Interoperable Information Exchange, Resource Discovery, and Service Quality Monitoring Across Virtual Organizations in Distributed Research Infrastructures

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"
der Georg-August-Universität Göttingen
im Promotionsprogramm PCS
der Georg-August University School of Science (GAUSS)

vorgelegt von

Tibor Kálmán
aus Kaposvár, Ungarn

Göttingen, 2016

Betreuungsausschuss

Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Oswald Haan
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG)

Prof. Dr. Dieter Hogrefe
Institut für Informatik, Georg-August-Universität Göttingen

Mitglieder der Prüfungskommission

Referent: Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG)
Institut für Informatik, Georg-August-Universität Göttingen

Korreferent: Prof. Dr. Ulrich Sax
Institut für Medizinische Informatik, Georg-August-Universität Göttingen

Weitere Mitglieder der Prüfungskommission

Prof. Dr. Jens Grabowski
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Oswald Haan
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG)

Prof. Dr. Dieter Hogrefe
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Caroline Sporleder
Institut für Informatik, Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 08. November 2016.

Abstract

Recently, distributed research infrastructures were built up to feed the emerging resource demand of research institutes and enable researchers from various disciplines to access cutting edge technologies. More recently, the different research disciplines began to approach each other in the mutual goal to answer increasingly complex research problems. It has therefore become increasingly important for research infrastructures to interoperate. Exchanging resource descriptions coming from the information and monitoring systems of different research infrastructures is among the most important steps to reach interoperability.

The aim of this thesis is to develop and realise a concept of an interoperable information and monitoring system, which can be used transparently with different existing distributed research infrastructures. This concept also extends the scope of existing information services by including resource properties relating to quality of service and organisational items. We focus on the information and monitoring systems, because of their importance for the other components of distributed research infrastructures. Exchanging resource descriptions, discovering resources, and monitoring the quality and availability of computing and storage resources are essential for executing jobs or transferring data in distributed research infrastructures.

Based on a requirement analysis of application scenarios we identify the information needed for exchanging resource descriptions and provide a theoretical model which is capable to include that information. We also present a theoretical approach for a schema mediation process. For modeling resource descriptions in distributed research environments many information schemas and data models exist. In addition to the theoretical analysis, we discuss the evaluation of information schemas being utilized in productional environments. Based on these results, a concept of an automated resource exchange process and a respective generic monitoring architecture supporting it are provided.

In addition to our theoretical analysis, concept, and models, a proof of concept for distributed research infrastructures has been developed. We also design and develop a simulation framework for service-oriented, multi-middleware environments that is based on automated system deployment and service provisioning. The evaluation is done by artificial simulation experiments in the simulation framework and in real system scenarios, which demonstrate the applicability of our concept.

Acknowledgements

The help and support of many people made this dissertation possible. First of all, I would like to thank my supervisor Prof. Dr. Ramin Yahyapour for helping me with his experience and knowledge to finalize my work. Also, I am very grateful to Prof. Dr. Oswald Haan and Prof. Dr. Bernhard Neumair, who made it possible for me to conduct research in a stimulating environment and provided advice and directions in the early phases.

I am also grateful to Prof. Dr. Ulrich Sax for agreeing to act as a co-referee for this thesis as well as to Prof. Dr. Jens Grabowski, Prof. Dr. Oswald Haan, Prof. Dr. Dieter Hogrefe and Prof. Dr. Caroline Sporleder for participating as members of my defense committee.

As I conducted research on distributed infrastructures, several people were involved at different times and locations. It is my privilege to recognize the important support they provided. I would like to thank all my colleagues at the D-MON project for their interest in my research topic and for their experienced insights. They helped me transform my research vision into reality and into production level concepts for a national research infrastructure. I am very grateful to Prof. Dr. Jens Grabowski's Software Engineering of Distributed Systems (SWE) Group, which supported me in applying my results as a teaching environment for practical courses and always gave me valuable scientific feedback when I needed it. I am furthermore deeply thankful to the administrators and representatives of various communities who patiently answered all my questions and provided a lot of monitoring information. I received important support from the colleagues of GoeGrid, Instant-Grid, MediGrid, DARIAH and ePIC.

Moreover, I want to thank my current and prior colleagues at the GWDG for the nice work atmosphere, as well as my friends and fellow doctoral students for their moral support, which made my day-to-day work balanced and varied. I would like to give special thanks to the two proof-readers of this work.

Last, but not least, I want to thank my family for their patience and endless support.

Contents

1. Introduction 1
1.1. Problem Statement 2
1.2. Contribution 3
1.3. Impact 4
1.4. Thesis Structure 6

2. Motivation 7
2.1. Computing Paradigms in Distributed Research Infrastructures 7
2.2. Infrastructures Under Study 10
2.2.1. Grid Infrastructures 10
2.2.2. Cloud Computing 11
2.3. Exchanging Resource Information and Monitoring in Distributed Research Infrastructures 13
2.4. Problems of Exchanging Resource Information and Monitoring in Distributed Research Infrastructures 15

3. Modelling Heterogeneous Resource Information in a Generic Way 17
3.1. Exchanging Resource Information Using a Generic Information Model 18
3.2. Actors of a Distributed Research Infrastructure 20
3.3. Requirements for Resource Information and Monitoring Data 23
3.3.1. Community Users 25
3.3.2. Community Administrators 26
3.3.3. Community Services 27
3.3.4. Infrastructure Administrators and Resource Providers 28
3.3.5. Infrastructure Services 28
3.4. Required Entities for Modeling Distributed Research Infrastructures 29
3.5. State of the Art of Exchanging Resource Information in Research Infrastructures 32
3.5.1. Examination of Middlewares 33
3.5.2. Examination of Information and Monitoring Systems 33
3.5.3. Examination of Information and Data Models 37
3.6. Generic Information Model for Monitoring Distributed Research Infrastructures 38
3.6.1. The GLUE v2.0 Information Model 38
3.6.2. The D-GRDL Information Model 42
3.6.3. Other Information Models 44
3.7. Mapping of Existing Information Models onto the Generic Information Model 44
3.7.1. ETL, a Data Warehousing Technique 44
3.7.2. ETL Process for Interoperable Monitoring of Distributed Research Infrastructures 45
3.7.3. Extraction of Resource Information and Monitoring Data 46
3.7.4. Transformation of Resource Descriptions 47
3.7.5. Loading of Resource Information and Monitoring Data 48
3.8. Schema Mapping and Schema Matching 48
3.9. Analysis of results 49

4. Implementing an Interoperable, Unified Monitoring and Information System for Distributed Research Infrastructures 51
4.1. Scenarios . . . 52
4.1.1. Research Infrastructure of the German e-Science Initiative . . . 52
4.1.2. Pan-European e-Infrastructure . . . 53
4.1.3. Instant-Grid . . . 54
4.2. Requirements for an Interoperable, Unified Monitoring and Information System . 55
4.2.1. Generic Requirements for Services of Distributed Research Infrastructures 55
4.2.2. Specific Requirements for an Interoperable, Unified Monitoring and Information System . . . 56
4.3. Design of an Interoperable Monitoring and Information System . . . 57
4.3.1. Basic Architecture . . . 57
4.3.2. Transformation of Resource Infromation and Monitoring Data . . . 60
4.3.3. Monitoring in a Community-specific Context . . . 61
4.3.4. Distributed Setup . . . 65
4.3.5. Discovering Monitoring and Information Services . . . 67
4.4. Implementation and Case Study . . . 69
4.4.1. D-MON Monitoring System . . . 69
4.4.2. ETL for Globus Toolkit, UNICORE, and gLite . . . 69
4.4.3. Community-aware Monitoring . . . 77
4.4.4. Service Discovery and Integrated Service Registry . . . 79
4.5. Related Work . . . 81
4.6. Analysis of results . . . 83
4.6.1. Experiences . . . 85

5. A Framework for the Simulation of Heterogeneous Information Services 87
5.1. A Simulation and Demonstration Environment . . . 87
5.2. Technical Concept . . . 88
5.2.1. Automated Self-Configuration . . . 89
5.2.2. Ready-to-Use Features and Applications . . . 90
5.3. Automated Middleware Deployment . . . 91
5.3.1. Dynamic Deployment of UNICORE 6 . . . 91
5.3.2. Dynamic Deployment of Globus Toolkit . . . 93
5.3.3. Security Environment . . . 94
5.3.4. Customization Options . . . 95
5.4. Use Case: Practical Course . . . 97
5.5. Connectivity with Research Infrastructures . . . 97
5.5.1. Secure Submission Client of Productional Research Infrastructures . . . 98
5.5.2. Accept Workload from Productional Research Infrastructures . . . 98
5.6. Related Work . . . 99
5.7. Analysis of Results . . . 100

6. Describing, Monitoring and Publishing Resource Quality in Distributed Research Infrastructures 103
6.1. Managing Distributed Research Infrastructures . . . 104
6.1.1. Activities of Managing Distributed Research Infrastructures . . . 104
6.1.2. Monitoring Resource Quality . . . 105
6.2. Quality Metrics and Key Performance Indicators . . . 106
6.2.1. State of the Art Measuring Resource Quality . . . 107
6.2.2. Suggested Key Performance Indicators and Quality Metrics . . . 108
6.3. Service Quality in the Data Model . . . 109
6.3.1. The Currently Used D-GRDL Data Model . . . 110

6.3.2. GLUE v2.0 Data Model for Computing Resources 110
6.3.3. GLUE v2.0 Data Model for Storage Resources 112
6.4. Data Sources of Quality Information 113
6.5. Case Study 1: Extractor for the Jawari Benchmarking System 113
6.5.1. Jawari Benchmarking Toolkit 114
6.5.2. Extracting the Results of the Static Single Task Benchmark 114
6.6. Case Study 2: Extending a Traditional Monitoring Framework for Extracting Quality Information 116
6.6.1. Homogeneous Monitoring of Heterogeneous Stateful Services 116
6.6.2. Architecture and Implementation 118
6.7. Related Work 120
6.8. Analysis of Results 120

7. Conclusion 122
7.1. Summary 122
7.2. Thesis Conclusions 122
7.2.1. Generic Information Modeling of Heterogeneous Resource Descriptions and Monitoring Data 122
7.2.2. Generic Architecture, Implementation and Proof of Concept 123
7.2.3. Automated system deployment and productional infrastructures 124
7.2.4. Additional Metadata for Resource Descriptions 124
7.3. Outlook 125
7.3.1. Automated System Provisioning and Service Deployment 125
7.3.2. Evolution of Paradigms 126
7.3.3. Extension of Data and Information Modeling 126

Bibliography 129

A. Appendix: Size and Structure of the Monitoring Data 145
A.1. Size of Monitoring Data 146
A.1.1. Filesize MDS4: National RI 146
A.1.2. Filesize MDS4: Community RI 146
A.1.3. Filesize BDII: National RI 147
A.1.4. Filesize BDII: RI Site 147
A.2. Structure of Monitoring Data 148
A.2.1. GLUE2 Records: National RI with MDS4 149
A.2.2. GLUE2 Records: Community RI with MDS4 150
A.2.3. GLUE2 Records: National RI with BDII 151
A.2.4. GLUE2 Records: RI Site with BDII 152
A.2.5. GLUE2 Records: National RI with CIS (Test-Instance) 153

B. Appendix: Selected Mapping Tables 154

C. Appendix: Performance and Stress Tests with Grinder 157
C.1. Stress Test Tool 157
C.2. Portal Interface for D-MON 158
C.3. Grinder Results 159

List of Figures

2.1. XaaS model of Cloud Computing 12
2.2. Exchanging Resource Information in Distributed Research Infrastructures Without an Interoperable Information System 14

3.1. Data Modeling for Developing a Generic Information and Data Model for Information- and Monitoring Systems of Distributed Research Infrastructures . . 19
3.2. A Structured, 5-Steps, Generic Modeling Process 20
3.3. Use Case Diagram of an Interoperable, Integrated, Community-based Information and Monitoring System 21
3.4. Required Entities for Modeling Resources of Distributed Research Infrastructures 30
3.5. Utilizing Middlewares in Distributed Research Infrastructures 32
3.6. Computing Entities of the GLUE v2.0 Schema 41
3.7. Storage Entities of the GLUE v2.0 Schema 42
3.8. Generic ETL Process for a Data Warehouse 45
3.9. ETL Process for Interoperable Monitoring of Distributed Research Infrastructures 46
3.10. Schema Mediation for Distributed Research Infrastructures 47

4.1. Monitoring and Information System Based on a Privileged System 57
4.2. Autonomous, Middleware-independent Monitoring and Information System . . . 59
4.3. Schema Mediation 60
4.4. Use Case Diagram of an Interoperable, Integrated, Community-based Monitoring and Information System 62
4.5. Access Control Utilizing Policy Enforcement Points and Policy Decision Points . 63
4.6. Using VO-attributes in Integrated Monitoring 64
4.7. Central-, Federated- and Clustered Storage Component for an Autonomous, Middleware-Independent Service 66
4.8. Information Service Discovery for the Integrated, Interoperable Monitoring System 68
4.9. Schema Mapping for the MDS4 Site Entity and GLUE v2.0 AdminDomain . . . 73
4.10. An Example for Monitoring in Community-aware Context: the D-MON Integrated Monitoring System 78
4.11. Integrated Discovery Service for an Integrated Monitoring Service 80

5.1. Software Components to Establish the Automated Configuration at Startup 89
5.2. Network Interface Configuration for Direct as well as Translated Access 90
5.3. Dynamically Configured UNICORE 6 Architecture of IG 92
5.4. Components of the IG Globus Toolkit Edition 94
5.5. New Setup Phases During the Boot Process of the IG UNICORE Edition 97
5.6. IG Environment Deployed in the Practical Course 98

6.1. Activities of Managing Distributed Research Infrastructures 105
6.2. Resource Matching by GWES 107
6.3. Entities of the GLUE v2.0 Schema and Their Attributes Related to Describe Quality of Computing Resources 110
6.4. RGID Interface for the Jawari Grid Benchmarking Service 114

6.5. Extending a Traditional Monitoring System for Collecting Information Related to Quality Metrics 118
6.6. Possible Monitoring Time-Out during Test Execution: Stateful Service Data Is Not Destroyed 119
6.7. Using Additional Time-Out for Proper Clean-Up in Case of a Monitoring Time-Out 119

A.1. Size of MDS4 Monitoring Data in a National Research Infrastructure 146
A.2. Size of MDS4 Monitoring Data in a Research Infrastructure Serving a Community 146
A.3. Size of BDII Monitoring Data in a National Research Infrastructure 147
A.4. Size of BDII Monitoring Data in a Research Infrastructure Serving a Community 147
A.5. Mean of GLUE2 Records: National Research Infrastructure with MDS4 149
A.6. Mean of GLUE2 Records: Community Research Infrastructure with MDS4 . . . 150
A.7. Mean of GLUE2 Records: National Research Infrastructure with BDII 151
A.8. Mean of GLUE2 Records: Research Infrastructure Site with BDII 152
A.9. Mean of GLUE2 Records in CIS: Testbed for a National Research Infrastructure . 153

C.1. The Architecture of Grinder: Console, Workers, and Agents 157
C.2. The D-MON Portal Interface 158
C.3. Stress Test Controlled and Monitored by the Grinder Console 159
C.4. Test Results and Statistics Shown by the Grinder Console 159
C.5. Grinder Console Showing the Grinder Agents Running 40 Workers and 40 Threads 160

List of Tables

2.1. Essential characteristics of the Cloud model defined by NIST 12

3.1. Overview of Middleware Systems and Distributed Research Infrastructure Initiatives 34
3.2. Overview of Information- and Monitoring Services in Distributed Research Infrastructures 36
3.3. Examination of Information Models 39

4.1. Examination of Architectures of an Interoperable Information System 58
4.2. Main Advantages and Disadvantages of Distributed Storage Components Compared to Centralized Storage Components 67

6.1. The Quality Levels of a Computing Service Defined by the GLUE v2.0 Schema . 111
6.2. The Health States Defined by the GLUE v2.0 Schema 111
6.3. The Benchmark Types Defined by the GLUE v2.0 Schema 112
6.4. Benchmark.ID is a Suggested New Attribute to the GLUE v2.0 Schema 113
6.5. WSRF-based Services in GT4 and UNICORE6 for Remote Testing 117

A.1. MDS4, BDII, CIS Sources for Monitoring Data 145
A.2. Mean of GLUE2 Records: National Research Infrastructure with MDS4 149
A.3. Mean of GLUE2 Records: Community Research Infrastructure with MDS4 . . . 150
A.4. Mean of GLUE2 Records: National Research Infrastructure with BDII 151
A.5. Mean of GLUE2 Records: Research Infrastructure Site with BDII 152
A.6. Mean of GLUE2 Records in CIS: Testbed for a National Research Infrastructure . 153

B.1. Example Mapping for the AdminDomain Entity of the GLUE v2.0 Schema . . . 154
B.2. Example Mapping for the AdminContact Entity of the GLUE v2.0 Schema . . . 154
B.3. Example Mapping for the AdminDomainLocation Entity of the GLUE v2.0 Schema154
B.4. Example Mapping for the ComputingService Entity of the GLUE v2.0 Schema . 155
B.5. Example Mapping for the ComputingShare Entity of the GLUE v2.0 Schema . . 156

C.1. Grinder Results of Test Cases Simulating Portal Users 160
C.2. Grinder Results of Test Cases Simulating ETL-Adaptors 160

Acronyms

API *Application Programming Interface*

ARPANET *Advanced Research Projects Agency Network*

AWS *Amazon Web Services*

BDII *Berkeley Database Information Index*

CA *Certification Authority*

CaaS *Container as a Service*

CD *Compact Disc*

CDI *Collaborative Data Infrastructure*

CIM *Common Information Model*

CIS *Common Information Service*

CIP *Common Information Provider*

CNCF *Cloud Native Computing Foundation*

D-GRDL *D-Grid Resource Description Language*

D-Grid *German Grid Initiative*

D-MON *D-Grid Monitoring Project*

DEISA *Distributed European Infrastructure for Supercomputing Applications*

DGI *German Grid Initiative*

DHCP *Dynamic Host Configuration Protocol*

DMTF *Distributed Management Task Force, formerly "Desktop Management Task Force"*

DOA *Digital Object Architecture*

DONA *Digital Object Numbering Authority*

DSML *Directory Services Markup Language*

EaaS *Everything as a Service*

EC2 *Elastic Compute Cloud*

EGEE *Enabling Grids for E-sciencE*

EGI *European Grid Infrastructure*

EGI-Engage *Engaging the Research Community towards an Open Science Commons*

EGI-InSPIRE	*Integrated Sustainable Pan-European Infrastructure for Researchers in Europe*
EMI	*European Middleware Initiative*
ePIC	*European Persistent Identifier Consortium*
ETL	*Extract, Transform, Load*
FIA	*Future Internet Assembly*
GB	*Gigabyte*
GENI	*Global Environment for Network Innovations*
GGF	*Global Grid Forum*
GIIS	*Grid Information Index Service*
gLite	*Lightweight Middleware for Grid Computing*
GLUE	*Grid Laboratory Uniform Environment*
GLUE v1.1	*Grid Laboratory Uniform Environment Version 1.1*
GLUE v1.3	*Grid Laboratory Uniform Environment Version 1.3*
GLUE v2.0	*Grid Laboratory Uniform Environment Version 2.0*
GLUE v2.0 Schema	*Grid Laboratory Uniform Environment Schema Version 2.0*
GMA	*Grid Monitoring Architecture*
GOCDB	*Grid Operations Centre DataBase*
GRDB	*Grid Resource DataBase*
GRIS	*Grid Resource Information Service*
GRRS	*Grid Resource Registry Service*
GSI	*Grid Security Infrastructure*
GT	*Globus Toolkit*
GT4	*Globus Toolkit 4*
GWES	*Generic Workflow Execution Service*
GWorkflowDL	*Grid Workflow Description Language*
IaaS	*Infrastructure as a Service*
IANA	*Internet Assigned Numbers and Names Authority*
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
IG	*Instant-Grid*
ITU	*International Telecommunication Union*
JKS	*Java Key Store*

JSDL *Job Submission Description Language*

JSON *JavaScript Object Notation*

JSR *Java Specification Request*

KPI *Key Performance Indicator*

LDAP *Lightweight Directory Access Protocol*

LDIF *LDAP Data Interchange Format*

LRMS *Local Resource Management System*

MB *Megagabyte*

MDS2 *Metacomputing Discovery Service*

MDS4 *Monitoring and Discovery Service version 4*

MPI *Message Passing Interface*

NAS *Network Attached Storage*

NAT *Network Address Translation*

NFS *Network File System*

NGI *National Grid Initiative*

NIST *National Institute of Standards and Technology*

OCC *Open Commons Consortium, formerly the Open Cloud Consortium*

OCCI *Open Cloud Computing Interface*

OCI *Open Container Initiative*

OGF *Open Grid Forum*

OGSA *Open Grid Services Architecture*

OGSA-DAI *Open Grid Services Architecture - Data Access and Integration*

OSG *Open Science Grid*

P2P *Peer-to-Peer*

PaaS *Platform as a Service*

PAP *Policy Administration Point*

PBS *Portable Batch System*

PDP *Policy Decision Point*

PEP *Policy Enforcement Point*

PIP *Policy Information Point*

PKCS12 *Public-Key Cryptography Standards #12*

PKI *Public Key Infrastructure*

PMA *Policy Management Authority*

POV-Ray *Persistence of Vision Raytracer*

PXE *Preboot Execution Environment*

RA *Registration Authority*

RDA *Research Data Alliance*

REST *Representational State Transfer*

RGID *Reliable Grid Information Database*

R-GMA *Relational Grid Monitoring Architecture*

RSL *Resource Specification Language*

RSS *Rich Site Summary, formerly "RDF Site Summary", often called "Really Simple Syndication"*

S3 *Simple Storage Service*

SaaS *Software as a Service*

SLA *Service Level Agreement*

SOA *Service Oriented Architecture*

SOC *Service Oriented Computing*

SQL *Structured Query Language*

TFTP *Trivial File Transfer Protocol*

TSI *Target System Interface*

TTL *Time-To-Live*

UCC *UNICORE Command Line Client*

UI *User Interface*

UMD *Unified Middleware Distribution*

UML *Unified Modeling Language*

UNICORE *UNiform Interface to COmputing REsources*

UNICORE6 *UNiform Interface to COmputing REsources v6*

UR *Usage Record*

URC *UNICORE Rich Client*

URI *Uniform Resource Identifier*

URL *Uniform Resource Locator*

USB *Universal Serial Bus*

VO *Virtual Organization*

VOMS	*Virtual Organisation Membership Service*
VOMRS	*Virtual Organisation Membership Registration Service*
WN	*Worker Node*
WSRF	*Web Services Resource Framework*
XaaS	*Everything as a Service*
XML	*eXtensible Markup Language*
XPath	*XML Path Language*
XQuery	*XML Query Language*
XSL	*eXtensible Stylesheet Language*
XSLT	*eXtensible Stylesheet Language Transformation*
XUUDB	*UNICORE User Database*

1. Introduction

Research opportunities and outcomes in a broad range of scientific disciplines are nowadays directly dependent on the quality and availability of computing and storage resources. Well established research infrastructures are for example commonly utilized in the arts and humanities [27, 136, 137, 96], astrophysics [12, 14], biology [44], climatology [24, 39], computational chemistry [205], environmental sciences [104, 124], health and life sciences [53, 144, 67, 40], and high energy physics [192].

Research infrastructures are designed to provide innovative technologies as well as to satisfy the growing resource demand of universities and research institutes. They offer research services to users across the globe and help to form scientific and scholarly communities by granting single sign-on access for all community members. Furthermore, research infrastructures facilitate citizen science, i.e. the participation of amateur scientists in the research process. The increasing number and scope of international and interdisciplinary research collaborations have led to a growing demand for distributed research infrastructures, which provide a federated network of service centers, data repositories and knowledge hubs. Distributed research infrastructures are in the focus of this work; they are designed to provide seamless access to computing and storage resources owned by different organizations and providers. However, distributed research infrastructures can only partly fulfill the technical requirements for international research collaborations because of their large heterogeneity.

Various computing paradigms have been utilized to build up research infrastructures [115, 187]. Grid and cloud systems are nowadays the most commonly used computing paradigms. While cloud computing's services available on demand and on a self-service basis have become increasingly significant in recent years, grid systems still play an important role in computational research. The importance of grids can on the one hand be attributed to long term investments in grid systems. On the other hand, grids are more suitable for some application profiles: Grids provide a better support of the traditional, batch system-based computation and data-intensive applications by integrating high performance computing and storage systems via high-speed interconnect networks.

The integration of paradigms leads to hybrid infrastructures [189], which can combine the advantages of both systems. For instance, there are frameworks, that support instantiating of customized computing grids on a cloud infrastructure [107, 191, 224], but research has also been conducted to provide solutions for the reverse case, where a cloud infrastructure is built on grid or cluster resources [139, 189]. Furthermore, cloud computing aims to play a major role in the Internet of Services, by allowing on-demand deployment of applications, services, platforms, and infrastructures [162]. With the emerging need for cooperation between distributed research infrastructures [106], there is a need for convergence: The competing computing paradigms have several similarities, and their common challenges need to be addressed. Later in this work we provide an overview of the computing paradigms utilized to build up distributed research infrastructures, as well as the technology trends and the research advances behind these paradigms.

Distributed research infrastructures are distributed systems that offer a single system view by using middlewares to connect the different software components of a distributed system. Middlewares manage and coordinate the resources of a research infrastructure and fulfil a range of essential functions regarding security, data management, job execution, information systems, and monitoring.

The ultimate distributed system concept promises a single system view and seamless access to resources [51, 213]. This vision has been realized in different ways by various research domains, and specific solutions for these domains have evolved. As a consequence, we face a situation where the research infrastructures achieve the single system view by deploying slightly or substantially different middlewares. The following example shows the heterogeneity of research infrastructures, even if the infrastructures are based on the same computing paradigm: While the high energy physics community utilizes the gLite [97] middleware, the astrophysics community uses the GT [79] middleware, and computational chemistry projects are based on the UNICORE [221] middleware. Additionally, one scientific domain can utilize various application-specific versions of the same middleware. We refer exemplary to the various life science communities and the slightly different versions of the GT middleware they utilize [144, 143, 67, 40]. The heterogeneity of middleware utilization is further increased by the fact that the same middleware can be utilized either in a domain-specific way [205], or to access continent-wide supercomputing infrastructures [59, 180].

Today's distributed research infrastructures are isolated legacy systems with no or minimal interoperability, where a common access is usually obtained by parallel deployment of different middlewares, like in the case of the German e-Science Initiative [164] or the Pan-European e-Infrastructure EGI [72]. Interoperability of research infrastructures is however important, especially since (1) the different research disciplines are more and more approaching each other in order to answer increasingly complex research problems [106], or (2) federated environments aim to support dynamic expansion of capabilities and scaling of applications [37, 63]. Establishing interoperation between research infrastructures is furthermore important for preventing vendor lock-in and enabling the seamless change of a resource provider. At the level of the computing paradigms, interoperability needs to be achieved through the provision of a unified infrastructure view and a unified access to the resources. At the level of the implementation, an interoperation of the different middlewares is essential. The original goal of distributed research infrastructures - a uniform access to resources - should not be forgotten. A community-aware authorized access is of particular importance in distributed research infrastructures, where users and providers do not know each other and the usage of the infrastructure is based on mutual trust. All services and infrastructures should therefore guarantee that users only obtain access to the subset of resources dedicated to their community.

Interoperability between the basic components (e.g. security, job execution, data management, and information systems) is the first step to reach uniform research infrastructures. Exchanging resource descriptions, discovering resources and monitoring services are tasks of the information- and monitoring systems. These tasks are essential for executing jobs or managing data in distributed research infrastructures. The focus of this dissertation lies on the interoperation of the information- and monitoring systems. We chose this focus, because information- and monitoring systems are of high importance for all other components of a distributed research infrastructure. Neither we aimed to deliver community-specific solutions, nor we designed monitoring systems related to the recent technology trends. Much rather this work addressed the common challenges of the competing computing paradigms due to identifying several similarities and the research progresses behind these paradigms.

We continue with the problem statement and the open research questions we address in this work.

1.1. Problem Statement

Here we just give an overall view on the problem space we address and we provide the list of open research questions we answer. We show our motivation in Chapter 2 extensively, and we give there a detailed description of the weaknesses of recent monitoring and information systems in distributed research infrastructures. We also show that those are a hindrance for interdisciplinary work.

The list of open research questions for which this thesis presents a solution:

- In the context of **uniform representation** of the information:
  - What kind of information is needed to *describe the resources* in distributed research infrastructures?
  - How to *model the information*?
  - How to choose a generally accepted *data model* for exchanging detailed resource information?
- In the context of **uniform view to all data** via community aware, *authorized access*:
  - *From what sources* can resource information and monitoring data be collected?
  - How to design an automated monitoring process and a *monitoring architecture* supporting it?
  - How to *authorize access* to resource descriptions and monitoring data on a community-aware manner?
- In the context of **automated** system deployment and **provisioning**:
  - What technical *concepts* can allow to set up a self-configured and independent multi site and multi user distributed research infrastructure?
  - How can such environments be *connected with productional* distributed research infrastructures?
  - Which *customization concepts* can provide a suitable system design for automated deployment of software components?
- In the context of additional information about **resource quality** and performance:
  - What *metrics and indicators* are important for describing and measuring resource quality in distributed research infrastructures?
  - How can the the quality information be *monitored, published, and exchanged*?
  - *From what sources* can the resource quality information be collected?
  - How can the quality information be described and modeled with the *generic data schema*?

The goal of this work is neither delivering community-specific solutions, nor designing information- and monitoring systems related to the recent technology trends. Much rather we address the common challenges of the competing computing paradigms due to identifying several similarities and the research progresses behind these paradigms (Section 2.4). That way we can focus on a generic interoperable monitoring architecture for distributed research infrastructures which is based on abstract requirements and standards. It is also independent from the specific middleware implementations and extensible for future distributed computing paradigms.

In the following Section, we continue with summarizing the contributions of this thesis.

1.2. Contribution

This thesis improves the interoperability of distributed research infrastructures through enabling the exchange of resource descriptions based on a generic data model.

We present a homogeneous monitoring- and information system for our application domain that is independent from the middlewares and extensible for future distributed computing models. The presented solution does not require changes in the existing distributed research infrastructures.

The contributions of this dissertation are the following:

- Abstract requirements for a generic information model are collected to monitor distributed research infrastructures in an interoperable manner.[1]

[1]We collected the initial requirements together with colleagues from the D-MON project.

- A discussion of information models, monitoring- and information systems, and middlewares recently utilized in our application domain to exchange resource information.[2]
- A generic interoperable monitoring architecture for distributed research infrastructures is presented.[3]
- The *Extract, Transform, Load* (ETL) data-warehousing technique is adapted to integrate monitoring data from heterogeneous sources.
- Exemplary mappings into a generic information model are provided for the most widely used information models.
- A method to discover heterogeneous resources in distributed research infrastructures is described.[4]
- A concept is presented to describe, monitor, and publish the quality and performance of resources.[5]
- A solution for automated system deployment, which allows to set up a self-configured multi site and multi user distributed research infrastructure.[6]
- A framework is developed to simulate heterogeneous information services.[7]
- A prototypical implementation of an interoperable information- and monitoring system is developed.[4]
- Case studies are made to evaluate the concept.

1.3. Impact

The impact of this dissertation in theory and practice can be seen in the following list of publications. The results of this work have been peer-reviewed and published in journal articles as well as in several international workshop and conference proceedings:

- NRIN Vol. 20(1-2): Kálmán, T.; Tonne, D.; Schmitt, O. *Sustainable Preservation for the Arts and Humanities*, 2015
- IEEE CALS2011: Kálmán, T. *Assessment of Resource Quality for Service Level Agreements in Life Science Grids*, 2011
- IAS Vol. 5: Kálmán, T. and Rings, T. *A UNICORE-based Multi Site and Multi User Grid Environment for Demonstration, Education, and Testing Purposes*, pages 1-10, 2010
- IEEE EDUCON2010: Rings, T.; Aschenbrenner, A.; Grabowski, J.; Kálmán, T.; Lauer, G.; Meyer, J.; Quadt, A.; Sax, U. and Viezens, F. *An Interdisciplinary Practical Course on the Application of Grid Computing*, pages 149-155, 2010
- CHEP2010: Kálmán, T. *Extension of a Traditional Monitoring Framework to Support Efficient Monitoring of Stateful Grid Services (short paper)*, 2010
- CGW08: Baur, T.; Breu, R.; Kálmán, T.; Lindinger, T.; Milbert, A.; Poghosyan, G. and Romberg, M. *Adopting GLUE 2.0 for an Interoperable Grid Monitoring System*, pages 149-155, 2009

[2]We describe our application domain and the utilized computing paradigms in Chapter 2.

[3]We designed the initial architecture together with colleagues from the D-MON project. The German Grid Initiative, furthermore, adopted our results as its grid monitoring architecture.

[4]Our results were adopted as a production level service by the German e-Science Initiative.

[5]As a use case we chose the german health and life science communities. We proposed our results to the health and life science communities for uptake.

[6]Our results were adopted to deploy productional services for the European Persistent Identifier Consortium, as well as for the DARIAH infrastructure.

[7]We used and enhanced the results of the Instant-Grid project, on which we previously worked together with colleagues. Our results were applied as a teaching environment for practical courses at the University of Goettingen, Germany and at the Monash University, Australia.

