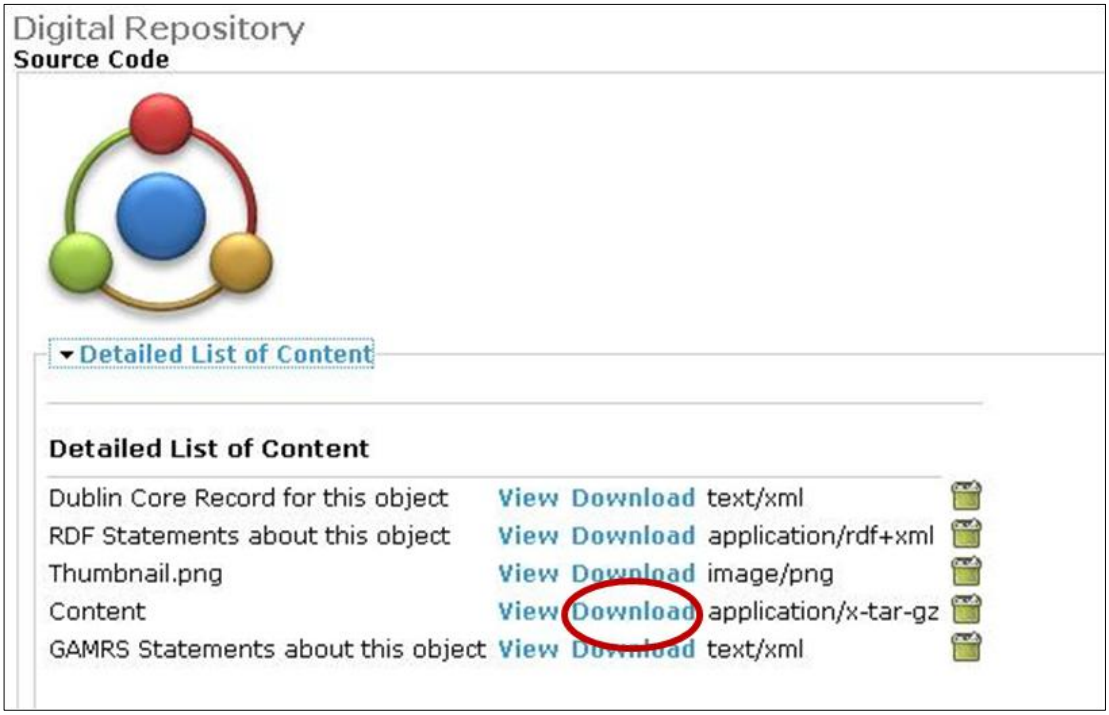


Figure 4-6: BSoft source code – content download

Further, the BSoft installation was tested with the help of the BSoft test suite, which had been downloaded from the GAMRS repository as well. The test suite contains three types of files: *test input file(s)*; the *running script* – which represents the correct command line(s) needed to run the application correctly; and *test output*

files – which represent the correct output when the application is run with the test input file(s) found in the test suite. Tests have shown a correct installation of BSoft within the virtual machine.

Finally, the virtual machine was stored on the *Virtual Machine Storage* server and a reference to it was added in GAMRS.

SCENARIO 4: USE GAMRS IN DISTRIBUTED INFRASTRUCTURES OTHER THAN GRID

	CHALLENGE	OBJECTIVE	GOALS
SCENARIO 4	R4: Versatility	O2: Repository model	- To demonstrate that GAMRS can be used on distributed infrastructures other than Grid (O2). - To show that the storage of application-related objects following the categorization proposed in the GAMRS repository model (O2) helps expand the area of usage of GAMRS with new scenarios previously unavailable in traditional solutions.

SCENARIO 4 tests how virtual machine-embedded Grid applications stored in GAMRS can be run on virtualized infrastructures (and implicitly on cloud infrastructures).

Figure 4-7 shows how GAMRS can be used in conjunction with virtualized/cloud infrastructures. The user can search GAMRS for a certain application (action 1) which s/he wants to deploy and run on a virtualized infrastructure. If the collection of application-related objects contains the virtual machine-embedded object, the user can then download it on his/her computer (action 2). Next, the user can connect to a cloud gateway or virtualization hypervisor access interface and upload the virtual machine on the virtualized infrastructure (action 3). The machine will remain saved in the pool of virtual machines and the user can instantiate it (even multiple copies, creating a cluster) any time s/he wants to use it to solve his/her problems (action 4).

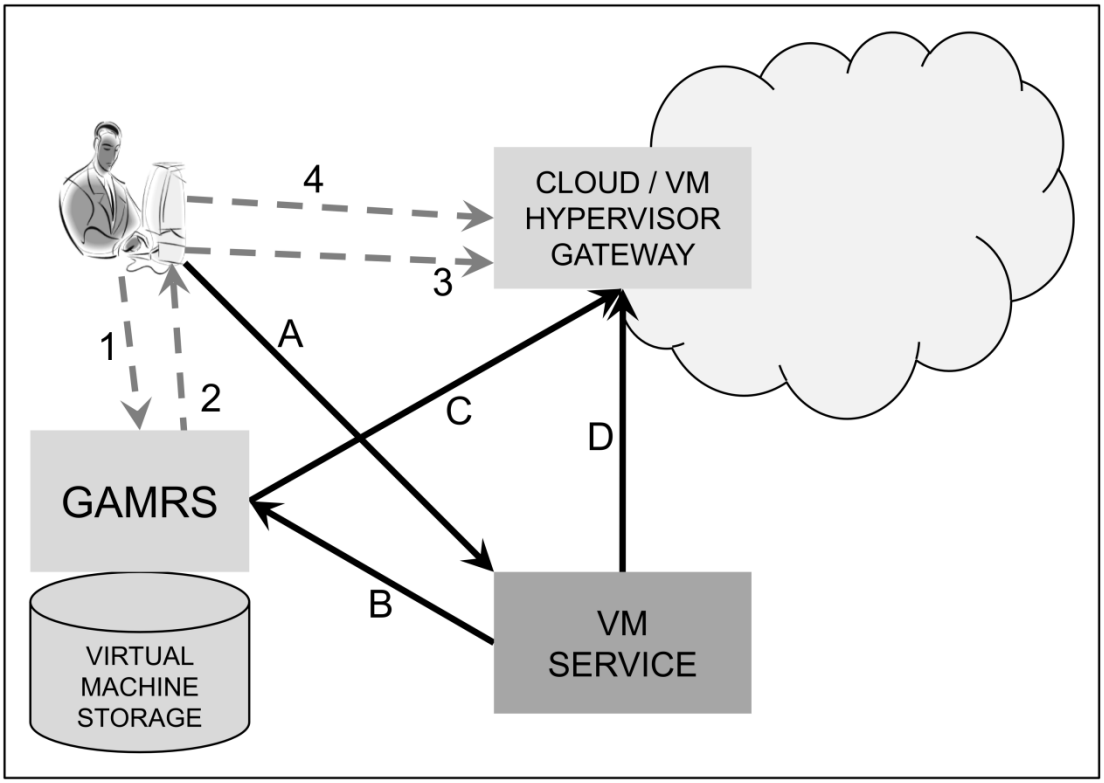


Figure 4-7: Using GAMRS in virtualized architectures

Traditionally, the transfer of the virtual machine from the repository to the user's computer and from the user's computer to the virtualized infrastructure represented a drawback. Virtual machines are usually large in size (i.e. several gigabytes) and they can be costly in terms of network usage and disk usage, hampering the normal utilization of the user's computer. However, this scenario implemented a solution, which overcomes this drawback. This involved the creation of a new service (*VM Service*), which is able to communicate both with GAMRS and the virtual infrastructure hypervisors.

The virtual machine-embedded application object stored in GAMRS contains in its associated metadata the path to its storage location, the protocol needed for access and staging the object, and the virtualization technology type used in the creation of the virtual machine (in this scenario the standard Open Virtualization Format was used – a format known by most virtualization hypervisors currently in production).

The cloud/virtualized hypervisors can be stored as provider objects in the GAMRS. They too have metadata associated with them, such as location, methods of access, protocols and security information. The user submits to the VM-Service the information about the virtual machine-embedded application and about the hypervisor on which s/he would like to deploy the virtual machine (action A). The VM-Service finds the virtual machine archive on GAMRS (action B) and initiates a direct transfer of that archive between the GAMRS storage and the virtualized hypervisor storage (action C). Upon successful completion, the VM-Service connects to the hypervisor access interface (action D) and unpacks the virtual machine files. Next, it registers the virtual machine as a template in the pool of virtual machines and, if so specified by user, can also clone an instance for the user, power the instance on and, preferably, start a VNC (Virtual Network Computing) server [163] on the virtual machine to enable remote desktop connections to it. The user can then access the application through the cloud interface or directly through a VNC viewer.

The suite of actions described above was successfully implemented using the following objects: the BSoft VM-embedded application specified in SCENARIO 3; a VMWare ESXi virtualization hypervisor [164] and VMWare server that were installed at the Centre for Parallel Computing (University of Westminster) and which acted as a gateway to the virtualized infrastructure; and the new VM Service, which was developed from scratch in the Java programming language and was deployed as a web service in a GlassFish_v3 container [165].

Appendix H contains a snapshot taken after the successful completion of SCENARIO 4. On the column on the left-hand side it shows the BSoft virtual machine deployed as a template on the VMWare server (*Template-Bsoft*) and a powered-on clone of the virtual machine (*alex-Bsoft*). On the right-hand side one can see the desktop of the *alex-Bsoft* machine through which users can interact with the virtual machine. Moreover, the snapshot shows an example of such interaction: the user started a shell console and listed the contents of the BSoft test suite that was installed on the virtual machine in SCENARIO 3.

SCENARIO 5: IDENTIFY SIMILAR OR IDENTICAL GRID APPLICATIONS USING GAMRS

	CHALLENGE	OBJECTIVE	GOALS
SCENARIO 5	R2.4: Find similar applications	O3: MRDL O4: Matchmaking	- To show that matchmaking modules proposed in this research (O4) are suitable for the identification of similar applications stored in Grid repositories (O3). - To show that the newly-proposed entropy-generated stop-list can improve the accuracy of string-distance methods when applied to matching applications stored in Grid repositories (O4).

The *syntactical module* of the *GAMRS Matchmaking service* was implemented in Java programming language under the Eclipse platform. Implementation of the MRDL parser was also implemented in Java using the MRDL model developed under the EMF plug-in of Eclipse. Since GEMLCA and NGS AR repositories share a common set of applications, the syntactic module was tested on their application description documents. myExperiment could not be used in this scenario because it did not have any set of applications common with either NGS AR or GEMLCA.

The syntactic modules correctly identified the identical applications which fall under the fast-track comparison test {*designated running site, binary location path, list of arguments*}; namely, the algorithm identified the applications, which had their binaries already deployed on the NGS Grid infrastructure and were described both in GEMLCA and NGS AR. However, the algorithm failed to identify identical applications in the following scenario: the application exposed by NGS AR had its binary already deployed on the NGS Grid infrastructure; the same application was exposed by GEMLCA but required binary staging.

Full MRDL comparisons returned inconclusive results due to the lack of information comprised in real-case application description documents. Furthermore, the extra functionality of the syntactical module (i.e. binary hash sums and test suites) could not be tested in real-life scenarios as it involves the usage of the new application-related object additions proposed in the GAMRS repository model.

The matching of applications using *test suites* (GAMRS application-related objects) was tested in the application-running module.

* * *

The *string-distance* module of the *GAMRS Matchmaking service* was implemented in Java programming language under Eclipse platform. Implementation of the eleven string-distance functions used in this analysis was provided by two Java archives: Lingpipe [167] and Stringmetrics [168].

This scenario used application descriptions that had been retrieved from three Grid application repositories: the NGS AR, the CHARON/iSoftrepo, and the EGEE application repositories. Given that NGS AR contained too few application descriptions to build a consistent training set, these application descriptions were grouped together with those from CHARON/iSoftrepo. Application descriptions were therefore grouped in three sets: *corpus123* (containing 123 descriptions retrieved from NGS and CHARON/iSoftrepo), *corpus246* (containing 246 descriptions retrieved from EGEE) and *corpus369* (containing 369 descriptions, i.e. all descriptions found on the three repositories). Techniques based on term frequencies and term importance probabilities were trained separately on each of these three sets in order to analyze the variance in the accuracy of string-distance methods when trained on different corpora.

The test corpus was comprised of a subset of descriptions retrieved from CHARON/iSoftrepo and NGS repositories, which were compared in pairs of two: each application description against each one of the remaining descriptions. This resulted in 4372 comparison cases, out of which 631 represented cases of similar applications.

Several decision groups were considered for each string-distance method and these groups were implemented with the help of *match-intervals*. If the score of a method had a value within the match-interval, the two application descriptions were considered *similar*, otherwise they were considered *dissimilar*. The match-intervals (i.e. decision groups) were constructed in an incremental way (e.g. [0.0, 0.1], [0.0,

0.2]... [0.0, 1.0]) and were used to analyse how the accuracy of the matching methods varied with the relaxation or restriction of decision rules.

The accuracy of string-distance methods was compared with the help of the following measurements:

- *False positives* = how many times the method had marked as similar two applications which were different from each other.
- *False negatives* = how many times the method had marked as dissimilar two applications which were similar.
- *Precision* = the number of true positives (i.e. the comparison cases that correctly marked the applications as similar) divided by the total number of comparison cases that marked the applications as similar (i.e. the sum of true positives and false positives).
- *Recall* = the number of true positives divided by the total number of cases that compared two similar applications (i.e. the sum of true positives and false negatives).
- *F1* = the harmonic mean of precision and recall.

In addition to the measurements mentioned above, the matchmaking methods were also compared using the *average precision* and the *maximum F1* value of each method. Precision and recall need to be balanced, and when *F1* is maximized, both precision and recall are set to acceptable values.

All eleven string-distance methods were run with their respective matching-intervals and different entropy thresholds, using the three training corpora described above. The analysis of results collected in this exercise showed that overall, token-based matchmaking methods performed better than edit-distance metrics. This can be seen in the chart in Figure 4-8, which depicts the maximum F1 values obtained in the analysis:

(Note: in the next figures, the following notations have been used for the string-distance methods: TFIDF for TFIDF/Cosine; JW-TFIDF for Jaro-

Winkler_TFIDF/Cosine; JSD for Jensen-Shannon Divergence, JM-JSD for Jelinek-Mercer/Jensen-Shannon Divergence; D-JSD for Dirichlet/Jensen-Shannon Divergence; CASE for Case edit-distance; DL for Damerau-Levenshtein distance; FIXED for Fixed Weight edit-distance; DICE for Dice distance; JACCARD for Jaccard distance; and JW for Jaro-Winkler distance).

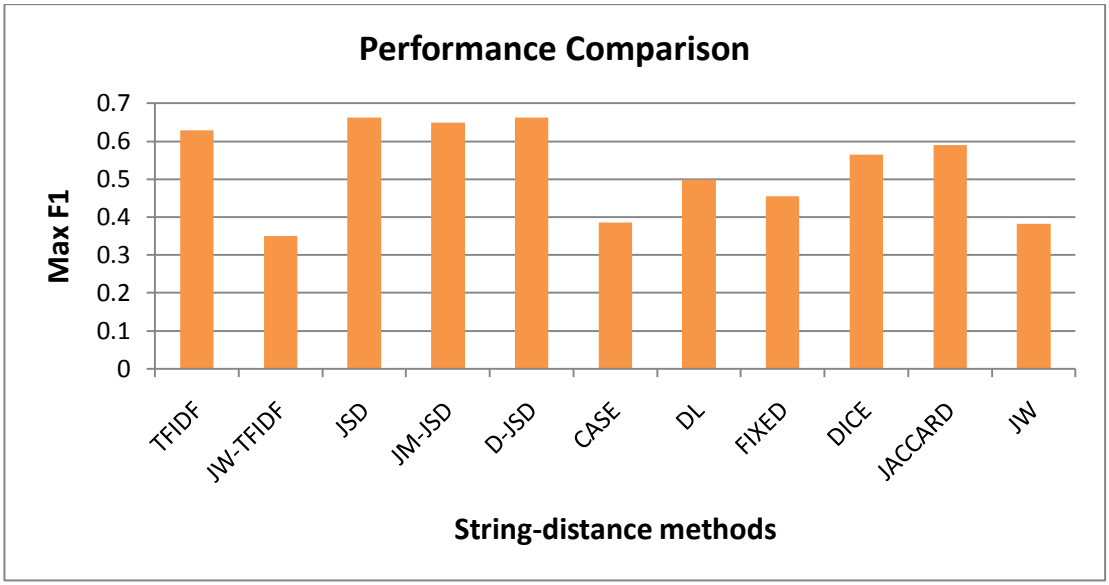


Figure 4-8: Maximum F1 value for string-distance methods

TFIDF and JSD-related methods showed the best performance of all methods under analysis, their maxF1 score reaching values between 0.62-0.67. Although the Jaccard and Dice methods are token-based methods as well, they exhibited lower performances than TFIDF and JSDs techniques (i.e. maxF1 values between 0.55-0.59).The explanation is that, as opposed to TFIDF and JSDs, which use probabilistic approaches in their calculus and base their decision on the frequency of terms (i.e. importance), DICE and JACCARD consider all terms of equal importance. Consequently, DICE and JACCARD methods record a higher number of false positives than in the case of TFIDF and JSD, thus recording a lower F1 score than TFIDF and JSD.

Edit-distance metrics (CASE, DL, FIXED, JW) performed poorer than the other methods, recording maxF1 values between 0.38-0.5. These methods base their decision on the number of operations with characters (i.e. insertion, deletion, substitution or transposition) needed to transform one string into the other. Edit-

distance metrics (CASE, DL, FIXED, JW) seem more suitable for comparing short strings rather than whole paragraphs of text.

The hybrid method (JW-TFIDF) showed the lowest performance out of all methods tested in the string-distance module (i.e. maxF1 value of 0.35). Even though the Jaro-Winkler coefficient used in this implementation (i.e. 0.9) was big enough to detect only small typos, the method wrongly coupled too many similar words. This determined a significant increase in the number of false positives, hence the low performance of the JW-TFIDF technique.

In the case of TFIDF it was observed that the average precision increases with the relaxation of matching rules. For example, the best average precision was obtained when applied on the decision groups [0-0.8] and [0-0.9] (as shown in Figure 4.9). Moreover, the max F1 values were obtained in the same decision intervals.

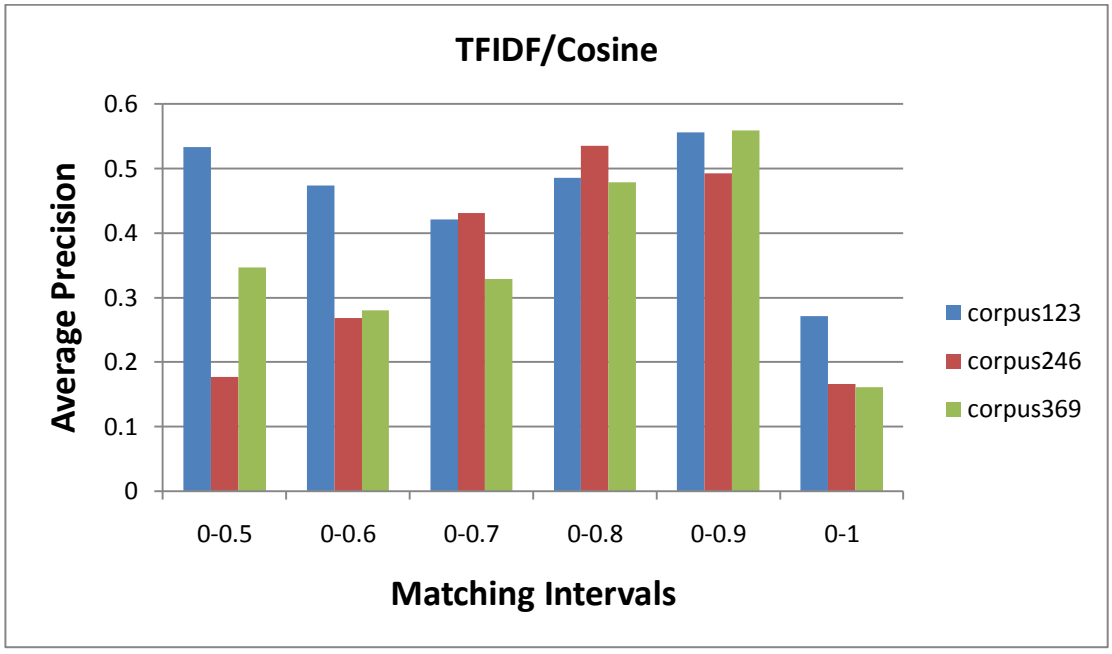


Figure 4-9: Variation within six matching intervals of the average precision of the TFIDF/Cosine method when trained on the three corpora

The figure shows that, when trained on corpus123 or corpus369, TFIDF/Cosine recorded the best precision values (i.e. 0.56-0.57) for the decision group [0-0.9]; when trained on corpus246, the method recorded the best average precision value (0.54) for the decision group [0-0.8]. The analysis showed that for stricter

matching rules (e.g. decision intervals lower than [0-0.6]) TFIDF/Cosine recorded low average precision values (i.e. 0.18-0.35), the only exception being the case when the method was trained on the corpus123, from which the test corpus was created.

The suite of JSD methods achieved the best average performance and the highest F1 score when applied to the [0.3-1] decision group (as shown in Figure 4-10). The test corpus was constructed as a subset from the 123-application description corpus. Training the JSD on the 123-application description corpus showed a slightly better performance of the technique (i.e. average precision in the [0.3-1] decision group: 0.53-0.55) than when trained on the 246-application corpus, which does not contain descriptions found in the test corpus (i.e. average precision in the [0.3-1] decision group: 0.52-0.53). This is explained by the fact that a method trained on a corpus, which already contains the terms and description documents found in the test corpus, gives a more accurate view of the term importance and term probability distributions than when trained on a corpus which does not contain the application descriptions found in the test corpus.

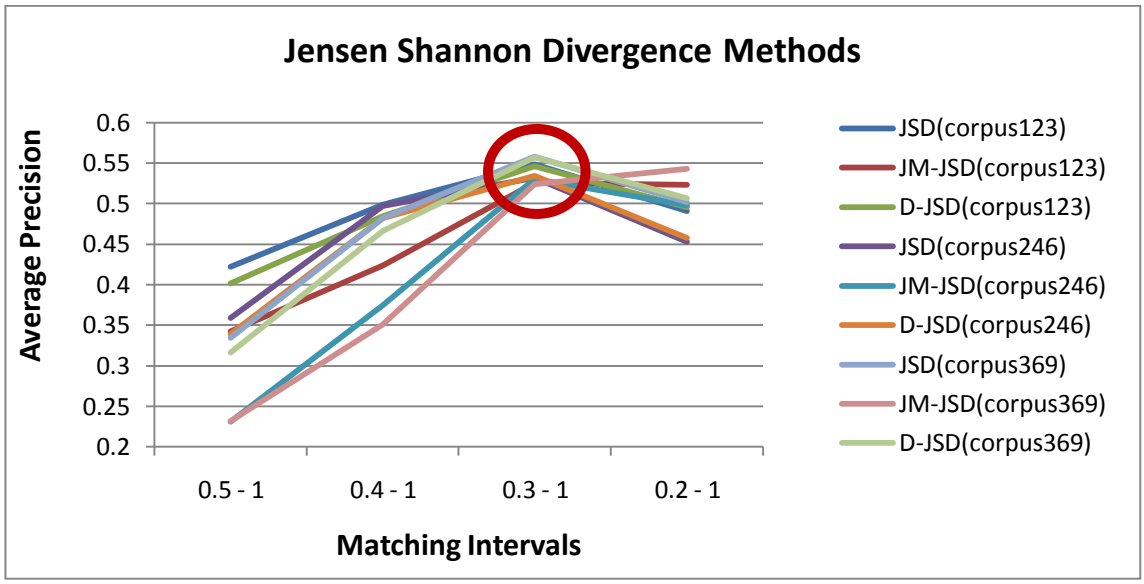


Figure 4-10: Variation within four matching intervals of the average precision of the suite of JSD methods when trained on the three corpora

Moreover, the results showed that training the JSD functions on the 369-application corpus further optimized the accuracy of these methods (i.e. average precision in

the [0.3-1] decision group: 0.53-0.56) by giving a more comprehensive and more precise view of the term probability distributions in the corpus.

The use of entropy-generated stop-list in combination with TFIDF showed no concluding results as the average precision and F1 scores were comparable between the cases where a stop-list was used and the cases where a stop-list was not used (i.e. best results were obtained for entropy threshold values in [0.06-0.1]). However, when applied in conjunction with JSD techniques, the entropy-generated stop-list using thresholds between [0.07-0.12] showed an improvement of both average precision and F1 scores throughout the cases considered in this analysis. An example can be seen in Figure 4.11, where the use of stop-list with such thresholds determined a significant increase in the average precision of JSD methods, notwithstanding what corpus was used for training.

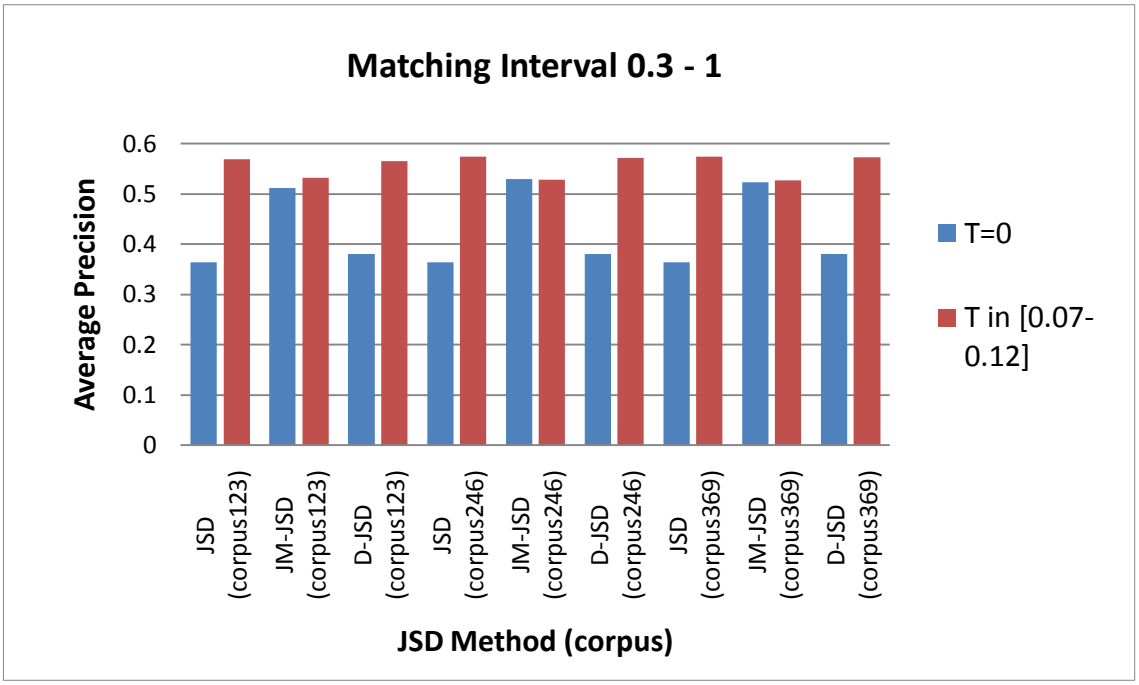


Figure 4-11: Comparison between the average precision of the JSD methods with no stop-list and the same JSD methods using a stop-list with a threshold between [0.07-0.12]

For the decision group [0.3-1] the average precision of JSD and D-JSD methods applied without a stop-list recorded an average precision ranging between [0.37-0.39]. For the same decision group, when using the entropy-generated stop-list with a threshold between [0.07-0.12], the same two methods recorded an overall increase in the average precision, with values ranging between [0.56-0.58]. The

JM-JSD method recorded better values than JSD and D-JSD even without the use of stop-lists (i.e. average precision [0.51-0.52]). Nevertheless, the entropy-generated stop-list with a threshold between [0.07-0.12] improved slightly the average precision of JM-JSD to [0.52-0.53].

* * *

Due to the time constraints of this research, the *application-running* module was not integrated in the GAMRS Matchmaking service. The tests involving this module were done manually using the NGS Grid infrastructure and the Grid user interface exposed on the NGS site [38] via a Java applet called gsiSSH-Term [169].

To that end, I analyzed the Grid applications exposed on the NGS Grid core sites and collected test suites (i.e. set of input files, running script and the set of output files) for 30 applications (out of 38 at the time the analysis has been made). Figure 4-12 shows the list of applications for which a test suite was created.

Each of these applications was run on Grid using the set of input files and the running script available in the test suite. The output was retrieved and compared against the set of output files available in the test suite using the *diff* Linux software [170] – a very common and reliable piece of software, which points out differences between files. For a better understanding of the results, the applications were run several times with the same set of input files and the process of output comparison was repeated after each run.

The conclusion of the tests was that the application-running module can accurately identify identical applications that exhibit deterministic behaviour. Since for the same set of input files, the application always returns the same set of output files, identical deterministic applications are correctly identified by this method. However, the way the comparison of output files was implemented, identical non-deterministic applications could not be identified by the application-running method.

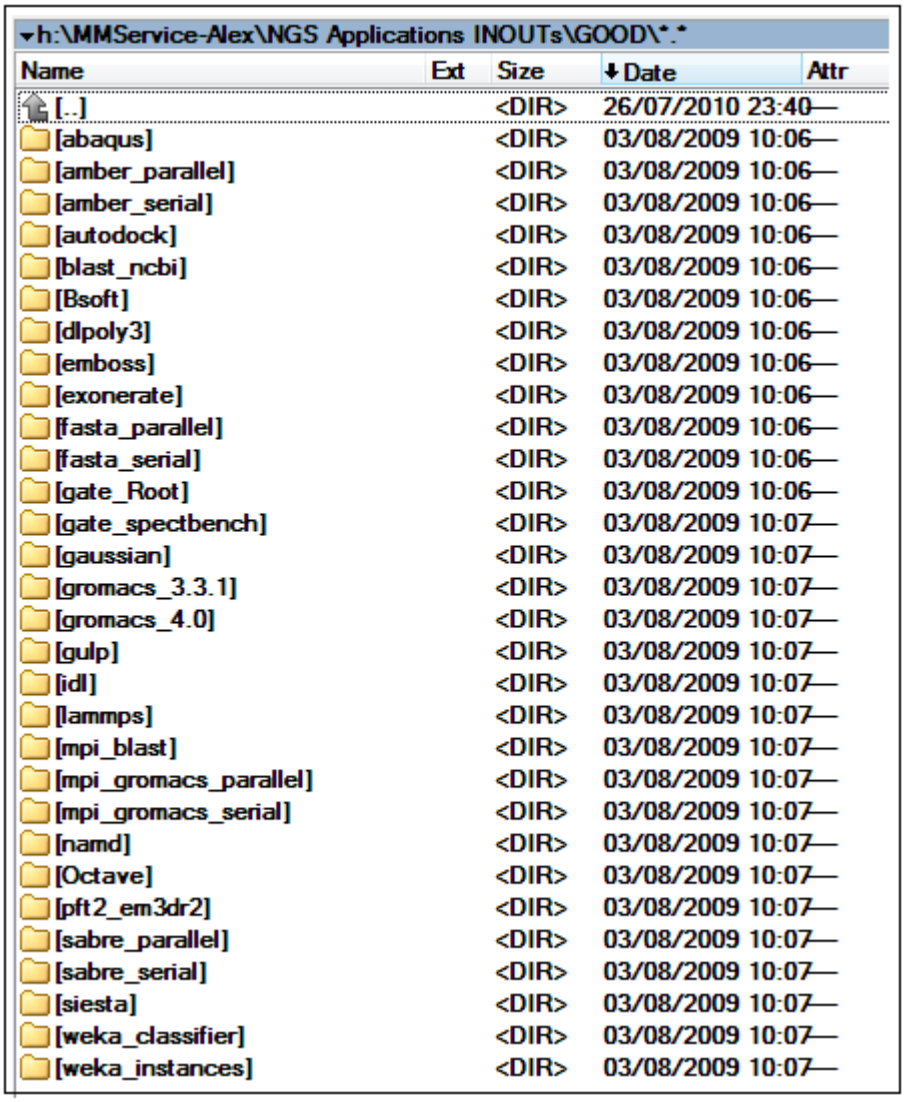


Figure 4-12: Grid applications used for testing the application-running module

* * *

Due to the scarceness of data (i.e. binaries) stored in the few production Grid application repositories capable of storing binaries, the *binary matching* module could not be tested in real-case scenarios. Nevertheless, its fundamental test of hash sums can be explained theoretically and is based on the property of *collision resistance* of hash algorithms (i.e. in our case, what is the probability that two different binaries generate the same hash sum). Tests on binary hash sums generated with *strong collision resistant* algorithms (such as SHA-512, which had no known collisions identified at the time this thesis was written) should identify correctly identical applications. Moreover, even those algorithms for which collisions

have been found (such as SHA-1 or MD5) have such a low probability of collisions (i.e. $p(collision)_{SHA-1} = 10^{-86}$ and $p(collision)_{MD5} = 0.55\text{x}10^{-19}$) [166], that they can be safely used to generate hash sums of Grid application binaries. In cases where such algorithms are used (for example SHA-1) if two binary hash sums are found to be identical, the mathematically-correct decision of the GAMRS syntactic module should be *"The two applications are identical with a probability of* $1 - 10^{-86}$*"*. However, for obvious practical reasons, the syntactic module returns just *"The two applications are identical"*.

.

Chapter 5

Conclusions

All the five scenarios designed in Chapter 4 in order to test the functionality of GAMRS were implemented successfully. The results of this testbed confirm that GAMRS brings relevant contributions to existing solutions in the area of Grid. With the combined capabilities of the three core services – *Publisher, Meta-Repository and Matchmaking*, GAMRS sets the milestone for a new generation of Grid application repositories able to support different distributed computing architectures, while being easily accessible to both human users and services.

The GAMRS architecture shows that Grid technology can be combined with web technologies to provide a wide range of interfaces to make applications easily accessible both to human users and services. As part of SCENARIO 1, the pilot GAMRS solution implemented for this research was able to successfully connect three Grid application repositories (GEMLCA, myExperiment and NGS Application Repository) to the system, making their applications visible and usable through the GAMRS interfaces. SCENARIO 2 showed that GAMRS can also expose applications to the web via these interfaces and can support *Search* operations on metadata associated to objects stored in it.

At the same time, the successful storage and deployment of the BSoft application on GAMRS as part of SCENARIO 3 proved that GAMRS can function as a

repository in its own right, by storing applications and application-related objects and exposing them to the web.

The new repository model is richer than traditional Grid application repository models, and it expands the area of usage of Grid applications to other distributed architectures such as cloud architectures and application-on-demand architectures as demonstrated in SCENARIO 4. Moreover, GAMRS can store Grid application-related objects such as binaries, source code, dependencies, documentation and test files. These objects can be used by Grid administrators for application deployment processes.

In addition to that, the virtual machine-embedded approach allows applications to be run in their native environment making the porting of application on different machine architectures or different operating systems unnecessary. This research implemented a pilot VM-Service and used a VMWare virtualized infrastructure to successfully deploy and run applications that are stored as virtual machine-embedded objects in the GAMRS, thus demonstrating the application's usability not only in Grid but also in cloud computing or other virtualized technologies.

Finally, the successful implementation of SCENARIO 5 showed that the GAMRS Matchmaking service works and is able to find identical or similar applications among different Grid application repositories.

The syntactic module can be successfully used to identify identical applications in the following scenarios: when the applications binary is present and stored in GAMRS; when the application binary hash sum is present in MRDL; when the application test suite is present in GAMRS; and when the full binary path referenced inside the two MRDL documents under comparison are pointing to the same Grid resource and application. Scenarios other than these four failed due to the lack of information present in real-case application descriptions – for example, none of the application description documents found stored in repositories connected to GAMRS contained values in the fields used for description of application resources.