- JGC Vol. 7(3): Baur, T.; Breu, R.; Kálmán, T.; Lindinger, T.; Milbert, A.; Poghosyan, G.; Reiser, H. and Romberg, M. *An Interoperable Grid Information System for Integrated Resource Monitoring based on Virtual Organizations*, pages 319-333, 2009

Contribution to a book chapter was made with relation to this dissertation:

- Hahn, H.; Kálmán, T.; Kohlmann, W.; Kollatz, T.; Neuschäfer, M.; Pielström, S.; Puhl, J.; Stiller, J.; Tonne, D. *Handbuch Digital Humanities*, ISBN 978-3-7375-6818-0, 2015

Several not peer-review articles and technical reports were written and several workshop talks were given in the context of this thesis:

- Kálmán, T.; Kong, X.; Schwardmann, U. *Die digitale Forschungsinfrastruktur DARIAH-DE: Angebotspalette für die Geistes- und Kulturwissenschaften. Bibliothek Forschung und Praxis, 40(2), pp. 234-243*, 2016
- Harmsen, H.; Kálmán, T.; Wandl-Vogt,E. *DARIAH meets EGI. Inspired, 19*, 2015
- e-IRG2014: Kálmán, T.; Wandl-Vogt,E. *DARIAH-ERIC: Towards a Sustainable, Social and Technical European e-Research Infrastructure for the Arts and Humanities*, [Abstract], 2014
- LZA2012: Kálmán, T.; Kurzawe D.; Schwardmann U. *European Persistent Identifier Consortium - PIDs für die Wissenschaft*, pp. 151-168, 2012
- ALLHAND10: Kálmán, T. *Information about the Grid Environment at GoeGrid*, 2010
- MONWS09: Kálmán, T. *Überwachung von Globus Toolkit v4.x Diensten mit Nagios*, 2009
- MONWS09: Lindinger, T. and Kálmán, T. *D-MON: Site und System Monitoring im D-Grid*, 2009
- MONWS08: Kálmán, T. *Reliable Grid Information Database (RGID) and the Jawari Interface*, 2008,
- MONWS07: Kálmán, T. *[D-]Grid Resource Definition Language (D-GRDL) and the Reliable Grid Resource Database*, 2007
- Baur, T.; Kálmán, T.; Lindinger, T.; Milbert, A.; Poghosyan, G. and Romberg, M. *Middleware-Übergreifendes Monitoring: Evaluierung und Auswahl von Komponenten*, D-Grid Report, 2008.

The author identified the topics and co-supervised one Master thesis and one student's project with relation to the overall topic of this thesis:

- Al-Taheri, K. *A Namespace and User Management Service for Virtualized Persistent Identifier Systems*, Master's Thesis, 2016
- Al-Taheri, K. *Prototyping a User Management Service for Persistent Identifier Systems*, Student's Project, 2015

Furthermore, the tools and technologies created during the development of this thesis have been developed further. The Nagios Plugins for Globus Toolkit, for instance, were actively used for interoperable monitoring of the *German Grid Initiative* (D-Grid). The Discovery Service for Grids was also adopted as a production level service in D-Grid. An other example is a special edition of Instant-Grid, which is applied as a teaching environment for practical courses at the University of Göttingen, Germany and at the Monash University, Australia. It is also used as a demonstration environment for student projects and a master thesis. Furthermore, our solutions for automated middleware deployment and for self-configured demonstration environments were adopted to deploy productional services for the European Persistent Identifier Consortium, as well as for the DARIAH infrastructure.

1.4. Thesis Structure

This thesis is structured in the following way. In Chapter 2, we describe the motivation of our work and the scope of this thesis. We provide a brief overview of the computing paradigms utilized to build up distributed research infrastructures, as well as the technology trends and the research advances behind them. Following that, we shift our focus to the common issues and similarities of these concepts when it comes to exchanging resource information. Afterwards, we highlight a number of research questions we try to answer.

Chapter 3 presents our 5-steps information modeling process and discusses the modeling of heterogeneous resource information in a generic way. First, we identify the information required for exchanging resource descriptions based on a requirement analysis of application scenarios, which is our first contribution. For modeling resource descriptions in distributed research environments many information schemas and data models exist. In addition to the theoretical analysis, we discuss the information schemas recently being utilized. We also present a theoretical approach for a schema mediation process for distributed research infrastructures, which is the next achievement of our work.

In Chapter 4, we continue our work by using the results of our theoretical analysis regarding the information demand in distributed research infrastructures, as well as the theoretical approach for a schema mediation process, and we develop a proof of concept for grid environments. We design an automated resource description exchange process and a respective generic monitoring architecture supporting it, which is our next contribution.

Within Chapter 5, we describe a solution for automated middleware deployment and self-configured demonstration environments, that allows to set up a self-configured and independent multi site and multi user distributed research infrastructure. We present the technical concepts including the automatic configuration, ready-to-use features, and applications. We applied this environment as our simulation framework, but it is also suitable for demonstrating, developing, and testing purposes of distributed computing environments.

Describing, monitoring, and publishing the quality and performance of resources enables to perform essential management activities in distributed research infrastructures. This is the focus of Chapter 6. To demonstrate the practical relevance of our work, we chose the German life science communities. We start by describing their quality metrics, afterwards we identify further key performance indicators that can be used by the life science communities for defining service levels that can be agreed on. We discuss how information systems can publish and exchange the quality information, as well as how the information model can handle them. Finally, we present two approaches to measure and monitor the quality metrics in multi-middleware environments: an external benchmarking system and traditional monitoring system.

Lastly, in Chapter 7 we summarize the thesis, discuss its contributions and findings, and we also point out the limitations of the current work. Beyond that, we outline possible research items which extend or refine the results and methods presented in this thesis, and we state directions for future work.

2. Motivation

In this chapter we describe the motivation of our work and the scope of this thesis. We provide a brief overview of the computing paradigms utilized to build up distributed research infrastructures, as well as the technology trends and the research advances behind them. Following that, we shift our focus to the common issues and similarities of these concepts when it comes to exchanging resource information. Afterwards, we highlight a number of research questions we try to answer through this thesis.

Since the research problems are increasingly complex, recently, the disciplines began making advances to each other. Few dispute that interdisciplinarity and the resulting cooperation are changing the way in which we engage in the research process. This development, of course, also affects the supporting technology which backs the infrastructures. Indeed, research infrastructures are required to bring resources together and they also need to enable cooperations between their users [106].

From the technical point of view it means that a research infrastructure should provide the same or complementary functionality as another research infrastructure. Since the various distributed research infrastructures were built up by using various paradigms, we shift our focus to the similarities of the competing computing paradigms. That way, we can address their common challenges.

In the following, we first give a brief overview of the main computing paradigms, which are used to build up distributed research infrastructures, as well as the technology trends and the research advances behind them. The examined paradigms are: cloud computing, cluster computing, future internet, grid computing, market-oriented computing, peer-to-peer computing, service-oriented computing, (web)service computing, and utility computing. Although we do not provide detailed descriptions of autonomy oriented computation, jungle computing, meta computing, sky computing, and volunteer computing for the sake of brevity, we give references to further reading. Following that, in Section 2.2, we continue with analyzing the systems we apply in the case studies. Principally, they contain grid and cloud systems. Afterwards, Section 2.3 introduces the common challenges of exchanging resource information and monitoring in such systems. This is the basis of our work's problem definition, which we present in Section 2.4.

2.1. Computing Paradigms in Distributed Research Infrastructures

In the following, we give a brief overview of the main concepts of distributed resource sharing which are used to build up distributed research infrastructures. The list of examined paradigms is arranged fully in alphabetical order and neither sorted by any priority nor in chronological order.

cloud computing The cloud computing paradigm [155] is based on a service provisioning model, which foresees services available on demand and on a self-service basis. The users of the cloud systems access the services whenever and wherever they need them, like it is in the case of other utilities. The payment is also made in the same way: users need to pay service providers when they use the services of the clouds. The usage is monitored and accounted on a transparent way for both the consumer and provider. This reduces the operational costs and consumers no longer need to set up and maintain an own complex infrastructure. We examine cloud computing in details in Subsection 2.2.2.

cluster computing Cluster computing utilize parallel and/or distributed computing techniques to solve computationally intensive tasks across networks of computers. For that purpose, cluster computing relies on operationally independent computers (nodes). The nodes are similar type of machines tightly-coupled and interconnected by fast, dedicated networks.

The nodes and computing tasks are managed by a local resource and queue management system, which provides a single system view to the users. The cluster users do not use the worker nodes directly, but they submit jobs to the resource and queue management systems, which distribute the computing tasks to the nodes depending on scheduling policies and on the characteristic of the tasks. Cluster systems are usually not directly accessible from the Internet, since they are deployed in private networks inside one administrative domain. The interfaces of local resource and queue management systems provided for users and client applications are proprietary and not standardized. Since extensive research has been done on cluster computing, we do not examine the local resource and queue management systems in details here. We refer the interested reader to [16, 75, 177].

Future Internet Several initiatives enable research on distributed and networked systems with the goal to design new network architectures and evaluate the Future Internet. Such exemplary initiatives are *Global Environment for Network Innovations* (GENI) [23], *Future Internet Assembly* (FIA) [76], and ITU-T Study Group 13 on Future networks including cloud computing, mobile and next-generation networks [121].

The Future Internet aims to be more secure, manageable, scalable, efficient, better at handling mobile nodes, and many Future Internet's features and functions are based on virtualised resources [91]. The virtualisation of resources has been a topic for a long time. The Future Internet benefits from the use of virtual components, such as virtual networks and virtual machines. Utilising the virtual components it is easy to assemble higher level services and automate service provisioning. However, the management of services and their related *Service Level Agreement* (SLA)s through the complete service lifecycle still remains challenging in the Future Internet [35].

grid computing The computing grid is used on the analogy of the electrical power grid. Fosters et al. describes a vision in [84], where on-demand computational power is delivered to the consumers by networks similarly to the electrical power network. Further analogies are: the reliability of the electrical grids and the easy usage of the grids.

Grid computing aims to support increasingly large networks and the integration of many heterogeneous, dynamic *virtual organizations*. To provide the necessary software infrastructure for distributed resource sharing, collaborative efforts were undertaken [72, 214], and generic grid libraries and grid middlewares were developed [79, 97, 221].

While other computing paradigms have been increasingly significant, grid systems are still utilized and play an important role in computational research. This importance does not simply depend on the long-term funding, but there are application profiles that are more appropriate for grids, since grids better support the traditional, batch system-based computation and data intensive applications by integrating high performance computing and storage systems via high-speed interconnect networks. The grid computing paradigm is examined in details in Subsection 2.2.1.

market-oriented computing Distributed computing resources and their users can also be referred as producers and consumers of an artificial economy. Accordingly, their individual activities can be seen in terms of production and consumption of commodities. This is the basic idea of market-oriented computing, which applies economic models to allocate distributed computing resources efficiently [74, 226].

In this way, economic theories helps the equilibrium on the market by controlling the supply and demand of distributed computing resources. Furthermore, scheduling approaches can be considered as computing the competitive equilibrium of an artificial economy. Economic principles give the basis of decisions to maximize resource utility or resource consumption by varying optimization objectives, and different demands, including different price models.

peer-to-peer computing *Peer-to-Peer* (P2P) computing [161] uses a highly distributed and decentralized network architecture, where interconnected nodes (peers) share resources amongst each other. The individual nodes in the network are considered equal and usually consumers nabs suppliers of available resources at the same time. The shared resources are directly available to other participating peers and therefore, no centralised management is necessary.

The P2P concept was utilized in many application domains, but it became widely used by file sharing systems and online music services for unlicensed content. Although it has created much controversy over legality and fair use, P2P computing has recently emerged as a viable business model for the distribution of digital objects [33]. The productive research and developement in addition to the "initial file sharing applications has shown high resilience to failures, tolerance to network performance variations, and improved scalability to huge numbers of peers" [116]. It also increased the level of privacy and the failure tolerance by varying network performance.

service oriented computing, (web-)services computing *Service Oriented Computing* (SOC) and (Web-)Services Computing utilize services as technological building blocks to provide the capability for the distributed systems to cooperate. To develop applications *“and to build the service model, service oriented computing relies on the* Service Oriented Architecture *(SOA), which is a way of reorganizing software applications and infrastructure into a set of interacting services.”* [174] The web services allow businesses to create, discover, publish, and utilize business functions over the Internet. Methods exist to allow the automatic composition of web services [186], which greatly helps to integrate and manage the business processes dynamically.

utility computing Utility computing is a concept which is certainly not new. Providing computing services similar to other usual essential services (utilities) still stayed the basic idea of utility computing, even though the concept of distributed resource sharing evolved with the technologies.

Before networked businesses and the always-on Internet were widely available, rather models based central facility concepts existed. The designers of an operating system predicted in 1965 a computer facility operating as a utility, *“like a power company or water company“* [50]. In 1966 it was envisioned, that the computer industry would come to resemble a public utility *“in which many remotely located users are connected via communication links to a central computing facility“* [175].

The birth of the *Advanced Research Projects Agency Network* (ARPANET) in 1969 was the beginning of internetworking standalone computing facilities and this led to a worldwide system of computer networks, to the Internet. Scientists spotted well the new possibilities of emerging computer networks and started to predict the distributed manner of the utility computing. For example, [142] says: *“As of now, computer networks are still in their infancy. But as they grow up and become more sophisticated, we will probably see the spread of ‘computer utilities’ which, like present electric and telephone utilities, will service individual homes and offices across the country“*. This vision involves computer utilities, which can be accessed whenever and wherever consumers need them, like it is in the case of any other utility service. The computer utility infrastructures are provided using “invisible” networks and their users pay to the service providers when they use the services of the infrastructure.

Other paradigms Further distributed computing paradigms exist, which are not examined here for the sake of brevity. We refer the interested reader to *autonomy oriented computation* [126], *complete computing* [125], *jungle computing* [201], *meta computing* [209], *sky computing* [140, 176] or *volunteer computing* [197].

Extensive research has been done, whether computing can become established as a new utility by making use of the mentioned paradigms. We refer exemplary to [82] for grids, to [38] for clouds, and to [116] for peer-to-peer computing. Foster says in [82] that *"a combination of technology trends and research advances make it now feasible to realize the Grid vision and to put in place both a new international scientific infrastructure and new tools based on that infrastructure that can, together, meet the challenging demands of 21st Century science"*. Buyya et al. point out that *"Cloud computing is a new and promising paradigm delivering IT services as computing utilities"* [38]. Iamnitchi et al. found that *"Peer-to-Peer Computing has established itself ... in the general area of distributed systems"* and *"peer-to-peer computing is associated with inherently decentralized, self-organizing, and self-coordinating large-scale systems"* [116].

With the emerging need for cooperation of distributed research infrastructures, there is a need for convergence: the competing computing paradigms have several similarities and their common challenges have to be adressed.

2.2. Infrastructures Under Study

In this Section we examine the grid and cloud paradigms, which are recently utilized to set up distributed research infrastructures we analyzed and applied in the case studies.

2.2.1. Grid Infrastructures

The computing grid is used on the analogy of the electrical power grid. Fosters et al. describes a vision in [84], where on-demand computational power is delivered to the consumers by networks similarly to the electrical power network. Further analogies are: the reliability of the electrical grids and the easy usage of the grids.

Grids became one of the central pillars of recent distributed research infrastructures due to long-term investitions. Well established grid systems are commonly utilized in a broad field of scientific domains, such as high energy physics [231], astrophysics [12, 14], life sciences [40], [144], medicine [143], [67], biology [44], computational chemistry [205], climatology [24], [39], agriculture [202] or humanities [96], [215].

Grid computing aims to support increasingly large networks and the integration of heterogeneous, dynamic *virtual organizations* [85]. To provide the necessary software infrastructure for distributed resource sharing, collaborative efforts were undertaken [72, 214], and generic grid libraries and grid middlewares were developed [79, 97, 221].

The grid middlewares provide single system view and seamless access to computing and storage resources, which are owned by different organizations. In recent years different grid system implementations were evolved to support the original concept of the grid. As a consequence, we face a situation nowdays, where the grid initiatives achive the single system view by deploying slightly or strongly different grid middlewares.

Grid middlewares offer security services, which provide authentication and authorization functionalities. Grid systems require personalised user certificates for running computational tasks or managing data transfers. The grid resources (hosts and services) are also certified, however, a common approach that the grid services run on one host are identified by the same host certificate. The user certificates and host (service) certificates are issued by a registered national grid *Certification Authority* (CA). The grid users and the owners of grid resources need to identify themselves personally before a certificate is issued. To ease this process a CA may point out a *Registration Authority* (RA), which operates on the behalf of the CA and is usually located near to the users.

Since resource sharing in grid computing is usually not supported between individual grid users and resource providers directly, the grid users require membership in a *Virtual Organization* (VO). The membership authorises the users for using the resources of the VO.

The interfaces to the grid are: the VO usually runs a dedicated *User Interface* (UI) machine or a user-friendly grid portal to job submission and data management transactions.

A grid-based research infrastructure may provide different software environments and hardware architectures. The applications therefore need to be prepared for the target software and hardware environments. Since the target *Worker Node* (WN) of a grid site is usually not accessible directly, the grid sites often run a UI machine, which has an identical software and hardware environment like the WNs. The applications can be pre-compiled on the UI machine and the distribution of the compiled application to each target WN of the grid is part of the grid jobs. An other common approach is to distribute the source code to the target WN, where the application is compiled before execution. In this case, the source code transfer and the compilation are parts of the grid jobs.

The computational tasks and data transfers are described using a middleware specific language, like *Resource Specification Language* (RSL) or standardised languages, like *Job Submission Description Language* (JSDL). This job description is submitted to the grid middleware, which enables further job management, including monitoring or cancellation of grid jobs and data transfers.

While other computing paradigms have been increasingly significant, grid systems are still utilized and play an important role in computational research [72, 214, 94], as well as business models for the industry exist [109, 198]. This importance does not simply depend on the long-term funding, but there are application profiles that are more appropriate for grids, since grids better support the traditional, batch system-based computation and data intensive applications by integrating high performance computing and storage systems via high-speed interconnect networks.

Even though several workarounds try to ease the certificate management for non-experts [88, 105, 166], the security model of grids [225] is still very complex [61]. That fact significally lowered the acceptance of grids. Grid computing, however, addresses several important issues of distributed computing [42] and some of the concepts have turned out well and have been widely accepted. We exemplary point out the results of the research work made for a more reliable, secure, uniform, and high performance file transfer [3, 141]; as well as the concept of sharing resources within a VO [85].

2.2.2. Cloud Computing

More recently, many research communities are opting for pay-per-use models. Distributed computing paradigms, which support this business model, have also often been the basis of research infrastructures. Cloud computing is one of these concepts. It is based on a pay-per-use business model, which makes it possible that consumers pay when they use the cloud infrastructures. That way research communities can reduce their operational costs, because they neither need to set up all complex infrastructure components, nor they need to maintain and operate them on their own. Cloud computing's provisioning model foresees services available on demand. The users of the infrastructures access the services whenever and wherever they need them. Cloud computing has emerged as a popular computing model of distributed research infrastructures [106, 133, 187, 191, 193, 224, 233].

During the recent years cloud systems have been widely deployed not only by research institutes, but also by the industry; which enables to bring distributed resource sharing to the public.

The *National Institute of Standards and Technology* (NIST) identifies in [155] the following properties for cloud systems: on-demand self-service, broad network access, resource pooling, rapid elasticity, measured service. An overview of these characteristics is shown in Table 2.1.

Property	Description
On-Demand Self-Service	The consumers instantiate services on an automated, self-service basis. The computing, storage, and network capabilities are provided according to the demand of consumers.
Broad Network Access	The consumers access the cloud services using common interfaces over the network. The services are accessed by various client platforms.
Resource Pooling	The physical resources of a cloud provider are pooled dynamically into virtual resources, which are utilized by multiple consumers in a multi-tenant manner. The comsumer has no detailed control over the location of the provided capabilities.
Rapid Elasticity	The provided virtual resources are allocated and released rapidly according to the demand. Capabilities often appear without boundaries (any time, any quantity).
Measured Service	Cloud systems utilize a metering capability to control and optimize resources automatically (pay-per-use). The usage is monitored and accounted on a transparent way for both the consumer and provider.

Table 2.1.: Essential characteristics of the Cloud model defined by NIST [155]

According to the most popular classification of service layers in cloud systems [155], 3 layers are identified: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), and *Software as a Service* (SaaS). The term *Everything as a Service* (XaaS) or *Everything as a Service* (EaaS) is often used to sum the service layers up. Figure 2.1 illustrates these layers.

On the IaaS level virtualized resources, e.g., computing, storage, and network services, are offered. The infrastructure can be managed and monitored by cloud infrastructure APIs. An example for one of the first commercial cloud offers on the IaaS layer is *Amazon Web Services* (AWS) [4], where Amazon provides resizable, virtualized compute (*Elastic Compute Cloud* (EC2)) and storage (*Simple Storage Service* (S3)) capacity. Rackspace [183] is an other well known company, which serves the hosting need of its consumers and promises reliable services delivered through its worldwide data centers built on virtualisation technologies. The PaaS layer provides cloud platform APIs for developers to control, load balance, provision, and manage their developer environments and deployed services. The developers have no control over the resources on the

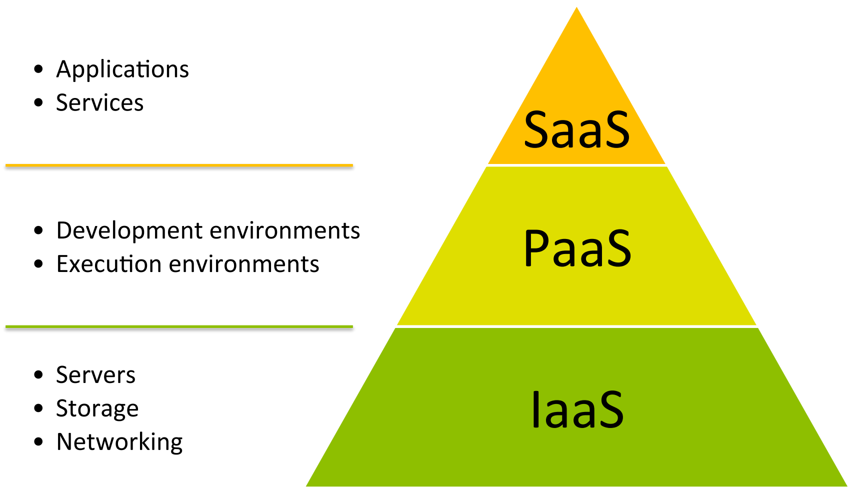


Figure 2.1.: XaaS model of Cloud Computing

infrastructure level. Examples for PaaS offers are Google App Engine [100], the cloud service platform Windows Azure [160], and Heroku (acquired by Salesforce now) [196]. The SaaS layer enables that consumers do not need to worry about installation, setup, and running of their applications. Web interfaces are provided for the end-users with similar functionalities as the local software installations. The SaaS layer enables the users to change the configuration of the applications, but they do not have control over the underlying infrastructures or platforms. The SaaS market is growing rapidly. Examplary services are: Google Apps (like Gmail, Google Docs, Google Drive, etc), and Microsoft Office 365[159].

Clouds can be deployed employing the following models: private, public, hybrid, and community clouds. The resources of public clouds are shared between multiple customers, while private clouds dedicate the resources to a single organisation or a single customer. Usually, the access to a private cloud is also restricted to a single organisation. Cloud environments, which use resources of both public and private clouds, are the hybrid clouds. A group of organisations, a community, can set up a community cloud, where the resources are shared between the participating parties. The access to a community cloud is provided to the participating organizations only.

The cloud computing paradigm gave to distributed research infrastructures a new potential and perspective [41, 13]. However, some of the research questions of distributed resource sharing still remains an important topic.

For a detailed comparison between grid computing and cloud computing, along with the challenges they face, and for an overview of their characteristics we refer to [195] and [87]. In the following section, we shift our focus to the similarities of the competing computing paradigms. That way, we can address their common challenges.

2.3. Exchanging Resource Information and Monitoring in Distributed Research Infrastructures

Distributed research infrastructure concepts became mature and large collaborative efforts were undertaken to set up production quality infrastructures, like country- and continent-wide grid infrastructures [11, 72, 73, 94, 164, 214], or domain-specific cloud infrastructures [137].

A reliable operation and usage of a distributed research infrastructure is increasingly dependent on monitoring. In the monitoring process, the status of a research infrastructure is observed, particularly its resources, its services, as well as its processes including user jobs. Research infrastructure monitoring is furthermore building the basis for checking and controlling the compliance of an infrastructure's service quality with the *Service Level Agreement* (SLA) made between provider and research community. Finally, monitoring allows for service benchmarking, correct accounting and billing, as well as scheduling of resource and service usage.

To fulfill its purpose, research infrastructure monitoring relies on data provided by information services. However, large infrastructures with stakeholders from different organizations often use incompatible technical realizations, apply heterogeneous data structures to encode their data, and offer different interfaces to provide it. Reason for this is either the absence of standards or the design of best practice approaches that neglect existing standards. The fact that many standards are still evolving, and that transitions to new versions must be incorporated into the infrastructures, is further complicating the situation. Figure 2.2 depicts such a scenario. The provisioning of high-quality infrastructure operations subsequently requires solutions for interoperation and integration of research infrastructure information services and monitoring data. In addition, monitoring data has to be available independently of the specific implementation or the organization that provides the access.

It is therefore important that the access control to the research infrastructures is community-specific. The various actors of the research infrastructures require that the data is specifically related to the services and resources they operate. Information services should therefore provision

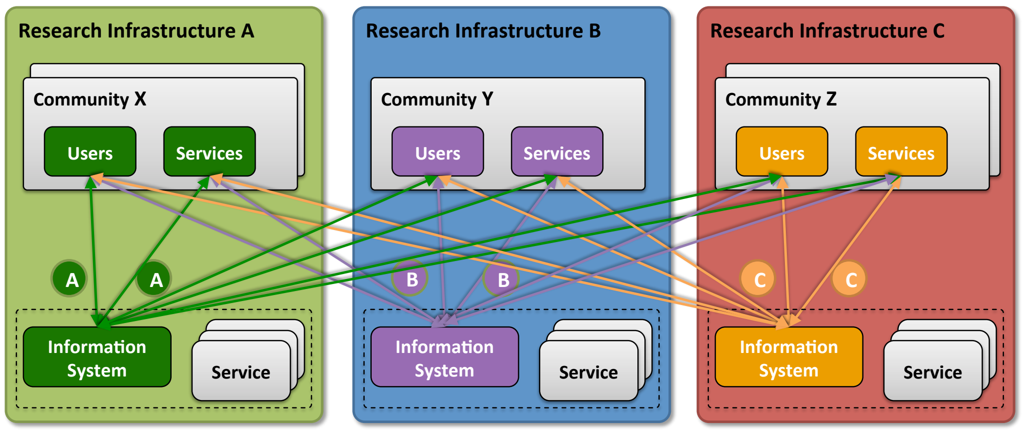


Figure 2.2.: Exchanging Resource Information in Distributed Research Infrastructures Without an Interoperable Information System

the gathered data in accordance with the organizational context of a user, i.e. the VO he or she is a member of. Such a mechanism is called *VO-awareness* in this work.

A common approach to obtain interoperability is the definition and adoption of common open standards [152] and architectures. This approach relies heavily on the standardization and implementation processes, which are often costly and politically charged because implementation and roll-outs involve different research communities. A coupling of the architectures is therefore a reasonable alternative in some scenarios.

In other scenarios, the inclusion of non-intrusive components such as bridges is necessary for the integration of research infrastructures. This is for example the case where there is no common standard, or where the parallel operation of multiple different implementations (e.g. different middlewares) is favourable. System integration has been in the focus of researchers for a long time. One example is the work of Hegering et al. [108], who describe (1) a multi-architectural platform, (2) management gateways and (3) multi-architectural agents as three different kinds of architectural bridges that can be applied. Our work concentrates on the realization of a management gateway as a bridge between different monitoring and information services. It is furthermore pointed out by Hegering et al. [108] that an integration of infrastructures can be achieved by

- bridging information (e.g. data description schema),
- bridging communication (e.g. interfaces),
- bridging organizational models (e.g. roles),
- and bridging functional models (e.g. queries).

In this work, we concentrate on building a bridge as a mean to realize successful data exchange and enable information and communication models to be compatible in an agile way. Such a bridge is furthermore facilitating the retrieval of monitoring data according to roles and organizations.

Several problems become apparent by analysing the distributed research infrastructure scenarios and concepts with respect to monitoring (cf. [18, 20, 47, 48, 138, 146, 223]). A major shortcoming of big infrastructures such as the research infrastructure set up by the German e-Science Initiative [2] is the creation of several autonomous (monitoring) service components, which might result in several logical infrastructures without data interchange. This loss of interoperability can lead to major problems in the operation of the research infrastructures, where the parallel deployment of multiple different implementations is favourable. We exemplary mention the case if a scheduler used in one middleware is unaware of resource allocations that belong to foreign middleware components. Another potential problem is that some parts of a complex job might rely on resources available in infrastructures, whose resources are managed by different middleware implementations and as a consequence resource information is not available for the schedulers. In this way, proper resource allocation is not possible and jobs cannot be mapped to the necessary resources, since the multiple sources provide only a fraction of the desired resource descriptions information. This substantially complicates the operation of comprehensive monitoring and information services. Obviously, in that situation a more homogeneous and comprehensive approach is desirable.

With the adoption of virtualization technologies new challenges were raised also for the operation of distributed research infrastructures. These challenges also imply the information systems and monitoring services [15, 47, 179, 222]. To allow the efficient operation of virtualized environments the monitoring components must be able to handle the characteristics of virtualized environments. Also the information systems should have a proper differentiation between physical- and virtualized resources. The up-to-date information on services and resources enables to design suitable provisioning strategies, which helps to avoid the violations of *Service Level Agreement*s (SLAs).

From the other point of view, a main disadvantage of most monitoring- and information services is their focus on physical entities, which ignores the actual mapping of resources and services onto research communities. Employing a community-centric approach would be more user-friendly, since only the information that belongs to their community would be extracted and presented to the users. Information that is irrelevant for the users, e.g. the status of the providers' cluster, would be filtered and not presented to the users. A further advantage of community-centric approaches is the possibility of community-based privacy protection.

To ensure a smooth job scheduling as well as the operation and maintenance of resources and services, all areas of the research infrastructure need to be interoperable. These areas include authentication and authorization, job execution, data transfer, scheduling services, as well as the the interoperability of the monitoring and information systems.

Integrated monitoring services, which dynamically provide exhaustive information about the actual state of components from multiple monitoring systems, are a step towards integrated and interoperating distributed research infrastructures. In the following we describe the problems we identified and address in this work.

2.4. Problems of Exchanging Resource Information and Monitoring in Distributed Research Infrastructures

Since each of the paradigms promise the unified view of resources, their services should also support providing a unified infrastructure view. In case of exchanging resource descriptions and monitoring information it means an unified view of data and information for the entire research infrastructure.

Therefore, the respective components, e.g. the monitoring- and information systems, should provide:

- uniform *representation* of the information,
- uniform view to *all data* via community aware, authorized access,
- *automated* system *deployment* and provisioning,
- *additional information* about resource quality and performance.

The uniform representation of information should hide the heterogeneity of data structures and different naming conventions, utilized by the various research infrastructures. It should present the different underlying monitoring systems as a homogeneous data source. The problems to be solved in this context are:

- What kind of information is needed to describe the resources in distributed research infrastructures?

- How to model the information?
- How to choose a generally accepted data model for exchanging detailed resource information?

The uniform data access should allow to get an overall view of the data provided by various data sources and research infrastructures. However, the community-aware, authorized access should present only information, which belongs to the community. The monitoring and information systems should filter out all information with no use to the users. In addition, this filtering should ensure privacy protection, based on community membership. We identified the following open questions in this context:

- From what sources can resource information and monitoring data be collected?
- How to design an automated monitoring process and a monitoring architecture supporting it?
- How to authorize access to resource descriptions and monitoring data on a community-aware manner?

A solution should allow automated system deployment, which supports to set up a self-configured and independent multi site and multi user distributed research infrastructure. It should also be suitable for demonstrating, developing, and testing purposes of distributed computing environments. The main objectives of this topic to be addressed are:

- What technical concepts can allow to set up a self-configured and independent multi site and multi user distributed research infrastructure?
- How can such environments be connected with productional distributed research infrastructures?
- Which customization concepts can provide a suitable system design for automated deployment of software components?

Among the integrated data and uniform access to the monitoring information, monitoring and information systems should describe, monitor, and publish additional information about the quality and performance of resources. The related problems to be solved are:

- What metrics and indicators are important for measuring resource quality in distributed research infrastructures?
- How can the quality information be published and exchanged?
- From what sources can the resource quality information be collected?
- How can the quality information be modeled with the generic data schema?

This work addresses these issues. Chapter 3 discusses the problems of system integration and lays the focus on information modelling, schema mediation, but it also presents a process for gathering resource information. In Chapter 4, an approach for a homogeneous monitoring and information system is described. The presented system effectively observes stateful services offered by the heterogeneous middlewares on a VO-aware manner. A solution for automated system deployment, which allows to set up a self-configured and independent multi site and multi user distributed research infrastructure, is described in Chapter 5. The question of how quality information about the resources of distributed research infrastructures can be provided in a standardized way is answered in Chapter 6.

This chapter described the motivation of our work and the scope of this thesis. First, we provided a brief overview of the computing paradigms utilized to build up distributed research infrastructures. We shifted our focus to the common issues of these concepts and highlighted a number of research questions related to exchange resource information we try to answer.

3. Modelling Heterogeneous Resource Information in a Generic Way

In this Chapter, we present our information modeling process for exchanging resource descriptions in distributed research infrastructures. We identify the main actors and their information demand, which is the first achievement of our work. Starting with a requirement analysis we outline generic entities of a theoretical information model, which is capable to describe the main characteristics of heterogeneous, distributed research infrastructures. In addition to the theoretical analysis, we discuss the information systems and information schemas recently being utilized. We also present a theoretical approach for a schema mediation process for distributed research infrastructures, which is our next contribution.

We focus on exchanging resource information in heterogeneous, distributed research infrastructures, as well as providing easy access to information on status of available resources, services and activities. Previously, we showed in Chapter 2 that different computing paradigms were utilized to set up distributed research infrastructures. We therefore introduce a solution which is independent from the applied computing paradigms.

We start by describing the scope of our motivation for information modeling in Section 3.1. We define our *information modeling process* which consists of five steps. Subsequently, we outline these five steps in details.

The first step of the modeling activity is to identify the main *actors of distributed research infrastructures* by surveying various research infrastructures (Section 3.2). This enables us to define a high-level use-case model that describes how an interoperable, integrated monitoring and information system could be used to exchange resource descriptions and monitoring data. We continue with analyzing the abstract information demand of the identified actors in great detail. These demands are derived from their respective usage scenarios. The *requirements for resource information and monitoring data* are summarized from the view points of the various actors in Section 3.3. Based on the requirement analysis we identify a basic set of information which describes the main characteristics of a heterogeneous, distributed research infrastructure. This lays the foundations for identifying the required *generic entities of a generic data model* (Section 3.4). The next step of our information modeling process is to determine generic models at a conceptual level (information models) as well as at a lower level of abstraction (data models). We therefore shift our focus on the current *state-of-the-art* of exchanging resource information in our application domain in Section 3.5. We examine research initiatives, their infrastructures, the applied middlewares, the utilized information- and monitoring systems, as well as the information- and data models. The examination motivates our work to determine generic schemas at a conceptual level and we have a detailed view on the information models and their capabilities in Section 3.6. The purpose of the last step of our information modeling process is to link a theoretical, generic information model to actual services. We therefore present both, an approach for a *schema mediation* process as well as outline a concept to adapt a data warehousing technique for integrated monitoring of distributed research infrastructures (Section 3.7). We consider an interoperable information system as a data warehouse, which enables us to investigate, how the *Extract, Transform, Load* (ETL) process can physically integrate the heterogeneous monitoring data from the different monitoring systems into a central repository.

Finally, we highlight our findings and contributions, and analyze the main results of this Chapter in Section 3.9.

Later in this work (Chapter 6) we discuss how providers, users, and services can exchange quality information regarding infrastructure resources. We extend the generic data models described in this Chapter with further attributes to enable publishing quality information in a standardized form.

3.1. Exchanging Resource Information Using a Generic Information Model

To exchange resource information in distributed research infrastructures, a standardised process and a common understanding of how to describe the resources are required. Therefore, the information modeling is a crucial part of the success. Information modelling or data modelling goes beyond collecting data demands and information needs of various actors: it defines the principles of how the information is aimed to be structured and handled.

Information modeling or data architecture is therefore often used as an analogy to the activity of architects [207]. The design works of architects describe visions, where the architectural requirements are captured. Similarly to that, data modeling is also a design activity. Data modeling analyzes the information which meets the requirements of various actors, e.g. customers, providers, and services. Based on that, a basic set of information is identified, relationships between the information types are explored, and properties of various entities are determined.

Simple solutions are not expected in design activities. Likewise, we cannot assume that data modelers always deliver the same solution, but rather several solutions may meet the same requirements.

According to Buxton et al. [36] the conceptual information modeling aims to *“capture real-world data requirements in a simple and meaningful way which is understandable”*. For that purpose, several information modeling approaches exist and are in use. We refer exemplary to [32, 36, 150, 207, 227, 229]. In this work, we apply “strategic data modeling” as it is defined by Whitten in [229]. Accordingly, our modeling activity is part of our information system strategy. We define an overall vision for exchanging resource information and monitoring data in distributed research infrastructures. Then we suggest an architecture for an interoperable information and monitoring system.

Our goal is to enable access to the resource information and exchanging resource descriptions on an interoperable manner. We although have to go through the whole information modeling process, even if we only need the resource information. West et al. [227] gives an overview of how data models are developed and used today. For our data modeling activities, we adapted West’s process and created a specific one for our application domain, namely to enable exchanging resource information and monitoring data in distributed research infrastructures. Figure 3.1 depicts the specific process.

Based on the information requirements of the various actors of distributed research infrastructures, a conceptual data model should be defined first. This logical model is then intended - according to the technical requirements of the infrastructures - to create detailed, implementation specific models. The first model is the logical information model, the latter is the physical data model. Information models and data models capture the properties of different resources.

In this work, we use the terms *information model* and *data model* in the same way as RFC344 does. The RFC3444 [181] discusses the differences between information models and data models in detail. It also gives an overview of the role that the various existing specifications of standardization bodies - like *Distributed Management Task Force, formerly "Desktop Management Task Force"* (DMTF), *Internet Engineering Task Force* (IETF), or *International Telecommunication Union* (ITU) - play in the world of information models and data models.

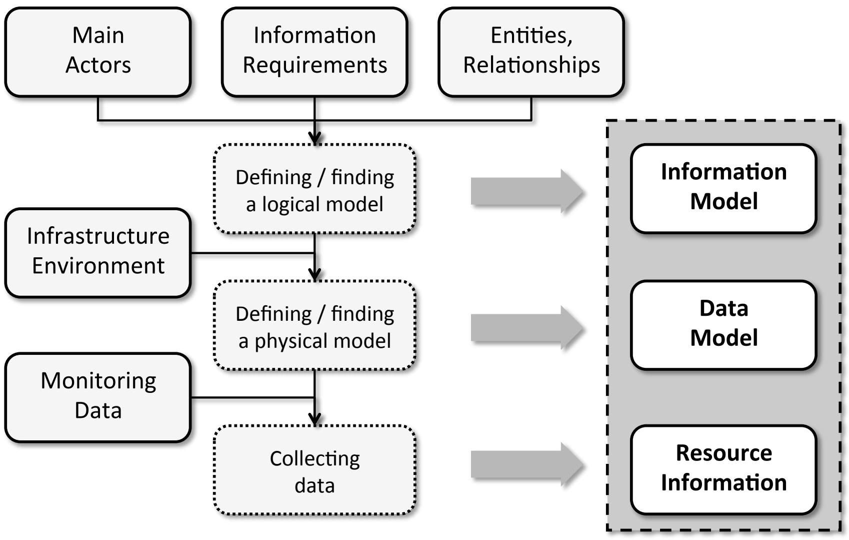


Figure 3.1.: Data Modeling for Developing a Generic Information and Data Model for Information- and Monitoring Systems of Distributed Research Infrastructures

For a detailed comparison of information- and data models we refer to [181]. In the following, we give a brief overview about the main characteristics of information- and data models:

Information model or information schema:
An *information model* or *information schema* is defined by use cases and is the result of the first step of a modeling activity. The information model focuses on the relation of entities and aims to model managed objects at a conceptual level. Therefore, an abstract representation of entities, properties, relationships, and operations is built in an implementation-independent way using modeling languages such as *Unified Modeling Language* (UML) or natural, plain language. Regarding the information systems and monitoring services, the described entities may be real-world objects such as processor, memory, or organization name. But there may also be abstract entities such as service, which represents resources providing computational capacity or storage space.

Data model or data schema:
A *data model* or *data schema* represents the information model at a lower level of abstraction. Data models are usually defined in a given specification or language, for example, *eXtensible Markup Language* (XML), *Structured Query Language* (SQL), *LDAP Data Interchange Format* (LDIF). Compared to information models, data models are more detailed and include specific details related to the implementations. The same information model can be rendered into multiple, different data models.

Both, information models and data models capture the properties of the resources of distributed research infrastructures. It is therefore challenging to define which detail should be part of the information model and which abstraction is going to be counted among the data model.

To follow a structured way, we define a 5-steps modeling process. Figure 3.2 depicts the steps of our data modeling process.

The five steps of the modeling process are defined as follows:

1. Main Actors:
   We start our modeling process by identifying the main actors of distributed research infrastructures. This step is done in a way that leads to a solution which is independent from the computing paradigm utilized to set up the distributed research infrastructure (Section 3.2).

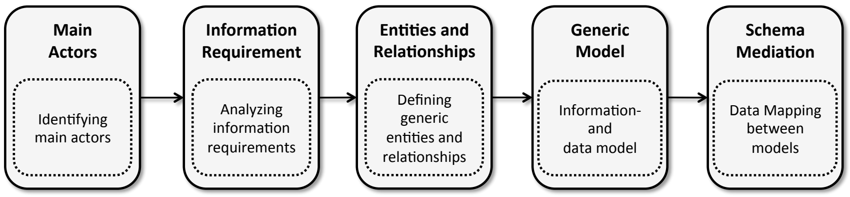

Main Actors
Identifying main actors
Information Requirement
Analyzing information requirements
Entities and Relationships
Defining generic entities and relationships
Generic Model
Information- and data model
Schema Mediation
Data Mapping between models

Figure 3.2.: A Structured, 5-Steps, Generic Modeling Process

2. Information Requirement:
   We analyze the requirements for resource information and monitoring data from the various view points of the identified actors (Section 3.3).

3. Entities and Relationships:
   After the investigation on the information needs of various actors, we continue to identify a basic set of information. We also explore relationships between the information types and determine properties of various entities (Section 3.4).

4. Generic Model:
   To model the entities of a distributed research infrastructure, we determine models at a conceptual level (information models) as well as at a lower level of abstraction (data models). In this step, we deliver a state of the art analysis of information- and data models utilized in our application domain. (Section 3.5 and Section 3.6)

5. Schema Mediation:
   Finally, we investigate how the existing information models can be mapped onto the generic model. We adapt a data integration technique to show how the generic model can be filled by the information models used by today's research infrastructures (Section 3.7).

In the following, we continue with the first step of our modeling process.

3.2. Actors of a Distributed Research Infrastructure

The goal of our data modeling process is to enable exchanging resource information and to provide easy access to information on status of available resources, services and activities. In this section we start our modeling process by identifying the main actors of distributed research infrastructures. Our data analysis process is organized in a way that leads to a solution which is independent from the computing paradigm utilized to set up the distributed research infrastructure.

In a first step, we define a high-level use-case model which describes how existing monitoring and information systems of distributed research infrastructures are currently used. Second, we add a new component - the interoperable, integrated, community-based information and monitoring system -, and re-define the high-level use case model to show how this new component changes the usage scenarios. We apply this model to identify the abstract actors of research infrastructures and to identify the data which is required by these actors.

We surveyed various distributed research infrastructures that serve various research disciplines (c.f. the arts and humanities [27, 136, 137, 96], astrophysics [12, 14], biology [44], climatology [24, 39], computational chemistry [205], environmental sciences [104, 124], health and life sciences [53, 144, 67, 40], and high energy physics [192]). We also assessed their information demand and reviewed technical reports [18] as well as a survey conducted among resource providers and communities of an e-science initiative study [93].

Based on that, we create a high-level overview about how an interoperable, integrated, community-based information and monitoring system is used. Our high-level use case diagram is depicted in Figure 3.3.

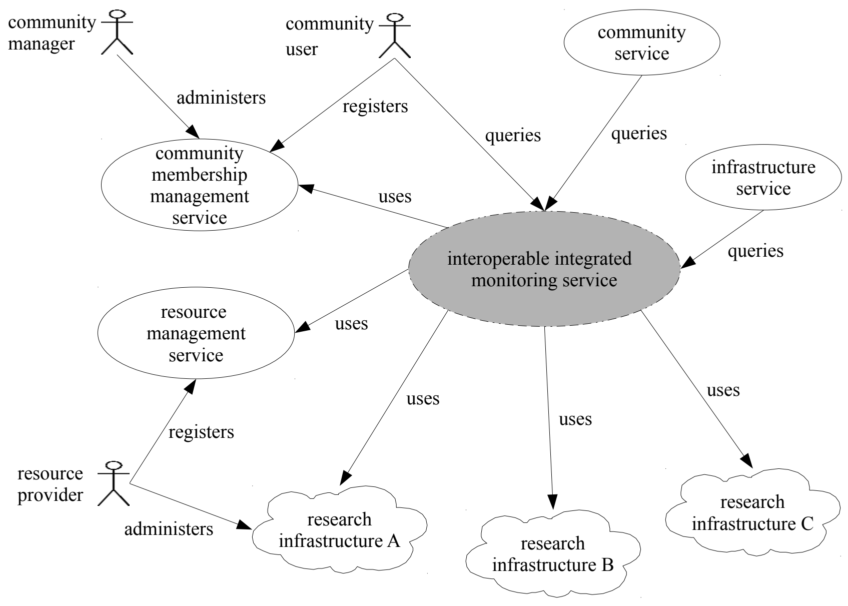

community manager
community user
community service
administers
registers
queries
queries
community membership management service
uses
infrastructure service
interoperable integrated monitoring service
queries
resource management service
uses
uses
uses
uses
registers
resource provider
administers
research infrastructure A
research infrastructure B
research infrastructure C

Figure 3.3.: Use Case Diagram of an Interoperable, Integrated, Community-based Information and Monitoring System

We identified the following actors in distributed research infrastructures: community users, community administrators, community services, infrastructure administrators or resource providers, infrastructure services. In the following we describe these main actors and their usage scenarios:

community users A user in a community uses resources of the community to store research data and to run computing tasks. Jobs can use input data (e.g. data stemming from experiments) and store outputs from data analyses. Users have different approaches concerning the selection of their utilized resources: While some users are mainly concerned about a quick processing of their computing tasks, others want to target a specific resource which has the optimal architecture for their application. The trustworthyness and the enforcement of specific policies on the resources - like data protection policies or long-term preservation policies - are also important requirements for resource selection in some usage scenarios. We refer examplery to [27, 67, 96, 136, 137, 144, 219].

In addition, users should see information about the progress of their jobs or workflows, which could in the case of close cooperations include other jobs of the community or virtual organization. This is necessary both to monitor computing quotas and to recognize atypical behavior and thus potential errors during the program execution. By default, a user should however only see his own job. Information on the job status also include information on where the job runs, how much time it has used, how long it had been standing in the queue, or whether it is active or waiting.

Users of a distributed research infrastructure should also be able to discover computing and storage resources. The service discovery is one of the most important use case for the monitoring and information systems.

We give an overview on the required resource information and monitoring data by community users in Section 3.3.1.

community administrators A representative of a community has the task to manage the community users. The operational concepts of research infrastructures require representatives for each community, who can admit to membership for new users, but also reject, disable or remove the users from the community - with the help of a central user management service, like for instance, VOMRS [62], VAPOR [158], DARIAH AAI [137, 56]. The computing and storage resources are managed by the respective resource providers and their administrators. Therefore, the resource administration does not belong to the scope of community administrators. The same applies to the creation of low-level resource usage statistics. This is usually created per community on a regular basis by the resource providers. The creation of a new virtual organization for the community is carried out by a central service after consultation with the concerned community speaker and the resource providers.

The community administrators require also high-level overview on the capacity of the provided storage and computing resources. For that purpose the information should be aggregated and the information must be trustworthy enough.

To diagnose abnormal behavior and find problems during program executions or data transfers, very detailed diagnostic information might be necessary. However, community administrators should only see the tasks, which belong to their community.

We analyze the required resource information and monitoring data by community administrators in Section 3.3.2.

community services There are core services which are useful for a particular community or for a research domain. Responsibilities of these services include amongst others resource discovery, resource brokering, scheduling, and monitoring.

The community services should be able to discover all computing and storage resources of a distributed research infrastructure which are available for the community. Supporting the discovery of such services is one of the most important use cases for the monitoring and information systems.

Detailed information about the current state of computing and storage resources of the distributed research infrastructure helps to choose between services which are of the same value in other respects. In order to assign computing tasks to computing resources or select storage services in an optimal way, accurate and up-to-date information is needed.

To monitor the provided services and infrastructure, information on the overall state of the distributed research infrastructure should be available for some community services.

The required resource information and monitoring data by community services is listed in Section 3.3.3.

infrastructure administrators and resource providers According to operational concepts of distributed research infrastructures, the computing and storage resources are managed by the respective resource providers [78].

The resource providers should offer their services on a manner, which is conform to the SLAs and policies agreed on. To act on problems rapidly, fresh information about the actual state of the services is needed. Automatic test systems and monitoring services might help the administrators to do so.

The infrastructure administrators require both, a high-level overview as well as detailed diagnostic information on the provided storage and computing resources.

For high-level overviews like the overall capacity of the storage and computing resources, the information can be aggregated. However, that information must be trustworthy all the time. The same applies to the creation of usage statistics and accounting, which should be done by the resource provider on a regular basis and per community.

The very detailed diagnostic information is necessary to monitor computing quotas as well as to recognize atypical behavior and thus potential errors during the program execution. If necessary, an infrastructure administrator should be able to see all jobs that are currently running on the resources. The infrastructure administrator's interests also include information such as where a job runs, how much time it has used, how long it had been standing in the queue, or whether it is active or waiting.

We give an overview on the required resource information and monitoring data by infrastructure administrators and resource providers in Section 3.3.4.

infrastructure services According to operational concepts of distributed research infrastructures, several central services with high availability should be offered (c.f. [78]).

These core services do not belong to any particular community and are provided by the operators of the infrastructure. Such services are especially the user and community management service (like [62, 158]), the resource and resource provider management service (like [154, 2]), the information- and monitoring services (like [200, 77, 156]), and various other central services like for instance resource brokers and automatic test systems (like [123, 69]).

The infrastructure services might require a high-level overview as well as detailed diagnostic information about the current state of the provided resources. For high-level overviews the information can be aggregated, assuming that information is trustworthy all the time. The detailed diagnostic information might be necessary for services, which monitor quotas, check the current state of the computing and storage resources, or recognize atypical behavior and errors during the program execution.

The required resource information and monitoring data by community services is listed in Section 3.3.5.

We identified the main actors and their usage scenarios in our application domain (c.f. Section 2.2). However, we point out that our procedure is generically applicable for further domains. In the following we continue with investigating the information demand of the identified main actors.

3.3. Requirements for Resource Information and Monitoring Data

In this section we analyze in detail the requirements for resource information and monitoring data that can be derived from respective usage scenarios[8]. We therefore give an overview of the demands on monitoring data and resource information from the perspectives of the main actors, which we identified in Section 3.2. Based on these demands we identify common, abstract requirements on monitoring data and resource information. The abstract requirements are laid down to identify a generic data model later in this work (Section 3.6). Such a generic data model is the particular basis for exchanging resource information between different environments and it is utilized in the following chapters. This Section is partly adapted from [18].

The freshness of the information is a crucial requirement for monitoring and information systems. We therefore define a reasonable update frequency for the different kind of monitoring data and resource information. It helps to build up a performant monitoring system and prevents the overloading of the various underlying systems that are periodically queried for fresh resource information.

We classify the monitoring data and resource information in four different groups: (1) static information, (2) dynamic information, (3) organizational information, and (4) information for

[8]We collected the initial requirements together with colleagues from the D-MON project. Our results were furthermore adopted by the German e-Science Initiative to build up its grid monitoring architecture.

improving the quality of service. We point out that there are both more detailed classification (for example, the EGI profile [34] describes 5 different levels), as well as more simple classifications exist (c.f. resource matcher component of the GWES [114]).

In the following we give a brief description of these four groups:

static information This includes generic, descriptive information about the computing resources, which has a static nature or it changes with low frequency. The information is almost always required to find appropriate resources for executing computing tasks or for data management operations. Examples are:

- service setup (service type, service endpoint URL, etc.)
- hardware setup (architecture, number of CPUs, RAM size, etc.),
- software setup (operating system, installed software, etc.),
- description of the environment (local resource management system, etc.)

dynamic information This includes status information about the computing and storage resources that give an overview of the current state of the resources. This information has a more dynamic nature, and it changes with higher frequency. Such examples are:

- status of the computing resources (free CPU, free RAM, etc.),
- status of the storage resources (free storage space, etc.),
- status of the local resource management system (queue status, etc.),
- maintenance and downtime information

organizational information This contains information with organizational nature. Examples are:

- addresses of the contact persons (resource administrators, community managers, etc.)
- central service addresses (ticket- and support-systems)
- list of all resources available for the community

information for improving the quality of service This information is not necessarily required for executing computing jobs or managing data transfers. However, this additional, very specific information may help selecting ”better“ resources, which can meet specific requirements in a better way. Examples are:

- detailed job information about the computing tasks submitted by the user,
- detailed diagnostic information about file transfers,
- benchmark results, benchmarking data, scores

In the previous section (Section 3.2) we gave an overview about the main actors of distributed research infrastructures. In line with this, we continue by giving an overview of the demands on monitoring data and resource information from the perspectives of these actors, namely:

- community users (Section 3.3.1)
- community administrators (Section 3.3.2)
- community services (scheduler, broker, etc) (Section 3.3.3)
- infrastructure administrators (Section 3.3.4)
- infrastructure services (meta-scheduler, etc) (Section 3.3.5)

We describe these requirements in detail to lay the foundations for a generic data model outlined in Section 3.6.

3.3.1. Community Users

Community users interact directly with the resources of a research infrastructure or use community specific services, which hide the underlying distributed research infrastructure. The various usage scenarios require various views: in addition to the view of all available resources, status and static information, also views of each resource are to be offered with detailed dynamic information. To diagnose and analyse atypical behaviour of computing tasks or data transfers, detailed information on the job execution or data management should also be provided. The following resource information and monitoring data is required for community users:

Static information required for submission:
This information has a static nature or changes with low frequency. On the other hand, the accuracy of this information is critical for the community users to find appropriate resources for computing tasks or for data management operations.

- static information about the computing resources:
  - Processor (number of nodes, number of CPUs, number of cores)
  - Memory (RAM per queue, average RAM per core, RAM per node, e.g. the maximum RAM available for a job)
  - local resource management system (batch or queueing system)
  - architecture, operating system, installed software (version, path, license)
- static information about the storage resources:
  - total storage capacity
  - local storage management system

Dynamic information:
This includes information with a more dynamic nature or changes with higher frequency. The information is about the current state of the computing and storage resources.

- status of the computing resources
  - maintenance and downtime information
  - status of the computing tasks submitted by the user
- status of the storage resources
  - maintenance and downtime information
  - status of the data transfers initialized by the user

Organizational information:
This contains information with organizational nature or describes administrative structures.

- List of computing resources available for the community
  - contact addresses of the administrators, ticketing systems
- List of storage resources available for the community
  - contact addresses of the administrators, ticketing systems

Information for improving quality of service:
This information is not necessarily required to execute computing jobs or managing data transfers. But this additional, specific information may help to choose between services and resources which are equivalent in other respects.

- Information regarding computing tasks:
  - detailed job information about the jobs submitted by the user
  - the number of the jobs submitted by the user
  - active cores / total cores

- Information regarding research data management:
  - the storage space used by the user
  - detailed information about the data entities stored by the user
  - used space / total capacity
  - bandwith (site, resources)

3.3.2. Community Administrators

In order to be able to fulfill its role, the community administrator requires information about the members of the community, about their activities on the resources and information about the resources themselves. The required views by community administrators include the state of the available computing and data resources both in high-level overview as well as per resource. The current state of the services should be grouped and aggregated by location or service group. The community administrators should see only those jobs, which belong to the community. To be able to analyze and diagnose potential errors, the computing tasks and data transfers should be grouped by services, resource providers or community users. Community administrators may require the following resource information and monitoring data:

Static information required for submission:

The information demand regarding the static and slow-changing data is very similar to the demand of the community users. The accuracy of this information is also critical for the community administrators.

- static information about the resources:
  - computing resources: the same static information like community users
  - storage resources: the same static information about the resources like community users

Dynamic information:

Similarly, the demand on information with a more dynamic nature is similar to the community users. However, the community representatives are usually authorized to see a broader amount of information about the current state of the computing and storage resources, and the activities of the community users. In addition to the information required by the community users, administrators might also need:

- status of the computing resources
  - the number of jobs submitted by the community
  - the number of jobs submitted by a specific community user
- status of the storage resources
  - the storage space used by the community
  - the storage space used by a specific community user

Organizational information:

Among the tasks of the community representatives are administrative and organisational tasks, like contacting resource providers to negotiate the deployment of new services or to endorse the community members on the resources. The information demand on organizational data and administrative structures include:

- static information about the (resource) provider:
  - information about the institute or organization
  - contact details of the institute
  - contact details of the administrators

    - location of the resources
    - provided resources, resource descriptions
- Service Level Agreements (human readable)
- list of resources available for the community
- members of the community

Information for improving quality of service:
The community representatives are usually authorized to see a broader amount of information about the current state of the computing and storage resources, and the activities of the community users.

- availability, reliability of the resources accessible by the community
- extra service information for community specific services
- all jobs of the community (information collected from all computing resources)
- all storage used by the community (information collected from all storage resources)

3.3.3. Community Services

The information demand is very similar to the demand of the community users and community administrators. Since a community service often acts on behalf of the users or administrators, the services require at least the same resource information. The community services are usually authorised to see a broader amount of information. The accuracy of this information is also critical for the community services.

Static information and dynamic information:

- computing resources: the same demand as community users and administrators
- storage resources: the same demand as community users and administrators

Organizational information:

- list of available resources, technical resource descriptions
- members of the community

Information for improving quality of service:
The community representatives are usually authorized to see a broader amount of information about the current state of the computing and storage resources, and the activities of the community users. This information is not necessary to execute computing tasks or to transfer data successfully. The following, additional information may however help to choose services and resources that meet specific conditions better:

- test results, benchmark data, scorings
- resource accreditation status (test or production)
- Service Level Agreements (machine readable)
- Usage Policies (maximum storage capacity, maximum CPU slots, maximum waiting time, etc.)
- status of jobs (number of currently running/waiting/blocked jobs, number of performed/executed jobs, job history)
- advance reservations (for future planning)
- estimated response time, backlog, etc.
- status of storage (number of currently running/scheduled data transfers, number of performed transfers, data transfer history)

- business models, pricing models, etc.

3.3.4. Infrastructure Administrators and Resource Providers

The infrastructure administrators and the resource providers are responsible for offering services with high availability and reliability, which meet the SLAs agreed on. To recognize abnormal occurrences as soon as possible, fresh information about the actual state of the infrastructure is absolutely essential. Additional systems, like automatic test systems and monitoring services, might produce hugely useful monitoring data, which helps the administrators to act on problems rapidly. The administrators require the data grouped according to communities, resource users, resource providers, or service groups. The information demand of the infrastructure administrators and the resource providers include the following.

Static information and dynamic information:
The information demand regarding the static and slow-changing data is very similar to the demand of the community users and the community administrators. The accuracy of this information is also critical for the community administrators.

- computing resources:
  - all static and dynamic information required by users and services
- storage resources:
  - all static and dynamic information required by users and services

Organizational information:
One of the tasks of the infrastructure administrators and resource providers is to contact the community administrators, or users, or services administrators in case of incidents. Likewise, the deployment of new services or the endorsement of new community members on the resources are tasks which demand detailed information on organizational data. This includes:

- computing resources:
  - list of available computing resources for a specific community
  - contact addresses of the administrators
  - accurate environment description (version numbers, etc.)
- storage resources:
  - list of available storage resources for a specific community
  - contact addresses of the administrators
  - accurate environment description (version numbers, etc.)
- list of administrators of a specific community, contact addresses, etc.
- list of users registered in a specific community, contact addresses, etc.

3.3.5. Infrastructure Services

The information demand is very similar to the demand of the community administrators, community services and infrastructure administrators. However, the infrastructure services are usually authorised to see a broader amount of information.

Infrastructure services furthermore demand high-level overview as well as detailed diagnostic data about the current state. From the information system's point of view it means to aggregate data, but also offer fine-grained monitoring information for services which monitor quotas, check the current state of the resources, or recognize atypical behavior of services. The accuracy of this information is also critical for the community services. The infrastructure services may require the following information:

Static information and dynamic information:

The following, additional information may help to choose services and resources that meet specific conditions better. The accuracy of this information is also critical for the infrastructure services.

- computing resources: the information required by a community service, but for all communities
- storage resources: all the information required by a community service, but for all communities

Organizational information:

In addition to the information required by the community users and administrators, the information demand on organizational data also include:

- list of available resources by a community, also for all communities
- members by a community, for all communities

Information for improving quality of service:

The central infrastructure services are usually authorised to see a broader amount of information about the resources and the activities of the communities. This information is not necessary to execute computing tasks or to transfer data. The following, additional information may although help infrastructure services, like a meta-scheduler, to choose services and resources that meet specific requirements in a better way.

- test results, benchmark data, scorings, for all communities
- resource accreditation status (test or production), for all communities
- Service Level Agreements (machine readable), for all communities
- business models, pricing models, etc., for all communities

In this Section we investigated the information needs of the main actors of distributed research infrastructures in detail. We continue our work by identifying a basic set of information to lay the foundations for a generic data model.

3.4. Required Entities for Modeling Distributed Research Infrastructures

In the previous sections we first gave an overview on the main actors of the distributed research infrastructures (Section 3.2) and investigated on their generic usage scenarios. We then shifted our focus to analyzing the requirements for resource information and monitoring data from the perspectives of these actors. We continued by giving an overview of the demands on monitoring data and resource information in great detail (Section 3.3). This lays the foundations for identifying the main characteristics of resource descriptions, and as a consequence, analyzing the required generic entities of a generic data model.

The interoperability of monitoring and information systems requires an exchange of resource descriptions and monitoring data. Offering different protocols or interfaces by the services of the distributed research infrastructures is not necessarily a problem. The basis of homogeneous monitoring is a precise and shareable resource description. The systems, which are about to interoperate, must be able to describe common components of the distributed research infrastructures. The existence of common entities in the information models is a prerequisite for interoperability.

It is therefore necessary to identify at least a basic set of information which can be exchanged. Based on this basic set of abstract information, it is possible to agree on a set of common properties that are necessary for interoperable monitoring. These properties are expressed by entities and attributes in the generic information model which captures the properties of different resources.

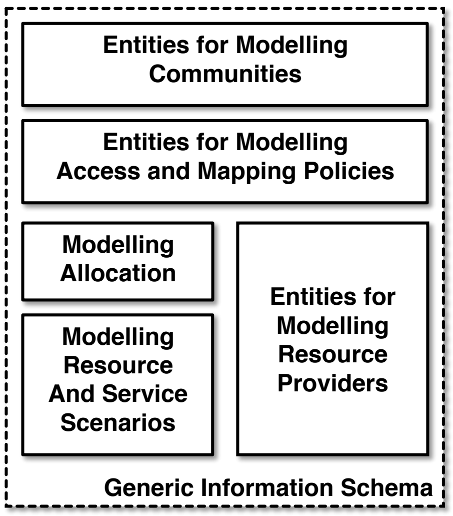

Entities for Modelling Communities
Entities for Modelling Access and Mapping Policies
Modelling Allocation
Modelling Resource And Service Scenarios
Entities for Modelling Resource Providers
Generic Information Schema

Figure 3.4.: Required Entities for Modeling Resources of Distributed Research Infrastructures

We found that a generic information model should be able to describe the following main characteristics of a distributed research infrastructure:

- communities,
- access policies and mapping policies,
- allocation of resources and services to communities,
- modeling resource and service scenarios, and
- modeling resource providers.

These characteristics, and consequently the basic set of information, are depicted in Figure 3.4). In the following, we briefly describe the characteristics we identified.

Modeling Communities Distributed research infrastructures enable homogeneous resource sharing and problem solving in dynamic, multi-institutional, virtual organizations [85]. Monitoring and information systems should allow communities to get an overview on the status of resources and services in a distributed research infrastructure even if it is inhomogeneous. Most systems still provide data from the view of resource providers and do not provide the resource descriptions and monitoring data in community-specific ways.

The modeling of communities, their institutions, and their members is therefore essential for coordinated access to resources which are shared among different physical and logical organizations.

The basic set of information regarding communities should contain simple descriptions, like where persons or legal entities are located. It should furthermore give information on contact persons and their contact details.

More complex entities of the model should for example describe the organizational relationships, since a physical or logical institution can participate in other organizations.

Communities and virtual organizations are usually helped by systems, which manage the roles and permissions of community members. In many cases it is therefore useful to describe such membership management systems in a generic model.

Modeling Access and Mapping Policies To share resources in a dynamic, multi-organizational environment requires information about the authorization of communities for the shared resources. A precise description of policies and resource utilization rules is the basis of enabling access for a group of actors in distributed research infrastructures.

The respective classified data, e.g. the policies for access and mapping, should be handled by a generic information schema.

Modeling Allocation of Resources and Services to Communities The consumers in distributed research infrastructures have various strategies regarding the consumption of shared resources. Several parameters may influence the resource utilization. We name exemplary the variable peak times, the pricing models, the alternating availability of own resources, and the variable community specific requirements.

Analogously, the resource providers define various policies and utilization targets for sharing their resources. The computing paradigms of our application domains, which we described in Section 2.2, enable resource providers to define suitable configuration for the various distributed research sharing scenarios. In the case of resources providing computing capacity, the resource providers usually configure these settings in their local resource management systems and set up several queues with different utilization targets for the computing tasks. The scheduler of the resource management system orchestrates the executions and controls the resource consumptions according to these settings.

The modeling of the allocation of resources to communities should therefore be part of a generic information model.

Modeling Resource and Service Scenarios The main goal of distributed research infrastructures is to enable homogeneous resource sharing and problem solving in a dynamic, multi-institutional environment [85]. The resource consumers require a homogeneous overview of the environment. A precise description of resources and services is the basis for a correct usage of the shared resources.

Therefore, information about the well-defined service usage points is required. We classify the data about the software components and service interfaces in this group. It does not only include the description of specific service functionalities, but also exposes the characterization of the technology used to implement the interface. Furthermore, it encloses the description of how the various activities can be created, monitored, and managed by the customers. Additional information about status information, like downtimes, can also be handled here.

A generic information schema should be able to model these service scenarios.

Modeling Resource Providers The modeling of the resource providers is an important part of the operation of the distributed research infrastructures. The resource providers of the distributed research infrastructures are actors, which have specific roles and privileges over resources. The permissions enable them to share resources, administer services, and define utilization policies for the shared resources. To operate the services of the infrastructure, they require an overview on the status of the shared resources in great detail.

The modeling of resource providers and their institutions should therefore be part of a generic information schema, which aims to describe distributed research infrastructures. Both, simple entities like geographical or logical locations, as well as complex entities like characterizing organizational relationships should be handled by the generic schema.

In this Section, we gave an overview on the main characteristics of a distributed research infrastructure that should be described by any generic information model. In the following, we survey the existing information schemas.