The string-distance module can be used in scenarios where no application-related object needed by other matchmaking methods exists stored in the repository. It comes as a last resort in trying to identify similar applications based only on the free text-description of the application. The module implemented a set of the most popular string-distance methods, which were tested against their ability to identify similar applications stored in Grid repositories. Overall, token-based matchmaking methods performed better than edit-distance methods and the hybrid method. Based on the analysis of test results, it was concluded that TFIDF/Cosine and Jensen-Shannon Divergence methods would be the best candidates for finding similar applications by comparing their free-text descriptions. Moreover, the results showed that the accuracy of those methods can be improved through the selection of particular matching intervals.

The same scenario also tested a new method of entropy-generated stop-list. The method proved to be very successful with Jensen-Shannon Divergence techniques: trimming the documents under comparison of their low-importance terms worked as an optimization method for JSD techniques, making them the most accurate method among the ones tested here.

Results also showed that training on different corpora can influence the accuracy of the matching methods. The analysis suggested that in order to acquire a more precise view of the term probability distributions in Grid application descriptions, it is better to induce training on corpora that include as many descriptions as available. And preferably, the training corpus should include application descriptions from the repositories where the two descriptions under comparison were retrieved from. Moreover, to make the matching more accurate, the two descriptions should first be added to the training corpus, and only then one should induce training and apply the trained matching methods on those two descriptions.

Nevertheless, due to the constraint-free aspect of what can be written in a free-text description of an application, even with the help of entropy-generated stop-list technique, the accuracy of these methods was around 60%. Consequently, my

suggestion is that this method is applied as a last resort, when no other methods (out of those tested by this research) can be used.

The application-running module returned excellent results in matching deterministic applications. With the help of test suites this module manages to successfully identify applications which exhibit a deterministic behaviour. However, this requires the presence of test suites in the repository. Furthermore, due to the limitations of method implemented for the comparison of output sets, the module fails to identify identical applications which exhibit non-deterministic behaviour.

Provided the application binaries are available in GAMRS, the binary matching method should successfully identify identical applications based on their hash sums. However, this method has its limitations, as it cannot identify identical applications which have their binary compiled for different operating systems. Nevertheless, many Grid infrastructures require their Grid sites to use only one or two operating systems on their execution nodes and this increases the chances that the binary matching can be used successfully.

* * *

As mentioned in Chapter 1, Section 1.2 this research started with an extensive critical analysis of the application repository solutions currently used in Grid, which resulted in the identification of a list of shortcomings associated with these solutions. I formally translated these shortcomings in a list of solution requirements related to the Grid application repository conceptual design and its functionality; while, at the same time, I took into consideration the roles such repositories would have on distributed computing frameworks other than Grid. Based on these specified requirements I set out four major research objectives:

- Objective O1: to design a service able to connect different types of Grid application repositories, but which would still function as a Grid application repository in its own right.

- Objective O2: to propose a new model for application repositories, which would achieve uniformity in Grid application presentation and would extend the functionality of these repositories beyond Grid.

- Objective O3: to find (or create) an application description language, which would provide uniformity in the presentation of Grid application descriptions; but would also allow Grid application repositories and the applications stored by them to be used in scenarios other than Grid, such as virtualisation; source code staging and compilation; or automatic application deployment.

- Objective O4: to design a matchmaking methodology and an algorithm able to process information about applications stored in different repositories and identify similar or identical applications.

In order to satisfy the four objectives mentioned above, I designed a novel type of Grid application repository – the Grid Application Meta-Repository System (GAMRS). GAMRS' theoretical design is described in Chapter 3 of this thesis.

Objective O1 was successfully met by creating a new Grid application repository architecture. The following table summarizes GAMRS' architectural features by comparison to the features exposed by other repository solutions.

Table 5-1: GAMRS architectural features vs. other solutions

	APPLICATION PUBLISHING	APPLICATION DISCOVERY	REPOSITORY OBJECT EXCHANGE & REUSE
GAMRS	- Graphical interface for human users; - HTTP/REST interface for services; - OGSI/WSRF Grid Service interface for Grid services.	- Intuitive web interface for human users; - Exposes HTTP/REST interface; - Exposes OGSI/WSRF Grid Service interface; - Supports connections to other repositories; - Exposes a system of identification of similar Grid applications; - Support for OAI-PMH protocol.	- Support for OAI-ORE protocol; - Support for FOXML documents.

BDII	- Publishing done by automated services via scripts that contain suites of console commands; - No graphical/web interface for human users.	- Console commands containing LDAP queries; - No OGSI/WSRF Grid service interface; - No Web visibility; - No HTTP/REST interface; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO
CHARON	- Command-line only for human users; - No access support for services;	- Collection of static Web pages; - No OGSI/WSRF Grid service interface - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO
GEMLCA	- Graphical interface for human users; - OGSI/WSRF Grid service interface for services.	- OGSI/WSRF Grid service interface; - Human users can find application information through PGRADE portals or using a GEMLCA Service Client; - No Web visibility; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO
NGS AR	- Graphical interface for human users - No access support for services	- JSR-168 web application interface – for human users; - No OGSI/WSRF Grid service interface; - No HTTP/REST interface; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO
GRIMOIRES	- Human users and services can register web services via UDDI clients.	- Visible to UDDI clients; - Visible to human users through a collection of static web pages.	N/A

myExperiment	- User-friendly web interface for human users; - HTTP/REST interface for services.	- Intuitive web interface for human users; - Exposes HTTP/REST interface; - No OGSI/WSRF Grid Service interface; - No connection to other repositories; - No system of identification of similar Grid applications; - No support for OAI-PMH protocol.	NO

Objective O2 was successfully met by creating a new Grid application repository model. The following two tables summarize the features of the GAMRS repository model by comparison to those exposed by other repository models.

Table 5-2: GAMRS repository model features vs. other solutions

	myExperiment	**NGS AR**	**GEMLCA**	**GUSE**	**CHARON/ iSoftrepo**	**GAMRS**
User	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
User-related objects	No	no	no	no	no	**YES**
User access policies	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
Application	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
Application access policies	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
Provider	No	no	no	no	no	**YES**
Provider-related objects	No	no	no	no	no	**YES**
Provider access policies	No	no	no	no	no	**YES**

Table 5-3: GAMRS repository model features vs. other solutions (application asset types)

	myExperiment	**NGS AR**	**GEMLCA**	**GUSE**	**CHARON/ iSoftrepo**	**GAMRS**
Description document	**YES**	**YES**	**YES**	**YES**	no	**YES**

Binaries	n/a	reference	**YES**	no	reference	**YES**
Source code	n/a	no	no	no	reference	**YES**
Library dependencies	possible (generic tag)	no	no	no	no	**YES**
Software dependencies	possible (generic tag)	no	no	no	no	**YES**
Documentation	Reference	no	no	no	reference	**YES**
Test files	possible (generic tag)	no	no	no	no	**YES**
VM embedded	No	no	no	no	no	**YES**
Licenses	**YES**	no	no	no	reference	**YES**
Hash sums	No	no	no	no	no	**YES**

Objective O3 was successfully met by extending an existing application description language (i.e. JSDL) with a new set of features which resulted in a novel Grid application description language called MRDL. The following table summarizes the features of MRDL by comparison to the features exposed by other application description languages.

Table 5-4: MRDL features vs. other solutions

	RSL	**JDL**	**xRSL**	**WS-GRAM**	**LCID**	**JSDL**	**MRDL**
Legacy compatibility	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES/ inherited**
Advanced features	partly	partly	partly	partly	partly	**YES**	**YES/ Inherited**
Different submission certificate and staging certificate	no	no	no	**YES (service only)**	no	no	**YES**
Multi-Grid data staging	no	no	no	**YES (service only)**	no	no	**YES**
Hash sums	no	no	no	no	no	no	**YES**

Advanced parameter/attribute descriptions	no	no	no	no	partly	no	**YES**
Multiple transfer protocols supported as URI definitions	no	no	**YES**	no	no	**YES**	**YES/ inherited**
Additional information (licenses, libraries, code for compilation)	no	no	no	no	no	no	**YES**
Application pre-run prerequisites	no	no	no	no	no	no	**YES**
Virtual machine staging	no	no	no	no	no	no	**YES**
Advanced parallel behaviour	partly	partly	partly	partly	partly	**YES**	**YES/ inherited**
Native extension	no	no	no	no	no	**YES**	**YES/ inherited**

Objective O4 was successfully met by creating the first Grid application matchmaking service able to identify similar or identical applications stored in different Grid repositories. For the matchmaking process I successfully identified different sources of information within the application-related objects stored in GAMRS and designed four matchmaking algorithms which are able to process these sources and decide over the similarity or identicalness of two Grid applications. Due to the limited timeframe of this PhD I had to restrict the research to only four matchmaking methods: syntactic, string-distance, binary matching, and application-running. However, the GAMRS Matchmaking service is extendable with other matching modules subject to further research.

The implementation of the pilot GAMRS-solution was subject to a series of constraints outlined in Chapter 4, Section 4.1. These constraints were put in place in order to simplify the development of the solution, but did not restrict the core functionality of GAMRS or its ability to meet the four objectives set out in this research.

The pilot-solution was used in five scenarios, which proved the GAMRS's novel features and showed how this solution met the research objectives. The following table gives an overview of the research objectives, the test scenarios designed to prove these objectives, and the results obtained.

Table 5-5: Degree to which test results met the research objectives

RESEARCH OBJECTIVE	TEST SCENARIOS	RESULT
O1: ARCHITECTURE - to connect to different Grid application repository technologies and solutions and to retrieve the applications stored in them; - to be accessed by OGSI/WSRF Grid Services; - to expose the applications to web via the HTTP/REST interface; - to support Search operations on metadata associated to objects stored in the repository; - to be accessed on WWW by any service equipped with a HTTP client; - to present the repository objects in a format that permits their exchange and reuse on other repositories which are built on technologies similar to that of GAMRS.	The following scenarios were used to test the functionality required by Objective O1: - Scenario 1: Connecting grid application repositories to GAMRS - Scenario 2: Searching in GAMRS - Scenario 3: Storing application-related objects in GAMRS	**OBJECTIVE SUCCESSFULLY MET** - successfully connected three application repositories and retrieved the applications stored in them; - successfully tested the OGSI/WSRF Grid interface; - successfully tested the HTTP/REST interface; - successfully tested the search operation - successfully used the WWW access interface; - successfully tested the OAI-ORE and FOXML formats.

O2: REPOSITORY MODEL		**OBJECTIVE SUCCESSFULLY MET**
- to function as a Grid application repository in its own right and allow users to publish objects inside GAMRS; - to store application-related objects following the categorization designed in the GAMRS repository model; - to be used on distributed infrastructures other than Grid.	The following scenarios were used to test the functionality required by Objective O2: - Scenario 3: Storing application-related objects in GAMRS - Scenario 4: Use GAMRS in distributed infrastructures other than Grid	- Successfully published objects inside GAMRS; - Successfully stored objects following the GAMRS repository model; - Successfully used GAMRS to run a Grid application in a cloud virtualized environment
O3: APPLICATION DESCRIPTION LANGUAGE - to convert application description documents in MRDL for a uniform presentation of applications; - to facilitate the application matchmaking process; - to be used in new scenarios (i.e. deployment, running, testing) previously unavailable in traditional solutions (such as source code staging and compilation, virtual-machine running and testing a correct application deployment and functioning with the help of test suites).	The following scenarios were used to test the functionality required by Objective O3: - Scenario 1: Connecting grid application repositories to GAMRS - Scenario 5: Identify similar or identical Grid applications using GAMRS	**OBJECTIVE SUCCESSFULLY MET** - Successfully converted three different types of ADL (i.e. LCID, Scufl and JSDL) to MRDL; - Successfully used description documents written in MRDL syntactic matchmaker and application running matchmaking module; - Successfully used a MRDL-formatted document to deploy a Grid application by staging the source code, followed by compilation and testing.

O4: APPLICATION MATCHMAKING		OBJECTIVE SUCCESSFULLY* MET
- to identify sources of information for the matchmaking process within the application-related objects stored in repository; - to show that the matchmaking modules proposed in this research are suitable for the identification of similar applications stored in Grid repositories; - to show that the newly-proposed entropy-generated stop-list can improve the accuracy of string-distance methods when applied to matching applications stored in Grid repositories.	The following scenario was used to test the functionality required by Objective O4: - Scenario 5: Identify similar or identical Grid applications using GAMRS	- Successfully identified the sources of information able to help with the matchmaking process (i.e. the application metadata, the application description document, binaries, test suites, the source code, hash sums, dependencies); - Successfully implemented and tested four matchmaking modules able to identify similar Grid applications. Result analysis concluded with several practical suggestions on the benefits and limitations of each module; - Successfully shown that the entropy-generated stop-list improved the accuracy of a whole class of string-distance methods (i.e. JSD) making them the most accurate from all the methods tested in this research, when applied to matching Grid applications

*Objective O4 was successfully met for the four matching modules described in this thesis. Implementation and analysis of other matchmaking techniques which may be suitable for identifying similar or identical applications stored in Grid repositories could be explored in future research.

Chapter 6

Contributions to Knowledge and Extensions

Chapter 5 showed that the four research objectives set out in this thesis were met within certain limits. GAMRS is a service able to connect different types of repositories and expose them through various interfaces to both human users and services. Moreover, GAMRS offers a new repository model, which achieves uniformity in Grid application presentation and extends the functionality of these repositories beyond Grid; and it uses an application description language, which allows for Grid application repositories and the applications stored by them to be used in scenarios other than Grid, such as virtualisation, source code staging and compilation, or automatic application deployment. GAMRS also includes a matchmaking algorithm able to process information about applications stored in Grid repositories and identify similar or identical applications.

This chapter summarizes the contributions brought by this research to the general knowledge of Grid application repositories and concludes with several suggestions on how this research could be extended in the future.

6.1. Contributions to Knowledge

This research makes four contributions to scientific knowledge in the area of Grid application repositories:

CONTRIBUTION 1: Novel architecture

The Grid Application Meta-Repository System proposes an architecture, which allows different Grid application repositories to be connected to the service, independent of their underlying technology. This provides users with access to the applications stored in all connected Grid repositories, and also presents applications in a uniform manner, regardless of the differences between the specific models of each repository. Furthermore, as opposed to the majority of existing Grid application repositories, this solution provides a standard OGSI/WSRF Grid Service interface that allows seamless integration with all other Grid Services, allowing them to access the content of non-OGSI repositories through the GAMRS. At the same time, the GAMRS architecture exposes a HTTP/REST API (Representational State Transfer, Application Programming Interface), which makes applications from all connected repositories visible to search engines on the Web. While until now most applications stored on Grid application repositories were invisible to the Web (mainly due to limitations of repository technologies used in implementation), GAMRS makes the application discovery easier for users – the HTTP/REST API provides a much simpler form of access for non-OGSI services than the Grid Service interface. Furthermore, GAMRS' access interfaces allow for the discovery of Grid application repository objects and associated metadata through OAI-PMH and OAI-ORE protocols, at the same time making these objects exchangeable and reusable between OAI-compliant repositories.

CONTRIBUTION 2: New application repository model

The Grid Application Meta-Repository Service proposes a new application repository model, which allows for inter-operability between various Grid application repositories and Grid services and extends the functionality of these repositories

beyond Grid. The new model helps store together the application and application-related objects, which makes them available to use across different Grid services. The proposed solution includes suggestions on the type of application-related data that repositories should store and also proposes scenarios where this model can be used in distributed computing designs other than Grid. The model gives a comprehensive description of a Grid application and is also able to function as a mediator between older application repository models. With the help of this model Grid applications can be easily deployed and ran on cloud systems. Furthermore, applications can be presented as embedded in virtual machines (VM) and therefore they can be run in their native environments. Also, by exposing applications as virtual machines, both administrators and users can easily deploy these applications on virtualized infrastructures. This procedure requires no prior knowledge of Operating System installation procedures, application installation procedures, the installation of software dependencies, or knowledge on how to configure the application. The new application repository model also contains the necessary description capability to allow users to describe and run commercial applications embedded in virtual machines, provided that a fee-based model is put in place for that.

CONTRIBUTION 3: Improved application description language

This research proposes a solution which enriches the functionality of old application description languages and provides answers to several interoperability problems, such as multi-Grid data staging and hybrid job-submission/data staging certificates. Furthermore, it proposes a life-cycle model for a Grid application describing the different states in which the application can be found (i.e. template, instance, deployment, and running) and sets a standard of how description languages can describe attributes, objects and actions associated with these states. The extensions to JSDL proposed by this research (such as the location of virtual machine-embedded application; the location of licenses, libraries and source code; and the new application running requirements: license acceptance, code compilation and VM-embedded) also allow for cross-operability with new generation technologies such as application-on-demand and virtualized systems.

MRDL, the new application description language obtained by expanding JSDL, provides the necessary description capabilities to refer to the application-related objects described by the GAMRS model. This permits Grid applications to be run using methods not employed on Grid until now, such as virtual machine-embedded or by source code staging and compilation.

CONTRIBUTION 4: Novel Grid application matchmaking service

This research also proposes a matchmaking service aimed at finding similar or identical applications stored in different Grid application repositories by analysing the information included in application description documents and application-related objects. The research identifies and describes several methods of matching Grid applications based on the application-related objects and, in the implementation phase, focuses on four classes of matchmaking techniques: syntactic, string-distance, application binary matching and application running. In conjunction with the application-related objects mentioned under Contribution 2, and with the help of the application description language described in Contribution 3 GAMRS proposes new algorithms for all four methods and analyzes the performance of these algorithms in real-case scenarios using data retrieved from production Grid repositories. Furthermore, GAMRS proposes and analyzes a new method of automatic processing of the training corpus through entropy thresholds for a better performance of the string-distance methods. This new method is not necessarily restricted to Grid applications; it can be used more broadly in natural language processing for the comparison of any two texts. Finally, based on the performance of the matchmaking modules, this research concludes with suggestions regarding which module would be suitable for which scenario and mentions the limitations or non-availability in other scenarios.

6.2. Future Extensions

In this thesis we have shown the benefits of the Grid Application Meta-Repository System as efficient solution to several current challenges in the field of Grid application repositories. However, the dynamics of distributed computing technologies will continue to generate new challenges and requirements in the time to come. Consequently, all four aspects of GAMRS – the architecture, the repository model, the application description language and the matchmaking service - were designed to be modular and extendible so that new features can be added effortlessly in order to permit GAMRS to be integrated easily with future technologies.