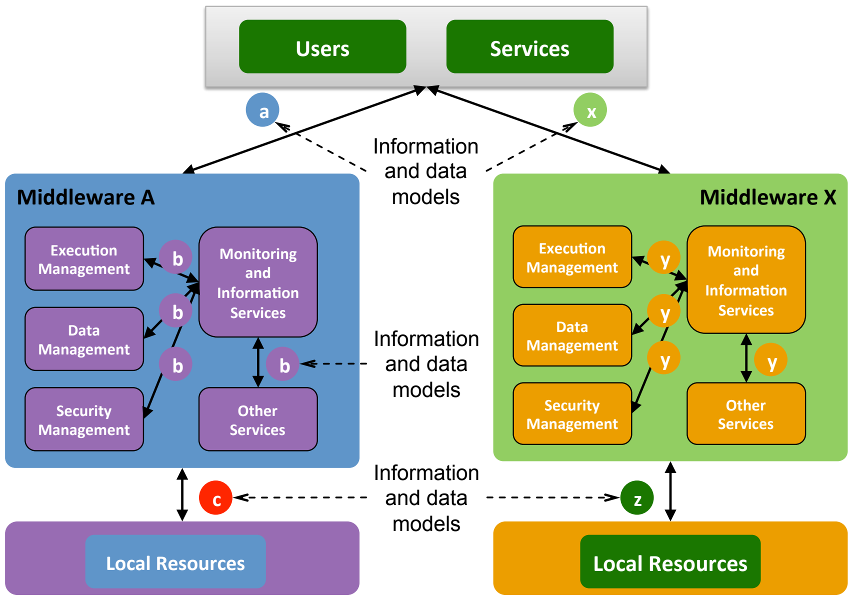


Figure 3.5.: Utilizing Middlewares in Distributed Research Infrastructures

3.5. State of the Art of Exchanging Resource Information in Distributed Research Infrastructures

We previously studied a number of usage scenarios employed in the modeling activities and identified abstract, common information demands. To link them to existing models utilized in distributed research infrastructures, we survey the main distributed research infrastructure initiatives of our application domain. In this section we therefore analyze their

1. middlewares,
2. monitoring- and information services,
3. as well as information- and data models.

The computing and storage resources of distributed research infrastructures are usually not directly available to its users. Figure 3.5 depicts a typical usage scenario. The resource providers or infrastructure initiatives agree with the communities or their representatives on the utilization of specific toolkits, libraries, and tools. These software components are connected by middlewares. The monitoring- and information services bundle the middleware components for exchanging resource information, for which specific information- and data models are used. In the following we briefly describe the functionalities of them:

Middlewares The middleware is the layer which lies between the consumers and providers. In distributed research infrastructures the services are typically provided through middlewares. The purpose of middlewares includes several core tasks like job execution, data management, resource allocation, providing security, and information management. Also the middlewares and their specific components are utilized to exchange resource information.

Monitoring and information services The software components of a middleware, that serve the same purpose, are typically bundled. The information and monitoring systems bundle the components of the middlewares, which collect, analyze, and provide information about the services and resources. The utilized interfaces and protocols are dependent on the information systems. The information and monitoring services are usually deeply integrated in the middleware and cannot be substituted by another information service.

If different types of information services were in use by the distributed research infrastructures, then also the resource descriptions typically use incompatible data structures and interfaces. Offering different protocols or interfaces in information systems is not necessarily a problem. The basis of homogeneous research infrastructure monitoring is a precise and shareable resource description.

Information and data models The information and data models capture the properties of different resources. One of the main problems of recently utilized distributed research infrastructures is that many incompatible information- and data models exist. They were derived from the use cases of different research infrastructure projects.

We studied and previously described a number of usage scenarios employed in the modeling activities and identified very abstract, but common information demands. We now try to link them to real services and therefore continue our work with surveying the main distributed research infrastructure initiatives of our application domain. First, we examine their utilized middleware systems in Subsection 3.5.1, which is followed by an overview of the monitoring and information services currently utilized by the middlewares in Subsection 3.5.2. Finally, we analyze their information and data models in Subsection 3.5.3.

3.5.1. Examination of Middlewares

In the following we examine the main distributed research infrastructure initiatives. The purpose of this examination is to show the characteristics of the research environments and to identify the technical backends of the distributed research infrastructures. Our survey is based on the following criteria:

- The *research infrastructure initiative* column shows the initiative that maintains the distributed research infrastructure.
- The *middleware* criterion lists the middleware systems deployed on the infrastructure.
- The *information and monitoring system* criterion reflects the information and monitoring system components of the middlewares that are relevant for exchanging resource information.

We summarize the results of the examination in Table 3.1. In general, our examination reveal that

- Several middlewares are currently used by the different research infrastructure initiatives.
- Several information and monitoring systems are in use.
- Own extensions exist in both middlewares and information systems.

For the sake of brevity, not all existing research infrastructure initiatives and middlewares are examined here. The interested reader may also review [187].

3.5.2. Examination of Information and Monitoring Systems

To have a better understanding on how the exchange of resource information between distributed research infrastructures could work, we examine the information and monitoring systems of the middlewares in this subsection. The result of the examination was also used in the *D-Grid Monitoring Project* (D-MON) project, and the D-Grid related part was published in [18].

In the previous subsection we showed that several middlewares are deployed by various distributed research infrastructure initiatives. The information and monitoring systems are crucial components of the middlewares and they are usually well integrated with the components of the

Research Infrastructure Initiative	Middleware	Information and Monitoring System
Asia Pacific Grid (APGrid)	Globus Toolkit 4.x	MDS4
AstroGrid	Globus Toolkit 2 / Globus Toolkit 4.x	Stellaris
China Research and Development Environment Over Wide-area Network (CROWN)	CROWN	CROWN GIMS
DEISA	UNICORE5, DESHL, Globus Toolkit	CIS with UNICORE6 (INCA)
European Grid Infrastructure (EGI)	gLite, UNICORE6, ARC, GT (UMD)	BDII, R-GMA, CIS, MDS (UMD)
EGI FedCloud TF	OpenNebula, OpenStack	BDII
German Grid Initiative (D-Grid)	Globus Toolkit 4.0, UNICORE6, gLite	MDS4, CIS, BDII
GridAustralia	APAC (Globus Toolkit 4.x / gLite)	MDS4
National Research Grid Initiative Japan (NAREGI)	NAREGI	NAREGI
NorduGrid	Advanced Resource Connector (ARC)	ARC
Open Science Data Cloud (OSDC)	Tukey Middleware (OpenStack)	Nova
Open Science Grid (OSG)	Virtual Data Toolkit (VDT), Condor-G	BDII/CEMon
TeraGrid	Globus Toolkit 2.x / 4.0	MDS4 with TeraGrid extensions
Worldwide LHC Computing Grid (WLCG)	gLite	BDII

Table 3.1.: Overview of Middleware Systems and Distributed Research Infrastructure Initiatives

middleware. This implies that the information systems are dependent on the middlewares. Since the middlewares are heterogeneous, the information and monitoring systems are also different.

We examined the information services of the most widely used middlewares with respect to the query language, data format, data model, interface, security model adopted, and information providers criteria:

- The *query language* criterion reflects the query language(s) that can be used to query the information system.
- The *data format* criterion shows the data format(s) which are supported by the information system.
- The *data model* criterion captures which data models (or abstract information models) are utilized for describing the resources of the distributed research infrastructure. Extensions are also shown if an information system used additional attributes.
- The *interface* criterion describes the interface provided by the information system.
- The *security model* criterion shows the security model adopted by the information system. If anonymous access was supported, it is also listed.
- The *information providers* criterion describes the sensors which collect monitoring data.

An overview of the results of our examination is shown in Table 3.2.

In brief, our examination reveals the following:

- Different versions of the same information system are in use which are not compatible with each other. We refer exemplary to MDS2 and MDS4, which are two versions of the information and monitoring system of the Globus Toolkit middleware.
- Different security models and policies are in use. We found that there are information systems which allow anonymous access to resource information, like *Berkeley Database Information Index* (BDII). On the other hand, there are information services that require clients to be identified by certificates, like *Monitoring and Discovery Service version 4* (MDS4).
- We can find both, interfaces providing complex functionality as well as very simple interfaces.
- Various data formats and query languages are utilized by the information systems to fit for community demands.

If different types of information services are in use by the distributed research infrastructures, then resource descriptions are typically provided by using incompatible data structures and interfaces. Offering different protocols or interfaces in information systems is not necessarily a problem. The basis of homogeneous research infrastructure monitoring is a precise and shareable resource description.

We therefore survey the information models in the following.

Middleware	**Globus Toolkit 4.x**	**Globus Toolkit 2.x**	**gLite**	**gLite**	**UNICORE6**	**Stellaris**	**ARC**	**TeraGrid**	**NAREGI**	**Usage Record**
Information Service	MDS4	MDS2	BDII	R-GMA	CIS	SPARQL over	LDAP	XQUERY over	SQL-Query	
Query Language	XQUERY over WS-RF	LDAP	LDAP	SQL-Query	XQUERY over HTTP	HTTP		WS-RF		
Data Format	XML	LDIF	LDIF	SQL	XML	XML + JSON	LDIF		SQL	XML
Data Model	GLUE1.1 with extensions	MDS Core Schema or GLUE1.1 (with extensions)	GLUE v1.2		DMTF CIM, Atom Syndication Format+ GLUE2	GLUE1.1	NorduGrid/ARC Schema (GLUE v1.2)	GLUE v1.3 (subset) (with extensions)	CIM (DMTF) (NAREGI extensions)	UR
Interface	OGSA/WS-RF + notification(WS-N) + Triggers	LDAP v3	LDAP	community specific + Web Services	XML Database & Atom-Feed (RSS) & OGSA/WSRF	Relational Database + HTTP (REST)	LDAP	OGSA/WS-RF	OGSA-DAI + Toolkit for SQL with JAVA API	
Security Model	GSI	LDAP (anonymous read)	LDAP (anonymous read)		GSI		LDAP (anonymous read) + Access Control Information (ACI)	GSI	GSI	
Information Providers	basic GT4 sensors + Information Aggregator Sources	basic GT2 sensors + specific extensions	basic GT2 sensors + specific Information Providers	AMON	Common Information Provider	application specific MD + adaptor for MDS4	basic ARC sensors + Local Information Tree (LIT)	Coordinated TeraGrid Software and Services (CTSS)	CIM Object Manager (CIMOM)	Usage Record v1.0

Table 3.2.: Overview of Information- and Monitoring Services in Distributed Research Infrastructures

3.5.3. Examination of Information and Data Models

A number of information and monitoring systems exists for different kinds of distributed research infrastructures and for different purposes. Our examination in the previous subsection showed that the information and monitoring systems utilize different information models for describing resources in distributed research infrastructures. We refer to Table 3.2, which summarizes our findings. If we abandoned the several varieties, extensions, and subsets of the models, we can reduce the number of the considered models, and we shift our focus to three core information models, which are the basis of several extensions.

In the following, we examine these three information models, namely the *Grid Laboratory Uniform Environment* (GLUE) Schema, the *D-Grid Resource Description Language* (D-GRDL) Schema, and the *Common Information Model* (CIM). The purpose of this examination is to show the characteristics of the most widely used information models based on the following criteria:

- The *organization* criterion shows the organization, which is responsible for the information model.
- The *background* criterion describes the circumstances and relationships of the maintainers and developers of the information model.
- The *driven by* criterion lists the participants who drive the further developments of the model.
- The *goal* criterion lists the attempts and aims of the model.
- The *focus* criterion reflects the targeted activities that the model can be relevant for.
- The *complexity* criterion captures how complex the model is and whether extensions are possible.
- The *acceptance* criterion shows the environments where the information model is utilized.
- The *implementations* criterion describes what kind of implementations exist.

We summarize the results of the examination in Table 3.3. In brief, our examination reveal the following:

1. Information modeling in distributed research infrastructures and in enterprise environments have different scopes:

   Modeling in distributed research infrastructures, as Andreozzi et al. points out in [8] for grids, is targeted at capturing the capabilities of research infrastructure services that should be advertised to any potential consumer in order to support activities such as resource selection for scheduling and high-level monitoring.

   In enterprise environments, information modeling mainly focuses on the management aspect of resources. The typical consumers of the monitoring information are the system administrators, who use the information to manage the resources.

   Because of the difference in scope, the information models utilized in enterprise environments do not cover all use cases of distributed research infrastructure environments.

2. Extendable information models of enterprise IT environments have high complexity [9]:

   The *Common Information Model* (CIM), for example, is a potential candidate for a suitable information model in distributed research infrastructures because of the extension and customization features foreseen in CIM.

   At the same time, these features make CIM a highly complex model, which is difficult to use for non CIM experts. Writing information providers for distributed research infrastructure resources is a very challenging task if an adequate knowledge of the model is missing.

3. Enterprise information models are not efficient in distributed research infrastructure environments:

   As the customization of information models of enterprise IT environments is a common way to bring specific extensions into the schema, several CIM implementations of extensions for distributed research infrastructures exist. We refer exemplary to [64], and [65], and [157]. However, because of the issues mentioned in the previous points, the implementations may also need to deal with performance problems ([65]).

Therefore, this examination motivates our work to have a detailed view on the two information models, which were designed for distributed research infrastructure environments, namely the *Grid Laboratory Uniform Environment* (GLUE) Schema and the *D-Grid Resource Description Language* (D-GRDL) Schema. In the following, we analyze whether the generic requirements for resource information and monitoring data can be fulfilled by the two schemas. For that purpose, we use the abstract requirements we identified in Section 3.3.

3.6. Generic Information Model for Monitoring Distributed Research Infrastructures

In this Section, which is partly adapted from [127, 128], we discuss the emerging *Grid Laboratory Uniform Environment Schema Version 2.0* [70], as well as the *D-Grid Resource Description Language* [232].

The *Grid Laboratory Uniform Environment Schema Version 2.0* (GLUE v2.0 Schema) is the latest published version of a standardization effort running within the *Open Grid Forum* (OGF) [169]. Regarding to our research conducted on the currently used middlewares (Section 3.5.1), the information and monitoring systems (Section 3.5.2), as well as the information and data models (Section 3.5.3), the most widely accepted information model is the GLUE Schema.

Following that, we focus on the *D-Grid Resource Description Language* (D-GRDL) [232], which was developed within the German DGI project for describing sets of e-Infrastructure resources. The D-GRDL is the current information model of the workflow scheduling system utilized by several German e-Science communities.

The purpose of both, GLUE schema and D-GRDL, is to define a standard information model for describing common entities in various distributed research infrastructure environments.

In Chapter 6 of this thesis, we discuss how providers, users, and services can exchange quality information regarding resources of distributed research infrastructures. We extend both of the currently used D-GRDL data model and the GLUE v2.0 Schema with further attributes to describe and publish the quality information in a standardized form. Those attributes are discussed for both schemas in Section 6.3.

3.6.1. The GLUE v2.0 Information Model

The GLUE Schema is a standard for describing distributed research infrastructure environments. It is used in several of today's existing middlewares, especially in their monitoring systems, and is therefore a suitable candidate for a common monitoring data model of a non-intrusive, interoperable monitoring system.

As our research on information and monitoring systems shows (Section 3.5.2 and Table 3.2), several different versions of the GLUE Schema are being used parallelly.

The first version of the GLUE information model (GLUE Version 1) aimed to help the interoperability activities between the users of distributed research infrastructures in the U.S. and Europe, namely the iVDGL and DataTAG projects. The Version 1 of the *Grid Laboratory Uniform Environment* Schema was released in 2002. It was a uniform information model to describe resources of grid computing infrastructures. The GLUE information model has evolved and several minor

	GLUE Schema	D-GRDL	Other Information Models, example: CIM
Organization	*Open Grid Forum* (OGF)	*German Grid Initiative* (DGI)	*Distributed Management Task Force, formerly "Desktop Management Task Force"* (DMTF)
Background	a working group of a community-initiated forum working on distributed computing activities	schema development is coordinated by a research organization	organization that develops and maintains open industry standards for system management in enterprise IT environments
Driven by	communities	german communities	industries
Goal	captures the capabilities of infrastructure services in order to support activities in distributed research infrastructure environments	offers a flexible framework for describing resources	defines the managed elements in an IT environment as a common set of objects
Focus	activities such as resource selection, inventory and high-level monitoring	selection of suitable resources (scheduling)	focuses on the management aspect of resources
Complexity	recent versions are more complex	simple, but flexible framework	too complex extensions, high learning and adopting curve
Acceptance	grid and cloud environments	life science communities, workflow engines	well accepted in enterprise environments
Implementations	different versions are being used parallel	several community extensions	extensions for distributed research infrastructures exist

Table 3.3.: Examination of Information Models

versions have been released, with the latest version being GLUE Schema v1.3. Nevertheless, the GLUE Schema could not fulfill several new requirements regarding resource administration, community-aware monitoring, or supporting further concepts. Therefore, a new working group, namely the GLUE Working Group [170] , was launched inside the *Open Grid Forum* (OGF) [169] in 2006.

The conceptual problems with the previously used versions of the schema, like the ComputingElement viewpoint of the cluster, the lack of handling multiple entry points to batch queues, or the limited description possibilities of storage resources, led to a decision by the GLUE Working Group of the OGF to redesign the schema. The process to redefine the information model and create a major new version of the schema took several years of work. Finally, the draft GLUE v2.0 recommendation was open for comments during 2008, and the emerging *Grid Laboratory Uniform Environment Version 2.0* (GLUE v2.0) information model [70] has been proposed as standard by the OGF.

This recent version of the GLUE information model (GLUE Version 2.0) is not backward compatible with the previous GLUE versions. Thus, bringing GLUE v2.0 into use requires proper planning, management, and guidance. Otherwise it may quickly lead to malfunctions of production level services, which rely on monitoring services and information systems.

There are different types of documents that are related to the GLUE v2.0 Schema: (1) the abstract information model specification document, (2) the data model reference realizations document, and (3) the profile documents of production scenarios. In the following, we give a brief overview on these documents, which we use intensively in our research.

Specification Document The current GLUE Specification v2.0 is approximately 70 pages long. It describes the GLUE Schema v2.0, which is an abstract information model. It is based on several usage scenarios [9] of current distributed research infrastructures.

The GLUE v2.0 schema presents a conceptual information model with mainly looking upon the entities of grid environments. The information model document uses natural language descriptions and provides graphical representation using UML class diagrams. Furthermore, an explanation is available in tabular form for each UML class. The explanatory tables include the following parts:

1. the generic characteristics of the entities (also included: the entity (or entities) from which it inherits and the description of what the entity represents.
2. the detailed list of properties of the class (with attribute names, data types, multiplicities, unit of measurements, descriptions) and the properties that are inherited from a parent class (for ease of reading)
3. the associations that the class holds with other classes (the associated reference includes the associated end class, key attributes, multiplicities, descriptions).

In the remainder of this document, we use the the latest currently available release [10] of the *GLUE Specification v2.0*, which is published by OGF.

Reference Realizations to Concrete Data Models Near the information model document, OGF also published a separate document which provides renderings to concrete data models. Currently, an *eXtensible Markup Language* (XML) schema, a *Lightweight Directory Access Protocol* (LDAP) schema, and a *Structured Query Language* (SQL) schema are supported. Although these data models represent the same relationships and concepts of the conceptual information model, they also contain simplifications targeted at improving query performance. In this work, we use some of the specific simplifications of the SQL data model without violating the concepts of the information model.

[9] GLUE Use Cases (document ID: 14621):`https://forge.ogf.org/sf/go/doc14621`

[10] GLUE Specification v2.0 (Document ID: 118): `http://redmine.ogf.org/dmsf_files/118?download=`

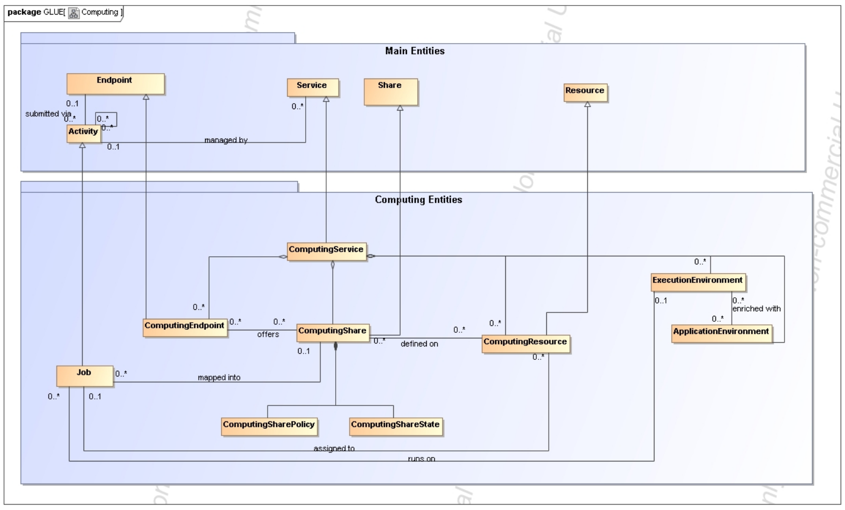


Figure 3.6.: Computing Entities of the GLUE v2.0 Schema (source: OGF[15])

Since the GLUE Schema has an active community, several enhancements have already been made to help implementing the particular data models as well as to define additional renderings of lightweight formats on textual basis, like *JavaScript Object Notation* (JSON) [52]. We refer exemplary to the JSON rendering [11] of the GLUE v2.0 Schema.

In the following, we use the *GLUE v2.0 Reference Realizations to Concrete Data Models (Document version 1)* which is the latest currently available release [12] published by OGF. For the XML data model we use the latest XML rendering [13] published by OGF.

Profile Documents Other additional documents, the profile documents, might be provided by the users of the GLUE Schema. A profile document describes usage scenarios and gives more information on the utilization of the GLUE Schema in production environments.

The *European Grid Infrastructure* (EGI) Profile [14] document, for instance, discusses clarifications of attributes which are optional in the conceptual model, but are considered to be mandatory in the EGI infrastructure.

This thesis mainly focuses on exchanging resource information regarding computing as well as storage capacities. Therefore, understanding how the GLUE v2.0 schema describes the entities of computing resources and storage resources is essential. In the following we briefly describe the computing and storage resource descriptions by the GLUE v2.0 schema:

Resources providing computing capacity A new and abstract entity, called ComputingService, models the resources providing computational capacity in the GLUE v2.0 information schema. The ComputingService is the main logical unit and aggregates further entities modeling computing capacity in distributed research infrastructures. The local resource manager system, e.g. batch system, and the batch queue information of the local resource manager system are expressed by ComputingShares. A ComputingSharePolicy defines the utilization policies of a computing share. The computing tasks, e.g. jobs or ComputingActivities, are submitted and monitored via interfaces provided by ComputingEndpoints

[11]JSON Rendering of GLUE v2.0: `https://github.com/OGF-GLUE/JSON`

[12]GLUE v2.0 Reference Realizations to Concrete Data Models (Document version 1): `http://redmine.ogf.org/dmsf_files/110?download=`

[13]GLUE v2.0 Reference Realization to XML Schema: `http://redmine.ogf.org/dmsf_files/110?download=`

[14]EGI Profile for the Use of the GLUE 2.0 Information Schema: `https://documents.egi.eu/public/ShowDocument?docid=1324`

[15]Computing Entities, OGF GLUE-WG `http://redmine.ogf.org/dmsf_files/114?download=`

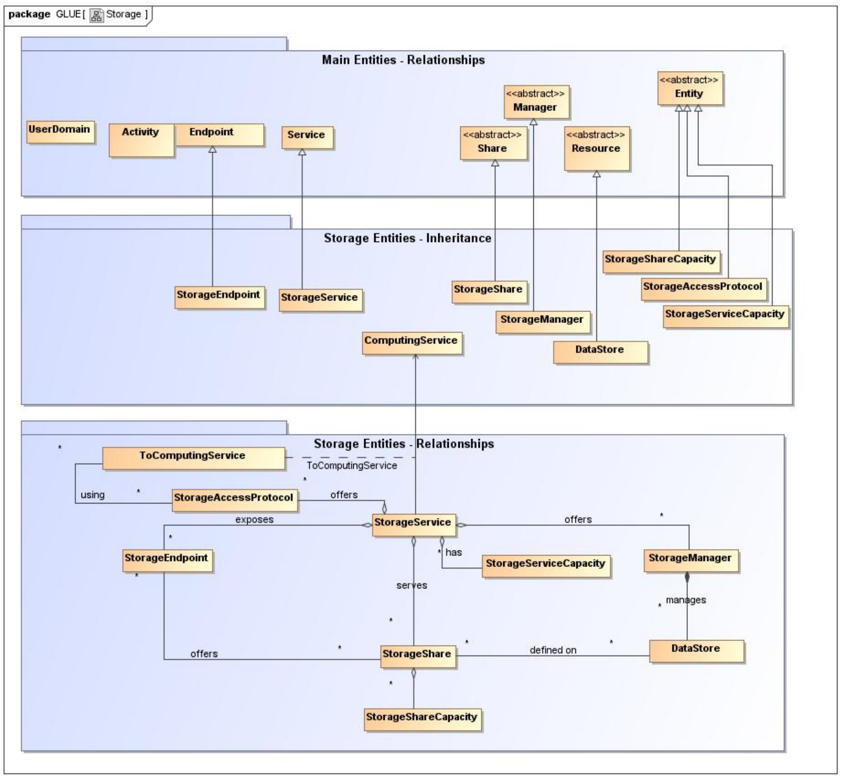


Figure 3.7.: Storage Entities of the GLUE v2.0 Schema (source: OGF[16])

that do not keep information about the jobs. Figure 3.6 depicts the detailed OGF GLUE-WG [170] model of the entities that are relevant for modeling computing capacity of resources.

Resources providing storage capacity The infrastructure resources, which provide storage capacities, are modeled by the GLUE v2.0 schema with the StorageService abstract entity. The StorageService logically aggregates the software and hardware components of distributed research infrastructures by exposing further entities. The StorageServices are usually provided via interfaces, which are modeled by StorageEndpoints, and accessible by using various data transfer protocols that are defined by StorageAccessProtocols. The total amount of storage, the amount of free storage, the amount of occupied storage, as well as the amount of already reserved storage are defined by the StorageServiceCapacity and StorageShare-Capacity. The StorageShare expresses the status information and the local storage manager system for a set of DataStores. The software components that manage storage systems are referred to as StorageManager. The logically homogeneous storage devices are modeled by DataStores. Figure 3.7 depicts the detailed OGF GLUE-WG [170] model of the entities that are relevant for modeling storage capacity of resources.

The GLUE v2.0 information model [70] and its possible new version is a suited candidate for a common information schema that describes the main characteristics of a distributed research infrastructure.

3.6.2. The D-GRDL Information Model

We gave a detailed overview on the application of the *D-Grid Resource Description Language* (D-GRDL) as well as several examples for its usage scenarios in our previous works [127, 128].

[16]Storage Schema, OGF GLUE-WG `http://redmine.ogf.org/dmsf_files/117?download=`

In this work, we only give a brief overview on the most important characteristics of the D-GRDL.

The *D-Grid Resource Description Language* [232] was developed within the German DGI project for describing sets of e-Infrastructure resources. The main motivation was to define a standard description model that uses the same form to store data about the resources of different research infrastructures. It furthermore enables both, to describe common simple entities (see Listing 3.2), as well as to aggregate resources of various environments (see Listing 3.2).

The D-GRDL is XML-based and designed to be very generic. That way, virtually all kind of objects of a distributed research infrastructure can be defined by the D-GRDL schema. The *D-Grid Resource Description Language* XML schema and its syntax are described in [232]. Practically, each entity and its properties are defined by the triple 'ident', 'type', and 'unit'.

```
<!-- # This is an example D-GRDL document to describe a Compound Type -->
  <simpleProperty ident="Load" type="Load_t">
    <field ident="Last1Min" type="int" unit="percent">99</field>
    <field ident="Last5Min" type="int" unit="percent">99</field>
    <field ident="Last15Min" type="int" unit="percent">2</field>
  </simpleProperty>
```

Listing 3.1: Describing Complex Types in a D-GRDL document: the Compound type

A disadvantage of the approach is that a concrete instance of the language has to be defined. The fact that D-GRDL determines only the syntax but not the semantics, makes two different forms of D-GRDL not necessarily being compatible with each other. There have already been different D-GRDL manifestations with the aim to attribute resource definitions in various GLUE dialects. Unfortunately, those D-GRDL dialects use older versions of the GLUE schema (V1.1 and V1.3), which are not compatible with the latest GLUE version [70] released by the OGF GLUE-WG [170].

The D-GRDL has many usage scenarios. Its application includes resource matching, resource mapping, storing aggregated resource data, integrity checking of descriptions, as well as a query and response language. If infrastructure resources were described in the D-GRDL format, the suitable resources for specific application purposes can be selected in an automated way [211, 212]. For that purpose, further services which understand D-GRDL can easily be utilized. We refer exemplary to the workflow service *Generic Workflow Execution Service* (GWES) [113]. That way, homogeneous descriptions of inhomogeneous infrastructures can be presented to the users, who solely focus on designing their workflows as well as applications [210].

The D-GRDL became the basic information model of the workflow scheduling systems utilized by several e-Research communities. The systems describe the resources of inhomogeneous infrastructures in the D-GRDL format. The data about the resources is always kept up to date and periodically refreshed by the ResourceUpdater. The resource descriptions are validated by a standalone component, called ResourceChecker. An other software component, the ResourceMatcher, compiles a D-GRDL resource query to an XQuery, sends this to the XML database, and produces a D-GRDL answer. The components and how they work together is outlined in [128].

```
<xml version="1.0">
<!-- # This is an example D-GRDL document to describe the grid host: grid-01.domainname.org -->
<resource uri="hardware:grid-01.domainname.org">

  <ofClass uri="urn:dgrdl:hardware"/>

  <name>grid-01</name>
  <description>this is a grid resource</description>

  <provides>
    <resourceRef uri="software:applicationA"/>
    <resourceRef uri="software:applicationB"/>
    <resourceRef uri="software:applicationC"/>
  </provides>

  <simpleProperty ident="WSRF.ManagedJobFactoryService" type="uri" unit="">
    https://server:8443/wsrf/services/ManagedJobFactoryService
  </simpleProperty>
  <simpleProperty ident="cpucount" type="int" unit="pcs">8</simpleProperty>
  <simpleProperty ident="cpuload" type="int unit="percent">10</simpleProperty>

</resource>
```

Listing 3.2: An Example D-GRDL Document to Describe a Resource Providing Computing Capacity

As a part of this work, we gave a solution to capture the policies and the resource utilization rules to enable access for a group of actors in distributed research infrastructures. We describe in [128], how the respective data for resource access and mapping can be handled by D-GRDL.

Furthermore, we describe how the D-GRDL schema can be used to describe resource quality information. Our solution is presented later in this work (Section 6.3).

3.6.3. Other Information Models

Several other information models exist, which are widely utilized for system management in enterprise IT environments. Research work has already been made to adopt these enterprise environment models for distributed research infrastructures ([65], [9], [64], and [157]). Generally said, while an adoption is often possible, the information models of enterprise system management cannot be used efficiently in distributed research infrastructures. Our examination in Section 3.5.3 showed that the main reasons for this are:

- the information models have different scopes
- they are not efficient in distributed research infrastructure environments
- it is difficult to extend them, or too complex extensions
- the learning and adoption curve is high.

Therefore, even though schemas like *Common Information Model* (CIM) are well established information models in IT environments, are not within the scope of this thesis. In case of necessity for integrating monitoring data of distributed research infrastructures with enterprise data schemas, we refer to the research works in that topic, like [65], [9], [157], and [64].

3.7. Mapping of Existing Information Models onto the Generic Information Model

In the previous sections we gave a detailed overview of how resource descriptions can be modeled. After defining the main actors of distributed research infrastructures and summarizing their information requirements, we analyzed the utilized middlewares as well as information and monitoring systems. We also evaluated the existing information and data models recently being utilized. Here we outline our concept which ensures the physical integration of heterogeneous data from multiple sources. We both explain our approach for a schema mediation process and define an *Extract, Transform, Load* (ETL) data warehousing process that we use for gathering and incorporating data from heterogeneous monitoring systems. This section is partly adapted from [21, 18].

3.7.1. ETL, a Data Warehousing Technique

An interoperable information and monitoring system must gather and incorporate data from heterogeneous monitoring systems. Data warehousing techniques ensure physical integration of heterogeneous data from various sources into a central repository. The central data repository is commonly referred to as data warehouse. If we considered the interoperable information system as a data warehouse, we can adopt the results of the extensive research that has already been conducted in the field of data warehousing.

To transform the complex source data into an integrated data structure efficiently, an ETL-process is defined in data warehousing. The *Extract, Transform, Load* (ETL) processes are the data integration tools in data warehousing. Figure 3.8 depicts the generic ETL process for a data warehouse. The ETL process extracts data from several, heterogeneous data sources by specific extractors (E). The original data providers may provide different interfaces and may use different protocols as well as data models. The extractors are accordingly provided. They may produce complete data source snapshots or differentials for incremental loading. The extracted data is

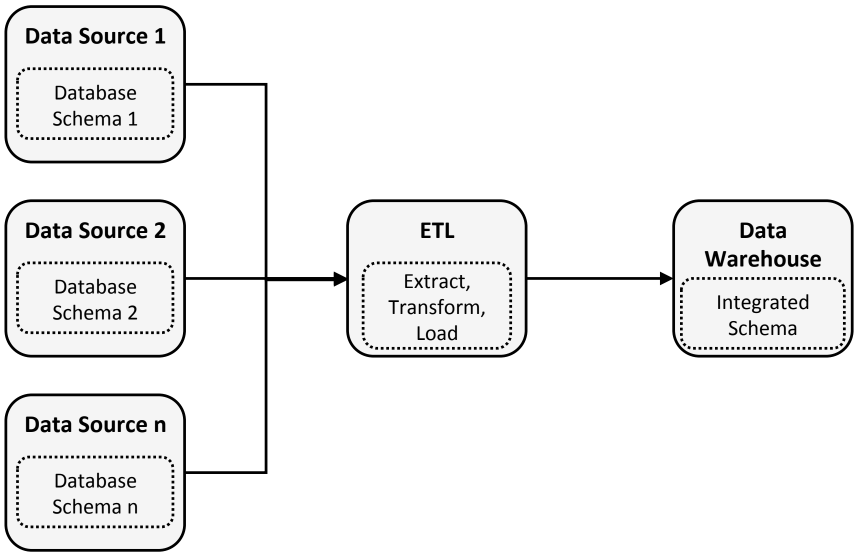


Figure 3.8.: Generic ETL Process for a Data Warehouse

parsed, checked, cleansed, and transformed (T) in a flexible way into a common data structure. As the last step of the process, the data is loaded (L) and stored in the target repository.

Designing an ETL process is a complex task and includes different programming and data modeling paradigms. The first steps of the data warehouse design usually require abstractions. Therefore, the design process starts with a higher level analysis of the data structure and the investigation on the content of the data sources. This enables the creation of a common information schema and a data model for a data warehouse. Related literature [32, 36] describes that there are several approaches to set up such abstraction processes.

It also needs to be specified how the attributes of the source data relate to the attributes of the data warehouse. This *logical data map* or *schema mapping* is of great importance. The schema mapping influences both the performance as well as the quality of the ETL process. On the one hand, too complex mappings may lead to very long parsing and transformation times, or they require more computing capacity for the execution. On the other hand, simple schema mappings may cause a lack of attributes in the target schema, which may result in missing data in the data warehouse.

3.7.2. ETL Process for Interoperable Monitoring of Distributed Research Infrastructures

In this section we discuss how to manage the process if we considered the interoperable information system as a data warehouse.

Our survey of distributed research infrastructures (Section 3.5.1, Section 3.5.2, and Table 3.2) shows that different types of monitoring and information systems are used by the research infrastructures. The different systems use incompatible data structures, different protocols and diverse interfaces to provide the resource descriptions.

To aim the goal of interoperable monitoring in such a heterogeneous environment, we adopt the ETL process for interoperable monitoring of distributed research infrastructures. Figure 3.9 depicts our approach. The monitoring data and resource information is gathered from the heterogeneous systems. The source systems are depicted on the left part of the Figure. The system specific extractors (E) use different protocols and data models to collect the monitoring data. The extracted monitoring data is filtered, parsed, and transformed (T) into a common data structure. Finally, the transformed monitoring data is loaded and stored (L) in the integrated information sys-

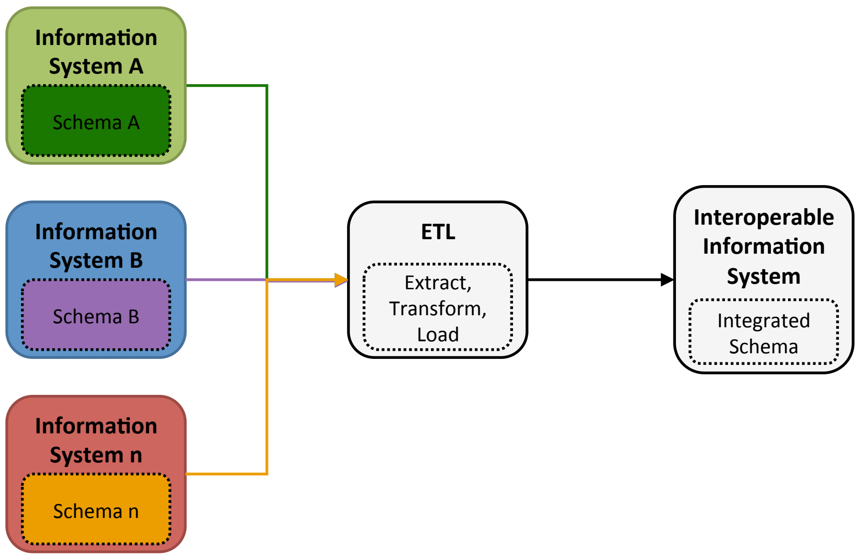


Figure 3.9.: ETL Process for Interoperable Monitoring of Distributed Research Infrastructures

tem which is shown on the right. This approach enables the physical integration of heterogeneous monitoring data and resource descriptions.

However, the physical integration of data sets may raise new issues in distributed research infrastructures, where the resources of virtual organizations are provided via abstract service entities. In such environments, the resource information of computational and storage resources may be available by multiple entry points (i.e. parallel deployment). Therefore, the integrated information system must augment data sets with the indication of the source system and the time of the data retrieval, at least. We use the term *provenance information* for the information that helps to trace back the derivation history of the monitoring data

We continue with giving a more detailed overview on the Extract, Transform, as well as Load processes in the following three subsections.

3.7.3. Extraction of Resource Information and Monitoring Data

The interface, protocol, and data schema is predetermined by the original information or monitoring service. To set up a communication channel, the standard client API of the middleware can be used. To encode, transport, and serialize monitoring data for reusing it in other contexts, the extractor process should output a common data format. The formats XML and *LDAP Data Interchange Format* (LDIF) are good candidates because of their attribute-value design. Our examination in Section 3.5.2 showed, that most of the information and monitoring systems either provide an interface to extract the data in XML or LDIF format, or offer the possibility to revert to existing tools.

Later in this work (Chapter 4.4) we examine carefully the information and monitoring systems of the distributed research infrastructure set up by the German e-Science Initiative. The three production level systems that we analyzed are the *Berkeley Database Information Index* (BDII), the *Common Information Service* (CIS), and the *Monitoring and Discovery Service version 4* (MDS4). We investigate how the original systems can be queried, how their interfaces can be utilized, and how the right query can be constructed. On that basis we create extractors for the information services BDII, CIS, and MDS4 with respect to the applied security and communication protocol. The process of extracting data from BDII, CIS, and MDS4, the three existing monitoring systems of our examined application domain, is realized by middleware-specific extractors (one per middleware).

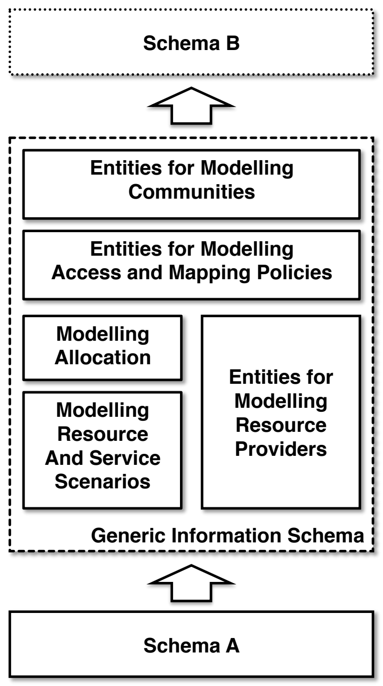

Schema B
Entities for Modelling Communities
Entities for Modelling Access and Mapping Policies
Modelling Allocation
Modelling Resource And Service Scenarios
Entities for Modelling Resource Providers
Generic Information Schema
Schema A

Figure 3.10.: Schema Mediation for Distributed Research Infrastructures

3.7.4. Transformation of Resource Descriptions

The source information schema and the source data model are determined by the original information systems. The target data schema is also defined by the chosen generic information model. It is not necessarily a problem when the systems utilize different protocols or interfaces. The systems, which are about to support the interoperability of the monitoring and information systems, must be able to describe common components of the distributed resource infrastructures.

The existence of common entities in the information models is a prerequisite for interoperability. A precise and shareable resource description is the basis of homogeneous, integrated, interoperable monitoring. In Section 3.4, we defined the generic entities that are required to describe the main characteristics of distributed research infrastructures, and found that a generic information schema should model the following main characteristics: (a) communities, (b) access policies and mapping policies, (c) allocation of resources and services to communities, (d) modeling resource and service scenarios, and (e) modeling resource providers. We depicted these characteristics on Figure 3.4.

Focusing on those entities and their attributes, it is possible to define abstract relationships between the attributes of the source information schema and the attributes of the target schema. Figure 3.10 illustrates how the entities can be used for transformation between the source and target schemas. Particularly, the generic information schema is used for the mediation of monitoring data. On the bottom, an abstract monitoring service that uses Schema A as its information model is depicted. The resource information and monitoring data are transformed onto a generic information model. The generic information model is composed of the entities we defined in Section 3.4. From that format, the data can be transformed into any other information model. We illustrated the possible formats with Schema B on the top of the Figure. Thus, it is possible to cross-provide monitoring data, for example from Monitoring System A into the interoperable monitoring service and from there into Monitoring System B or vice versa.

Transformation is a generic term which represents the abstract relationships of the attributes and the process to restructure the schemas. The transformation is defined by a triple: (a) a finite set of input attributes; (b) a finite set of output attributes and (c) a finite set of rules which describe how to (re)structure the attributes. On the one hand, the transformation may just include data cleaning tasks and various filtering operations to provide the necessary data. But on the other hand, the *transformation* between the schemas is more than a simple *translation*. In many cases, it also includes transformation tasks where the source data is also *transformed*. Such cases occur if the attributes were moved between the entities.

We also discuss later in this work (Chapter 4.4), how the data transformation can be implemented using *eXtensible Stylesheet Language Transformation* (XSLT), which is a popular language for processing XML data or transforming XML documents into other formats. The work was also part of designing and developing transformators for the D-MON adapters [55, 148], which transform the monitoring data from the information systems utilized by the distributed research infrastructure of the German e-Science Initiative.

3.7.5. Loading of Resource Information and Monitoring Data

The target information system or the data warehouse predetermines the interface, protocol, and data schema. Usually, the standard client API of the target system can be used to set up a communication channel.

Since relational databases, LDAP-directories, or XML-databases are commonly used for the realization of target repositories, the formats SQL, LDIF, or XML are often suitable candidates for the Load process.

Standard ETL processes - and their respective Load processes - typically overwrite the existing information in the data warehouse. In several environments this does not lead to any problems, since the changes are tracked in the source systems and there is no need to track the changes in the target repository. Generally, we can assume that in the case of the usage scenarios of our application domain, the source information and monitoring systems focus on the actual status of services and resources. They usually do not provide a functionality, which would help such tracking of changes. Therefore, the auditing of changes in an interoperable, integrated information system is of great importance. Later in this work (Chapter 4.4) we describe how we solved the audit and change-tracking problems utilizing data provenance.

In Chapter 4.4 we give a detailed description of implementing a target repository for monitoring data coming from the infrastructure of the German e-Science Initiative. We investigate how the extracted and transformed monitoring data can be loaded, how the interfaces of the target system can be utilized, and how the right query for the Load process can be constructed.

3.8. Schema Mapping and Schema Matching

The systems of heterogeneous research infrastructures need to exchange resource information and integrate monitoring data. Such data is structured according to heterogeneous formats and description schemas. This work refers to schema *mapping* as a set of rules and expressions, which describe how the data from the source system has to be transformed into the target system. The detection of such a specific transformation is referred to as schema *matching*.

As long as the number and complexity of the schemas employed is limited, schema mapping and matching can be conducted manually. Creating mappings by hand is however error prone [184]. With increasing complexity, it is therefore fundamental to automate the schema mapping and schema matching approaches as much as possible. This allows considering several models, as well as their mappings and matchings at the same time. Automated approaches can also reduce the costs of tracing the continuous evolvement of the schemas. With regard to our field of research, automation allows to handle more complex models of information systems and more complex formats of monitoring data. As a consequence, it allows an increase of search space for possible matchings. Compared with the early schema mapping systems, recent schema and ontology matching algorithms can deliver *“dramatic change in the performance"* [31].

The process of identifying if and how two schemas are semantically related [173, 184, 203] is one of the most essential tasks to integrate heterogeneous research infrastructures. There are usually a number of ways how the mapping problem between the mismatching models and differing heterogeneous formats can be solved. Subsequently, there are multiple possibilities of how such a transformation can be implemented.

Many schema mapping methods have been proposed. Since there is already a large strand of literature analyzing these methods, we do not describe these methods in detail. Instead, we refer to studies that describe and assess the most relevant approaches[184, 25, 203, 204, 173]. Extensive research has also been made to organize and classify matching systems, and several classifications exist. In the context of our work, we identified four areas where suitable and well-performing matching systems are needed: (a) label-based matching techniques, (b) instance-based matching techniques, (c) structure-based matching techniques, and (d) hybrid matching techniques.

3.9. Analysis of results

In this Section we summarize the Chapter and discuss its contributions and findings. The overall goal of our work is to enable the exchange of resource information in distributed research infrastructures and to provide easy access to information on status of available resources, services and activities. For that purpose, a standardized process and a common understanding of how resources are described are required. The information modeling ([32], [36], [207], [227] and [229]) is a crucial part of the success.

Our first contribution is a *5-step information modeling process* (Section 3.1), which is independent from the computing paradigm utilized to set up the distributed research infrastructure. The five steps are: (a) identifying the main actors of distributed research infrastructures, (b) analyzing the requirements for resource information from the view points of the various actors, (c) identifying a basic set of information which describes the main characteristics of an infrastructure, (d) determining models at a conceptual level (information models) as well as at a lower level of abstraction (data models), furthermore delivering a state of the art analysis of information- and data models utilized in our application domain, and finally, (e) investigating how the ETL data warehousing technique can be adapted to map existing information models onto a generic model.

To identify the main *actors of distributed research infrastructures*, we surveyed various distributed research infrastructures. This enables us to define a high-level use-case model that describes how an interoperable, integrated monitoring and information system could be used. We identified the following actors: (a) community users, (b) community administrators, (c) community services, (d) infrastructure administrators and resource providers, and (e) infrastructure services (Section 3.2).

We followed with analyzing the abstract *requirements for resource information and monitoring data* (Section 3.3). These demands are derived from respective usage scenarios of the main actors.

This laid the foundations for identifying the main *characteristics of resource descriptions*, and as a consequence, analyzing the required *generic entities of a generic data model*. We found in Section 3.4 that a generic information model should describe the following main characteristics of a distributed research infrastructure: (a) communities, (b) access policies and mapping policies, (c) allocation of resources and services to communities, (d) modeling resource and service scenarios, and (e) modeling resource providers. We introduced this as a basic set of abstract information, which is necessary for interoperable monitoring and for a shareable resource description.

We delivered a *state of the art analysis* of the distributed research infrastructure middlewares of our application domain, their information and monitoring systems, and their information- and data models.

- In general, our examination of the distributed research infrastructure *middlewares* (Section 3.5.1) reveal that: (a) several middlewares are currently used by the different research infrastructure initiatives, (b) several information and monitoring systems are in use, and (c) own extensions exist both in middlewares and information systems.

- By examining the *information and monitoring systems* (Section 3.5.2) we found that: (a) different versions of the same information system are in use that are not compatible with each other, (b) different security models and policies are in use (there are information systems allowing anonymous access to resource information, and information services requiring clients to be identified by certificates), (c) we can find both, interfaces providing complex functionality as well as very simple interfaces, and (d) various data formats and query languages are utilized by the information systems to fit for community demands.
- Our examination of the *information schemas and data models* (Section 3.5.3) reveal that: (a) information modeling in distributed research infrastructures and in enterprise environments have different scopes, (b) extendable information models of enterprise IT environments have high complexity, and (c) enterprise information models are not efficient in distributed research infrastructure environments.

Therefore, we decided not to focus on adopting these enterprise IT environment models for distributed research infrastructures. Instead, we refer to existing research that has been conducted to adopt these models to distributed research infrastructures, for example [65, 9, 157, 64]. The examination motivated us to have a detailed view on the two information models, which were designed for distributed research infrastructure environments, namely the *Grid Laboratory Uniform Environment* (GLUE) Schema and the *D-Grid Resource Description Language* (D-GRDL) Schema. We found in Section 3.6 that: (a) both the GLUE schema and the D-GRDL are capable to describe a set of common properties, which makes them suitable candidates for a standard information model; (b) regarding to our research, the GLUE Schema is the most widely accepted information model; (c) the D-GRDL is designed to be very generic and virtually all kind of objects of a distributed research infrastructure can be defined with it.

To link the theoretical information model to actual services, we outlined our concept to adapt the *Extract, Transform, Load* (ETL) data warehousing technique for integrated monitoring of distributed research infrastructures (Section 3.7). Through a three step Extract – Transform – Load process we ensure that the heterogeneous monitoring data from the different monitoring systems is physically integrated into a central repository. First, an adaptor connects to the native interfaces and extracts (E) data in the original format, then the data is transformed (T), and finally loaded (L) into a repository, which utilizes the generic information model. We considered the interoperable information system as a data warehouse, which is why we can apply research results made in the field of data warehousing, ETL processes, and schema mapping.

In this chapter, we laid the information modeling basis of exchanging resource information and monitoring data in distributed research infrastructures. We defined a 5-steps information modeling process, which is independent from the computing paradigm utilized to set up the distributed research infrastructure. We identified the main actors of our application domain and their information demand. Based on a requirement analysis, we provided a theoretical model which is capable to describe the main characteristics of resource descriptions and monitoring data. To link the generic entities of our model to existing systems, we delivered a state of the art analysis of information systems and information schemas. We concluded the section by analyzing our results. In the next Chapter, we continue our work by developing a proof of concept for the distributed research infrastructures of our application domain.

4. Implementing an Interoperable, Unified Monitoring and Information System for Distributed Research Infrastructures

In this Chapter, we use the results of our theoretical analysis regarding the information demand in distributed research infrastructures, as well as the theoretical approach for a schema mediation process outlined in the previous chapter, and we develop a proof of concept for grid environments. We design an automated resource description exchange process and a respective generic monitoring architecture supporting it, which is our next contribution.

This chapter summarizes our work on the topic of an interoperable, unified monitoring and information system for distributed research infrastructures [127, 128, 130, 131, 148], as well as [18, 21].

Here we describe a distributed monitoring architecture for an interoperable and integrated information service and its implementation within the distributed research infrastructure set up by the German e-Science initiative [2]. The developed service unifies heterogeneous monitoring data gathered from multiple resource providing organizations as well as different middlewares. It realizes data transformations between different data models and combines community memberships and *Virtual Organization* (VO) resource management policies with the unified monitoring data to enable appropriate community-aware authorization to monitoring information. An automated process for discovering information and monitoring systems of a distributed research infrastructure is also discussed. This service is the basis for an integrated and interoperable monitoring of distributed systems, which need to interact with different communities and their heterogeneous services.

In many distributed research infrastructures different kinds of information services are in use, which utilize different incompatible data structures and interfaces to encode and provide their data. An example for an infrastructure with heterogeneous middleware components is the distributed research infrastructure set up by the German Grid initiative. The various resources are available through multiple middlewares [2]. Site monitoring, service monitoring, and job monitoring are done in several different, partly concurrent ways [178], and an easy access to combined data is not available. The broad spectrum of monitoring tools ranges from monitoring systems based on different middlewares to monitoring systems based on middleware-independent architectures. Beside the co-existing information systems for the different middlewares; *Globus Toolkit* (GT), *Lightweight Middleware for Grid Computing* (gLite), and *UNiform Interface to COmputing REsources* (UNICORE), there are monitoring tools for special purposes [153], for monitoring levels [208], and for user-centric monitoring for special application environments, e.g. community specific job monitoring [54].

Homogeneous monitoring of such heterogeneous infrastructures with the monitoring data being accessible everywhere, independently of the middleware which provided it, is the basis for a consistent status reporting on the resources and services. Thus, interoperability or interoperation between the different information services in a heterogeneous, distributed research infrastructure is required.

The prerequisites to understand this Chapter includes knowledge about the computing paradigms utilized to set up distributed research infrastructures (Chapter 2), as well as our examinations of the middlewares (Section 3.5.1), the information and monitoring systems (Section 3.5.2), and the information and data models (Section 3.5.3). We outlined the generic entities of a theoretical information model that are capable to describe the main characteristics of heterogeneous, distributed research infrastructures in Section 3.4. We also refer to our analysis regarding to the determination of generic models at a conceptual level (information models) as well as at a lower level of abstraction (data models) (Section 3.6). Finally, we need to recall how the *Extract, Transform, Load* (ETL) data warehousing technique can be adapted to map existing information models onto a generic model. (Section 3.7). We furthermore refer to research works made on the field of data warehousing, ETL processes, and schema mapping (c.f. [32, 36] and [184, 203, 173], respectively):

The remainder of this Chapter is structured as follows. Section 4.1 describes the practical relevance of the problem using three examples as scenarios: (a) the distributed research infrastructure set up by the German e-Science initiative, (b) the distributed infrastructure of the *European Grid Infrastructure* (EGI)[17] , and (c) the demonstration infrastructure *Instant-Grid* (IG). Based on these scenarios, Section 4.2 discusses generic and specific requirements, while Section 4.3 presents a system design, which overcomes the identified issues. Then, Section 4.4 presents a case study to evaluate our concept and covers the details of the implementation of the D-MON monitoring system. In Section 4.5 we outline related work by introducing architectures, standards, interfaces utilized to exchange monitoring data and to approach the integration problem and community-aware data provisioning. Finally, we conclude with the analysis of our results (Section 4.6).

This chapter is partly adapted from our presentations [127, 128, 130, 131, 148], as well as from [18, 21].

4.1. Scenarios

In this Section we describe three distributed research infrastructure scenarios. We use the scenarios to identify common requirements for an interoperable, unified monitoring and information system, which dynamically provide exhaustive information about the actual state of the components of a heterogeneuos distributed research infrastructure.

4.1.1. Research Infrastructure of the German e-Science Initiative

The research infrastructure set up by the German e-Science initiative, D-Grid [2], enables users from many scientific fields, grouped into communities realized as VOs, to exploit distributed computing for their specific applications. It is a large and complex grid infrastructure with stakeholders from different organizations. In this environment incompatible technical realizations with heterogeneous data structures and interfaces emerge easily. The highly diverse sets of applications coming from the scientific communities are differently suited for the various available middleware solutions. Therefore, the infrastructure combines the three middleware installations of Globus Toolkit [79], gLite [97], and UNICORE [221] as well as dCache [90] and OGSA-DAI [10] for data management. In the D-Grid scenario, compute resources are offered through all three middleware solutions or through either of them. The communities use the middleware they are familiar with and which are best suited for their applications.

The communities constitute Virtual Organizations (VOs) which get access to the subset of resources contributed to that VO. In that sense a VO is not just a group of users but also consists of virtual resources and services, those which are available for use by the specific community. As a consequence, all services have to be VO-aware to allow different contexts of resource and service allocation with respect to community specific requirements and technology.

[17]This distributed infrastructure was established by the *Enabling Grids for E-sciencE* (EGEE), continued by the *European Grid Infrastructure* (EGI) and the *Integrated Sustainable Pan-European Infrastructure for Researchers in Europe* (EGI-InSPIRE) projects and being currently developed further by the *Engaging the Research Community towards an Open Science Commons* (EGI-Engage) project.

Monitoring such a complex infrastructure is an ambitious task as each of the middleware implementations has its own notion of, and tools for monitoring. Up to now, monitoring for the different middleware solutions is often done in a middleware specific way: Globus sites are monitored through the MDS4 [200] information service with a GridCSM Web interface [19]; gLite resources are monitored by Site Functional Test (SFT) [69] which bases on information from the BDII information service. The reports are displayed through a Web interface; the now deprecated UNICORE 5 sites were not monitored but UNICORE 6 sites use the Common Information Service (CIS) [156] that comes with a Google-maps user interface. Furthermore, middleware-specific monitoring systems for specific tasks have been set up like [130].

The result is a multitude of sources each providing only a fraction of the desired information. This complicates the operation of comprehensive information services, which should be usable from everywhere in the research infrastructure. Obviously, in that situation a more homogeneous and comprehensive approach is desirable.

Most of the information services in use have the disadvantage of being focused on physical entities thus ignoring the actual mapping of resources and services onto communities. A community-centric approach would simplify a user's life as only information belonging to their community would be extracted and presented to them, whereas all information they have no use of (such as the providers' cluster status) would be filtered. In addition, a community-based privacy protection is possible. Analyses of the D-Grid scenario and concepts with respect to monitoring (cf. [223],[20],[178]) revealed several issues: One of the major shortcomings of a big infrastructure is that building several autonomous (monitoring) service components may result in several logical infrastructures without data interchange. To tackle this issue for the monitoring service, the German e-Science initiative established the research project D-MON[55].

4.1.2. Pan-European e-Infrastructure

A large research infrastructure, distributed across Europe and the world, was established by the *Enabling Grids for E-sciencE* (EGEE) [11]. It was continued by the *European Grid Initiative* and later by the *Integrated Sustainable Pan-European Infrastructure for Researchers in Europe* (EGI-InSPIRE) project. Currently, it is being developed further by the *Engaging the Research Community towards an Open Science Commons* (EGI-Engage) project. The aim of the *European Grid Infrastructure* (EGI) is to support research activities across a broad range of disciplines from the arts and humanities to physics.

The scenario of EGI [72] also well demonstrates the transition between various computing paradigms. The infrastructure of the *European Grid Infrastructure* originally offered services on the basis of the grid computing paradigm. Nowadays, several compute cloud and storage cloud services were added to the EGI service portfolio by federating academic clouds from multiple cloud providers, for instance, EGI Federated Cloud [60].

The EGI links computing centers, data centers, and communities together by operating on a federated manner. Furthermore, EGI manages trusted software repositories, where research communities can look for reusable tools, as well as resource providers can find the latest versions of softwares required to set up distributed research infrastructure services. The *European Middleware Initiative* (EMI) is a collaboration of the major european middleware providers with the mission to deliver a consolidated set of middleware components for deployment in EGI as part of the *Unified Middleware Distribution* (UMD).

In such an infrastructure, the loss of interoperability of middlewares may lead to problems in the infrastructure's operation, as the following use-case illustrates. For example, a scheduler used in one middleware is unaware of resource allocation mappings and jobs that belong to foreign middleware components. Parts of a job may rely on resources available in a gLite based infrastructure such as *European Grid Infrastructure* (EGI), while other parts need resources available in a UNICORE based infrastructure such as *Distributed European Infrastructure for Supercomputing Applications* (DEISA) [94]. Besides interoperation of authorization systems, job submission, data transfer, and scheduling services, also the monitoring systems have to interoperate in order to allow a smooth overall job scheduling as well as resource and service operation and maintenance.

4.1.3. Instant-Grid

The *Instant-Grid* (IG) [119] is a live-CD bundled with a dynamically configured research infrastructure. The goals of Instant-Grid are a flexible and beginner-friendly demonstration-, test-, and development-environment for distributed research infrastructures [30]. With Instant-Grid it is possible to try out grid- and cluster technologies [29], and develop and test own applications in a distributed research infrastructure. Its original environment [28] was based on the *Globus Toolkit 4* (GT4) grid middleware, but as a part of this thesis extensions were made to enable the fully dynamic deployment of the *Portable Batch System* (PBS) and the *UNiform Interface to COmputing REsources v6* (UNICORE6) grid middleware [135]. The extensions provide maximum compatibility to "real", production-level distributed research infrastructures, since the same technology and software is used. We describe IG's solution for automated system deployment, the technical concepts including the automatic configuration, the ready-to-use features, and the applications in Chapter 5 in great details.

For the monitoring and information systems, the compatibility to production infrastructures leads to similar issues like those infrastructures face to. The pre-configured Instant-Grid delivers fully configured middlewares, like GT4 or UNICORE6, and showcase grid-applications from different areas, like graphics, simulation, and collaboration. In this environment incompatible technical realizations with heterogeneous data structures and interfaces emerge easily. Therefore, the monitoring of heterogeneous grid services and the diverse sets of applications from various scientific communities is done on in a middleware specific way: the GT4 sites use the MDS4 [200] information service, the UNICORE6 sites are monitored through the CIS [156]). In the case of Instant-Grid a more homogeneous approach would be desirable.

On top of the grid middleware, Instant-Grid provides further specialized services. One of the most important ones is the *Generic Workflow Execution Service* (GWES) [114], which is adopted for many projects and it is in use in several scientific communities. GWES is an advanced workflow system for orchestrating the distributed execution of applications on grid resources. An application workflow consists of several sequential and concurrent program executions as well as data transfers. GWES regularly requires detailed resource information to schedule the job executions efficiently and reliable.

Therefore, the *Grid Resource DataBase* (GRDB) [230] was developed. The GRDB provides static and dynamic information on grid resources. The contained data is continuously updated by information providers and is stored using the *D-Grid Resource Description Language* (D-GRDL) [232] information schema. The GRDB consists of two components, a daemon and a user interface component implemented as a portlet. GRDB is capable to extract information from two different sources, namely Ganglia and MDS4, but neither an integration of the information, nor concurrent uses of both sources is possible. Furthermore, the provided information is optimized for the scheduling capabilities of GWES. Also the dynamic configuration of IG is not fully supported, and there is no information provider for the information service of the UNICORE6 middleware.

Instant-Grid also supports the concept of the *Virtual Organization* (VO). However, to provide an easy-to-use demonstration environment, the IG users, the virtual resources and every service of a running *Instant-Grid* (IG) belongs to the same VO. This default behaviour can be changed for complex IG use cases, like [188], where various virtual communities have to be supported. The missing VO-awareness of the IG services, including the monitoring and information services, does not allow different contexts of resource and service allocation with respect to community specific requirements and technology.

An integrated monitoring service, which dynamically provides exhaustive information about the actual state of components from multiple information systems, would enable interoperable services in Instant-Grid.

4.2. Requirements for an Interoperable, Unified Monitoring and Information System for Distributed Research Infrastructures

In this Section, we discuss the requirements and challenges of integrated monitoring services, which dynamically provide exhaustive information about the actual state of components from multiple monitoring systems, and a step towards integrated and interoperating distributed research infrastructures.

Some requirements are related to general aspects and independent from the computing paradigms as well as concrete usage scenarios. We briefly discuss them in Section 4.2.1. Other requirements are more specific and related to the usage scenarios, we previously analyzed in Section (Section 4.1). These specific aspects are discussed in Section 4.2.2.

4.2.1. Generic Requirements for Services of Distributed Research Infrastructures

In the following, we discuss the requirements that are related to more general aspects. They are also independent both from the computing paradigms as well as from the concrete usage scenarios.

Scalability and extensibility One of the common goals for all developments within a distributed research infrastructure is their scalability and extensibility. We face challenges on the level of the resources, which are the basic building blocks of the infrastructures, since the resources are being more complex (number of processing cores of a CPU, storage components, etc.), whereby their management, including monitoring, is also being more complicated. In addition to the complexity of the resources, the size of the distributed research infrastructures is increasing. Although the users, communities, developers, and providers have already had a good understanding how they can deal with their distributed research infrastructures, creating federations of existing infrastructures is a new challenge.

Providing resource information and monitoring data nearly real-time is a common requirement for almost all systems of a distributed research infrastructure. Therefore, the monitoring and information systems should scale independently from the increasing number of participating communities, providers, and services,

Sustainability Supporting standards is crucial for the sustainability of an interoperable, unified monitoring and information system. The utilization of well established norms and specifications regarding protocols, data access interfaces, as well as data structures is the basis to exchange resource information and monitoring data in a uniform way. The system to be developed for our application domain should therefore be compliant with the *Grid Monitoring Architecture* (GMA) [218], the *Web Services Resource Framework* (WSRF) [81], *Open Grid Services Architecture - Data Access and Integration* (OGSA-DAI) [26], and the GLUE v2.0 Schema [70] while security aspects such as authentication and authorization have to be taken into account.

In multi-middleware setups it might be of interest to support an integration proxy approach. This would allow the middleware specific implementations to use the integrated monitoring data. Thus, it would help distribute computing tasks within the whole distributed research infrastructure, even for the resources that are under the control of other middleware implementations and therefore their real workload is not viewable for a single middleware.

Security Distributed research infrastructures have already addressed security questions and provide solutions for many common problems. For instance, encrypting sensible research data as well as ciphering the network traffic are well-known features. However, the concrete solutions of the various computing paradigms might differ widely from each other. Therefore, interoperating and federating such infrastructures is challenging. We refer exemplary to the grid and cloud environments. The grid infrastructures are mainly secured by certificates, while the cloud infrastructures are based on easy-to-use security solutions, and the *Public Key Infrastructure* (PKI) standards are usually not the main focus of a cloud-based infrastructure. Supporting the same security standards is necessary for better integration. We discuss the issues and requirements for authentication and authorization under paragraph *Community- and Role-based Access* in Section 4.2.2.

4.2.2. Specific Requirements for an Interoperable, Unified Monitoring and Information System

After giving a brief overview about the generic requirements, in the following we discuss the specific requirements which an interoperable, unified monitoring system should fulfill. In Section 4.1, we described scenarios of a distributed research infrastructure, which we use for deriving specific requirements. These requirements are adopted from [21].

Information Scope The scenarios of distributed research infrastructures in Section 4.1 showed, that a large number of communities, or *Virtual Organization*s, uses heterogeneous middleware installations. There are several potential uses for the information that must be served by the monitoring. The monitoring and information system should support the purposes of resource discovery, task scheduling, resource monitoring, job monitoring, accounting of resource usage, high-level overview of the infrastructure, low-level diagnostic information for operation of the infrastructure.

The scope of the usage may principally affect the design of the information system. We refer exemplary to the issues how often the fresh information is collected, whether the data has to be presented in great detail or aggregated, and whether the information are allowed to be cached.

Data Integration Based on our investigations on the presented infrastructure scenarios in Section 4.1 we found that to provide the overall information a concept is needed which ensures the physical integration of heterogeneous data from multiple sources. An interoperable information and monitoring system must gather and incorporate data from heterogeneous monitoring systems. Data warehousing techniques should help to ensure the physical integration of heterogeneous data from various sources in a non-intrusive way.

For that purpose, an automated process to exchange resource information and monitoring data has to be defined. Furthermore, a respective generic monitoring architecture supporting it, must also be designed.

Data Provenance The scenarios in Section 4.1 showed, that different middlewares may be installed on the same compute resources. When a data integration process collected resource information from those middlewares, duplicates might be created that falsify the results. Furthermore, if the heterogeneous middlewares delivered diverging information about the status of the resources, the integration process create diverging data sets belonging to the same resource.