The following list contains several suggestions on how GAMRS can be extended in the future. The list is by no means exhaustive; it describes the immediate extensions that can be done from the current state of the pilot solution to enrich GAMRS' capabilities:

ARCHITECTURE:

- The GAMRS Meta-Repository service can be extended with new adapters which would connect other repositories currently on production on Grid, such as BDII, GRIMOIRES, EGEE, EDGeS and CHARON/iSoftrepo.
- The GAMRS architecture is suitable to be cascaded in order to construct federated GAMRS regions. Multiple GAMRS can be connected together through their access interfaces (or each one can reference the others as *Provider* entities in its repository), thus helping users and services to discover and access Grid application stored by any of them.
- GAMRS can be extended with a fee-based model to include commercial applications. GAMRS already stores virtual machine-embedded applications and application licenses. Once such a fee-based model is put in place, users could gain access to applications by accepting the

terms of the license and paying a fee proportionate with the length of time s/he uses the application for.

- GAMRS can be extended with a submission engine able to process all MRDL extensions. This would allow GAMRS' submission engine to submit and run the application as virtual machine-embedded or by automatic source-code compilation. At the same time, the submission engine could be used to automatically process hybrid authentication for the deployment and running phases of the Grid application.

REPOSITORY MODEL:

- The GAMRS repository model can be extended to describe new authentication methods, which started to gain terrain in recent years and which are being considered for adoption on Grid infrastructures, such as Shibboleth or SAML/XACML implementations.

APPLICATION DESCRIPTION LANGUAGE:

- The GAMRS MRDL language schema can be extended to accommodate not only the description of stand-alone applications – as it does here, but also the description of applications presented as Web or Grid Services and applications exposed as workflows on the Grid infrastructure.

APPLICATION MATCHMAKING:

- The string-distance module of the GAMRS Matchmaking service is extendible and further string-distance methods can be added to the ones already implemented, such as Needleman-Wunsch, Witten-Bell, Katz and Knesser-Ney.
- The GAMRS Matchmaking service can be extended with a module, which also implements semantic matchmaking methods (including Latent Semantic Analysis) that can help identify similar applications stored in the Grid repositories connected to GAMRS.
- Another valuable extension of the GAMRS Matchmaking service would

be the design and implementation of an aggregation model able to combine together the scores returned by different matching modules, with the aim of delivering an even more accurate solution to the problem of finding similar applications stored in Grid repositories.

Publications

- Kiss, Tamas and Tudose, Alexandru and Kacsuk, Peter K. and Terstyanszky, Gabor (2007) *SRB data resources in computational grid workflows*. In: Cox, Simon J., (ed.) Proceedings of the UK e-Science All Hands Meeting 2007, Nottingham UK, 10th - 13th September. National e-Science Centre, Edinburgh, pp. 643-650. ISBN 9780955398834
- Kiss, Tamas and Tudose, Alexandru and Terstyanszky, Gabor and Kacsuk, Peter K. and Sipos, Gergely (2008) *Utilizing heterogeneous data sources in computational grid workflows*. In: Danelutto, Marco and Fragopoulou, Paraskevi and Getov, Vladimir, (eds.) Making grids work: Proceedings of the CoreGRID Workshop on Programming Models Grid and P2P System Architecture Grid Systems, Tools and Environments 12-13 June 2007, Heraklion, Crete, Greece. Springer, pp. 225-236. ISBN 9780387784472
- Tudose, Alexandru and Terstyanszky, Gabor and Kacsuk, Peter K. and Winter, Stephen (2010) Grid Application Meta-Repository System: Repository interconnectivity and cross-domain application usage in distributed computing environments. In: Data Driven e-Science: Use Cases and Successful Applications of Distributed Computing Infrastructures (ISGC 2010), Taipei, Taiwan. Springer, pp. 277-292. ISBN 9781441980137.
- Tudose, Alexandru and Terstyanszky, Gabor and Kacsuk, Peter K. and Winter, Stephen (2010) *Using string-distance methods to identify similar applications in Grid repositories*, In: Proceedings of the UK e-Science All Hands Meeting 2010, Cardiff, UK. - *forthcoming*

Bibliography

[1]. Kacsuk P., and Kiss T., *Towards a scientific workflow-oriented computational World Wide Grid*. In: CoreGRID Technical Report, 2007, http://westminsterresearch.wmin.ac.uk/5400/1/Kacsuk_Kiss_2007_final.pdf [accessed on 31/7/2010]

[2]. Foster I., Kesselman C., Nick J., Tuecke S., *The Physiology of the Grid*, 2002, http://www.globus.org/alliance/publications/papers/ogsa.pdf [accessed on 31/7/2010]

[3]. Workflow Management Coalition, *The Workflow reference Model*, Document Number WFMC-TC-1003, Brussels, 1995, http://www.wfmc.org/standards/docs/tc003v11.pdf [accessed on 31/7/2010]

[4]. The Open Grid Services Architecture WG, 2006, https://forge.gridforum.org/projects/ogsa-wg [accessed on 31/7/2010]

[5]. OASIS, *Business Process Execution Language version 2.0 specifications,* http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf [accessed on 31/7/2010]

[6]. IBM, *Business Process Execution Language for Web Services version 1.1*, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf [accessed on 31/7/2010]

[7]. Addis M., Ferris J., Greenwood M., Li P., *Experiences with e-Science workflow specification and enactment in bioinformatics*, In: e-Science All Hands Meeting 2003, 2-4 September 2003, East Midlands Conference Centre, Nottingham. pp. 459-466

[8]. xScufl specifications, http://www.ebi.ac.uk/~tmo/mygrid/XScuflSpecification.html [accessed on 31/7/2010]

[9]. (P-GRADE) Parallel Grid Run-time and Application Development Environment, http://www.p-grade.hu/main.php?m=1 [accessed on

31/7/2010]

[10]. Oinn T., et al., *Taverna: a tool for the composition and enactment of bioinformatics workflows*, In: Bioinformatics 20 (17), 2004, pp. 3045–54. doi:10.1093/bioinformatics/bth361, PMID 15201187

[11]. Hull D., et al., Taverna: a tool for building and running workflows of services, In: Nucleic Acids Res. 34 (Web Server issue): W729–32. doi:10.1093/nar/gkl320, 2006, PMID 16845108

[12]. Ludäscher B., et al., 2006, *Scientific Workflow Management and the Kepler System,* In: Special Issue: Workflow in Grid Systems. Concurrency and Computation: Practice & Experience 18(10), 2006, pp.1039-1065.

[13]. Altintas, I., et al., *Kepler: An Extensible System for Design and Execution of Scientific Workflows*, In: Proceedings of the 16th international Conference on Scientific and Statistical Database Management 2004, SSDBM. IEEE Computer Society, Washington, DC, 423. DOI= http://dx.doi.org/10.1109/SSDBM.2004.44.

[14]. Triana workflow system, http://www.trianacode.org/ [accessed on 31/7/2010]

[15]. WS-PGRADE/gUSE, http://www.guse.hu/?m=architecture&s=0 [accessed on 31/7/2010]

[16]. Tuecke S., et al., *Open Grid Services Infrastructure(OGSI)*, 2003, http://www.globus.org/alliance/publications/papers/Final_OGSI_Specification_V1.0.pdf [accessed on 31/7/2010]

[17]. Grimshaw A., *Grid Services Extend Web Services*, In: SOA Magazine, 2003, http://soa.sys-con.com/node/39829 [accessed on 31/7/2010]

[18]. Stephens T., *Knowledge: The essence of Meta Data: The Repository vs. The Registry*, 2005, http://www.dmreview.com/article_sub.cfm?articleId=1025672 [accessed on 31/7/2010]

[19]. Delaitre T., Goyeneche A., Kacsuk P., Kiss T., Terstyanszky G.Z., Winter S.C., *GEMLCA: grid execution management for legacy code*

architecture design, In: Proceedings for 30th Euromicro Conference, 2004, pp. 477-483, ISBN: 0-7695-2199-1

[20]. Delaitre T., et al., *GEMLCA: Running Legacy Code Applications as Grid Services*, Journal of Grid Computing, Volume 3, no.1-2, 2005, pg. 75-90, ISSN 1570-7873

[21]. UK National Grid Service (NGS) application repository portal, https://portal.ngs.ac.uk/JobProfiles.jsf [accessed on 31/7/2010]

[22]. Anjomshoaa A., et al., *Job Submission Description Language(JSDL)*, 2005, http://www.gridforum.org/documents/GFD.56.pdf [accessed on 31/7/2010]

[23]. Goble C. A., De Roure D. C., *myExperiment: social networking for workflow-using e-scientists*, In: Proceedings of the 2nd Workshop on Workflows in Support of Large-Scale Science, 2007, WORKS '07. ACM, New York, DOI= http://doi.acm.org/10.1145/1273360.1273361

[24]. myExperiment Portal web page, http://www.myexperiment.org [accessed on 31/7/2010]

[25]. Pacini F., *Job Description Language HowTo*, http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2-Document.pdf [accessed on 31/7/2010]

[26]. Pacini F., *JDL Attributes*, http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0_2.pdf [accessed on 31/7/2010]

[27]. RSL, *The Globus Resource Specification Language*, http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html [accessed on 31/7/2010]

[28]. xRSL, *Extended Resource Specification Language*, In: NORDUGRID-MANUAL-4 Reference manual, 2008, http:// www.nordugrid.org/documents/xrsl.pdf [accessed on 31/7/2010]

[29]. Field L., Schulz M. W., *Grid Deployment Experiences: The path to a production quality LDAP based grid information system,* In: Proceedings of the International Conference on Computing in High Energy and Nuclear

Physics (CHEP 2004), 2004.

[30]. Andreozzi S., Burke S., Field L., Konya B., *GLUE Schema Specification version 1.3*, 2004, http://glueschema.forge.cnaf.infn.it/Spec/V13 [accessed on 31/7/2010]

[31]. The OpenLDAP Project, http://www.openldap.org [accessed on 31/7/2010]

[32]. Request For Comments, *RFC 4511: Lightweight Directory Access Protocol (LDAP): The Protocol*, 2006, http://tools.ietf.org/html/rfc4511

[33]. CHARON/iSoftrepo Grid application repository web page, http://meta.cesnet.cz/charonwiki/isoftrepo/site.php?site=voce [accessed on 31/7/2010]

[34]. Kmunicek J., Kulhanek P., Petrek M., *CHARON System - Framework for Applications and Jobs Management in Grid Environment*, Proceedings of CGW05, 2006, http:// egee.cesnet.cz/sources/charon.pdf [accessed on 31/7/2010]

[35]. Kecskemeti G, et.al., Automatic deployment of Interoperable Legacy Code Services, In: Proceedings for UK e-Science All Hands Meeting, 2005, Nottingham, UK

[36]. Java Specification Request, *JSR 168: Portlet Specification* http://www.jcp.org/jsr/detail/168.jsp [accessed on 31/7/2010]

[37]. Lagoze C., et al., *Open Archives Initiative Protocol for Metadata Harvesting*, 2008, http://www.openarchives.org/OAI/openarchivesprotocol.html [accessed on 31/7/2010]

[38]. UK National Grid Service (NGS) web page, http://www.grid-support.ac.uk [accessed on 31/7/2010]

[39]. D Meredith, et.al, A JSDL Application Repository and Artefact Sharing Portal for Heterogeneous Grids and the NGS, In: Proceedings of the UK e-Science All Hands Meeting, 2007, Nottingham, UK

[40]. GRIMIORES Functional Specifications, http://www.omii.ac.uk/docs/3.4.0/GrimoiresDocumentation/funcspec.html

[accessed on 31/7/2010]

[41]. Grid Security Infrastructure, www.globus.org/security/overview.html [accessed on 31/7/2010]

[42]. Organization for the Advancement of Structured Information Standards (OASIS), Open Grid Services Infrastructure – Web Service Resource Framework (OGSI/WSRF), *WS-Resource Specifications version 1.2*, http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf [accessed on 31/7/2010]

[43]. Extensible Markup Language (XML), *Path Language Specifications version 1.0*, http://www.w3.org/TR/xpath [accessed on 31/7/2010]

[44]. Organization for the Advancement of Structured Information Standards (OASIS), Open Grid Services Infrastructure – Web Service Resource Framework (OGSI/WSRF), *WS-Notification Specifications, version 1.3,* http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf [accessed on 31/7/2010]

[45]. Organization for the Advancement of Structured Information Standards (OASIS), Open Grid Services Infrastructure – Web Service Resource Framework (OGSI/WSRF), *WS- Resource Lifetime Specifications, version 1.2*, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf [accessed on 31/7/2010]

[46]. myExperiment wiki page, http://wiki.myexperiment.org [accessed on 31/7/2010]

[47]. Project Trident, *A Scientific Workflow Workbench*, http://research.microsoft.com/en-us/collaboration/tools/trident.aspx [accessed on 31/7/2010]

[48]. Enabling Grid for E-Science (EGEE) Application Database, http://grid.ct.infn.it/egee_applications/index.php?page=1 [accessed on 31/7/2010]

[49]. Enabling Desktop Grid for E-Science application repository,

http://www.edges-grid.eu/web/edges/49 [accessed on 31/7/2010]

[50]. Johnson D., et al., A Middleware Independent GridWorkflow Builder for Scientific Applications, http://eprints.ecs.soton.ac.uk/18539/1/2009_-_18539 [accessed on 31/7/2010]

[51]. The Lattice Project web page, http://lattice.umiacs.umd.edu/gridservices.php [accessed on 31/7/2010]

[52]. Deutsche-Grid (D-GRID) Projects web page, http://www.d-grid.de/index.php?id=41&L=1 [accessed on 31/7/2010]

[53]. Australian Research Collaboration Service (ARCS) web page, http://www.arcs.org.au/products-services/systems-services/grid-services [accessed on 31/7/2010]

[54]. GridSAM submission interface web page, http://www.omii.ac.uk/wiki/GridSAM [accessed on 31/7/2010]

[55]. Global Grid Forum, *GRIDFTP Protocol Description, version 2,* http://www.ogf.org/documents/GFD.47.pdf [accessed on 31/7/2010]

[56]. Rajasekar A., et al., Storage Resource Broker - Managing Distributed Data in a Grid, In: Computer Society of India Journal, Special Issue on SAN, 2003, Vol. 33, No. 4, pp. 42-54

[57]. *Storage Resource Manager Interface Specifications, version 2.0*, https://sdm.lbl.gov/srm-wg/doc/srm.methods.v2.0.doc [accessed on 31/7/2010]

[58]. The GLOBUS Alliance web page, http://www.globus.org [accessed on 31/7/2010]

[59]. Globus toolkit version 2, *Documentation*, http://www.globus.org/toolkit/docs/2.4/ [accessed on 31/7/2010]

[60]. MPI-Forum, *Message Passing Interface (MPI) specifications, version 2.2*, http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf [accessed on 31/7/2010]

[61]. The Advanced Resource Connector (ARC),

http://www.nordugrid.org/middleware/ [accessed on 31/7/2010]

[62]. NorduGrid Projects web page, http://www.nordugrid.org [accessed on 31/7/2010]

[63]. Swegrid Project web page, http://www.snic.vr.se/projects/swegrid [accessed on 31/7/2010]

[64]. KnowARC Project web page, http://www.knowarc.eu [accessed on 31/7/2010]

[65]. Nordic Datagrid Facility (NDGF) Project web page, http://www.ndgf.org/ndgfweb/home.html [accessed on 31/7/2010]

[66]. Web Service Grid Resource Allocation and Management (WS-GRAM), http://www.globus.org/toolkit/docs/4.0/execution/wsgram/ [accessed on 31/7/2010]

[67]. Novotny J., et al., An Online Credential Repository for the Grid: MyProxy, 2001, http://www.globus.org/alliance/publications/papers/myproxy.pdf [accessed on 31/7/2010]

[68]. Request For Comments, *RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, http://www.ietf.org/rfc/rfc2459.txt [accessed on 31/7/2010]

[69]. Dresher M, et al., *JSDL parameter sweep extension*, http://www.ogf.org/documents/GFD.149.pdf [accessed on 31/7/2010]

[70]. Bai, X., *Resource matching and a matchmaking service for an intelligent grid,* In: International Journal of Computational Intelligence 1.3 (2004), pg.197

[71]. String metric, http://en.wikipedia.org/wiki/String_metric [accessed on 31/7/2010]

[72]. Damerau-Levenshtein distance metric, http://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance [accessed on 31/7/2010]

[73]. Smith-Waterman string distance metric, http://en.wikipedia.org/wiki/Smith-

Waterman [accessed on 31/7/2010]

[74]. Jaro Winkler distance metric, http://en.wikipedia.org/wiki/Jaro-Winkler_distance [accessed on 31/7/2010]

[75]. Needleman-Wunsch string distance metric, http://en.wikipedia.org/wiki/Needleman-Wunsch [accessed on 31/7/2010]

[76]. Monge-Elkan string distance metric, http://www.dcs.shef.ac.uk/~sam/stringmetrics.html#monge [accessed on 31/7/2010]

[77]. Term Frequency Inverse Document Frequency (TFIDF), http://en.wikipedia.org/wiki/Tf%E2%80%93idf [accessed on 31/7/2010]

[78]. Jaccard index, Tanimono coefficient, http://en.wikipedia.org/wiki/Jaccard_index [accessed on 31/7/2010]

[79]. Dice's coefficient, http://en.wikipedia.org/wiki/Dice%27s_coefficient [accessed on 31/7/2010]

[80]. Jensen-Shannon Divergence, http://en.wikipedia.org/wiki/Jensen-Shannon_divergence [accessed on 31/7/2010]

[81]. MacCartney B., *NLP Lunch Tutorial: Smoothing*, 2005, http://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf [accessed on 31/7/2010]

[82]. Cohen W., et al., *A Comparison of String Distance Metrics for Name-Matching Tasks*, In: International Joint Conference on Artificial Intelligence, 2003, http://www.cs.cmu.edu/~wcohen/postscript/ijcai-ws-2003.pdf [accessed on 31/7/2010]

[83]. Monge A. E., et al., *The field matching problem: Algorithms and applications*, In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996, pp. 267-270

[84]. Ankolekar A., et al., *Daml-s: Semantic markup for web services*, In: The emerging semantic web, by Isabel F. Cruz, 2002, pp.131-152, ISBN: 1586032550

[85]. Klusch M., Fries B., Sycara K., *Automated semantic web service discovery with OWLS-MX*, In: Proceedings of the Fifth international Joint Conference on Autonomous Agents and Multiagent Systems, 2006, AAMAS '06. ACM, pp. 915-922. DOI= http://doi.acm.org/10.1145/1160633.1160796

[86]. Sridhara G., et al., *Identifying word relations in software: a comparative study of semantic similarity tools*, presentation at MASPLAS '09 Mid-Atlantic Student Workshop on Programming Languages and Systems, (2009), Haverford College, Philadelphia, USA

[87]. Iosup, A., et al., *Inter-operating grids through delegated matchmaking*, In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC '07, pp. 1-12. DOI= http://doi.acm.org/10.1145/1362622.1362640

[88]. Montella R., et al., *An Integrated ClassAd-Latent Semantic Indexing Matchmaking Algorithm for Globus Toolkit Based Computing Grids*, In: Lecture notes in computer science, Volume 4967/2008, pg. 942-950, ISBN 978-3-540-68105-2

[89]. Liu C., et al., *Design and Evaluation of a Resource Selection Framework for Grid Applications*, In: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, 2002, HPDC-11 2002, pg 63-72, ISBN: 0-7695-1686-6

[90]. Condor Manual, http://www.cs.wisc.edu/condor/manual/v6.4/1_2Condor_s_Power.html [accessed on 31/7/2010]

[91]. Condor ClassAds, http://www.cs.wisc.edu/condor/classad [accessed on 31/7/2010]

[92]. Liu C., Yang L., Foster I., Angulo D., *Design and evaluation of a resource selection framework for GRID applications*, In: Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11), 2002, pp.63.