Therefore, an integrated information system must augment data sets with information that can help trace back the derivation history of the resource descriptions and monitoring data. We refer to that information as *provenance information*. A taxonomy of data provenance techniques is surveyed in [206]. It also gives a detailed view of data provenance research both in the scientific as well as in the business domains.

Community- and Role-based Access The scenarios in Section 4.1 showed, that a large number of communities consumes the services of the distributed research infrastructures, as well as there are several potential purposes that must be served by the monitoring. The monitoring of a distributed research infrastructure must therefore provide information on a need-to-know basis.

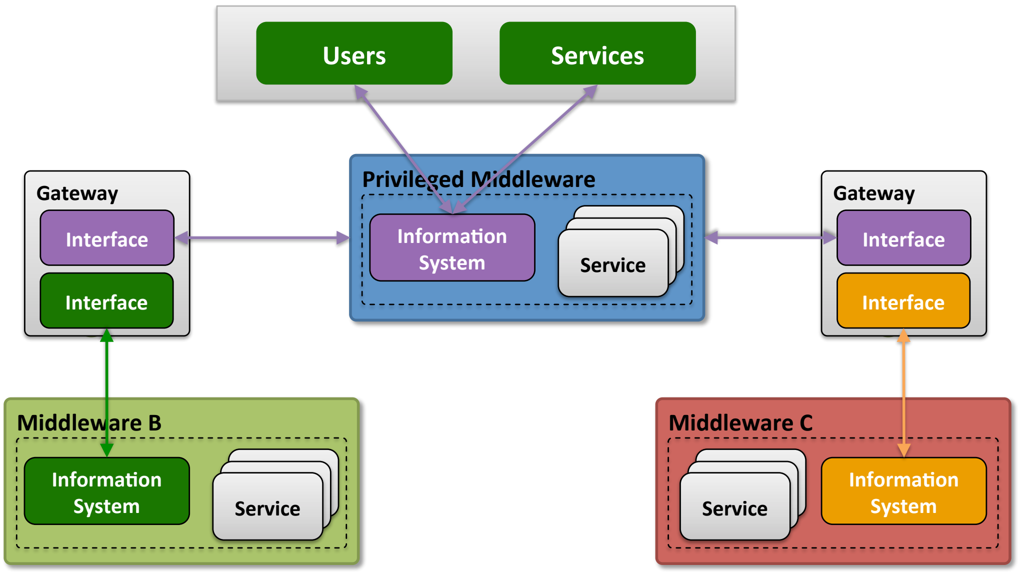


Figure 4.1.: Monitoring and Information System Based on a Privileged System

When it comes to authorization decisions, (1) the sovereignty of policy decision points, (2) the self-registration for citizen scientists, (3) the possible termination of community memberships, as well as (4) the persistence of user identities and their uniqueness across the infrastructures are the most challenging issues we identified. Both the community background of a requester, as well as the various roles acting on should be taken into account while the information system serves the request. The interoperable and uniform monitoring of such infrastructures collects information from all resources within a distributed research infrastructure, nevertheless, the monitoring system should provide proper access control to the integrated data. Therefore, the integrated monitoring system requires information about community memberships, roles and affiliations.

After discussing the main requirements, we shift our focus to discussing how a proper system design can fulfill the requirements.

4.3. Design of an Interoperable Monitoring and Information System

In this section the design options for an integrated monitoring system is described and we also discuss how the chosen options fullfill the requirements given in Section 4.2. The basis of the system to be designed is an architecture for the integration of several heterogeneous monitoring and information systems. On top of it interfaces to exchange resource descriptions, monitoring data, as well as communitymanagement information are modeled.

4.3.1. Basic Architecture

Several approaches to realize a monitoring system with the characteristics given above have been considered and their individual characteristics evaluated (we refer to our previous work [18], as well as to [147]).

In the following we examine three approaches: (a) building multiple bidirectional gateways between each pair of monitoring systems (*Multiple Bidirectional Gateways*), (b) defining one of the monitoring systems as the privileged system (*Privileged System*), and (c) an autonomous monitoring system which is independent from the middlewares (*Autonomous, Middleware-independent System*). The purpose of this examination is to show the characteristics of the architectures based on the following criteria:

	Multiple Bidirectional Gateways	Privileged System	Autonomous, Middleware-independent System
Data Exchange	between each pair of monitoring systems	to and from the privileged system	to and from the autonomous system
Nr of Components	the overall number of gateways needed equals the squared amount of monitoring systems	the amount of gateways needed to the linear amount of existing monitoring systems	gateways needed to the linear amount of existing monitoring systems
Scalability	hardly scalable	scales with the number of supported information providers	scales with the number of supported information providers
Data Duplication	no prevention	no prevention	no prevention
Real-Time Data Provision	can hardly be guaranteed	can be close to real-time	can be close to real-time
Dependency	self-developed and maintained	depends on the future development of the privileged middleware	self-developed and maintained
Capabilities	limited to the monitoring systems	limited to the privileged system	self-developed

Table 4.1.: Examination of Architectures of an Interoperable Information System

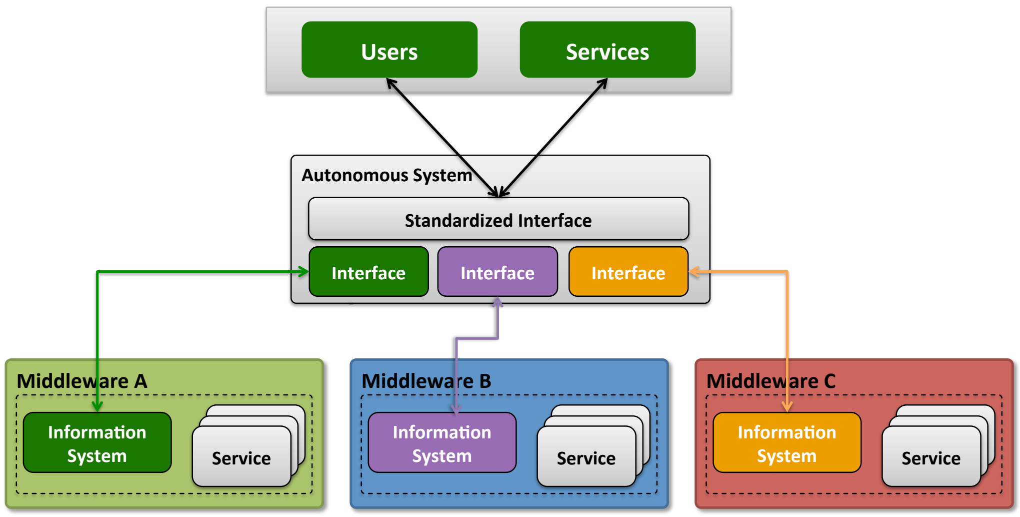


Figure 4.2.: Autonomous, Middleware-independent Monitoring and Information System

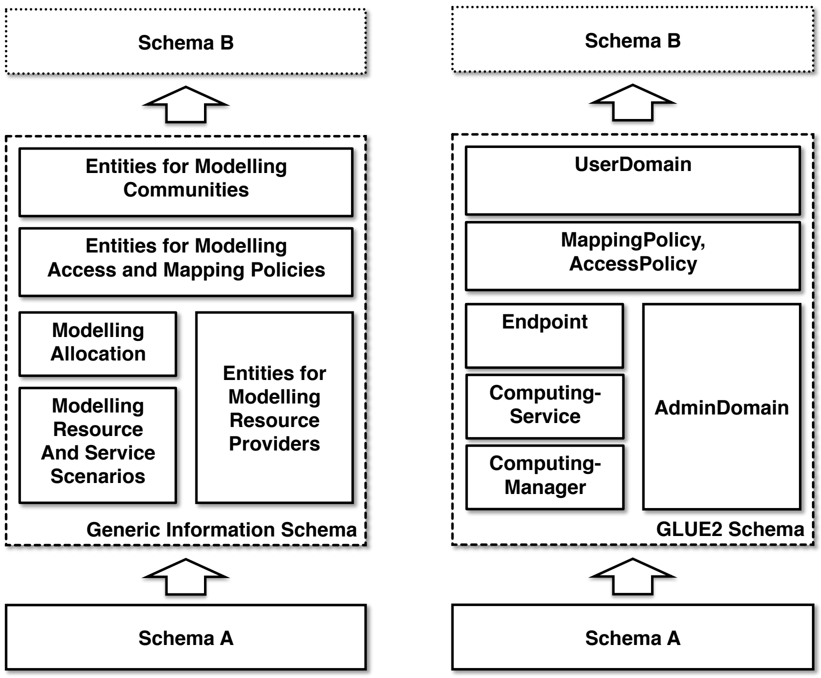
- The *data exchange* criterion describes how the monitoring data is exchanged between the involved systems.
- The *number of components* criterion captures how many components are required to realize the given approach.
- The *scalability* criterion shows how the approach scales.
- The *data duplication* criterion reflects whether the approach prevents data duplication, which results from different naming and conventions.
- The *real-time data provision* criterion reflects whether the provision of all data in real-time can be guaranteed.
- The *dependency* criterion shows whether the approach depends on the future development of the middlewares and other components.
- The *capabilities* criterion reflects what kind of limitations the systems have.

We summarize the results of our examination in Table 4.1. In brief, our examination reveal the following:

Multiple Bidirectional Gateways This approach realizes data exchange between different monitoring systems by building multiple bidirectional gateways between each pair of monitoring systems. The solution is hardly scalable as the overall number of gateways needed equals the squared amount of monitoring systems. Besides, preventing data duplication, which results from different naming and conventions, is a major challenge and the provision of all data in real-time can hardly be guaranteed.

Privileged System This alternative defines one of the monitoring systems as the privileged system to collect all monitoring information as shown on Figure 4.1. That way, for each monitoring system only one gateway is needed, which transfers data to and from the privileged system. This solution reduces the amount of gateways needed to the linear amount of existing monitoring systems, but it is limited to the capabilities of the privileged system and depends on the future development of the privileged middleware. Furthermore, it contradicts the demand of strong interoperability. It solves neither the problem of real-time data provision, nor duplicated data.

Autonomous, Middleware-independent System This approach is depicted on Figure 4.2. It uses an autonomous monitoring system which is independent from the middlewares. The autonomous, middleware-independent service utilizes a storage component, e.g. a database,


Schema B
Entities for Modelling Communities
Entities for Modelling Access and Mapping Policies
Modelling Allocation
Modelling Resource And Service Scenarios
Entities for Modelling Resource Providers
Generic Information Schema
Schema A
Schema B
UserDomain
MappingPolicy, AccessPolicy
Endpoint
Computing-Service
Computing-Manager
AdminDomain
GLUE2 Schema
Schema A

(a) Schema Mediation Utilizing Generic Entities for Distributed Research Infrastructures

(b) Schema Mediation Utilizing the GLUE v2.0 Information Schema

Figure 4.3.: Schema Mediation

with gateways connected to the underlying native monitoring services. This method would scale with the number of supported information providers, while being independent of a privileged middleware.

The disadvantages of the first and second options lead to consider the third one as our preferable basic architecture. The architecture of an autonomous, middleware-independent monitoring system motivates our work in the following subsections.

4.3.2. Transformation of Resource Infromation and Monitoring Data

Gathering and incorporating heterogeneous monitoring data and resource information builds the core of the interoperable monitoring and information system to be developed. To transform the complex monitoring data into an integrated data structure efficiently, an automated ETL-process is defined. We recall our investigations from Section 3.7 how the *Extract, Transform, Load* (ETL) data warehousing technique can be adapted to map existing information models onto a generic model. The monitoring information is collected asynchronously by middleware specific extractors (E), which use different protocols and data models. The extracted resource information and monitoring data is parsed, checked and transformed (T) in a flexible way into a common data structure, e.g. the GLUE v2.0 schema. Finally the monitoring data will be stored (L) in the target repository.

For a common data structure utilized by the target repository, we use our results regarding the generic entities of a theoretical information model, that are capable to describe the main characteristics of heterogeneous, distributed research infrastructures (Section 3.4). We recall the results of our theoretical analysis regarding the information demand in distributed research infrastructures (Chapter 3): after identifying the *main actors* of the distributed research infrastructures, investigating on their *generic usage scenarios*, as well as analysing the *requirements for resource information* and monitoring data from the perspectives of these actors, we identified the main *characteristics of resource descriptions*, and we found that a generic information model should describe the following main characteristics of a distributed research infrastructure (Section 3.4): (a) communities, (b) access policies and mapping policies, (c) allocation of resources and services to communities, (d) modelling resource and service scenarios, and (e) modelling resource providers. The characteristics of these basic set of information, and consequently the generic entities, are depicted on Figure 4.3a.

The GLUE v2.0 schema describes the main characteristics of a distributed research infrastructure. We define the mappings between our generic entities and the GLUE v2.0 entities as follows:

- community modeling: UserDomains,
- access policies and mapping policies: MappingPolicy and AccessPolicy,
- allocation of resources and services to communities: Endpoint,
- modelling resource and service scenarios: ComputingService and ComputingManager
- modelling resource providers: AdminDomains

We depicted the entities of the GLUE v2.0 schema on Figure 4.3b.

Particularly, our design uses the GLUE v2.0 schema for the mediation of resource information and service monitoring data as illustrated in Figure 4.3b: The automated ETL process gathers data from the connected monitoring services using schema A. All data is transformed into the generic information model, GLUE v2.0. From that format, the data can be transformed into any other schema B. Thus, this design enables to cross-provide monitoring data, for example from a monitoring system using schema A into the interoperable monitoring service and from there into a monitoring system utilizing schema B or vice versa.

4.3.3. Monitoring in a Community-specific Context

In order to design a community-aware monitoring system which regulates access to the monitoring data in a community-specific context, proper authentication and authorization methods are necessary. Several computing paradigms, utilized to set up distributed systems, allow to share resources on a community basis. For instance, grid computing incorporates the concept of the *Virtual Organization* (VO), which allows to use a subset of physical resources contributed to that VO. The virtual organizations can be constituted by communities and a VO is not just a group of community users, but it also consists of virtual resources and services, which are provided for the specific community. Another example is the cloud computing paradigm, which utilizes access control based on federated identity management. The single-sign-on concepts of clouds enable users to access independent services using credentials provided by organizations, which maintain identity information within a federation.

Hence, the research infrastructures have to provide community-aware services. The necessity of community-aware infrastructure services has been discussed in [17], as well as in our previous works [18, 21]. Community-aware services allows *"different contexts of resource and service allocation with respect to community specific requirements and technology"*. [21] In order to provide monitoring data and exchange resource descriptions, we detail the design of an integrated, community-aware monitoring- and information service.

The recent information services utilized by our application domains described in 4.1, have been focused on monitoring physical entities. The mapping of the resources and services onto communities is usually ignored. We revive from our previous work [21] that *"a VO-centric approach would simplify a user's life as only information belonging to their VO would be extracted and presented to them, whereas all information they have no use of (such as the providers' cluster status) would be filtered. In addition, a VO-based privacy protection is possible"*.

Therefore, to enable community-specific information provision, the identity of the affected community must be included in the monitoring data so that it can be related to the resources and services the community has allocated.

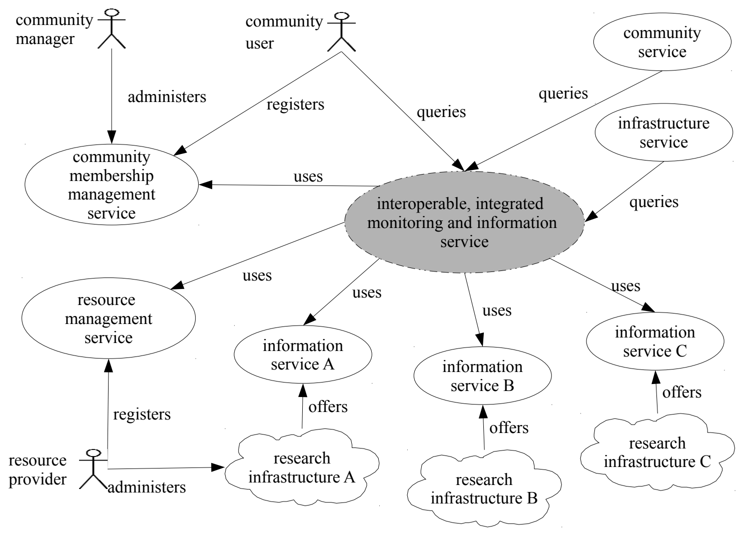


Figure 4.4.: Use Case Diagram of an Interoperable, Integrated, Community-based Monitoring and Information System

Furthermore, to regulate access to the monitoring data in a community-aware manner, proper authentication and authorization methods are essential. Each interface, which provides access to the monitoring and information system, should include the identity of the querying entity (user, community, or service) that wants to retrieve the data. With this information, the community's membership management service can be asked for the entity's authorization.

As a part of our modeling activity described in Section 3, we defined a high-level use-case model which describes how existing monitoring and information systems of distributed research infrastructures are currently used. We also added there a new component - the interoperable, integrated, community-based information and monitoring system (see Figure 3.3). In this Section we further detail the design and extend the use-case diagram with the existing monitoring and information systems of the research infrastructures. Our extended use case diagram is depicted in Figure 4.4. In the middle the integrated, interoperable monitoring and information service is depicted, which uses the information services of the underlying infrastructures to collect status information. An integrated monitoring system also relies on community membership management services, which have information can be asked for the entity's authorization. We depicted these components on the left side of the Figure. On the right, community-specific services and generic infrastructure services are modeled, which make use of the standardized interface of an integrated monitoring service. Since these services usually act on the behalf of a user, we do not differentiate between the users' queries and services' queries, and consider them as same.

We continue with applying the policy enforcement scenario and extending the design of an integrated, community-aware service that provides monitoring data and resource descriptions.

4.3.3.1. Policy Enforcement Scenario

The access control in distributed research infrastructures is often based on a policy enforcement scenario. A policy enforcement scenario utilizes the components *Policy Enforcement Point* (PEP) and *Policy Decision Point* (PDP) to decide if access to protected resources shall be granted or not. Figure 4.5 describes how a typical policy enforcement scenario processes:

1. The requests to access a protected resource are intercepted by the *Policy Enforcement Point*.

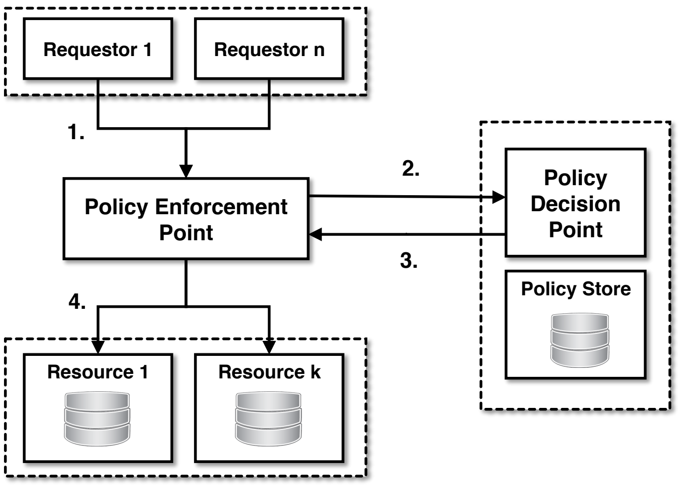


Figure 4.5.: Access Control Utilizing Policy Enforcement Points and Policy Decision Points

2. To authorize the request, the *Policy Enforcement Point* forwards it to a *Policy Decision Point*.

3. The *Policy Decision Point* evaluates the authorization decision request depending on the context, by which the *Policy Decision Point* may refer to a Policy Store in some implementations. The *Policy Decision Point* issues the decision and returns the result to the *Policy Enforcement Point*.

4. The decision of the *Policy Decision Point* is enforced by the *Policy Enforcement Point*: if there were sufficient privileges, then the PEP allows access to the protected resource, else it blocks the access.

In the work at hand, we apply and extend these ideas for an integrated monitoring system:

- The monitoring and information service of distributed research infrastructures should act as *Policy Enforcement Point*. The monitoring and information system intercepts request for accessing monitoring data and it enforces the *Policy Decision Point*'s decision.
- The *Policy Decision Point* should be the same component that is also used by other services of the distributed research infrastructures. Generally speaking, the community-membership or VO-management systems are components, which evaluate and issue such authorization decisions.

Figure 4.6 describes how the interoperable, integrated monitoring and information service acts as a *Policy Enforcement Point*. In order to regulate access to the monitoring data, the monitoring system needs to know the identity of the querying entity (user or service) that wants to retrieve the data. The users of a distributed infrastructure register themselves in a community using the community's membership management service (1). After the community manager granted the necessary permissions (2), users may use the services of the community. The user's identity is included for each query to the monitoring system (3). The monitoring service uses this information to ask the community membership management PDP for the entity's authorization (4). The authorization information also contains the querying entity's communities and thus it can be determined which information is allowed to see. Authentication and authorization methods can be based on community membership management systems. They define the membership of accessing entities to specific communities and this information can be used to find out which community's data an accessing entity is authorized to retrieve (5). As soon as credentials of a querying entity are known, the query can be filtered and the corresponding monitoring data and resource descriptions can be provided (6).

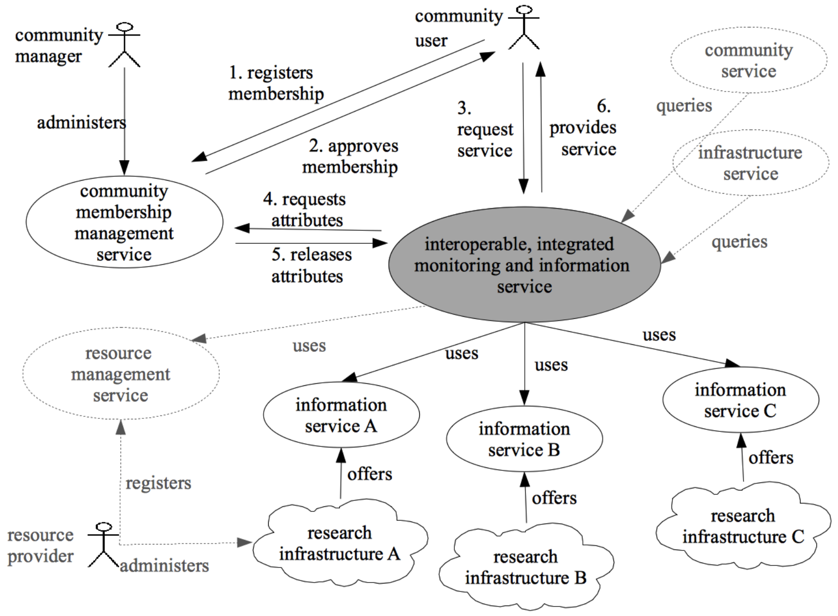


Figure 4.6.: Using VO-attributes in Integrated Monitoring

Our aim is to design a community-aware monitoring system. Such a system can be realized by using external *Policy Decision Point* (PDP) or policy engines which have knowledge about the actual mapping of resources and members onto communities [19]. Here we use the term *Policy Decision Point* in both organizational and technical meaning. In the following we give a brief overview how we distinguish between the organizational and technical characteristics of PDPs:

Organizational Policy Decision Point An organizational (sometimes also referred as political) PDP denotes a board or body, *"which aggregates and coordinates the multilateral relationships of the different contractors in a [distributed research infrastructure] such as resource and service providers or customers organized [as communities and VOs]. The organizational PDP may be hidden so that the contractual or political mappings are not explicitly known."* [21].

Technical Policy Decision Point A technical PDP defines the mappings *"in a technically and deterministically processable representation. Unlike the organizational PDP it must not be hidden. The representation can then be used as a template to compose the [communities] as well as related monitoring data according to the community's actual allocations of resources and services."* [21].

In this work, we use the terms *Policy Enforcement Point* and *Policy Decision Point* in the same way as RFC3198 [228] does. However, we also point out that access control architectures and concrete implementations of security models introduce further terms, like *Policy Administration Point* and *Policy Information Point*. In the following, we also describe these two terms briefly:

Policy Administration Point A *Policy Administration Point* is a service, tool, or interface that provides support for creating, editing, and managing policies or policy sets. That way, a *Policy Administration Point* (PAP) simplifies the implementation of changes, because policies can be deployed to defined architectural entities. A PAP is also referred to as *Policy Management Authority* (PMA) as well as *Security Identity Governance and Administration* by several enterprise implementations.

Policy Information Point The *Policy Information Point* component provides additional - usually external - information on attributes if necessary. We described above how a typical policy enforcement scenario processes. After the PEP transferred the request details to a PDP for an authorization decision (Step 2. on Figure 4.5), the PDP evaluates the authorization decision request depending on the context. If there were attributes not part of the request, a *Policy Information Point* (PIP) provides external information to help the PDP's authorization decision.

In this work we focus on regulating access to the monitoring data and resource information. To answer that purpose, we consider the PAP and the PIP as constituents of the PDP. We however refer the interested reader to our other works on the field of distributed research infrastructures that give a deeper insight into community-specific access control based on identities maintained within a federation as well as single-sign-on concepts utilizing federated identity management [136, 137].

In Section 4.4.3 we describe the implementation details for our application domain, as well as the components and services which may serve as a *Policy Decision Point* for the scenarios outlined in Section 4.1.

4.3.4. Distributed Setup

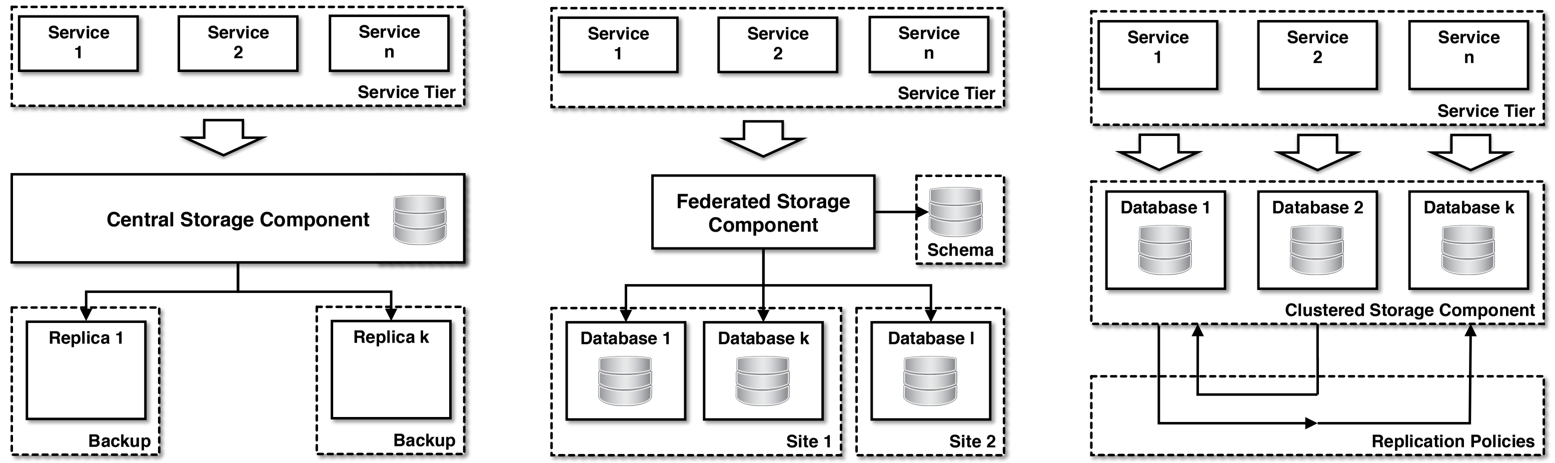
The architecture of an interoperable, integrated monitoring and information system should be scalable and efficient. This implies low resource consumption and short response time for accessing data. The selected core architecture is an autonomous, middleware-independent service which utilizes a storage component with gateways connected to the underlying native monitoring services. The storage component can be set up in central, clustered, or federated manner, where the latter can be realized by horizontal and vertical partitioning. In the following we briefly summarize the main characteristics of the four options:

Central Storage Component A monitoring service with a central storage has to contain the integrated data from all connected site information services. This setup will not scale in large-scale scenarios, because a frequent update rate induced by a large number of gateways and sites as well as many client queries will cause high load in the central component.

Clustered Storage Component An alternative is to use clustered storage systems distributed among the connected sites. This architecture is well known from database clusters but it has the disadvantage of being too unstable when distributed over the Internet.

Federated Setup with Horizontal Partitioning A federated setup connects multiple autonomous services as federated services. The autonomous services are located directly at the resource providers' sites and query only the monitoring systems of the local provider. This approach is very similar to horizontal partitioning utilized in distributed database systems: Horizontal partitioning (also referred to as sharding) splits the resource information across sites, like distributed database systems split tables within databases by *row* and distributes them across multiple databases. Through horizontal partitioning the monitoring data and resource information can be accessed with no interruption by utilizing repartitioning operations when necessary. The federated services realize an overlay network which distributes the monitoring data and allows a further optimization by using aligned data distribution algorithms and protocols.

Federated Setup with Vertical Partitioning Distributing the storage component physically across multiple locations is also possible by vertical partitioning of the data. Vertical partitioning spreads different *types* of resource information at different sites, like distributed database systems spread different *columns* of a table at different sites. As a consequence, the advantages and disadvantages are very similar to horizontal partitioning - except that



(a) Central Storage Component (b) Federated Storage Component (c) Clustered Storage Component

Figure 4.7.: Central-, Federated- and Clustered Storage Component for an Autonomous, Middleware-Independent Service

combining data across vertical partitions is more challenging because it requires joins, instead of using unions.

We summarize the main advantages and disadvantages of distributed storage components compared to centralized storage components on Table 4.2.

Advantages	Disadvantages
Reliability and availability increased	Implementation costs and software complexity
Expansion is modular	Processing overhead
Business unit (site) autonomy	Less or no local control over data
Better performance for certain queries	Inconsistent query time
Shorter communication ways	Data integrity issues
Lower data communication costs	Location of data untransparent
Location transparency	Backup vulnerability
Analytic processing efficiency	

Table 4.2.: Main Advantages and Disadvantages of Distributed Storage Components Compared to Centralized Storage Components

Distributed storage components deliver data from where they are stored to where queries are processed. Extensive research has already been made on the possible data delivery alternatives and their rich design space. Characterization along the dimensions (1) delivery modes, (2) frequency, and (3) communication methods was given in [168]. According to that:

1. The data delivery alternatives are (a) pull-only, (b) push-only, and (c) hybrid.
2. The typical frequency measurements, which are used to classify the regularity of data delivery, are (a) periodic, (b) conditional, and (c) ad-hoc or irregular deliveries.
3. The ways in which servers and clients communicate for delivering information to the clients are: (a) unicast and (b) one-to-many.

Comparisons of design strategies for distributed storage components have also been made in detail. We refer exemplary to [111], that describes the following five organization models for distributed databases: (1) centralized database with distributed access, (2) replication with periodic snapshot update, (3) replication with near real-time synchronization of updates, (4) partitioned with one logical database, and (5) partitioned with independent, nonintegrated segments.

We used the above classifications of data delivery alternatives and distributed design strategies of storage components for our implementation. In Section 4.4 we describe the implementation details for our application domain and for the scenarios outlined in Section 4.1.

4.3.5. Discovering Monitoring and Information Services

To retrieve resource information from the underlying monitoring systems, an exact list of information services is required. This precise list of entry points is vital for the ETL process, which requires automatically generated, general set of information about the monitoring services from

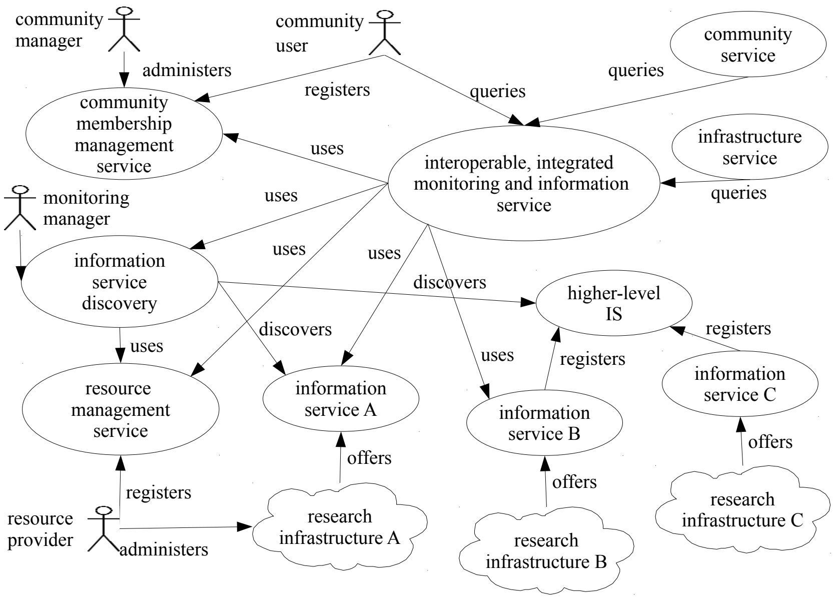


Figure 4.8.: Information Service Discovery for the Integrated, Interoperable Monitoring System

various (technical) platforms. Unfortunately, no central service exist, which would currently provide such information.

An integrated registry for different type of information services is needed that respects the technically incompatible monitoring services. To describe information services in an interoperable way, the description schema of such a registry should support our generic information model. Also other standards (for interfaces, organizational requirements, etc) should be supported.

We considered several approaches to realize an integrated information-service registry and evaluated their individual characteristics[18]. In our work [148] we examined three approaches: (a) extend an existing resource registry, (b) extend an existing information service, and (c) integrated service discovery using existing information services. In the following, we describe our preferred approach: the integrated service discovery using existing information services. This approach can deliver *dynamic data* and is based on an external, *independent component*. This model uses the federated information service infrastructure of technically different middlewares to discover all the registered information services in real time. If the monitoring infrastructure was built hierarchically, then the underlying monitoring services can also be discovered (See Figure 4.8). Using this information, a dynamically generated, common, standard-based registry of all registered information services of technically different infrastructures can be established. The exact monitoring data can be gathered directly from the underlying monitoring systems by adaptors using the integrated service registry. Consequently, the hierarchy and the corresponding latency can be skipped.

This approach provides a sustainable solution for grids using different monitoring infrastructures and requires neither dependency from the underlying grid middlewares nor resource providers to be involved. Used together with an interoperable, integrated grid monitoring system, it can deliver the most up-to-date status of different grid infrastructures.

[18]We designed three models for the D-MON project. The German Grid Initiative, furthermore, adopted one of our models as its discovery architecture for information systems of distributed research infrastructures.

4.4. Implementation and Case Study

In this Subsection we describe the implementation details of an interoperable and integrated information service within the distributed research infrastructure set up by the German e-Science Initiative[19]. The expertise of the German grid initiative D-Grid [164] provided an excellent basis for studying and learning about distributed research infrastructures and computing paradigms. The work together with colleagues from the D-MON project enabled to experience and apply production-like grid and cloud environments. Our results were adopted as a production level service by the German Grid Initiative.

This heterogeneous research infrastructure provided a perfect ecosystem to evaluate our concept for a distributed monitoring architecture for an interoperable and integrated monitoring- and information system. This section is partly adapted from our presentations [127, 128, 130, 131, 148], as well as from [18, 21].

4.4.1. D-MON Monitoring System

The *D-Grid Monitoring Project* (D-MON) [55] has made it its mission to lay the foundations for a research infrastructure-wide monitoring- and information system, which works with the various standards as well as external tools. We briefly presented the scenario of the distributed research infrastructure set up by the German e-Science Initiative in Section 4.1 and we referred to concepts with respect to monitoring (cf. [223, 20, 178]). In the D-Grid scenario, compute resources are offered through three middleware solutions, Globus Toolkit [79], gLite [97], and UNICORE [221] or through either of them. Storage resources are accessed via the dCache [90] and OGSA-DAI [10] middlewares. The communities use the middleware they are familiar with and which are best suited for their applications.

To gather resource information and monitoring data within the heterogeneous D-Grid infrastructures is a complex task, as each of the middleware implementations has its own tools and solutions for observing the status of the various resources. The monitoring is usually realized in a way specific to the utilized middlewares: Globus Toolkit sites use the *Monitoring and Discovery Service version 4* (MDS4) [200] information service, UNICORE 6 sites are monitored through the *Common Information Service* (CIS) [156], and gLite sites use the *Berkeley Database Information Index* (BDII) information service. Various tools for specific purposes are also utilized.

A D-Grid-wide monitoring system is relevant for users of the various communities as well as for the administrators of the individual resource provider centres. The D-MON system can monitor the status of resources, services and jobs that are running in the various middleware environments. It combines data unification and categorization with policies for community membership, VO resource management, and data transformations between different data models. It also realizes community-aware access to monitoring data gathered from multiple resource providing organizations as well as from various middlewares.

4.4.2. ETL for Globus Toolkit, UNICORE, and gLite

Our concept for an interoperable, integrated monitoring and information system foresees a thee-steps process (extract, transform, and load) between the middleware-specific information services and the integrated information database. This process of (1) extracting data from BDII, CIS, and MDS4, the three existing monitoring systems in D-Grid, and (2) transforming this data to the GLUE 2.0 schema, and (3) loading it into the target database is realized in D-MON by middleware-specific gateways, one per middleware. The extractor components of the gateways operate with the standard client APIs of the middlewares. The data transformation is implemented using Extensible Stylesheet Language Transformations (XSLT), which is a popular language for processing

[19]We designed the initial architecture together with colleagues from the D-MON [55] project. The German Grid Initiative [164], furthermore, adopted our results as its grid monitoring architecture.

XML data or transforming XML documents into other formats. This choice presented itself as most of the information services provide their data in XML format. The data is augmented with the identifiers of the information provider component to keep track of data provenance. In the following we highlight key aspects of the implementation details of the steps extract, transform, and load.

4.4.2.1. Extraction of Monitoring Data

The monitoring data collected by the various sensors can be extracted from the monitoring and information services of the middlewares. Because of historical reasons, several different implementations of monitoring and information services are utilized in grids. The different implementations support different interfaces, data formats, and sensors. Generally said, the utilization of a concrete implementation depends on the middleware, the community (virtual organization), the resource provider, and the project. The most common monitoring services and the most widely deployed middlewares in D-Grid are presented below. Since our aim is to use these services for extraction of monitoring data and resource information, we also briefly describe their available interfaces, data formats, and sensors.

MDS4: the Information System of Globus Toolkit v4 The *Monitoring and Discovery Service version 4* (MDS4) is the standard information service to monitor resources and services in a distributed research infrastructure based on the *Globus Toolkit 4* (GT4). The Globus Toolkit provides an open source collection of services that support *Web Services Resource Framework* (WSRF) [81] specifications and follow *Open Grid Services Architecture* (OGSA) [80] architectural principles. The MDS4 offers notification mechanisms and triggers. Via the OGSA and WSRF compliant interfaces it offers access for services, such as schedulers, monitoring portlets, benchmark platforms, that require access to resource information or monitoring data. Several WSRF based GT4 services (RFT, WS-GRAM) publish their monitoring information into MDS4 automatically. Implementing additional information providers are also possible, for instance, for job monitoring. MDS4 keeps the monitoring data in a non-persistent way, in the memory allocated for the MDS4 web service. There is currently no way to archive the monitoring data or query historical monitoring information.

In the following, we describe some details, which differ from a default installation and are specific to the MDS4 deployment in D-Grid. The infrastructure of the GT4 resource monitoring of the D-Grid is based on a 3-tier architecture. On the lowest level there are MDS4 site-indices. An MDS4 site-index provides resource information about the site. Depending on the topology of the grid site, more then one Globus Toolkit installations can exist and consequently, more MDS4 instance can exist at the site. In this case, the information services are arranged in a local hierarchy and one of the MDS4 installations is chosen for acting as a Site-Index. All other information services of the site are then registered to the chosen Site-Index.

The MDS4-Site-Index provides monitoring data to one or more Community-Indeces. D-Grid maintains and publishes a list of the service *Uniform Resource Locator* (URL)s of the community indices. The users of the community or other grid services of the virtual organization use the URL of an MDS4-Community-Index to get information about the GT4 resources available for the community. The Community-Index registers itself to the central D-Grid MDS4 Index. The central D-Grid MDS4 operates on the top of the Globus monitoring hierarchy. In the D-Grid set-up two redundant MDS4 installation exist and the community indices provide resource information for both of them. The top-level MDS4 has information about the community and the site MDS4 indices.

A further important difference from the default installation of the Globus Toolkit that a default installation does not publish hostnames, but instead publish IP addresses of the monitored hosts. However, grid users and grid services usually work with hostnames. To enable interoperation with other services utilizing hostnames, for example resource registry, all MDS4s must be configured to publish the fully qualified domain name instead of the IP address of the Globus Toolkit frontends. The required entries are shown in Listing 4.1.

```
<xml version="1.0">
<!-- [...] -->
<!-- This publishes the hostname instead of the IP address -->
<globalConfiguration>
  <parameter name="logicalHost" value="globus-mds4.domain.com"/>
  <parameter name="publishHostName" value="true"/>
  <!-- [...] -->
</globalConfiguration>
```

Listing 4.1: Publishing Hostnames for Monitoring instead of IP Addresses

Our extract process utilizes the *wsrf-query* client application of the Globus Toolkit 4. MDS4 and *wsrf-query* support searches via XPath and XQuery.

CIS: the Information System of UNICORE The UNICORE6 middleware provides a *Common Information Service* (CIS) monitoring system, which accumulates static and dynamic monitoring data of various UNICORE6 services. The CIS stores the monitoring data in an *eXtensible Markup Language* (XML) database. CIS publishes the data via *Rich Site Summary, formerly "RDF Site Summary", often called "Really Simple Syndication"* (RSS) feed in Atom Syndication Format. Furthermore, the CIS monitoring data can also be queried via standard query mechanisms, such as *XML Query Language* (XQuery) and *XML Path Language* (XPath). A map-based user interface is also available to display the CIS's aggregated resource information.

As part of the UNICORE6 components, an information provider extracts the monitoring data and resource information at the UNICORE sites. This information provider component, called the *Common Information Provider* (CIP), passes the data to the CIS on request. The current information model of the CIS is the GLUE2 Schema, while previous implementations used the *Common Information Model* (CIM) Schema.

The CIS, as it is deployed in D-Grid, provides resource information as well as data for system monitoring, but currently no detailed job-monitoring and accounting data are available. It enables authenticated clients only, which is why our implementation utilizes the *UNICORE Command Line Client* (UCC) to extract data from the CIS,.

MDS2: the Information System of Globus Toolkit v2 The *Metacomputing Discovery Service* (MDS2) is based on the OpenLDAP software and represents a decentralized monitoring architecture. The information provider components collect dynamic and static information on the resources and make them available in *LDAP Data Interchange Format* (LDIF) format of the *Lightweight Directory Access Protocol* (LDAP). The MDS2 also implements a caching mechanism by identifying the information with a *Time-To-Live* (TTL) field. The monitoring data and resource information is stored on a *Grid Resource Information Service* (GRIS), which runs on all resources. Each GRIS can provide data to a *Grid Information Index Service* (GIIS), which is responsible for the monitoring data on the site level.

The MDS2 hierarchy can easily be extended, because a GIIS can act both as client and server. Two schemas can be utilized as GRIS and GIIS data format: (1) the MDS Core Schema and (2) the GLUE v1.1 Schema.

Our implementation takes advantage of MDS2's standard compatibility: Since MDS2 is based on the *Lightweight Directory Access Protocol* (LDAP) v3 standard, both a GRIS and a GIIS can be queried by a normal LDAP client.

BDII: the Information System of gLite The *Berkeley Database Information Index* (BDII) is the current monitoring service of the *Lightweight Middleware for Grid Computing* (gLite) [7] middleware. It is based on the established monitoring service of older Globus Toolkit versions (MDS2). The BDII serves as a data source: it stores the monitoring data and makes it available via *Lightweight Directory Access Protocol* (LDAP).

Like in the MDS2 design, a local *Grid Resource Information Service* (GRIS) runs on the resources and it gathers dynamic and static information from extensible information providers. A *Berkeley Database Information Index* then serves as a cross-domain aggregator who collects the data of all GRIS components. The BDII hierarchy is freely stackable: a BDII can act as client as well as server. Thus, it can also collect data from other *Berkeley Database Information Index* (BDII) installations.

Although MDS2 was marked by its developers as deprecated, we found that MDS2 and especially BDII continue to be extremely important components in our implementation domain.

Our implementation uses that BDII is based on the *Lightweight Directory Access Protocol* (LDAP) v3 standard. That way, a BDII can be queried by a normal LDAP client.

RGID/D-GRDL (Life Sciences and Instant-Grid) The *Reliable Grid Information Database* (RGID) [127, 128] is based on the description language *D-Grid Resource Description Language* (D-GRDL) [232], which is intended for resource descriptions in the D-Grid. Currently, it is primarily applied in MediGRID and Instant-Grid to support the scheduling of tasks and to provide monitoring data [127, 128, 135].

The RGID can be used as an information provider on a local resource, as well as an aggregator for other monitoring services (e.g. MDS4, Ganglia). The resource information is stored in a native *eXtensible Markup Language* (XML) database (eXist) and are in XML format (D-GRDL). Thus, a transformation in other data formats is easily possible.

The resource information is accessible via JSR-168 compliant portlets, which can be integrated in standard portal frameworks supporting the *Java Specification Request* (JSR) standard. Since RGID utilizes a native XML database, standard query mechanisms such as *XML Query Language* (XQuery) and *XML Path Language* (XPath) are also supported.

For the implementation work presented in this thesis is important that RGID is able to combine resource information with the test results of other monitoring- and benchmarking services [127, 132].

4.4.2.2. XSL Transformations for Schema Interoperability

The next implementation step is to transform the extracted data to the GLUE 2.0 schema. For the transformation we define schema mappings, which are sets of rules and expressions that describe how the data from the source system has to be transformed into the target system. As most of the source information services provide their data in XML format, our implementation utilizes *eXtensible Stylesheet Language Transformation* (XSLT), which is a popular language for processing XML data or transforming XML documents into other formats. In the following, we describe the main characteristics of our XSL transformations.

A Note on Schema Interoperability and Information Lossless Our implementation focuses on the monitoring- and information systems BDII, CIS, MDS4 as source systems. In the previous Subsection we described the information models, which are utilized by these systems.

Our schema mapping (our transformation) describes which attribute of which entity should be mapped onto which attribute of which GLUE v2.0 entity. We needed to define such precise field-to-field mappings for the most important entities of the utilized schemas. This can however be a challenging issue if the schemas were not interoperable.

An example for schema interoperability is the way, how the various schemas model computational resources. In the GLUE v2.0 information schema, a new abstract entity - called ComputingService - was defined to model the resources, which provide computational capacity. This is the main logical unit and aggregates further entities modeling computing capacity in distributed research infrastructures. The batch queue information of the local resource manager system is

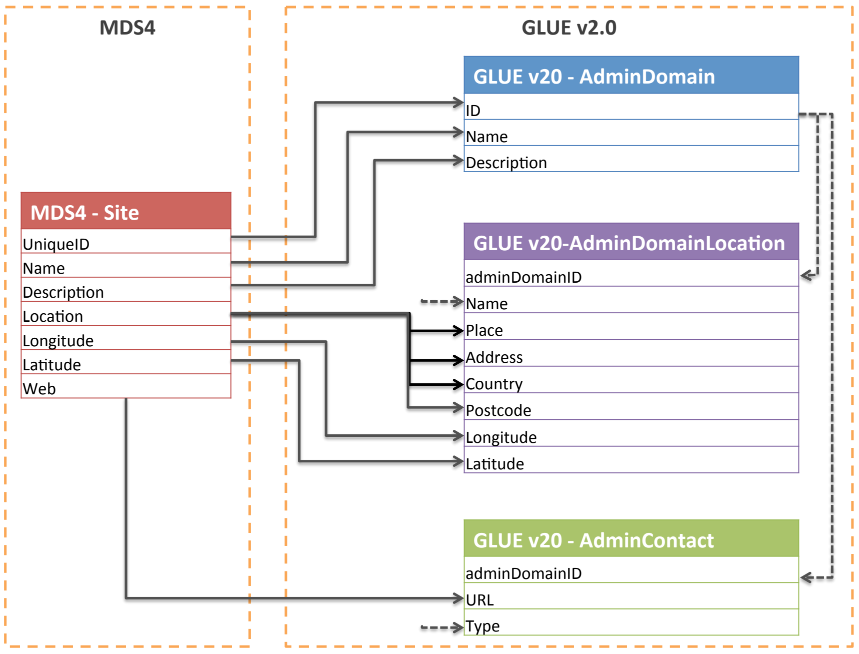


Figure 4.9.: Schema Mapping for the MDS4 Site Entity and GLUE v2.0 AdminDomain

expressed by ComputingShares. The jobs are submitted and monitored via interfaces provided by Endpoints that do not keep information about the jobs.

We defined a logical mapping table, which describes how the attributes of the BDII, CIS, MDS4 entities are to be mapped to the attributes of the GLUE v2.0 ComputingService and to the GLUE 2.0 ComputingShare entities. The precise field-to-field mapping for ComputingService is presented at Table B.4 (Appendix). For ComputingShare, the Table B.5 (Appendix) shows, how the attributes can be filled by using the existing information services of the middlewares gLite, UNICORE6, and Globus Toolkit 4.

In the presented example it was possible to create one-to-one mapping between the attributes MDS4-BDII and GLUEv2 schemas. However, there is no precise mapping for CIS and ComputingShare, since the CIS version we worked with, does not publish informations on the queueing systems. Since our infrastructure scenario contains 3 parallelly deployed middlewares on every site, our logical schema mapping fills the generic model's major attributes by other middlewares. That way, the information about running or waiting jobs in a batch queue can also be exchanged.

A Further Note on Schema Mapping There are scenarios, where defining a one-to-one mapping is challenging. The transformation of MDS4's *Site* entity into the GLUE v2.0 information model demonstrates such a scenario.

The Site entity describes resource and service providers, but it does not exist in the GLUE v2.0 information model. The GLUE v2.0 schema describes resource providers by defining the *AdminDomain*, *AdminDomainLocation*, and *AdminContact* entities. Therefore, the Site entity needs to be split into new entities. A further significant difference between the models that an administration domain may have more locations and contact addresses.

Our mapping transforms the name and description of a provider into the AdminDomain entity, while the location information of a site is transformed into the AdminDomainLocation entity. The various contact addresses of the site (email address, website) are stored in the AdminContact entity. Note that our mapping does transform the unique identifier of the site into the 3 new entities, because the Location and Contact entities use the unique identifier of the Domain as a foreign key. Figure 4.9 illustrates the transformation of the Site entity of the MDS4 information model.

Figure 4.9 also demonstrates that not the complete entity, but an attribute may have to be splitted into more new attributes. The source schema utilizes the Location attribute to store address information of a resource provider. The AdminDomainLocation entity of the GLUE v2.0 schema has a refined design: Name, Place, Address, Country, and Postcode attributes can be used.

A more detailed description of the logical mappings are provided in the Appendix. The mapping table of the AdminDomain entity is described in Table B.1 (Appendix), the AdminDomainLocation entity in Table B.3 (Appendix), and the AdminContact in Table B.2 (Appendix).

Transformation example for MDS4 In the previous subsection (Section 4.4.2.1) we discussed MDS4, the monitoring system of the Globus Toolkit 4 middleware. We recall, that MDS4 can be queried using a WSRF interface and it provides output in XML format. As MDS4 is based on *Grid Laboratory Uniform Environment Version 1.1* (GLUE v1.1) and additional information providers like GeoMaint [95] provide site related information in *Grid Laboratory Uniform Environment Version 1.3* (GLUE v1.3), we implemented the *eXtensible Stylesheet Language* (XSL) transformations for MDS4 to parse, check and map the fields with respect to the different versions of GLUE.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ns0="http://mds.globus.org/index"
  xmlns:ns1="http://mds.globus.org/glue/ce/1.1"
  xmlns:ns11="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-01.xsd"
  xmlns:geomaint="http://infnforge.cnaf.infn.it/glueinfomodel/Spec/V12/R2"
  xmlns:ns="http://infnforge.cnaf.infn.it/glueinfomodel/Spec/V12/R2" exclude−result−prefixes="html">
  <!−− Using the GLOBUS Toolkit and GeoMaint namespaces −−>

  <!−− Creating text based output −−>
  <xsl:output method="text" encoding="ISO-8859-1"/>

  <!−− Identifying the data source by labels−−>
  <xsl:variable name="informationProvider">MDS4</xsl:variable>
  <xsl:param name="sourceAddress" select="'wsrfclient'" />
  <xsl:variable name="insertTime">CURRENT_TIMESTAMP</xsl:variable>

  <!−− The input XML is a valid MDS4 XML document −−>
  <xsl:template match="/">

    <!−− ... some other templates are removed from this example ... −−>
    <xsl:apply−templates select="//ns1:ComputingElement"/>
  </xsl:template>

  <!−− A template for the MDS4 ComputingElement entries −−>
  <xsl:template match="//ns1:ComputingElement">

    <!−− Trying to find the not empty HostName −−>
    <xsl:if test="not(count(./node()[local-name()='Info']/@ns1:HostName)=0)">

      <!−− Creating SQL output by parsing the attributes −−>
      INSERT INTO GLUE20.ComputingShare (
        serviceID, localID, mappingQueue, runningJobs, totalJobs, waitingJobs,
        informationProvider, sourceAddr, insertTime
      ) VALUES (
        "<xsl:value-of select="./node()[local−name()='Info']/@ns1:HostName"/>",
        "<xsl:value-of select="./@ns1:UniqueID"/>",
        "<xsl:value-of select="./@ns1:UniqueID"/>",
        "<xsl:value-of select="./ns1:State/@ns1:RunningJobs"/>",
        "<xsl:value-of select="./ns1:State/@ns1:TotalJobs"/>",
        "<xsl:value-of select="./ns1:State/@ns1:WaitingJobs"/>",
        "<xsl:value-of select="$informationProvider"/>",
        "<xsl:value-of select="$sourceAddress"/>",
        <xsl:value−of select="$insertTime"/>)


      <!−− Updating if monitoring data with the same identifier already exists −−>
      ON DUPLICATE KEY
      UPDATE
        serviceID = VALUES(serviceID),
        localID = VALUES(localID),
        mappingQueue = VALUES(mappingQueue),
        runningJobs = VALUES(runningJobs),
        totalJobs = VALUES(totalJobs),
        waitingJobs = VALUES(waitingJobs),
        informationProvider = VALUES(informationProvider),
        sourceAddr = VALUES(sourceAddr),
        insertTime = VALUES(insertTime);

    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

Listing 4.2: XSL Transformation for a ComputingElement of the MDS4 Schema

Listing 4.2 illustrates an example of the XSL Transformations of the MDS4 schema. The example transforms the batch queue information of the local resource manager system from the format, which is used by the MDS4 schema into the format, which is required by the the GLUE v2.0 schema. The XML output of the MDS4 source is the input for our XSL transformation and it is parsed by the lines up to 33. The subsequent lines the respective SQL query is formed for the ComputingShare component of the GLUE v2.0 schema. To identify the data source, lines 16 - 18 define specific labels that we use for adding data provenance information. An XSL template is defined in line 21 and it is applied to the matching MDS4 elements and their child nodes, mainly the ComputingElement of the MDS4 schema. Whenever new monitoring information is inserted into the database, a unique identifier is used. The lines 31 - 46 show how the data for the ComputingShare component is parsed. To prevent re-insert monitoring data again when the same identifier is already stored in the database, an UPDATE SQL query is created (lines 50 - 60) instead of an INSERT SQL query.

Transformation for CIS The information system of the UNICORE 6 middleware, CIS, already utilizes the GLUE v2.0 schema internally. Thus, special conversion or rearrangement of the data is not needed. Our transformation process is therefore simple: It merely verifies the CIS output, which is an XML representation of the resource information, and it maps the data to the respective SQL commands.

This example shows that not every information service delivers monitoring data for all GLUE v2.0 attributes. There are cases, when the mapping table contains many empty fields because some entries only exist in one schema and not in the other. For example, the CIS implementation we worked with does not provide batch queue information. Consequently, totalJobs, runningJobs, maxTotalJobs, maxRunningJobs attributes of the ComputingShare entity are missing. In such infrastructures where more information systems provide monitoring information from the same resources (parallel deployment) and the resources can be explicitly identified (the same unique identifier is configured in all monitoring systems), it should be checked whether another information system can deliver the missing attributes. Since our infrastructure scenario contains 3 parallelly deployed middlewares on every site, the generic model can be filled on a major attribute basis by other middlewares.

We found that the scenarios of our application domain - described in Section 4.1 in great detail - achieve interoperation by installing different middlewares on the same compute resources. Our data integration process collects monitoring data and resource information from those middlewares. Duplicates of resource information would falsify the results. The transformation process, therefore, also enriches the data with provenance information that can help trace back the derivation history of the resource descriptions and monitoring data.

Transformation example for BDII We described BDII, the information system of the gLite middleware, in Section 4.4.2.1. We found that BDII is based on the LDAP and provides output in LDIF. Since our design foresees transformations using XSL, we implemented our BDII extractor to provide the resource information in *Directory Services Markup Language* (DSML) format. DSML is an XML representation of the directory service information stored in BDII.

We illustrate an example of our XSL Transformations for BDII in Listing 4.3. First, the XML file is parsed in lines up to 30. Then, the respective SQL query is formed for the AdminDomain component of the GLUE v2.0 schema. We use an XSL template in line 20 and match it to the DSML elements. New resource information is always identified by a unique identifier before it is inserted into the target database: the lines 27 - 43 show how the identifier for the AdminDomain component is parsed. Similarly to the presented MDS4 transformation example, whenever monitoring information with the same identifier is already stored, the transformation only updates it (lines 46 - 52). Finally, we refer to lines 10 - 12: Here we define dedicated labels to identify the source monitoring system and to integrate data provenance information.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dsml="http://www.dsml.org/DSML">
  <!−− Using the W3C XSLT and DSML namespaces −−>

  <!−− Creating text based output −−>
  <xsl:output method="text" encoding="ISO-8859-1" />

  <!−− Identifying the data source by labels−−>
  <xsl:variable name="informationProvider">BDII</xsl:variable>
  <xsl:variable name="sourceAddress">iwrdmon−bdii.fzk.de</xsl:variable>
  <xsl:variable name="insertTime">CURRENT_TIMESTAMP</xsl:variable>

  <!−− The input XML is a valid DSML −−>
  <xsl:template match="/">
    <xsl:apply−templates select="//dsml:dsml/dsml:directory-entries/dsml:entry"/>
  </xsl:template>

  <!−− A template for the DSML entries −−>
  <xsl:template match="//dsml:dsml/dsml:directory-entries/dsml:entry">
    <xsl:variable name="DistinguishedName"><xsl:value−of select="@dn"/></xsl:variable>
    <xsl:choose>
      <!−− Trying to find the GlueSiteUniqueID −−>
      <xsl:when test="starts-with(@dn,'GlueSiteUniqueID')">

        <!−− Parsing the unique ID of the AdminDomain −−>
        <xsl:variable name="DistinguishedNameLocal">
        <xsl:value−of select = "substring-after(substring-before(
              $DistinguishedName,',mds-vo-name=local,o=grid'),'o-name=')" />
        </xsl:variable>

        <!−− Creating SQL output by parsing the attributes −−>
        INSERT INTO GLUE20.AdminDomain (
              ID, Name, Description, Distributed,
                informationProvider, sourceAddr, insertTime)
        VALUES ( "<xsl:value-of select="$DistinguishedNameLocal"/>",
              "<xsl:value-of select="dsml:attr[@name='GlueSiteName']/dsml:value"/>",
              "<xsl:value-of select="dsml:attr[@name='GlueSiteDescription']/dsml:value"/>
                -<xsl:value-of select="dsml:attr[@name='GlueSiteOtherInfo']/dsml:value"/>",
              "0",
              "<xsl:value-of select="$informationProvider"/>",
              "<xsl:value-of select="$sourceAddress"/>",
              <xsl:value−of select="$insertTime"/>)

        <!−− Updating if monitoring data with the same identifier already exists −−>
        ON DUPLICATE KEY
        UPDATE
              Name = VALUES (Name),
              Description = VALUES (Description),
              Distributed = VALUES (Distributed),
              sourceAddr = VALUES (sourceAddr),
              insertTime = VALUES (insertTime);

      </xsl:when>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

Listing 4.3: XSL Transformation for the BDII Schema (AdminDomain)

4.4.2.3. Loading Monitoring Data

ETL implementations, or generally speaking data transformations, are realized in a call-by-need manner or in a bulk manner. The first case keeps the data in the source storage component and defines the transformations as specific queries against the source storage component. That way, the updates in the source storage components are immediately and automatically available in the target component. Additionally, the source component - in case of a distributed or federated approach - would keep its autonomy and responsibility. In the bulk approach, a transformation brings the data in the target format and the data is physically loaded into the target storage component, which applies the integrated schema. We mention that in this case the updates in the source systems are not immediately reflected in the target storage component.

We found that the bulk approach is advantageous if (1) querying the source systems is expensive, or (2) the source systems are not updated frequently. As a result, the approach of performing a one-time bulk ETL operation is taken. The loader components store the transformed, homogeneous monitoring information in the D-MON integrated database.

During the implementation we faced several issues, because the released data models of the GLUE v2.0 Schema (and especially its SQL rendering) contained errors, which led to an inconsistent database schema after deploying it at our integrated storage component. We reported the identified failures to the D-MON project, which forwarded the issues to the OGF GLUE Working Group.

This work provides mappings for the stable entities and attributes of the information models. The mapping table is not fully completed and only the most important entities and attributes are mapped. We plan to extend the mapping table with further entities and attributes in the future.

4.4.3. Community-aware Monitoring

Our system design (Section 4.3) foresees the regulation of access to monitoring information in a community-aware way. For each query to be served, the monitoring system needs to know the identity of the entity that would retrieve the data. Using the identity information, the *Policy Decision Point* is asked whether the entity (the user or the service) is authorized. To provide proper authentification and authorization methods the *community- and virtual organization management systems* can be the basis.

From the perspective of a monitoring system, the community membership and virtual organization management systems are utilized to build up virtual organizational structures. Such systems enable a particular mapping of users and resources onto specific communities. Thus, this mapping provides the basis for authentication and authorization management, proper accounting, and especially for community-aware monitoring. Therefore, we briefly describe the functionalities of such systems utilized in our application domain:

VOMS Database The *Virtual Organisation Membership Service* (VOMS) database contains information on the members of a *Virtual Organization*, i.e. a list of users of individual communities. This enables to realize the mapping of users to one or more communities. VOMS is used in our application domain very occasionally, because the management of communities is realized centrally via VOMRS instead.

VOMRS Database The *Virtual Organisation Membership Registration Service* (VOMRS) [62] databases include, in principle, all the information of the VOMS databases. Additionally, a VOMS also offers other configuration options, such as the creation of sub-VOs. For each community a dedicated VOMRS instance is operated in our usage scenarios. The VOMRS database contains all registered users of the distributed research infrastructure. Usually, the contact information, the user certificate, and the memberships in communities are stored in the VOMRS.

GRRS The *Grid Resource Registry Service* (GRRS) [2] database contains information on all registered resources in D-Grid. In particular, it makes it possible to allocate a resource to the communities. The GRRS database contains all registered resources of the resource infrastructure. The resource information registered in GRRS includes: the name and short description of the resource, information about the installed middleware, the institution which operates the resource, the host and service certificates, a numeric resource identifier, the community-memberships, and the responsible contact addresses. The resource providers can register and manage their resources via a web interface.

GOCDB The *Grid Operations Centre DataBase* (GOCDB) [154] is the official repository for storing and presenting EGI topology and resources information. It consists mainly of participating *National Grid Initiative* (NGI)s, sites providing resources to the infrastructure, resources and services, maintenance plans for the resources, participating people, and their roles within EGI operations. The data are gathered and presented through a central interface [99], but provided and updated at regional level by participating NGIs.

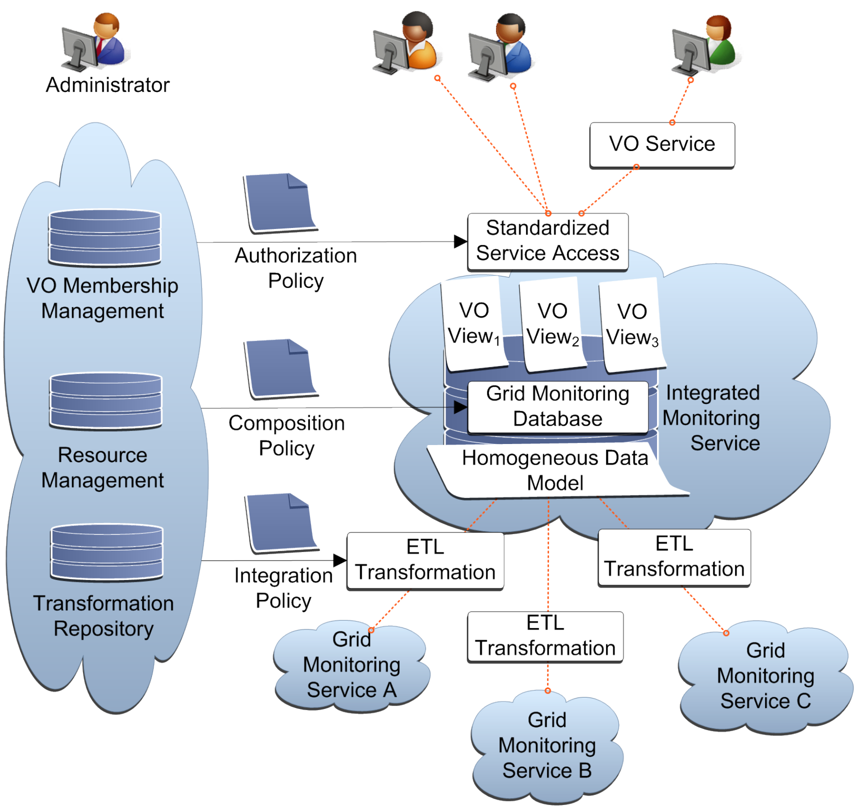


Figure 4.10.: An Example for Monitoring in Community-aware Context: the D-MON Integrated Monitoring System

In this work we focus on the community membership and virtual organization management systems utilized in the scenarios we described in Section 4.1. We however refer the interested reader to our other works on the field of distributed research infrastructures that describes the community-specific access control based on identities maintained within a federation and components required for single-sign-on [136, 137].

4.4.3.1. Community-aware Context in D-MON

We continue with showing how these services and components can be used for a community-aware monitoring system. For that we briefly revive the example of D-MON. The design we chose for the D-MON system is depicted on Figure 4.10. The components required for the integration of community management layers into monitoring architectures are shown on the left side. These components include: (a) the *Policy Decision Point*s and the information about community membership rules, (b) the resource registration systems and the information about resource composition of communities, and (c) the information about data transformation. The community-aware, integrated D-MON service uses these components as follows: the access for the various clients is granted only to members of the community. The necessary *authorization policy* is defined by the community membership rules, which are provided by the *VO Membership Management* system (PDP). The *composition policy* controls which resource's monitoring data is included in the community-related views. The D-MON system utilizes *integration policies* to define how the monitoring data is integrated into the GLUE v2.0 *generic data model* by the *ETL adaptors* and how it is stored in the target storage component of D-MON. The components at the bottom represent the various source systems and monitoring services which provide the monitoring data.

These policies direct the relation between resources, services, communities and monitoring data. This relation has to be converted into syntactically and semantically fitting data structures and it has to be integrated into the generic data model. All this lays the basis for community-aware monitoring of resources and services. The resources, services and components need to be identified in a unique way and such identifiers must be identical in the source systems, e.g. in the resource management system as well as in the source monitoring services. Otherwise, defining a deterministic mapping is not possible. During the implementation we found that the identifiers are not provided in a trustful way and especially the non-identical identifiers in the source systems created a particular challenge for the integration.