[93]. Hurault A., et al., *Using ontology for resources matchmaking in grid Middleware*, ftp://ftp.irit.fr/IRIT/ACADIE/HaraultAida.pdf [accessed on

31/7/2010]

[94]. Knublauch H., et al., *The protégé owl plugin: An open development environment for semantic web applications*, In: Third International Semantic Web Conference - ISWC 2004, Hiroshima, Japan

[95]. Web Ontology Language (OWL) Overview, http://www.w3.org/TR/owl-features/ [accessed on 31/7/2010]

[96]. Unicore WF as NAREGI-WFML, http://www.unicore.eu/summit/2008/presentations/06_Hoeing_WS-BPEL.pdf [accessed on 31/7/2010]

[97]. Sycara K., et al., *LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace*, In: Autonomous Agents and Multi-Agent Systems, 5, 2002, pp.173–203.

[98]. Sycara K., Lu J., Klusch M., Widoff S., Dynamic Service Matchmaking among Agents in Open Information Environments, In: Journal ACM SIGMOD Record, Special Issue on Semantic Interoperability in Global Information Systems, 1999, pp 47-53. DOI= http://doi.acm.org/10.1145/309844.309895

[99]. Sycara K., Lu J., Klusch M., Widoff S., *Matchmaking Among Heterogeneous Agents in the Internet*, In: Proceedings AAAI Spring Symposium on Intelligent Agents in Cyberspace, 1999, Stanford, USA, http://www.cs.cmu.edu/~softagents/papers/aaai4.pdf, [accessed on 31/7/2010]

[100]. Sycara K., Lu J., Klusch M., *Interoperability among Heterogeneous Software Agents on the Internet*, In: Technical Report CMU-RI-TR-98-22, 1998, CMU Pittsburgh, USA.

[101]. Klusch M, et al., *OWLS-MX: Hybrid OWL-S Service Matchmaking*, http://www-ags.dfki.uni-sb.de/~klusch/papers/owlsmx-aaai.pdf [accessed on 31/7/2010]

[102]. World Wide Web Consortium, *Feature Synopsys for OWL Lite and OWL*, http://www.w3.org/TR/2002/WD-owl-features-20020729/ [accessed on

31/7/2010]

[103]. Larry M. Deschaine L.M., Brice R.S., Nodine M.H., *Use of InfoSleuth to Coordinate Information Acquisition, Tracking and Analysis in Complex Applications*, In: Proceedings of Advanced Simulation Technologies Conference, 2000, http://www.argreenhouse.com/InfoSleuth/publications/astc2000.pdf [accessed on 31/7/2010]

[104]. Fowler J., Perry B., Nodine M., Bargmeyer B., *Agent-Based Semantic Interoperability inInfoSleuth*,In: SIGMOD Record 28:1, 1999, pp. 60-67.

[105]. Nodine M., Fowler J., Perry B., *An Overview of Active Information Gathering in InfoSleuth*, In: Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications, 1999, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.6762&rep=rep1&type=pdf [accessed on 31/7/2010]

[106]. Nodine M., et al., *Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth*, In: Proceedings of the International Conference on Data Engineering, 1999, pp 358-365.

[107]. Nodine M., Perry B., Unruh A., *Experience with the InfoSleuth Agent Architecture*, In: Proceedings of AAAI-98 Workshop on Software Tools for Developing Agents, 1998, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.46.4133&rep=rep1&type=pdf [accessed on 31/7/2010]

[108]. Singh M., et al., *Facilitating Open Communication in Agent Systems*, In: Lecture Notes in AI, v. 1365, Intelligent Agents IV: Agent Theories, Architectures, and Languages, 1998, pp. 281-296.

[109]. Bayardo R., et al., *Semantic Integration of Information in Open and Dynamic Environments*, In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1997, pp. 195-206.

[110]. Arni F., et al., *The deductive database system LDL++*, http://arxiv.org/abs/cs/0202001v1 [accessed on 31/7/2010]

[111]. Martin D., et al., Bringing Semantics to Web Services: The OWL-S Approach, In: First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004) 6-9, 2004, San Diego, California, USA, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.2273&rep=rep1&type=pdf [accessed on 31/7/2010]

[112]. Noy N., Musen M., *PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions*, In: Proceedings in Eighteenth National Conference on Artificial Intelligence, 2002, pp 744-750

[113]. Pallis G., et al., *Effective Keyword Search for Software Resources Installed in Large-Scale Grid Infrastructures,* In: IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, 2009, pp 482-489, ISBN 978-0-7695-3801-3

[114]. Indumathi D., Chitra A., *Software Agent Based Search Engine Using Grid Technology*, In: Academic Open Internet Journal, v.16, 2005, http://www.acadjournal.com/2005/v16/part6/p8 [accessed on 31/7/2010]

[115]. Fielding R. T., Taylor R. N., Principled Design of the Modern Web Architecture, ACM Transactions on Internet Technology (TOIT) 2 , pp 115–150, ISSN 1533-5399

[116]. Fielding R. T., Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, 2000, University of California, Irvine

[117]. AUTODOCK application web page, http://autodock.scripps.edu [accessed on 31/7/2010]

[118]. Open Archives Initiative web page, http://www.openarchives.org [accessed on 31/7/2010]

[119]. Open Archive Initiative – Object Reuse and Exchange, *OAI-ORE specifications and user guide*, *version 1.0*, http://www.openarchives.org/ore/1.0/toc [accessed on 31/7/2010]

[120]. Fedora Commons repository framework web page, http://www.fedora-commons.org [accessed on 31/7/2010]

[121]. ePrints repository framework web page, http://www.eprints.org [accessed on 31/7/2010]

[122]. dSpace Repository framework web page, http://www.dspace.org [accessed on 31/7/2010]

[123]. Maslov A., *Adding OAI-ORE Support to Repository Platforms,* In: The 4th International Conference on Open Repositories, 2009, Atlanta, Georgia, http://journals.tdl.org/jodi/article/download/749/640 [accessed on 31/7/2010]

[124]. Fedora Commons FOXML language, *Introduction to Fedora Object XML*, http://www.fedora-commons.org/documentation/3.0b1/userdocs/digitalobjects/introFOXML.html [accessed on 31/7/2010]

[125]. GridWay web page, http://www.gridway.org/doku.php?id=start [accessed on 31/7/2010]

[126]. Japanse National Research Grid Initiative (NAREGI) web page, http://www.naregi.org/index_e.html [accessed on 31/7/2010]

[127]. Service Oriented Collaborations for Industry and Commerce (GRIA) web page, http://www.gria.org [accessed on 31/7/2010]

[128]. Genesis II Project web page, http://www.cs.virginia.edu/~vcgr/wiki/index.php/The_Genesis_II_Project [accessed on 31/7/2010]

[129]. EGEE Project web page, http://www.eu-egee.org [accessed on 31/7/2010]

[130]. SEE-Grid Project web page, http://www.see-grid.org [accessed on 31/7/2010]

[131]. EELA Grid Project web page, http://www.eu-eela.eu/index.php?option=com_content&task=view&id=26&Itemid=50 [accessed on 31/7/2010]

[132]. EUMedGrid Project web page, http://www.eumedgrid.eu [accessed on 31/7/2010]

[133]. EU-India Grid Project web page, http://www.euindiagrid.eu [accessed on

31/7/2010]

[134]. EUChinaGrid Project web page, http://www.euchinagrid.org [accessed on 31/7/2010]

[135]. Baltic-Grid II Project web page, http://www.balticgrid.org [accessed on 31/7/2010]

[136]. TRIPLE: subject–predicate–object expression in Resource Description Framework - *Concepts and Abstract Syntax*, http://www.w3.org/TR/rdf-concepts [accessed on 31/7/2010]

[137]. Sycara K., et al., *Automated discovery, interaction and composition of Semantic Web services*, In: Journal of Web Semantics, Vol. 1, No. 1, 2003, pp. 27-46

[138]. Li L., Horrocks I., *A software framework for matchmaking based on semantic web technology*, In: Proceedings of the 12th international Conference on World Wide Web, 2003, WWW '03, pp 331-339, DOI= http://doi.acm.org/10.1145/775152.775199

[139]. Request For Comments, *RFC 2616: Hypertext Transfer Protocol (HTTP) 1.1*, http://www.ietf.org/rfc/rfc2616.txt [accessed on 31/7/2010]

[140]. Tarrant D., et al., *Using OAI-ORE to Transform Digital Repositories into Interoperable Storage and Services Applications*, In: Code{4}lib Journal, v. 6, 2009, ISSN 1940-5758, http://journal.code4lib.org/articles/1062 [accessed on 31/7/2010]

[141]. Virtual Organization Membership Service (VOMS), http://edg-wp2.web.cern.ch/edg-wp2/security/voms/voms.html [accessed on 31/7/2010]

[142]. Deterministic algorithm, http://en.wikipedia.org/wiki/Deterministic_algorithm [accessed on 31/7/2010]

[143]. Oracle White Paper, *Oracle Application Server 10g – Grid Computing*, Oracle Corporation, 2005, http://www.oracle.com/technology/tech/grid/collateral/OracleAS10g_gcwp.p

df [accessed on 31/7/2010]

[144]. Kusnetzky D., Olofson C., *Oracle 10g: Putting Grids to Work*, Oracle Corporation, 2004, http://www.oracle.com/technology/tech/grid/collateral/idc_oracle10g.pdf [accessed on 31/7/2010]

[145]. McKee B., et al, Introducing IBM WebSphere Service Registry and Repository, Part1: A day in the life of WebSphere Service Registry and Repository in the SOA life cycle, 2006,http://www.ibm.com/developerworks/websphere/library/techarticles/0609_mckee/0609_mckee.html [accessed on 31/7/2010]

[146]. McKee B. et al, *Introducing IBM WebSphere Service Registry and Repository, Part2: Architecture, APIs, and content*, 2006, http://www.ibm.com/developerworks/websphere/library/techarticles/0609_mckee2/0609_mckee2.html [accessed on 31/7/2010]

[147]. Global Grid Forum, Application Contents Service (ACS) Specification 1.0, May 2006, http://www.ogf.org/documents/GFD.73.pdf [accessed on 31/7/2010]

[148]. Aloisio G., et al., *WebGRelC: Towards Ubiquitous Grid Data Management Services*, 2006, http://www.cogkit.org/GCE06/papers/CameraReady_116.pdf [accessed on 31/7/2010]

[149]. Java Commodity Kit web page, http://www.globus.org.cog.java [accessed on 31/7/2010]

[150]. Islandora/ Drupal – Fedora commons modules, http://vre.upei.ca/dev/islandora [accessed on 31/7/2010]

[151]. gSearch web page, http://www.fedora.info/download/2.2/services/genericsearch/genericsearch-1.1.zip [accessed on 31/7/2010]

[152]. gSearch installation notes, http://www.fedora-

commons.org/download/2.2/services/genericsearch/doc/index.html [accessed on 31/7/2010]

[153]. Fedora commons OAI-PMH provider, http://downloads.sourceforge.net/fedora-commons/oaiprovider-1.2.zip [accessed on 31/7/2010]

[154]. Fedora commons OAI-PMH provider installation notes, https://wiki.duraspace.org/display/FCSVCS/OAI+Provider+Configuration+Reference [accessed on 31/7/2010]

[155]. Dublin Core Metadata initiative, *Dublin Core Metadata Element Set, version 1.1*, http://dublincore.org/documents/dces [accessed on 31/7/2010]

[156]. Fedora commons OAI-ORE provider, http://sourceforge.net/projects/oreprovider [accessed on 31/7/2010]

[157]. Fedora commons OAI-ORE provider installation notes, http://oreprovider.sourceforge.net/examples.html [accessed on 31/7/2010]

[158]. ORE User Guide – Resource Map implementation in Atom, http://www.openarchives.org/ore/1.0/atom [accessed on 31/7/2010]

[159]. Apache HTTP server web page, http://httpd.apache.org [accessed on 31/7/2010]

[160]. NGS Certification Authority web page, https://ca.grid-support.ac.uk/cgi-bin/pub/pki?cmd=getStaticPage&name=index [accessed on 31/7/2010]

[161]. Eclipse Modeling Framework web page, http://www.eclipse.org/modeling/emf [accessed on 31/7/2010]

[162]. Bernard's Software Package (BSoft) web page, http://lsbr.niams.nih.gov/bsoft [accessed on 31/7/2010]

[163]. Virtual Network Computing System, http://en.wikipedia.org/wiki/Virtual_Network_Computing [accessed on 31/7/2010]

[164]. VMWare ESXi hypervisor web page, http://www.vmware.com/products/esxi/ [accessed on 31/7/2010]

[165]. GlassFish application container version 3, https://glassfish.dev.java.net [accessed on 31/7/2010]

[166]. Access Data, White Paper, *MD5 collisions – The effect on computer forensics*, http://www.accessdata.com/media/en_us/print/papers/wp.md5_collisions.en_us.pdf [accessed on 31/7/2010]

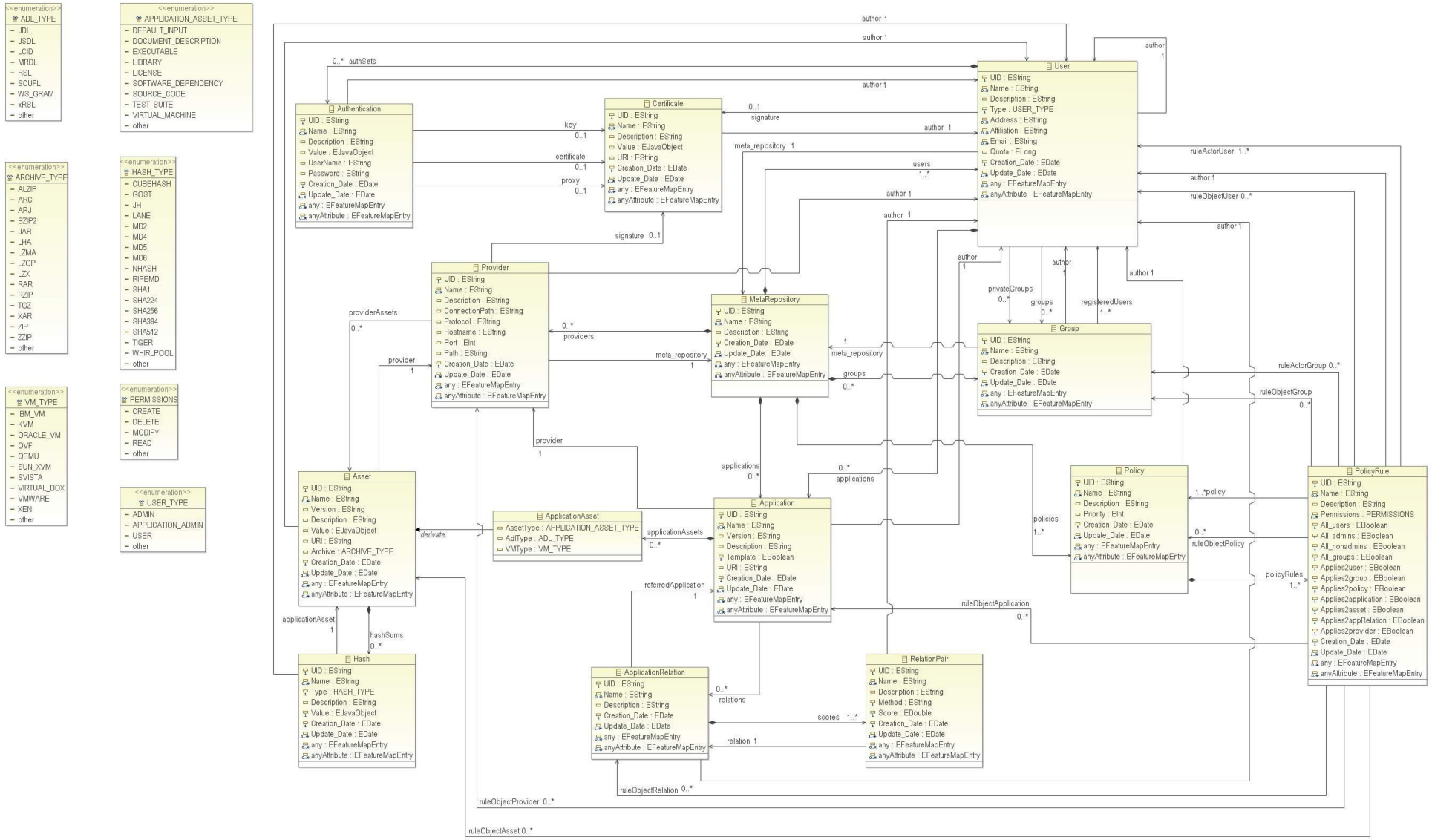
[167]. LingPIPE, *String Comparison Tutorial*, http://alias-i.com/lingpipe/demos/tutorial/stringCompare/read-me.html [accessed on 31/7/2010]

[168]. Stringmetrics web page, http://www.dcs.shef.ac.uk/~sam/stringmetrics.html [accessed on 31/7/2010]

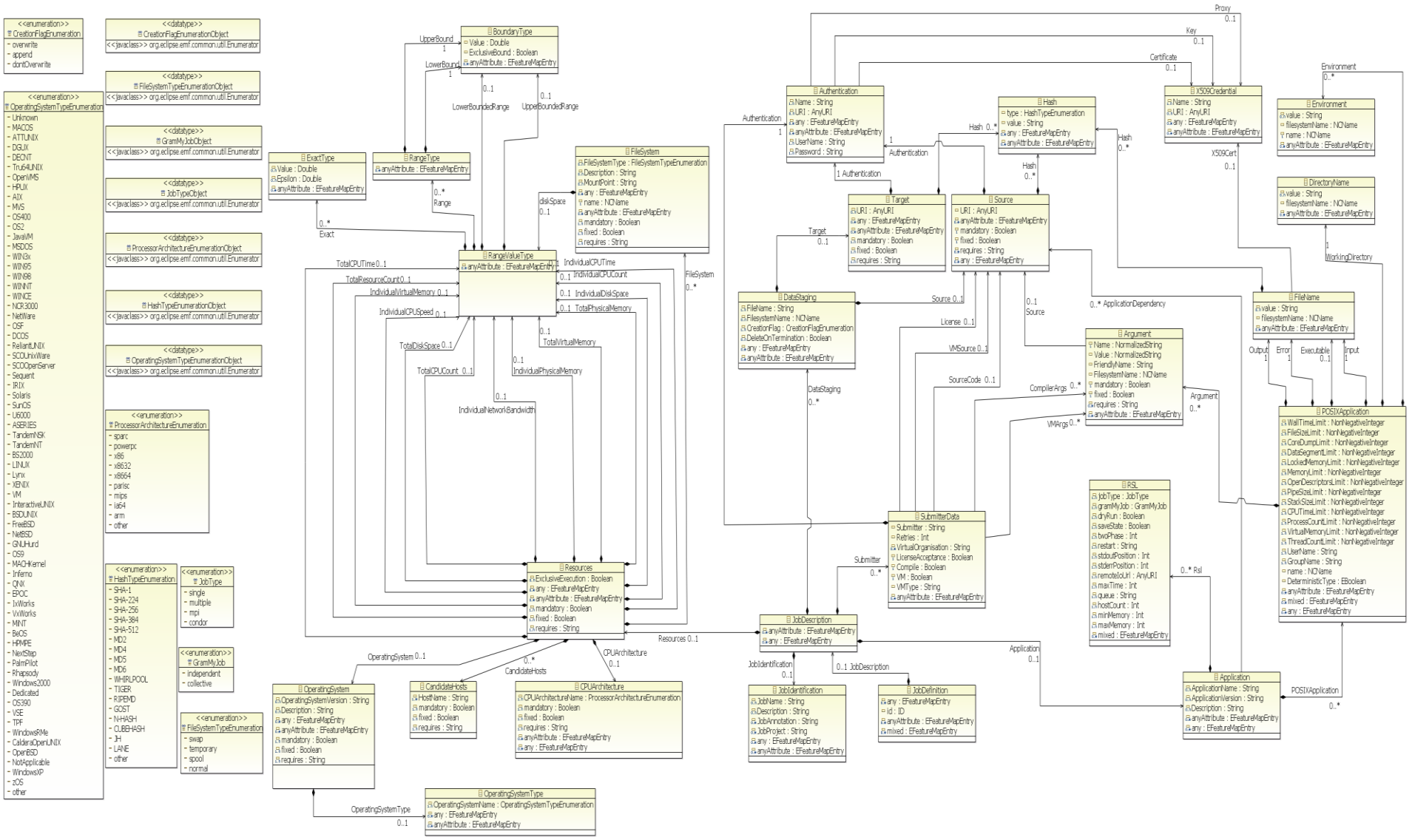
[169]. GsiSSH-Term web page, http://www.ngs.ac.uk/tools/gsisshterm [accessed on 31/7/2010]

[170]. Linux diff online manual, http://linux.about.com/library/cmd/blcmdl1_diff.htm [accessed on 31/7/2010]

Appendix A: GAMRS Repository Model



Appendix B: MRDL Application Description Language



Appendix C: String-distance Methods tested in GAMRS

The **Damerau-Levenshtein** distance metric compares two finite sequences of symbols (i.e. strings) and returns the distance between them by counting the minimum number of operations needed to transform one string into the other. An operation is defined as an insertion, deletion, substitution or transposition of two characters. The lower the Damerau-Levenshtein score is, the more similar the strings are, with '0' meaning the strings are identical.

The **Case** distance metric is a variation of Damerau-Levenshtein and computes the similarity score of two strings in the same fashion. The difference is that the Case metric is case-insensitive (i.e. makes no difference between uppercase and lowercase letters) and it does not count punctuation signs in its similarity score. Like the Damerau-Levenshtein score, the lower the Case distance is, the more similar the strings are.

The **Fixed Weight** string metric is another variation of the Damerau-Levenshtein distance. While Damerau-Levenshtein makes no distinction between the importance of each operation, Fixed Weight metric allows the assignment of different weights to the four operations. In the tests done as part of this research it was used an implementation of Fixed Weight with the weights suggested for English text in [167]:

$$w_{insertion} = 0.8\,;\ w_{deletion} = 1.0\quad w_{substitution} = 0.9 \text{ and } w_{transposition} = 0.9$$

The Fixed Weight score is the same as in the methods discussed above: the lower the distance, the more similar the strings.

The **Jaro-Winkler** similarity distance is also based on character matching. For two strings s_1,s_2, the Jaro distance is defined as:

$$j_{s_1,s_2} = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) \quad (1)$$

where m is the number of matching characters; t is the number of transpositions needed to obtain the same sequence of matching characters in both strings; and |s| denotes the number of characters in string s.

Winkler modified the Jaro distance so that the new distance gives better scores to strings that start identically for a prefix of length L. Mathematically, the Jaro-Winkler distance is defined in relation to the Jaro distance:

$$\mathrm{jw}_{s_1,s_2} = \mathrm{j}_{s_1,s_2} + pL(1 - \mathrm{j}_{s_1,s_2}) \quad (2)$$

where j_{S_1,S_2} is the Jaro distance for strings s_1,s_2; L is the length of the common prefix at the start of the string (with a threshold value of 4 characters; L ≤ 4) and p is a constant used to adjust the score upwards for having common prefixes. The standard value for this constant is 0.1. The higher the Jaro-Winkler score is, the more similar the strings are. A score of 0 corresponds to no similarity, while 1 equates to an exact match.

The **TFIDF** (term frequency, inverse document frequency) is a weight widely-used in natural language processing and information retrieval, which gives a measure of how important a word (i.e. token or term) is to a document in a collection. TFIDF relates the importance of the word with the number of occurrences of that word both in the document under analysis and in the entire collection of documents (i.e. "corpus" in specialized literature). TFIDF specifies that: "The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus."[18]

For the mathematical representation of TFDIF we have used the following notations:

t = term; basic entity in token-based analysis; the equivalent of a word in natural language

d = document; a set of terms; in our case, a paragraph of text written in English, which represents a free-text description of a Grid application

C = *corpus*; a collection of documents

$c(t)_d$ = the number of occurrences of the term t in document d

$c(t)_C$ =
the number of occurrences of the term t in the collection of documents C

$|d|_t = cardinality\ of\ d\ in\ relation\ to\ t =$ the number of terms in document d

$|C|_d = cardinality\ of\ C\ in\ relation\ to\ d =$ the number of documents in corpus C

$|C|_t = cardinality\ of\ C\ in\ relation\ to\ t =$ the number of terms in corpus C

$|C|_d^t = cardinality\ of\ C\ in\ relation\ to\ d\ and\ t$ = the number of documents in corpus C which contain the term t

Given these, the *term frequency* of the term t in document d is the ratio between the number of occurrences of the term t in document d and the total number of terms contained in document d.

$$tf(t)_d = \frac{c(t)_d}{|d|_t} \quad (3)$$

The *term frequency* of the term t in a corpus C is the ratio between the number of occurrences of the term t in all documents in corpus C and the overall number of terms contained in all documents in corpus C.

$$tf(t)_C = \frac{c(t)_C}{|C|_t} \quad (4)$$

The *inverse document frequency* of a term t used in documents contained in a corpus C is a measure of the general importance of the term t in corpus C and is obtained by dividing the total number of documents in corpus C by the number of documents in C which contain the term t, and then taking the logarithm of that quotient.

$$idf(t)_C = log\frac{|C|_d}{|C|_d^t} \quad (5)$$

The mathematical formula of TFIDF can now be obtained by combining formulas (3) and (5):

$$tfidf(t)_d^C = \frac{c(t)_d}{|d|_t} log \frac{|C|_d}{|C|_d^t} \quad (6)$$

This is the most widely-used formula for TFIDF and we also used this mathematical expression for the analysis of the paragraphs of free-text descriptions of Grid applications. However, mathematical variants such as (7) and (8) can also be encountered in the literature, but they are rare.

$$tfidf(t)_d^C = \frac{c(t)_d}{|d|_t} log \frac{|C|_d - |C|_d^t}{|C|_d^t} \quad (7)$$

$$tfidf(t)_d^C = \frac{c(t)_d}{|d|_t} log \frac{|C|_d}{1+|C|_d^t} \quad (8)$$

As mentioned before, TFIDF is not a distance but a weight. Subsequently, from each document one can construct a vector of TFIDF weights relative to the frequency of the terms in the document. A common way to compare two vectors is to measure the cosine value of the angle between these vectors. Mathematically, if A and B are vectors over a vector space $\cup$ (i.e. in our case R^n), the cosine of the angle between A and B is calculated as follows:

$$\cos\theta = \frac{A \bullet B}{|A||B|} \quad (9)$$

where $A \bullet B$ represents the *dot product* between vectors A and B, and |x| represents the length of the vector.

On R^n, the intuitive notion of the length of vector $x = [x_1, x_2, \ldots, x_n]$ is captured by the following formula:

$$|X| = \sqrt{{x_1}^2 + {x_2}^2 + \ldots + {x_n}^2} \quad (10)$$

The **TFIDF/Cosine distance** between two documents d_1 and d_2, (both elements of the same collection C) is defined as the cosine value of the angle between their vectors of TFIDF weights. Mathematically, using formulae (6), (9) and (10) the TFIDF/Cosine distance is calculated as:

$$cossim(d_1, d_2)_C = \frac{\sum_i tfidf(t_i)^C_{d_1}\ tfidf(t_i)^C_{d_2}}{\sqrt{\sum_i (tfidf(t_i)^C_{d_1})^2}\sqrt{\sum_i (tfidf(t_i)^C_{d_2})^2}} \quad (11)$$

An important consequence of using formula (6) for TFIDF is that its values will always be positive. Consequently, the cosine similarity of two documents will range from 0 to 1, with 1 meaning the same vectors of term frequencies, hence very similar documents.

The hybrid similarity distance **Jaro-Winkler/TFIDF** uses the same logic as the TFIDF/Cosine method, except it tries to eliminate small typos that may occur in free texts. Specifically, for the first stage, it uses the Jaro-Winkler distance to compare the terms contained in two documents and suggests that pairs of terms with very high similarity scores (i.e. usually over 0.9) are considered as just one term. In a second stage, this method computes the TFIDF vectors and uses the *cossim* formula (11) to obtain the similarity score of the two documents. Essentially, this method tries to eliminate the cases when a word is misspelled and, in particular, the case where the error consists of just two consecutive characters being swapped – which would require just one transposition of those characters to make the word correct again.

Example

For example, the word *Grid* can be incorrectly written as *Gird*. In this case, using formula (1) with $m = |s_1| = |s_2| = 4$ and $t = 1$, it gives a Jaro distance $j_{\text{Grid,Gird}} \approx 0.9167$. Implicitly, using the formula (2) with $p = 0.1$ and $L = 1$ it gives a Jaro-Winkler score of $jw_{\text{Grid,Gird}} \approx 0.9167 + 0.1 \times 1 \times (1 - 0.9167) \approx 0.925$.

Because $jw_{\text{Grid,Gird}} \geq 0.9$, this solution suggest that, for any two documents contained within the corpus under analysis, instead of regarding *Grid* and *Gird* as two

different terms, they should be regarded as one and the same term and the frequency vectors should be constructed accordingly.

The **Jensen–Shannon Divergence** (JSD – also known in literature as information radius or total divergence to the average) is a widespread method of measuring the similarity between two or more probability distributions.

The general definition of the Jensen-Shannon divergence, which allows for the comparison of two or more distributions, is:

$$JSD(P_1, \dots, P_n) = H(\sum_{i=1}^{n} \alpha_i P_i) - \sum_{i=1}^{n} \alpha_i H(P_i) \quad (12)$$

where $\{\alpha_i: i=1,2,\dots,n\}$ are the weights for the probability distributions $\{P_i: i=1,2,\dots,n\}$ and $H(P_i)$ is the Shannon entropy for distribution P_i. However, in the majority of cases formula (12) is used to compare only two probability distributions at one time and the weights associated with them are selected as $\alpha_1 = \alpha_2 = 1/2$; hence,

$$JSD(P_1, P_2) = H\left(\frac{P_1+P_2}{2}\right) - \frac{H(P_1)+H(P_2)}{2} \quad (13)$$

The Jensen-Shannon divergence relates to the concept of information entropy and the most popular entropy used in computer science is Shannon's entropy. The concept of Shannon's entropy relies on the concept of uncertainty:

For a random variable *X* with *n* outcomes $\{x_i: i = 1,2, \dots, n\}$ each with a probability of happening $p(x_i)$, the *uncertainty* associated with each outcome is defined as:

$$u(x_i) = log \frac{1}{p(x_i)} = -\log\left(p(x_i)\right) \quad (14)$$

This definition captures the following idea: "the lower the probability $p(x_i)$ of an event to happen, the higher the uncertainty $u(x_i)$ associated with that event". The logarithm is used to provide the additivity characteristic for independent uncertainty (i.e. $\log(ab) = \log(a) + \log(b)$). The average uncertainty associated with the random variable *X* is defined as:

$$AVGU(X) = \sum_{i=1}^{n} p(x_i)u(x_i) = -\sum_{i=1}^{n} p(x_i)\log\left(p(x_i)\right) \quad (15)$$

For a random variable X with n outcomes $\{x_i: i=1,2,...,n\}$, each with a probability of happening $p(x_i)$, Shannon's entropy $H(X)$ is defined as "the average uncertainty associated with the random variable *X".* Hence, formula (15) gives the mathematical representation of the Shannon's entropy:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log\left(p(x_i)\right) \quad (16)$$

This formula is used in natural language processing and information retrieval and is the one we used in our research for the comparison of Grid application descriptions. In our case the free-text paragraph that represents the description of a Grid application can be seen as a probability distribution of a set of terms $\{t_i: i=1,2,...,n\}$ (i.e. words), each having a probability of occurrence $p(t_i)$. Therefore, a solution to find similarities between two documents/paragraphs is to use the Jensen-Shannon divergence and find out to what extent the probability distributions of their terms differ from the average. If these probability distributions are close to the average, it is highly likely that the two documents describe similar applications.

In our research we have analyzed three types of methods that use the Jensen-Shannon divergence. These methods differ among each other in terms of the probability function they use to calculate the occurrence of a term in a document.

The first one, the **classic Jensen-Shannon divergence** – or simply *Jensen-Shannon divergence* – uses the term frequency (formula 3) as $p(t_i)_d$ – probability of a term t_i to occur in a document d:

$$p(t_i)_d = tf(t_i)_d = \frac{c(t_i)_d}{|d|_{t_i}} \quad (17)$$

The second and third Jensen-Shannon methods are methods of interpolation, which combine the importance of the term in the document with the importance of the term in the whole corpus.

The **Jelinek-Mercer/Jensen-Shannon** method uses the following technique of interpolating the term frequency in a document with the term frequency in the entire corpus:

$$p(t_i)_d^C = (1 - \lambda) tf(t_i)_d + \lambda\, tf(t_i)_C \quad (18)$$

For natural language processing, usually the value of λ is 1/2, which gives us the formula used by the Jelinek-Mercer/Jensen-Shannon method: Let *C* be a collection of documents $\{d_j: j=1,2,...,n\}$. The probability of a term $\{t_i: i=1,2,...,m\}$, to occur in d_j is given by the formula:

$$p(t_i)_{d_j}^C = \frac{1}{2} tf(t_i)_{d_j} + \frac{1}{2} tf(t_i)_C \quad (19)$$

Formula (19) shows that instead of focusing just on the importance of the term in the document (i.e. $tf(t_i)_{dj}$), the Jelenik-Mercer method combines it evenly (i.e. λ = 1/2) with the importance of the term in the whole corpus (i.e. $tf(t_i)_C$).

The **Dirichlet/Jensen-Shannon** method uses a different technique of interpolation between the term frequency in a document and the term frequency in the entire collection: Let C be a collection of documents $\{d_j: j=1,2,...,n\}$. The probability of a term $\{t_i: i=1,2,...,m\}$, to occur in d_j is given by the formula:

$$p(t_i)_{d_j}^C = \frac{c(t_i)_{d_j} + \mu tf(t_i)_C}{\mu + |d_j|_t} \quad (20)$$

where μ is a scaling factor. Usually, for natural language processing μ = 1. Thus, formula (20) becomes:

$$p(t_i)_{d_j}^C = \frac{c(t_i)_{d_j} + tf(t_i)_C}{1 + |d_j|_t} = \frac{c(t_i)_{d_j}}{1 + |d_j|_t} + \frac{tf(t_i)_C}{1 + |d_j|_t} \quad (21)$$

In most of the cases encountered in natural language processing $(1 + |d_j|_t)$ can be approximated with $|d_j|_t$ (i.e. the number of terms contained in a document is big enough so that a small variation of (+1) cannot change the result significantly). Hence, the Dirichlet/Jensen-Shannon method uses the following formula:

$$p(t_i)_{d_j}^C \cong tf(t_i)_{d_j} + \frac{tf(t_i)_C}{|d_j|_t} \quad (22)$$

Formula (22) shows that instead of focusing only on the importance of the term in the document (i.e. $tf(t_i)_{dj}$), the Dirichlet method combines it with the importance of the term in the whole corpus (i.e. $tf(t_i)_C$), scaled by the inverse number of the terms contained in the document (i.e. $1/|d_j|_t$).

Formulae (17), (19) and (22) are the most widespread forms of term occurrence probabilities that can be encountered in Jensen-Shannon divergence methods used in natural language processing. Consequently, we have used these forms for the analysis of the free-text part of application description documents.

The **Jaccard coefficient** is a measure of the similarity between two sample sets and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{23}$$

The **Jaccard distance** measures the dissimilarity between two sample sets and is obtained by subtracting the Jaccard coefficient from 1 – formula (24

$$J_d(X,Y) = 1 - J(X,Y) = \frac{|X \cup Y| - |X \cap Y|}{|X \cup Y|} \tag{24}$$

Similar to the Jaccard coefficient, **Dice's coefficient** is a measure of the similarity between sample sets. In natural language processing, for documents X and Y the coefficient is defined as:

$$D(X,Y) = \frac{2|X \cap Y|}{|X| + |Y|} \tag{25}$$

The **Dice distance** measures the dissimilarity between two sample sets and is obtained by subtracting the Dice coefficient from 1:

$$D_d(X,Y) = 1 - D(X,Y) = 1 - \frac{2|X \cap Y|}{|X| + |Y|} \tag{26}$$

The lower the Dice distance (or Jaccard distance) is, the more similar two strings are.

Appendix D: Repository Frameworks

Name	Description Usage	Security	CRUD Search	User-defined meta-model	User-friendly interface	HTTP REST interface	OAI-PMH	OAI-ORE
ePrints	Open source Available for download	HTTPS supported	CRUD supported Search supported	Supported	Web-interface	Supported	Supported	Not yet supported
Fedora	Open source Available for download	HTTPS supported	CRUD supported Search supported	Supported	Web-interface and JAVA client	Supported	Supported	Supported
ACS	Open source Available for download	GSI supported	CRUD supported Search supported	Supported	Not provided	Requires development	Not supported	Not supported
Oracle 10g	Open source Available for download	HTTPS supported	CRUD supported Search supported	Supported	Graphical client provided	N/A	Not provided	Not provided
IBM WebSphere	Commercial software Available for download	HTTPS Supported	CRUD supported Search supported	Supported	No client provided by default; requires development	N/A	N/A	N/A
WebGRelC	Open source No release candidate at the time this report was being written	GSI supported	CRUD supported Search supported	Partial support	JAVA client and JSR-168 compliant client	Not provided	Not provided	Not provided
JAVA CoG Kit	Open source Available for download	* JAVA CoG Kit is a development environment; in theory, it supports all the requirements specified by this research. However, all of them have to be engineered from community–added code in combination with newly written code.						

Colour codes:

Requirement fulfilled

Requirement fulfilled, but a better alternative is preferred

Requirement not fulfilled; needs further development

Appendix E: OAI-PMH *GetRecord* query example

http://192.168.1.68:8080/fedora/oai?verb=GetRecord&identifier=oai:example.org:gamrs:application549&metadataPrefix=oai_dc

<OAI-PMH xmlns=http://www.openarchives.org/OAI/2.0/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
<responseDate>2010-05-26T14:56:36Z</responseDate>
<request verb="GetRecord" metadataPrefix="oai_dc"
identifier="oai:example.org:gamrs:application549">http://192.168.56.101:8080/fedora/oai</request>
<GetRecord>
<record>
<header>
<identifier>oai:example.org:gamrs:application549</identifier>
<datestamp>2010-05-26T15:35:54Z</datestamp> </header>
<metadata>
<oai_dc:dc xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/
http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
<dc:title>AMBER</dc:title>
<dc:creator>Alex Tudose</dc:creator>
<dc:description>
This is an example-job of Amber for parallel run.The input files required for this parallel example-run of sander.LES.MPI can be staged to your home (or working) directory by clicking on "DataStaging" page/tab. When you submit the job and have "Stage all data when submitting job" checked on, these files will automatically be uploaded to your selected directory on your selected run host. If your working directory will be different from your home directory, you can define a new directory on the "Detail" page/tab, "WORKINGDIR" box.The executable should be given as the FIRST argument (on the "Arguments page) and not in the executable box. This box should always have the value /usr/ngs/AMBER or /usr/ngs/AMBER_9_0 if you want to run version 9.0.
</dc:description>
<dc:identifier>gamrs:application549</dc:identifier>
</oai_dc:dc> </metadata> </record>
</GetRecord></OAI-PMH>

Appendix F: OAI-ORE document of a GAMRS object – example


```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <id>info:fedora/gamrs:application549</id>
  <title type="text">AMBER </title>
  <updated>2010-05-26T19:35:54.370Z</updated>
  <author><name>fedoraAdmin</name></author>
  <category term="Active" scheme="info:fedora/fedora-
system:def/model#state"></category>
  <category term="2009-10-29T21:38:20.329Z" scheme="info:fedora/fedora-
system:def/model# createdDate"></category>
  <icon>http://www.fedora-
commons.org/images/logo_vertical_transparent_200_251.png</icon>
<entry>
  <id>info:fedora/gamrs:application549/DC</id>
  <title type="text">DC</title>
  <updated>2010-05-26T19:35:54.370Z</updated>
  <link href="info:fedora/gamrs:application549/DC/2010-05-26T19:35:54.370Z"
rel="alternate"></link>
  <category term="A" scheme="info:fedora/fedora-
system:def/model#state"></category>
  <category term="X" scheme="info:fedora/fedora-
system:def/model#controlGroup"></category>
  <category term="true" scheme="info:fedora/fedora-system:def/model#
versionable"></category>
</entry>
<entry xmlns:thr="http://purl.org/syndication/thread/1.0">
  <id>info:fedora/gamrs:application549/DC/2009-10-27T07:17:52.541Z</id>
  <title type="text">DC1.0</title>
  <updated>2009-10-27T07:17:52.541Z</updated>
  <thr:in-reply-to ref="info:fedora/gamrs:application549/DC"></thr:in-reply-to>
  <category term="http://www.openarchives.org/OAI/2.0/oai_dc/"
scheme="info:fedora/fedora-system:def/model#formatURI"></category>
  <category term="Dublin Core Record for this object" scheme="info:fedora/fedora-
system:def/model# label"></category>
  <category term="378" scheme="info:fedora/fedora-
system:def/model#length"></category>
  <content type="text/xml">
  <oai_dc:dc xmlns:oai_dc=http://www.openarchives.org/OAI/2.0/oai_dc/
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/
http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
<dc:title>AMBER</dc:title>
<dc:identifier>gamrs:application549</dc:identifier></oai_dc:dc></content>
</entry>
(...)
<entry>
<id>info:fedora/gamrs:application549/MRDL</id>
<title type="text">MRDL</title>
<updated>2009-10-29T11:51:50.080Z</updated>
(...)
<id>info:fedora/gamrs:application549/MRDL/2009-10-29T11:51:50.080Z</id>
<title type="text">MRDL.0</title>
<updated>2009-10-29T11:51:50.080Z</updated>
<thr:in-reply-to ref="info:fedora/gamrs:application549/MRDL"></thr:in-reply-to>
<category term="Description in GAMRS ADL" scheme="info:fedora/fedora-system:def/model#label"></category>
<category term="5380" scheme="info:fedora/fedora-system:def/model#length"></category>
<content type="text/xml">
<uk.ac.wmin.cpc.mrp.parsers.mrdl:JobDescription (...)>
<JobIdentification>
<JobName>AMBER (parallel example)</JobName>
<Description>AMBER
This is an example-job of Amber for parallel run. The input files required for this parallel example-run of sander.LES.MPI can be staged to your home (or working) directory by clicking on "DataStaging" page/tab.(...)</Description></JobIdentification>
<Application>
<ApplicationName>AMBER </ApplicationName>
<ApplicationVersion>10.00</ApplicationVersion>
<POSIXApplication>
<Executable>/usr/ngs/AMBER</Executable>
<Argument>sander.LES.MPI</Argument>
<Argument>-O</Argument>
<Argument>-i</Argument>
<Argument>md.in</Argument>
(...)
<ProcessCountLimit>4</ProcessCountLimit>
</POSIXApplication>
</Application>
<DataStaging>
<FileName>md.in</FileName>
<FilesystemName>WORKINGDIR</FilesystemName>
<CreationFlag>overwrite</CreationFlag>
<DeleteOnTermination>false</DeleteOnTermination>
<Source>

<URI>gsiftp://ngs.rl.ac.uk:2811/apps/amber/examples/parallel_example/md.in</URI>
          </Source>
       </DataStaging>
(...)
</uk.ac.wmin.cpc.mrp.parsers.mrdl:JobDescription></content>
</entry>
<entry>
   <id>info:fedora/gamrs:application549/AUDIT</id>
   <title type="text">AUDIT</title>
   <updated>2009-10-29T21:38:20.329Z</updated>
(...)
  </entry>
  <entry xmlns:thr="http://purl.org/syndication/thread/1.0">
   <id>info:fedora/gamrs:application549/AUDIT/2009-10-29T21:38:20.329Z</id>
   <title type="text">AUDIT.0</title>
 (...)
<category term="Audit Trail for this object" scheme="info:fedora/fedora-system:def/model#label"></category>
   <content type="text/xml">
     <audit:auditTrail xmlns:audit="info:fedora/fedora-system:def/audit#">
       <audit:record ID="AUDREC1">
         <audit:process type="Fedora API-M"></audit:process>
         <audit:action>ingest</audit:action>
         <audit:componentID></audit:componentID>
         <audit:responsibility>fedoraAdmin</audit:responsibility>
         <audit:date>2009-10-29T21:38:20.329Z</audit:date>
         <audit:justification>Ingested from local file /root/foxml_all/apps/sec/gamrs_AMBER(parallel_example)_template_549</audit:justification>
</audit:auditTrail>
   </content>
  </entry>
</feed>

Appendix G: BSoft application – FOXML

```
<?xml version="1.0" encoding="UTF-8"?>
<foxml:digitalObject VERSION="1.1" PID="gamrs:application1"
  xmlns:foxml="info:fedora/fedora-system:def/foxml#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="info:fedora/fedora-system:def/foxml#
http://www.fedora.info/definitions/1/0/foxml1-1.xsd">
<foxml:objectProperties>
  <foxml:property NAME="info:fedora/fedora-system:def/model#state" VALUE="Active"/>
  <foxml:property NAME="info:fedora/fedora-system:def/model#label" VALUE="BSoft"/>
  <foxml:property NAME="info:fedora/fedora-system:def/model#ownerId" VALUE="fedoraAdmin"/>
  <foxml:property NAME="info:fedora/fedora-system:def/model#createdDate" VALUE="2009-10-
26T14:53:01.233Z"/>
  <foxml:property NAME="info:fedora/fedora-system:def/view#lastModifiedDate" VALUE="2009-10-
29T11:51:50.080Z"/>
</foxml:objectProperties>
<foxml:datastream ID="AUDIT" STATE="A" CONTROL_GROUP="X" VERSIONABLE="false">
  <foxml:datastreamVersion ID="AUDIT.0" LABEL="Audit Trail for this object" CREATED="2009-10-
26T14:53:01.233Z" MIMETYPE="text/xml" FORMAT_URI="info:fedora/fedora-
system:format/xml.fedora.audit">
  <foxml:xmlContent>
  <audit:auditTrail xmlns:audit="info:fedora/fedora-system:def/audit#">
  <audit:record ID="AUDREC1">
    <audit:process type="Fedora API-M"/>
    <audit:action>addDatastream</audit:action>
    <audit:componentID>TN</audit:componentID>
    <audit:responsibility>fedoraAdmin</audit:responsibility>
    <audit:date>2009-10-26T19:13:46.259Z</audit:date>
    <audit:justification></audit:justification>
  </audit:record>
  <audit:record ID="AUDREC2">
    <audit:process type="Fedora API-M"/>
    <audit:action>addDatastream</audit:action>
    <audit:componentID>ASSETS</audit:componentID>
    <audit:responsibility>fedoraAdmin</audit:responsibility>
    <audit:date>2009-10-26T19:59:26.524Z</audit:date>
    <audit:justification></audit:justification>
  </audit:record>
(...)
  </audit:auditTrail></foxml:xmlContent></foxml:datastreamVersion>
</foxml:datastream>
<foxml:datastream ID="DC" STATE="A" CONTROL_GROUP="X" VERSIONABLE="true">
<foxml:datastreamVersion ID="DC.0" LABEL="Dublin Core Record for this object"
CREATED="2009-10-26T15:18:12.424Z" MIMETYPE="text/xml"
FORMAT_URI="http://www.openarchives.org/OAI/2.0/oai_dc/" SIZE="913">
<foxml:xmlContent>
<oai_dc:dc xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
```

instance" xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/ http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
<dc:title>BSoft</dc:title>
<dc:creator>Alex Tudose</dc:creator>
<dc:description>This is an example of running the PFT3DR supplied tutorial example, which uses a tcsh script 'iteration_01-09.tsch' to run pft2, em3dr2 and Bsoft programs in a sequence of 9 steps to process an image file 'polyoma_images.pif'.</dc:description>
<dc:publisher>NGS Application Repository</dc:publisher>
<dc:type>INSTANCE</dc:type>
<dc:format>v1.5.4</dc:format>
<dc:identifier>gamrs:application1</dc:identifier>
<dc:source>https://portal.ngs.ac.uk/JobProfiles.jsf</dc:source>
<dc:relation></dc:relation>
</oai_dc:dc></foxml:xmlContent></foxml:datastreamVersion>
</foxml:datastream>

<foxml:datastream ID="RELS-EXT" STATE="A" CONTROL_GROUP="X" VERSIONABLE="true">
<foxml:datastreamVersion ID="RELS-EXT.0" LABEL="RDF Statements about this object" CREATED="2009-10-26T14:56:10.562Z" MIMETYPE="application/rdf+xml" FORMAT_URI="info:fedora/fedora-system:FedoraRELSExt-1.0" SIZE="492">
<foxml:xmlContent>
<rdf:RDF xmlns:fedora-model="info:fedora/fedora-system:def/model#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rel="info:fedora/fedora-system:def/relations-external#">
<rdf:Description rdf:about="info:fedora/gamrs:application1">
<rel:isMemberOf rdf:resource="info:fedora/gamrs:applications"></rel:isMemberOf>
<fedora-model:hasModel rdf:resource="info:fedora/gamrs:ApplicationModel"></fedora-model:hasModel>
</rdf:Description>
</rdf:RDF></foxml:xmlContent></foxml:datastreamVersion>
</foxml:datastream>

<foxml:datastream ID="TN" STATE="A" CONTROL_GROUP="M" VERSIONABLE="true">
<foxml:datastreamVersion ID="TN.0" LABEL="Thumbnail.png" CREATED="2009-10-26T19:13:46.259Z" MIMETYPE="image/png">
<foxml:contentLocation TYPE="INTERNAL_ID" REF="http://192.168.56.101:8080/fedora/get/gamrs:application1/TN/2009-10-26T19:13:46.259Z"/>
</foxml:datastreamVersion>
</foxml:datastream>

<foxml:datastream ID="ASSETS" STATE="A" CONTROL_GROUP="E" VERSIONABLE="true">
<foxml:datastreamVersion ID="ASSETS.0" LABEL="Application Assets" CREATED="2009-10-26T19:59:26.524Z" MIMETYPE="text/xml">
<foxml:contentLocation TYPE="URL" REF="http://local.fedora.server/fedora/get/gamrs:appassets"/>
</foxml:datastreamVersion>
<foxml:datastreamVersion ID="ASSETS.1" LABEL="Application Assets" CREATED="2009-10-26T20:27:04.786Z" MIMETYPE="text/xml">
<foxml:contentLocation TYPE="URL" REF="http://local.fedora.server/fedora/get/gamrs:app1assets"/>
</foxml:datastreamVersion>
</foxml:datastream>

<foxml:datastream ID="gamrsapplication" STATE="A" CONTROL_GROUP="X" VERSIONABLE="true">
(...)

<foxml:datastreamVersion ID="gamrsapplication.5" LABEL="GAMRS Statements about this object" CREATED="2009-10-27T07:35:35.273Z" MIMETYPE="text/xml" SIZE="1111">
<foxml:xmlContent>
<gamrsApplication>
 <name>PFT3DR-Bsoft</name>
 <version>v1.5.4</version>
 <description>This is an example of running the PFT3DR supplied tutorial example, which uses a tcsh script 'iteration_01-09.tsch' to run pft2, em3dr2 and Bsoft programs in a sequence of 9 steps to process an image file 'polyoma_images.pif'.</description>
 <owner>http://192.168.56.101/fedora/repository/gamrs:user1</owner>
 <template>TRUE</template>
 <adltype>JSDL</adltype>
 <reference>http://192.168.56.101/fedora/repository/gamrs:application1</reference>
 <provider>http://192.168.56.101/fedora/repository/gamrs:provider1</provider>
 <externalref>https://portal.ngs.ac.uk/JobProfiles.jsf</externalref>
 <asset>http://192.168.56.101/fedora/repository/gamrs:app1assets1</asset>
 <asset>http://192.168.56.101/fedora/repository/gamrs:app1assets2</asset>
 <asset>http://192.168.56.101/fedora/repository/gamrs:app1assets3</asset>
 <asset>http://192.168.56.101/fedora/repository/gamrs:app1assets4</asset>
 <relation>
  <peer></peer>
  <score></score>
 </relation>
 <other></other>
</gamrsApplication></foxml:xmlContent></foxml:datastreamVersion></foxml:datastream>

<foxml:datastream ID="ADL" STATE="A" CONTROL_GROUP="X" VERSIONABLE="true">
<foxml:datastreamVersion ID="ADL.0" LABEL="Description in native ADL" CREATED="2009-10-27T07:37:35.467Z" MIMETYPE="text/xml" SIZE="10549">
<foxml:xmlContent>
 <jsdl:JobDefinition xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
xmlns:rsl="http://www.ggf.org/namespaces/2004/11/jsdl-rsl-1.0.xsd"
xmlns:sweep="http://schemas.ogf.org/jsdl/2007/04/sweep"
xmlns:sweepfunc="http://schemas.ogf.org/jsdl/2007/04/sweep/functions">
 <jsdl:JobDescription>
  <jsdl:JobIdentification>
   <jsdl:JobName>PFT3DR / Bsoft Iteration</jsdl:JobName>
   <jsdl:Description>PFT3DR - iteration_01-09.tsch Tutorial example.

---
template configured by J.Churchill (SCT,RAL,STFC) Sept 2008

This is an example of running the PFT3DR supplied tutorial example, which uses a tcsh script 'iteration_01-09.tsch' to run pft2, em3dr2 and Bsoft programs in a sequence of 9 steps to process an image file 'polyoma_images.pif&apos.This portal example is slightly different from other portal examples, as the staging step includes staging the 'executable' , which in this case is a script. Unfortunately, at the time of writing (Sept 2008), the data staging uses an API which does not retain executable file permissions. So this example needs to be run in two steps where (a) Is the data staging step and (b) Runs the job. In between (a) and (b) you will need to login to the run host and change the file permissions to executable (chmod +x file_name) on the iteration script and the script which starts it, called 'run_iteration'. The script 'run_iteration' simply loads the appropriate environment modules for PFT3DR (and Bsoft) so that these programs are in the environment path when it then runs 'iteration_01-09.tcsh'. These module load statements could be included in the iteration script but this example shows how you can run the tutorial unedited.For this example you need to create a small directory hierarchy on the system that will run the job. First create a top level sub directory for the job

to run in (you can use the 'Browse Host' page in the portal or login to the run host). In this subdirectory create the 'maps', 'resolution', 'particles' and 'run' directories. This portal example is setup to stage data into these 4 directories. Each of these directory locations is determined on the 'File Systems' page of the portal by the WORKINGDIR, MAPS, PARTICLES and RESOLUTION "file system", where WORKINGDIR is set to the location of the 'run' subdirectory and the others are self explanatory. The iterations script assumes it is in the 'run01 directory and the other 3 directories are at the same hierarchy level as 'run01'. This example assumes that the 4 directories are located for a mythical user ngs0341 under /home/ngs0341/pft3dr/run. You need to change all 4 directory locations to your ngs user id (ngsXXX) and the location of your run (replace pft3dr/run in the path). To do this click on the folder icon next to "WorkingDir" then 'Browse for dir on a grid host' on the 'File Systems' page. This takes you to the "Browse Host" page. Connect to the host you will run the job on by selecting from the hosts lists then clicking "Connect". Browse your files until you are in the directory you want as the working dir, then use the Actions drop down and select "Apply as Working dir" then click "OK" button. To create a new subdirectory to run in, use the Actions drop down and select "Create subdir" option. Find or create the other three directories and edit their locations directly into 'Mount Point' for each on the File systems page.To run this example job (or another job based on this template) first go to the 'DataStaging' page and click the 'Stage In Now' button to upload the files for the example. Now login to the run host and change the permissions on run_iteration and itermation_01-09.tcsh files to executable (eg chmod +x run_iteration). Then go to the "Submit/Run" page and check the box next to "Ok to overwrite the job status when re-submitting the job" and uncheck the box "Stage all data...". Then click the "Submit My Job" button. The status of the job should appear on the "Job Status" line. First it will say "Submitted". To update the status, click on "Check Job Status". When completed, go to the "Data Transfer" or "Browse Host" page and download your output. This example takes 10-20 minutes to run.For more information about running PFT3DR jobs on the NGS, please refer to: http://www.ngs.ac.uk/sites/ral/applications/ImageAnalysis/pft3dr.html or use the links on the 'Files/Links' page of this portal template.If you need help, please contact the NGS helpdesk (support@grid-support.ac.uk)
</jsdl:Description>
</jsdl:JobIdentification>
<jsdl:Application>
<jsdl:ApplicationName>PFT3DR / Bsoft</jsdl:ApplicationName>
<jsdl:ApplicationVersion>2.0.4 / 1.5.4</jsdl:ApplicationVersion>
<jsdl-posix:POSIXApplication>
<jsdl-posix:Executable filesystemName="WORKINGDIR">run_iteration</jsdl-posix:Executable>
<jsdl-posix:Output filesystemName="WORKINGDIR">iteration.out</jsdl-posix:Output>
<jsdl-posix:Error filesystemName="WORKINGDIR">iteration.err</jsdl-posix:Error>
<jsdl-posix:WorkingDirectory>/home/ngs0341/pft3dr/run/run03</jsdl-posix:WorkingDirectory>
<jsdl-posix:ProcessCountLimit>1</jsdl-posix:ProcessCountLimit>
</jsdl-posix:POSIXApplication>
<rsl:jobType>single</rsl:jobType>
</jsdl:Application>
<jsdl:Resources>
<jsdl:CandidateHosts>
<jsdl:HostName>ngs.rl.ac.uk:2119/lsf</jsdl:HostName>
</jsdl:CandidateHosts>
<jsdl:FileSystem name="WORKINGDIR">
<jsdl:FileSystemType>normal</jsdl:FileSystemType>
<jsdl:Description>The working job directory</jsdl:Description>
<jsdl:MountPoint>/home/ngs0341/pft3dr/run/run03</jsdl:MountPoint>
</jsdl:FileSystem>
<jsdl:FileSystem name="MAPS">
<jsdl:FileSystemType>normal</jsdl:FileSystemType>
<jsdl:Description>File system added by browsing host</jsdl:Description>

```
      <jsdl:MountPoint>/home/ngs0341/pft3dr/run/maps</jsdl:MountPoint>
    </jsdl:FileSystem>
    <jsdl:FileSystem name="PARTICLES">
      <jsdl:FileSystemType>normal</jsdl:FileSystemType>
      <jsdl:Description></jsdl:Description>
      <jsdl:MountPoint>/home/ngs0341/pft3dr/run/particles</jsdl:MountPoint>
    </jsdl:FileSystem>
    <jsdl:FileSystem name="RESOLUTION">
      <jsdl:FileSystemType>normal</jsdl:FileSystemType>
      <jsdl:Description></jsdl:Description>
      <jsdl:MountPoint>/home/ngs0341/pft3dr/run/resolution</jsdl:MountPoint>
    </jsdl:FileSystem>
  </jsdl:Resources>
  <jsdl:DataStaging>
    <jsdl:FileName>iterations_01-09.tcsh</jsdl:FileName>
    <jsdl:FilesystemName>WORKINGDIR</jsdl:FilesystemName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
    <jsdl:Source>
      <jsdl:URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/run/iterations_01-
09.tcsh</jsdl:URI>
    </jsdl:Source>
  </jsdl:DataStaging>
  <jsdl:DataStaging>
    <jsdl:FileName>polyoma_00.star</jsdl:FileName>
    <jsdl:FilesystemName>WORKINGDIR</jsdl:FilesystemName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
    <jsdl:Source>
      <jsdl:URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-
09/run/polyoma_00.star</jsdl:URI>
    </jsdl:Source>
  </jsdl:DataStaging>
  <jsdl:DataStaging>
    <jsdl:FileName>polyoma_images.pif</jsdl:FileName>
    <jsdl:FilesystemName>WORKINGDIR</jsdl:FilesystemName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
    <jsdl:Source>
      <jsdl:URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-
09/run/polyoma_images.pif</jsdl:URI>
    </jsdl:Source>
  </jsdl:DataStaging>
  <jsdl:DataStaging>
    <jsdl:FileName>run_iteration</jsdl:FileName>
    <jsdl:FilesystemName>WORKINGDIR</jsdl:FilesystemName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
    <jsdl:Source>
      <jsdl:URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-
09/run/run_iteration</jsdl:URI>
    </jsdl:Source>
  </jsdl:DataStaging>
  <jsdl:DataStaging>
    <jsdl:FileName>polyoma_3d.pif</jsdl:FileName>
    <jsdl:FilesystemName>MAPS</jsdl:FilesystemName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
```

<jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
<jsdl:Source>
<jsdl:URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/maps/polyoma_3d.pif</jsdl:URI>
</jsdl:Source>
</jsdl:DataStaging>
<jsdl:DataStaging>
<jsdl:FileName>polyoma_images.pif</jsdl:FileName>
<jsdl:FilesystemName>MAPS</jsdl:FilesystemName>
<jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
<jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
<jsdl:Source>
<jsdl:URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/maps/polyoma_images.pif</jsdl:URI>
</jsdl:Source>
</jsdl:DataStaging>
<jsdl:DataStaging>
<jsdl:FileName>README</jsdl:FileName>
<jsdl:FilesystemName>PARTICLES</jsdl:FilesystemName>
<jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
<jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
<jsdl:Source>
<jsdl:URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/particles/README</jsdl:URI>
</jsdl:Source>
</jsdl:DataStaging>
<jsdl:DataStaging>
<jsdl:FileName>README</jsdl:FileName>
<jsdl:FilesystemName>RESOLUTION</jsdl:FilesystemName>
<jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
<jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
<jsdl:Source>
<jsdl:URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/resolution/README</jsdl:URI>
</jsdl:Source>
</jsdl:DataStaging>
</jsdl:JobDescription>
</jsdl:JobDefinition></foxml:xmlContent></foxml:datastreamVersion>
</foxml:datastream>

<foxml:datastream ID="MRDL" STATE="A" CONTROL_GROUP="X" VERSIONABLE="true">
<foxml:datastreamVersion ID="MRDL.0" LABEL="Description in GAMRS ADL" CREATED="2009-10-29T11:51:50.080Z" MIMETYPE="text/xml" SIZE="8620">
<foxml:xmlContent>
<uk.ac.wmin.cpc.mrp.parsers.mrdl:JobDescription xmlns:uk.ac.wmin.cpc.mrp.parsers.mrdl="http://schemas.ggf.org/uk.ac.wmin.cpc.mrp.parsers.mrdl/2005/11/uk.ac.wmin.cpc.mrp.parsers.mrdl" xmlns:xmi="http://www.omg.org/XMI" xmi:version="2.0">
<JobIdentification>
<JobName>PFT3DR / Bsoft Iteration</JobName>
<Description>PFT3DR - iteration_01-09.tsch Tutorial example.

---
template configured by J.Churchill (SCT,RAL,STFC) Sept 2008

This is an example of running the PFT3DR supplied tutorial example, which uses a tcsh script 'iteration_01-09.tsch' to run pft2, em3dr2 and Bsoft programs in a sequence of 9 steps to process an image file 'polyoma_images.pif'This portal example is slightly different

from other portal examples, as the staging step includes staging the 'executable' , which in this case is a script. Unfortunately, at the time of writing (Sept 2008), the data staging uses an API which does not retain executable file permissions. So this example needs to be run in two steps where (a) Is the data staging step and (b) Runs the job. In between (a) and (b) you will need to login to the run host and change the file permissions to executable (chmod +x file_name) on the iteration script and the script which starts it, called 'run_iteration'. The script 'run_iteration' simply loads the appropriate environment modules for PFT3DR (and Bsoft) so that these programs are in the environment path when it then runs 'iteration_01-09.tcsh'. These module load statements could be included in the iteration script but this example shows how you can run the tutorial unedited.For this example you need to create a small directory hierarchy on the system that will run the job. First create a top level sub directory for the job to run in (you can use the 'Browse Host' page in the portal or login to the run host). In this subdirectory create the 'maps', 'resolution', 'particles' and 'run' directories. This portal example is setup to stage data into these 4 directories. Each of these directory locations is determined on the 'File Systems' page of the portal by the WORKINGDIR, MAPS, PARTICLES and RESOLUTION "file system", where WORKINGDIR is set to the location of the 'run' subdirectory and the others are self explanatory. The iterations script assumes it is in the 'run01 directory and the other 3 directories are at the same hierarchy level as 'run01'. This example assumes that the 4 directories are located for a mythical user ngs0341 under /home/ngs0341/pft3dr/run. You need to change all 4 directory locations to your ngs user id (ngsXXX) and the location of your run (replace pft3dr/run in the path). To do this click on the folder icon next to "WorkingDir" then 'Browse for dir on a grid host' on the 'File Systems' page. This takes you to the "Browse Host" page. Connect to the host you will run the job on by selecting from the hosts lists then clicking "Connect". Browse your files until you are in the directory you want as the working dir, then use the Actions drop down and select "Apply as Working dir" then click "OK" button. To create a new subdirectory to run in, use the Actions drop down and select "Create subdir" option. Find or create the other three directories and edit their locations directly into 'Mount Point' for each on the File systems page.To run this example job (or another job based on this template) first go to the 'DataStaging' page and click the 'Stage In Now' button to upload the files for the example. Now login to the run host and change the permissions on run_iteration and itermation_01-09.tcsh files to executable (eg chmod +x run_iteration). Then go to the "Submit/Run" page and check the box next to "Ok to overwrite the job status when re-submitting the job" and uncheck the box "Stage all data...". Then click the "Submit My Job" button. The status of the job should appear on the "Job Status" line. First it will say "Submitted". To update the status, click on "Check Job Status". When completed, go to the "Data Transfer" or "Browse Host" page and download your output. This example takes 10-20 minutes to run.For more information about running PFT3DR jobs on the NGS, please refer to:http://www.ngs.ac.uk/sites/ral/applications/ImageAnalysis/pft3dr.html or use the links on the 'Files/Links' page of this portal template.If you need help, please contact the NGS helpdesk (support@grid-support.ac.uk)
</Description>
</JobIdentification>
<Application>
<ApplicationName>PFT3DR / Bsoft</ApplicationName>
<ApplicationVersion>2.0.4 / 1.5.4</ApplicationVersion>
<POSIXApplication>
<Executable>/home/ngs0341/pft3dr/run/run03/run_iteration</Executable>
<Output>/home/ngs0341/pft3dr/run/run03/iteration.out</Output>
<Error>/home/ngs0341/pft3dr/run/run03/iteration.err</Error>
<WorkingDirectory>/home/ngs0341/pft3dr/run/run03</WorkingDirectory>
<ProcessCountLimit>1</ProcessCountLimit>
</POSIXApplication>
</Application>
<DataStaging>
<FileName>iterations_01-09.tcsh</FileName>

```
    <FilesystemName>WORKINGDIR</FilesystemName>
    <CreationFlag>overwrite</CreationFlag>
    <DeleteOnTermination>false</DeleteOnTermination>
    <Source>
      <URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/run/iterations_01-09.tcsh</URI>
    </Source>
  </DataStaging>
  <DataStaging>
    <FileName>polyoma_00.star</FileName>
    <FilesystemName>WORKINGDIR</FilesystemName>
    <CreationFlag>overwrite</CreationFlag>
    <DeleteOnTermination>false</DeleteOnTermination>
    <Source>
      <URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/run/polyoma_00.star</URI>
    </Source>
  </DataStaging>
  <DataStaging>
    <FileName>polyoma_images.pif</FileName>
    <FilesystemName>WORKINGDIR</FilesystemName>
    <CreationFlag>overwrite</CreationFlag>
    <DeleteOnTermination>false</DeleteOnTermination>
    <Source>
      <URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/run/polyoma_images.pif</URI>
    </Source>
  </DataStaging>
  <DataStaging>
    <FileName>run_iteration</FileName>
    <FilesystemName>WORKINGDIR</FilesystemName>
    <CreationFlag>overwrite</CreationFlag>
    <DeleteOnTermination>false</DeleteOnTermination>
    <Source>
      <URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/run/run_iteration</URI>
    </Source>
  </DataStaging>
  <DataStaging>
    <FileName>polyoma_3d.pif</FileName>
    <FilesystemName>MAPS</FilesystemName>
    <CreationFlag>overwrite</CreationFlag>
    <DeleteOnTermination>false</DeleteOnTermination>
    <Source>
      <URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/maps/polyoma_3d.pif</URI>
    </Source>
  </DataStaging>
  <DataStaging>
    <FileName>polyoma_images.pif</FileName>
    <FilesystemName>MAPS</FilesystemName>
    <CreationFlag>overwrite</CreationFlag>
    <DeleteOnTermination>false</DeleteOnTermination>
    <Source>
      <URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-
09/maps/polyoma_images.pif</URI>
    </Source>
  </DataStaging>
  <DataStaging>
    <FileName>README</FileName>
    <FilesystemName>PARTICLES</FilesystemName>
    <CreationFlag>overwrite</CreationFlag>
```

```
    <DeleteOnTermination>false</DeleteOnTermination>
    <Source>
      <URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/particles/README</URI>
    </Source>
  </DataStaging>
  <DataStaging>
    <FileName>README</FileName>
    <FilesystemName>RESOLUTION</FilesystemName>
    <CreationFlag>overwrite</CreationFlag>
    <DeleteOnTermination>false</DeleteOnTermination>
    <Source>
      <URI>gsiftp://ngs.rl.ac.uk/apps/pft3dr/examples/iterations_01-09/resolution/README</URI>
    </Source>
  </DataStaging>
  <Backend backendId="GT2" count="1" error="STDERR" jobType="single" output="STDOUT">
    <siteInfo id="0" jobManager="lsf">
      <site>ngs.rl.ac.uk:2119</site>
      <executable stage="false">
        <value>/home/ngs0341/pft3dr/run/run03/run_iteration</value>
      </executable>
      <paramPrefix>.</paramPrefix>
    </siteInfo>
  </Backend>
</uk.ac.wmin.cpc.mrp.parsers.mrdl:JobDescription>
</foxml:xmlContent>
</foxml:datastreamVersion>
</foxml:datastream>
</foxml:digitalObject>
```

Appendix H: Snapshot of Bsoft application deployed in a virtualized environment using GAMRS