4.4.3.2. Standardized User Interface

User interfaces are necessary to display resource information and query monitoring data from the various monitoring services. Commonly, the monitoring services are already equipped with own interfaces or specific interfaces can only work with certain monitoring services. This often leads to confusion about the capabilities of the underlying systems. Our previous work [22] gives an overview on the main user interfaces utilized in our application domain. As the selection of an user interface determines the entire underlying monitoring system, a decoupling of the underlying monitoring services is favored. For D-MON, the integrated monitoring system of D-Grid, we therefore decided to provide an own, independent user interface and developed a web client and a command line client.

Today's services are increasingly relying on the web: web-based interfaces support the business processes, operations, and their integrations. The complexity of web interfaces varies widely. Basic command line clients, from the other side, are standardized access interfaces, that can simplify the addition of higher level interfaces. The web client is based on distributed components, so called *portlets*, which support the encapsulation of web applications. As the developed portlets support the *Java Specification Request* (JSR) 168 specification, they can easily be plugged into standardized web toolkits and portal frameworks, like Gridsphere [165] or the Vine Toolkit [194]. The command line client uses the standardized database access interface and it is compliant with the *Open Grid Services Architecture - Data Access and Integration* (OGSA-DAI) [10] specification.

4.4.4. Service Discovery and Integrated Service Registry

In the following we describe the discovery and integrated registry system as we implemented for the D-MON project[20]. We also give an overview on the ETL prototypes developed for discovering information services in MDS4, BDII, and CIS; as well as on the information schema for a registry database used for discovering.

The implementation and evaluation of our concept was presented in [148]. Here we just shortly summarize the main components, which are also depicted on Figure 4.11:

Service Discovery Database It is the storage component of the integrated discovery service. It keeps a list of all monitoring and information systems, which were discovered by the service. It can provide various views on the data, depending on various criterias (e.g. community membership).

Service Discovery Adaptors These adaptors are responsible for collecting and maintaining the information. They are implemented as ETL adaptors and XSLT transformers for each platforms (e.g. MDS4, BDII, and CIS).

Monitoring Management This component also maintains descriptions about site resources, which are not registered in the central information services and thus, cannot be discovered automatically. It also keeps information on resource downtimes. It is implemented as a Black- and White-List Manager.

[20]We implemented the discovery adaptors and the integrated information service registry for the D-MON project. The implementation became part of the production level infrastructure of the German Grid Initiative.

Homogeneous Data Model The schema describes the required general set of information. The data model also unifies the service descriptions, which are coming from the monitoring services from various platforms. Our current implementation makes use of some additional attributes, which are not part of the GLUE v2.0 schema.

Integrated Monitoring Service Integrated monitoring services, like the D-MON monitoring service, rely on the unified information provided by the integrated discovery service. They use one or more of the provided service views as interface to query information from the integrated discovery service.

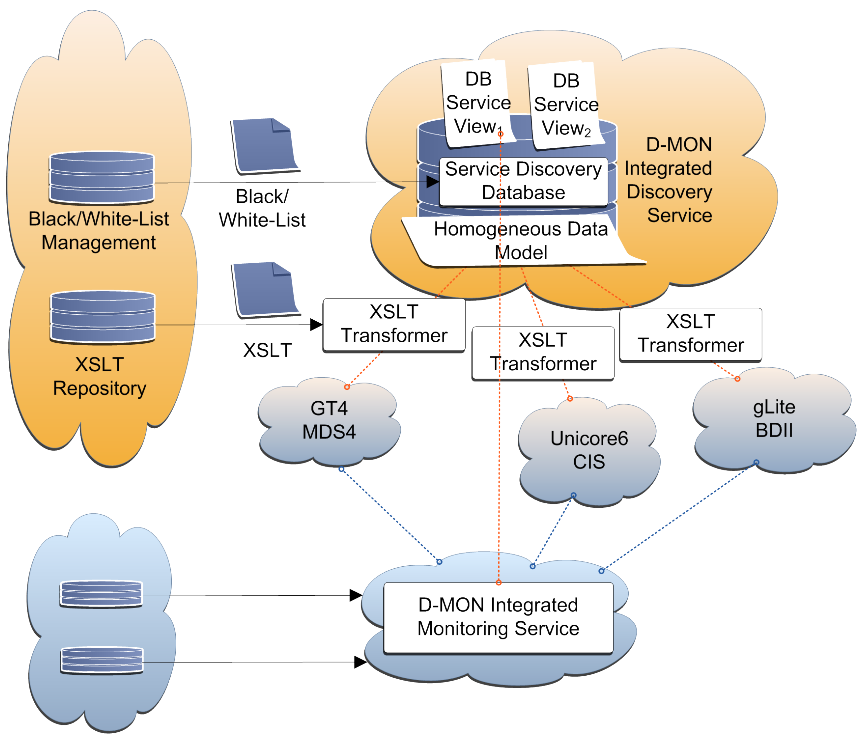


Figure 4.11.: Integrated Discovery Service for an Integrated Monitoring Service

The aim of our implementation was to utilize the *Extract, Transform, Load* (ETL) concept for discovering information services. That way, our ETL adaptors for information service interfaces can be reused. Our ETL adaptors for discovery are implemented as follows. The *Extractor* component aims to reuse our existing monitoring adaptors. Although we do not require all monitoring data from the source systems, but it eases the implementation and the maintenance of the overall system. We implemented all *Transformation* components newly, as we need to map the information into a new schema (e.g the discovery schema), which schema contains new entities. We kept the *Load* component very similar to that we described for integrated monitoring. In our implemented prototypes we either extended the storage schema, or we used a separated registry database.

Our implementation has the advantages, that (1) it can easily be extended by additional research infrastructures, (2) the integration of other technologies (middleware systems) is possible by developing a new discovery adapter, and (3) it can be be replaced by standardized central resource registration systems at any time.

The experiences we gathered during the development and testing of the system show that the bad data quality provided by the various source systems is a hindrance for the automatic detection of the information services. We exemplary refer to the site names and downtime information. The site names identify the resource providers and are required for assigning the resources to a site. Therefore, they should be carefully checked by the providers. However, we found that the different resources of the same provider disseminate different site identifiers. Another example is the downtime information. This is often not available in a uniform format, but as a free text. Both issues make difficult to implement automated ETL processes for service discovery.

We therefore derived policy implications, which have an impact on both the information model, as well as the process of the resource registration[21].

4.5. Related Work

In this Section we outline related work by introducing architectures, standards, interfaces utilized to exchange monitoring data as well as to approach both the integration problem and community-aware data provisioning.

Interoperable monitoring of distributed research infrastructures has been a topic since many years [235, 1], and standardization efforts have been made by several interoperability and standard initiatives. For grid systems, mainly the *Global Grid Forum* (GGF) and later the *Open Grid Forum* (OGF) drove the evolution of standards and promoted their adoption on the fields of grid infrastructures, as well as associated storage, networking and workflows. To address the problem of monitoring the resources and services of a grid infrastructure, multitude of concepts have been worked out with respect to monitoring architecture [218] and data schema [8]. For the cloud paradigm, the main initiatives being involved in standardization have been the *Institute of Electrical and Electronics Engineers* (IEEE) [117], the *Distributed Management Task Force, formerly "Desktop Management Task Force"* (DMTF) [68], the *Open Commons Consortium, formerly the Open Cloud Consortium* (OCC) [167], and the *Open Grid Forum* (OGF) [117]. While their works have a strong focus on interoperability, their activities are still diverse. IEEE's *Adaptive Management of Cloud Computing Environments Working Group* provided a description of an adaptive management environment, as well as focused on its components and the information needed and the communications needed to support and maintain highly dynamic environments in [117]. The DMTF addresses cloud management with multiple standards and working groups. The *Cloud Management Initiative* brings this work together for an integrated approach [68]. The original concept of the *Open Cloud Computing Interface* (OCCI) *Working Group* of the OGF was to design a flexible RESTful [182] *Application Programming Interface* for common tasks including management, monitoring, deployment and autonomic scaling in cloud environments. Its focus has been broadened and the standardization activities of the OCCI Working Group are now suitable to serve many other models and computing paradigms [171]. Consortiums, which manage infrastructure and operate services, also improve the state of the art of cloud standards. The *Open Commons Consortium, formerly the Open Cloud Consortium*, for example, is organized into different working groups, which enrich the standardization activities by developing reference implementations, designing components as well as by supporting data commons [167].

A multitude of concepts have been worked out to address the problem of monitoring the resources and services of a distributed research infrastructure. As a basic blueprint, we refer to the *Grid Monitoring Architecture* (GMA) [218], defined by the OGF. The GMA describes major components for a monitoring system as well as their essential interactions. It separates *data discovery* from *data transfer* by presenting a producer/consumer-based architecture for monitoring. This architecture supports resource discovery and information delivery between information producers and consumers by utilizing *directory services*. Each *information producer* first contacts the directory services to register themselves in the directory service. Several additional information, like the type of information to publish to the research infrastructure is also registered. When an *information consumer* wants to discover resource information of interest, it contacts the discovery services and asks for locating the respective producers. The next step might also be a direct interaction between the consumer and the discovered producer: the consumer does not necessarily

[21]We suggested extensions both in the information model, as well as in the policy for the resource registration. The German Grid Initiative adopted our extensions.

contact the directory services but the producer to request resource information or monitoring data. The producer sends the data back to the consumer directly. The GMA is designed to be scalable and to avoid single points of failure. The GMA architecture provided the basis to create sophisticated monitoring architectures, such as the Aggregator Framework of the MDS4 [199] or the *Relational Grid Monitoring Architecture* (R-GMA) [49]. To query the MDS4 the *Web Services Resource Framework* (WSRF) [81] can be used. To access an R-GMA based service, the standard SQL queries can be utilized, then R-GMA one distributed relational database. While these architectures represent different concepts, which have their advantages and disadvantages, their implementations are important parts of different middleware implementations (e.g Globus Toolkit and gLite).

The definition and adoption of common open standards and architectures is a usual approach to achieve interoperability [152]. However, this approach strongly relies on standardization and implementation processes, and roll-outs across different communities can be complicated, costly, and politically charged. A coupling of architectures can therefore be a good alternative in certain cases. Whenever it is preferable to operate different implementations simultaneously, e.g. different middlewares, non-intrusive components such as bridges are needed in order to enable integration. This is also the case in situations where there is no agreement on a common standard or no knowledge of such standard. The issue of system integration has already been addressed by researchers in the past. One example are Hegering et.al. [108], who describe three types of architectural bridges, namely multi-architectural platforms, management gateways, and multi-architectural agents. They further point out that bridging can refer to the integration of communication (e.g. interfaces), information (e.g. data description schemas), organization (e.g. roles), and functional models (e.g. queries). In the context of our research, we concentrate on management gateways that can serve as a bridge between separate monitoring services and information models, as well as enable the retrieval of monitoring data according to roles and organizations.

In the area of information models we refer to Section 3.5 and Section 3.6, where we surveyed and examined the various information schemas in great details. We briefly recall the example of the *Grid Laboratory Uniform Environment* (GLUE) schema, which originally started as a joint effort by the European DataGrid [71]) and DataTAG [57] projects as well as the international Virtual Data Grid Laboratory [122] project. After several years of standardization work (cf. GLUE Working Group of the OGF) today it became adopted by many major research infrastructure projects such as EGI [72] and *Open Science Grid* (OSG) [172]. We also recall that the GLUE v2.0 is the only model for grid resources that supports Virtual Organizations (cf. UserDomains in the schema). Since the manageability of research infrastructures becomes more important, the information schemas of enterprise IT environments, such as the *Common Information Model* (CIM) from DMTF can also be of growing importance. Research results (cf. [65], [9], [64], and [157]) show that it is possible to adopt these enterprise environment models for distributed research infrastructures. However, their different scopes [8] as well as their high complexity [9] are still hindrance to utilize them in our application domain. Other related information models are for example the *Usage Record* (UR) [149] from OGF and the *D-Grid Resource Description Language* (D-GRDL) [232]. The aim of UR and its interface (cf. *Resource Usage Service* (RUS) [45]) is the accounting and thus, it cannot provide general resource information, which are essential for monitoring of distributed research infrastructures. The D-GRDL is an adaptable, but very abstract framework for describing resource data.

A standardized interface for the exchange of state data has been proposed for both grids as well as clouds. For grid components the OGF worked out the *Web Services Resource Framework* (WSRF) [81]. While the WSRF has been widely utilized, different versions are in use. Also several popular monitoring frameworks just ignore the WSRF standard by choosing components that fit their requirements best. Other methods to exchange monitoring data are using *eXtensible Markup Language* (XML) or *Representational State Transfer* (REST). Since cloud systems have a strong focus on flexible REST [182] interfaces, a set of specifications for REST-based communication models have delivered through the various standardization initiatives. We name exemplary the OCCI, which is widely deployed in our application domain. The OCCI is a protocol and an *Application Programming Interface* (API), which was designed to enable interoperable tools for common cloud infrastructure management tasks including deployment, autonomic scaling and monitoring. It has since evolved into an flexible API and recently, it became suitable to serve many other models in addition to IaaS.

To provide community-aware services is one of the basic requirements for the distributed research infrastructures. The necessity of community-aware infrastructure services has been discussed in [17], as well as in our previous works [18, 21]. In the work at hand, in order to provide monitoring data and exchange resource descriptions, we detail the design of an integrated, community-aware monitoring system.

At last, we stress how we approach the integration problem. With the era of large-scale research infrastructures, the aggregation of monitoring data from distributed information sources has been a topic since many years [1, 235], and standalone monitoring and information services have been established by several communities. We refer exemplary to the community-specific solutions designed by the communities of astrophysics (cf. Stellaris [112]) and life sciences (cf. ResourceUpdater [143] and D-GRDL [232]). While our recent work has a similar approach, we had a stronger focus on interoperability. Instead of providing a solution for a dedicated community or supporting national initiatives only (cf. our previous works on Instant-Grid [135] and D-GRDL [127, 128]), we utilize several upcoming standards of our application domain, like the information model, the monitoring architecture as well as the communication models (cf. [70, 81, 171, 218]).

4.6. Analysis of results

The overall goal of this chapter is to describe the practical relevance of the problem and develop a proof of concept for existing distributed research infrastructures. This chapter also summarizes the results from our presentations [127, 128, 130, 131, 148], as well as from [18, 21].

First, we presented three exemplary infrastructure scenarios in Section 4.1: (a) the distributed research infrastructure set up by the German e-Science initiative, (b) the distributed infrastructure of the *European Grid Infrastructure* (EGI)[22] , and (c) the demonstration infrastructure *Instant-Grid* (IG).

We continued by investigating in their common challenges and requirements for an interoperable monitoring and information system (Section 4.2). We found that several requirements are related to general aspects and are independent from both the computing paradigms as well as concrete usage scenarios. Such *generic requirements* are, for example: (a) scalability and extensibility, (b) sustainability, or (c) security. We also found that other requirements are more specific and related to the usage scenarios. These *specific requirements* are discussed in Section 4.2.2 and include among others: (a) *Information scope* (supporting the purposes of resource discovery, task scheduling, resource monitoring, job monitoring, accounting of resource usage, high-level overview of the infrastructure, and low-level diagnostic information for operation of the infrastructure), (b) *Data integration* (ensuring the physical integration of heterogeneous data from multiple sources), (c) *Data provenance* (augmenting the data sets with information that helps tracing back the derivation history of the resource descriptions and monitoring data), (d) *Community- and role-based access* (providing proper access control to the integrated data by considering community memberships, roles and affiliations).

After discussing the main requirements, we shifted our focus in Section 4.4 to discussing how a proper system design can fulfill the requirements. We examined the following three approaches

[22]This distributed infrastructure was established by the *Enabling Grids for E-sciencE* (EGEE), continued by the *European Grid Infrastructure* (EGI) and the *Integrated Sustainable Pan-European Infrastructure for Researchers in Europe* (EGI-InSPIRE) projects and being currently developed further by the *Engaging the Research Community towards an Open Science Commons* (EGI-Engage) project.

based on several criteria: (a) building multiple bidirectional gateways between each pair of monitoring systems (*Multiple Bidirectional Gateways*), (b) defining one of the monitoring systems as the privileged system (*Privileged System*), and (c) an autonomous monitoring system which is independent from the middlewares (*Autonomous, Middleware-independent System*). We considered the option (c) as our preferable basic architecture because of the disadvantages of the other two approaches. We refer to our previous work [18], as well as to [147], which discuss several other approaches that have been considered to design a monitoring system.

This *autonomous, middleware-independent monitoring system* motivated our further work. We used our results from Section 3.4 to describe the main characteristics of heterogeneous, distributed research infrastructures. We showed in Section 4.3.2 that a mapping between our *generic* entities and the *GLUE v2.0* entities can be defined as follows: (a) modeling *communities*: UserDomains, (b) modeling *authorization policies*: AccessPolicy and MappingPolicy, (c) modeling *allocation of resources and services* to communities: Endpoint, (d) modeling *resource and service scenarios*: ComputingService and ComputingManager, and (e) modeling resource *providers*: AdminDomains. This enabled us to design transformations (Section 4.3.2), that use the GLUE v2.0 schema for the mediation of resource information. Thus, this design also enables to cross-provide monitoring data, for example from a monitoring system using schema A into the interoperable monitoring service and from there into a monitoring system utilizing schema B or vice versa.

The access control in distributed research infrastructures is often based on a policy enforcement scenario. In this work, we applied the policy enforcement scenario for integrated monitoring as follows: (a) the monitoring and information system should act as *Policy Enforcement Point*, (b) the *Policy Decision Point* (PDP) decides if access to protected resources is granted or not, (c) a *Policy Administration Point*, which supports the management of policies, as well as a *Policy Information Point*, which provides additional information on attributes, are both considered as constituents of the PDP. We also brought the following conclusions: (a) the *Policy Decision Point* for integrated monitoring should be the same component that is also used by other services of the distributed research infrastructures, and (b) the community- or VO-management systems are components, which evaluate and issue such authorization decisions. Readers interested in a deeper insight into community-specific access control are referred to our other works in the field [136, 137, 102].

The architecture of an interoperable, integrated monitoring and information system should be scalable and efficient. This implies low resource consumption and short response time for accessing data. The selected system architecture is an autonomous, middleware-independent service, which utilizes a storage component with gateways connected to the underlying native monitoring services. We presented and characterized four options of how the storage component can be set up, namely: (a) *central setup*, (b) *clustered setup*, (c) *federated setup with horizontal partitioning*, and (d) *federated setup with vertical partitioning*.

Distributed storage components "deliver" data from where they are stored to where queries are processed. We decided to rely on the extensive research work, which has already been made on the possible data delivery alternatives and their rich design space. We used the characterization discussed in [168] for: (1) *data delivery* alternatives, (2) typical *frequency measurements*, as well as (3) *data communication* models. For the characterization of the *design strategies for distributed storage* components we drew on [111].

We continued with providing a proof of concept and described the details of how our concept can be implemented within the distributed research infrastructure set up by the German e-Science Initiative[23] (Section 4.4). The expertise of the D-Grid [164] provided an excellent basis to (1) study distributed research infrastructures, (2) learn about computing paradigms utilized in our application domain, and (3) evaluate our concept for an interoperable monitoring architecture.

We defined an automated ETL-process for the information systems Globus Toolkit [79], gLite [97], and UNICORE [221] to gather, transform, and integrate their complex monitoring data

[23]We implemented the initial architecture together with colleagues from the D-MON [55] project and the German Grid Initiative (D-Grid) [164] adopted our results as its grid monitoring architecture.

efficiently. That way, we demonstrated how the *Extract, Transform, Load* (ETL) data warehousing technique can be adapted to map existing information models onto the GLUE v2.0 schema. The middleware specific extractors (E), which use different protocols and data models, collect the monitoring information asynchronously. XSL templates parse, check, and transform (T) the extracted resource information in a flexible way into a common data structure, e.g. the GLUE v2.0 schema. The transformed monitoring data is then stored (L) in the storage component of the target repository.

Finally, we outlined related work in Section 4.5 by introducing architectures, standards, interfaces utilized to exchange monitoring data and to approach the integration problem and community-aware data provisioning.

4.6.1. Experiences

Implementing a proof of concept and deploy our solution in a production level distributed research infrastructure brought us to the following conclusions:

Community-aware regulation of access to information We designed and implemented a community-aware monitoring system, which regulates the access to monitoring information using the identity information. We found that the scenarios of our application domain utilize federated identity management solutions with a security model [225], which is very complex [61]. To lighten the security management for non-experts, various workarounds have been introduced, we refer exemplary to [88, 105, 166]. In the future, we expect further harmonization of the recently used federated identity management systems. We point out that such a development might be a new challenge: the translation between identity management systems also means to achieve the *same trust level* in the various systems of the distributed research infrastructures. We also stress that the most challenging issues we identified are: (1) the sovereignty of policy decision points, (2) the identification of community memberships' expiration, (3) the self-registration possibilities for (citizen) scientists, as well as (4) the persistence of user identities and their uniqueness across the infrastructures.

Data provenance One of our achievements is that the integrated monitoring overcomes the limitations of the various middlewares, and it can collect monitoring data and resource information from heterogeneous middlewares. When the heterogeneous middlewares delivered diverging information about the resources, our integration process creates diverging data sets belonging to the same resource. To keep track of data provenance, e.g. the origin of the data, is therefore vital. We extended the database schema of our implementation with provenance information. Each transformation step and each data item is enriched with provenance information. For our scenarios, we defined (a) the source information or monitoring system, (b) the exact time and date of retrieval, and (c) the IP-address of the source system as necessary provenance information.

Necessity of naming standards and global unique identifiers Integrated, unified monitoring discloses the structural shortcomings of the monitored infrastructures. This may lead to duplicate entries that describe the same underlying resource. Globally unique identifiers across the infrastructures are needed to guarantee the consistency in tagging. The assignment of such identifiers, e.g. names and numbers, is an organizational issue. Similar issues have already been solved, for example by establishing the *Internet Assigned Numbers and Names Authority* (IANA) for the IP-addresses to autonomous systems in 1984, and by establishing the *Digital Object Numbering Authority* (DONA) for the *Digital Object Architecture* (DOA) [120] in 2014. Identifying resources in a unique, persistent way requires common naming standards or policies. These can be assigned at well-known policy decision points. For a deeper insight we refer the interested reader to our work in that field [134].

Lossless transformations and loss of accuracy By examining the information- and monitoring systems for our proof of concept we found that there are differences in semantics. Thus, the transformations may cause loss of information. Our implementation work focused on MDS4, BDII, and CIS as sources. We showed that it is possible to gather and transform the important data for the relevant values without a loss of accuracy.

The first contribution in this chapter is to develop a proof of concept for actual services of distributed research infrastructures. We presented three infrastructure scenarios and investigated in their common requirements for an interoperable monitoring and information system. We connected the requirements with the results of our theoretical analysis regarding the information demand in distributed research infrastructures, as well as the theoretical approach for a schema mediation process outlined in the previous chapter. We also designed an automated resource information exchange process and a respective generic monitoring architecture supporting it, which is our next contribution.

5. A Framework for the Simulation of Heterogeneous Information Services

In this Chapter, we describe a solution for automated system deployment, which allows to set up a self-configured and independent multi site and multi user distributed research infrastructure. We present the technical concepts including the automatic configuration, ready-to-use features, and applications. We applied this environment as our simulation framework, but it is also suitable for demonstrating, developing, and testing purposes of distributed computing environments.

This chapter summarizes our work on the topic of automated middleware deployment [28, 29, 30], as well as our results on the topic of self-configured demonstration and testing environments [135, 188].

We give an overview of the framework we designed and developed to set up a heterogeneous distributed infrastructure for simulation and testing purposes. We applied this environment as our *simulation framework* for distributed computing environments, which enabled us to automatically establish a self-configured and independent multi site and multi user distributed research infrastructure. But this framework is also suitable for *demonstrating, developing, and testing purposes* of distributed computing environments; because it allows the installation of research infrastructure middlewares on machines located in a local network or in virtualized environments without any previous knowledge of distributed research infrastructure technologies. We therefore also present an use case where we successfully applied this framework in teaching.

This Chapter is structured as follows: We briefly discuss our motivation in Section 5.1. In Section 5.2, we describe the system design and the technical concepts including the automatic configuration, ready-to-use features, and demo applications. Afterwards in Section 5.3, new features of the Instant-Grid UNICORE edition are presented. In Section 5.4 and 5.5, we describe the application of Instant-Grid in a practical course at the University of Göttingen, as well as the connectivity to existing productional distributed research infrastructures. In Section 5.6, we consider related work. Finally, we conclude with a summary and an outlook in Section 5.7.

This chapter is partly adapted from our previous works on the topic of automated middleware deployment [28, 29, 30], as well as on the topic of self-configured demonstration and testing environments [135, 188].

5.1. A Simulation and Demonstration Environment

Distributed computing environments offer major advantages to its users, solving extensive computational and storage problems, for example in the fields of financial modeling, earthquake simulation, and protein folding. We found that the scenarios of our application domain - described in Section 4.1 in great detail - provide service oriented environments and enables to share computational and storage resources that are spatially dispersed and belong to various organizations. The examined implementations utilized the grid paradigm. A grid system can generally be defined as a networked system "that coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service" [83].

On the other hand, establishing such a distributed computing environment requires a comprehensive understanding of grid technologies and entails a number of challenges. The usefulness and acceptance of distributed research infrastructures can be improved by providing solutions such as the Instant-Grid [119], which establishes a trustworthy multi user grid environment. The Instant-Grid sets up a self-configured, independent standalone grid. This standalone grid is based on the *Globus Toolkit 4* (GT4) [79] or the *UNiform Interface to COmputing REsources v6* (UNICORE6) [221] middlewares and enables users without expertise on grid technologies to set up a grid system.

Distributed computing systems based on the Instant-Grid can for example be applied for demonstration, development, and testing purposes. They are furthermore suitable for education purposes, since the grid applications can be utilized through a web browser and are configured inside the grid security environment. When used for education purposes, the Instant-Grid allows students to configure and administrate their own distributed computing system, including the development and testing of grid applications and their distribution. The system furthermore supplies various tools, a grid test environment that is independent from production, pre-installed grid applications, and a range of fully configured services. Being a closed distributed research infrastructure, it facilitates testing of grid components and applications. These applications automatically make use of all the available resources of the distributed infrastructure.

The Instant-Grid Project was started as a research project and was funded by the German Federal Ministry of Education and Research. Initially, Instant-Grid is based on Globus Toolkit 4 [79]. In this work we call this version the Instant-Grid *Globus Toolkit 4 edition*. We published our results in [28, 29, 30].

After the project was officially closed, further developments were driven by us. We used and enhanced the results of the project, and we started the development of an Instant-Grid based on the UNICORE 6 middleware. We call the new version the Instant-Grid *UNICORE edition*. The Instant-Grid UNICORE edition applies to the specific requirements of UNICORE 6 and provides an ideal interoperability test environment. We improved the basic functionality of Instant-Grid and added several new features. Improved functionalities that work at runtime include for example the service configuration, the automated discovery, and the user management. Furthermore, the new UNICORE edition also widened the applicability and allowed to utilize it as a teaching environment.[24] We summarized our improvements and results in [135, 188].

5.2. Technical Concept

This section describes the system design and the technical concepts including the automatic configuration, ready-to-use features, and demo applications of Instant-Grid. The boot mechanism of the Instant-Grid combines a *Preboot Execution Environment* (PXE) network boot mechanism with the Live CD-concept [101, 185], and enables its usage in cloud environments. The system ensures a fully automated service and network-setup during the boot process. A dedicated frontend, which can be booted in cloud, as well as from *Universal Serial Bus* (USB) or *Compact Disc* (CD), ensures an automatic setup of a self-configured distributed research infrastructure. The middleware depends on the edition of the Instant-Grid, e.g. based on GT4 or UNICORE6.

The dedicated frontend, i.e. the Instant-Grid server, is in charge of the network-based startup of other participating nodes. The frontend is informed at run-time about all relevant changes of the distributed infrastructure, for example the removal or addition of resources. In addition, it updates the resource databases and resource registries utilized in Instant-Grid. Altogether, this mechanism enables the utilization of preinstalled applications and features in the Instant-Grid environment.

[24]Our results were applied as a teaching environment for practical courses at the University of Göttingen, Germany and at the Monash University, Australia.

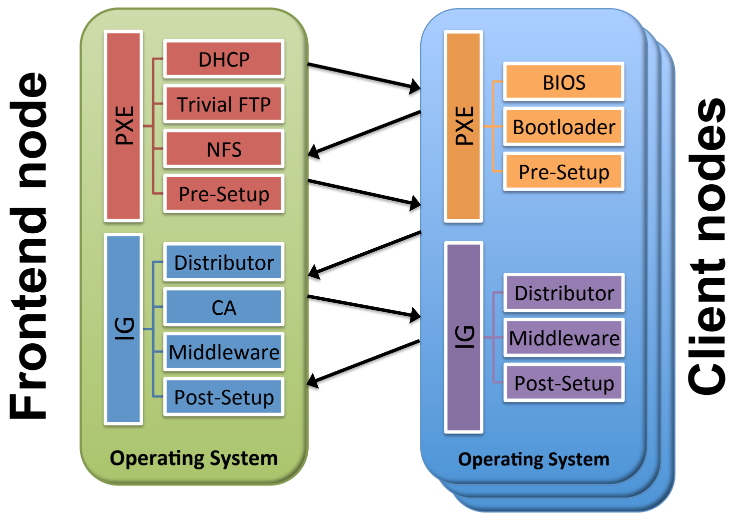


Figure 5.1.: Software Components to Establish the Automated Configuration at Startup

5.2.1. Automated Self-Configuration

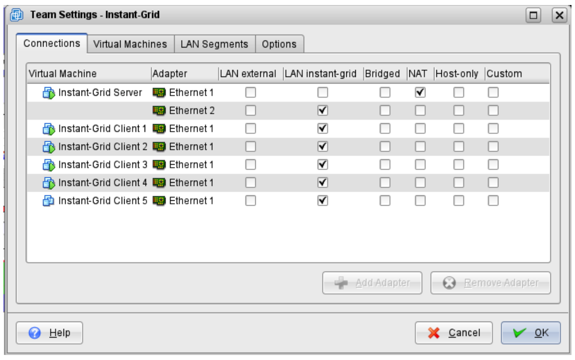
We gave a detailed technical description of our approach to deploy pre-configured middleware systems in a fully automated process in our previous works [28, 29, 30, 135]. Here we just highlight the main concept of the automated configuration at (a) the startup process and (b) during runtime:

Automated Configuration at Startup Technically, services to turn a computer lab into a grid system are based on the well established PXE mechanism, which is also used by several cloud environments as well as in Live-CD projects aiming at cluster setups. Figure 5.1 gives a schematic overview of the boot process. After booting the Instant-Grid image, the machine acts as *Dynamic Host Configuration Protocol* (DHCP), *Trivial File Transfer Protocol* (TFTP), and *Network File System* (NFS) server. The DHCP server is already configured to answer PXE boot requests from client nodes. Then, the kernel and needed boot files are transferred via TFTP. Finally, after booting the kernel, the nodes mount their root filesystem via NFS from the frontend. This startup process and all services are configured automatically and do not require any human intervention. The configuration steps that are done to deploy the middleware are described in Section 5.3.1 and Section 5.3.2.

Our implementation of this design aims to avoid most human intervention. This includes an automatic search for a suitable network interface on the frontend node as well as on the clients is performed. The frontend node can directly access the Internet through an optional external interface. On the client nodes all interfaces, except the one for the internal network, are disabled. In case an external interface is set up on the frontend node, the nodes can access the Internet via *Network Address Translation* (NAT). Figure 5.2 shows a screenshot of the network interface configuration that we utilized to set up a distributed research infrastructure for demonstration and teaching purposes [135].

To the best of our knowledge, all other implementations using PXE for building a cluster need further manual configuration steps before starting all required services.

Automated Configuration at Runtime An important idea about grid computing environments is that the resources are assumed to be dynamic. This means, client nodes can be started or stopped at any time the grid system exists. In case of Instant-Grid, the grid environment exists as long as the frontend node runs. To accommodate this idea, a mechanism has been implemented to discover new and lost nodes. This information is then distributed to all participating nodes. In Instant-Grid we developed such a mechanism for the frontend by probing the network for available nodes, updating the configuration, and provide it to all



Virtual Machine	Adapter	LAN external	LAN instant-grid	Bridged	NAT	Host-only	Custom
Instant-Grid Server	Ethernet 1	☐	☐	☐	✓	☐	☐
	Ethernet 2	☐	✓	☐	☐	☐	☐
Instant-Grid Client 1	Ethernet 1	☐	✓	☐	☐	☐	☐
Instant-Grid Client 2	Ethernet 1	☐	✓	☐	☐	☐	☐
Instant-Grid Client 3	Ethernet 1	☐	✓	☐	☐	☐	☐
Instant-Grid Client 4	Ethernet 1	☐	✓	☐	☐	☐	☐
Instant-Grid Client 5	Ethernet 1	☐	✓	☐	☐	☐	☐

Figure 5.2.: Network Interface Configuration for Direct as well as Translated Access

nodes via a specific exported NFS directory. On the client side, this directory is mounted and periodically checked in a preconfigured interval. In case of changes, the local configuration is updated.

The dynamic character of a grid system - as opposed to the static character of a cluster — necessitates the availability of an information service, which provides current data on status, resources, and services of the grid. In Instant-Grid, the hardware resources are monitored using the Ganglia Monitoring System [92], while the web service based CIS [156] of the UNICORE 6 or the MDS4 [200] of the Globus Toolkit 4 handle information about available services. Ganglia, CIS or MDS4 are started and configured without any user involvement at boot time. Additionally, Instant-Grid provides test routines to ensure the correct behavior of the registered resources and services. The availability and load information obtained by Ganglia can also be seen in a web frontend.

5.2.2. Ready-to-Use Features and Applications

We chose several examples to demonstrate the benefits of distributed research infrastructures and support the development of applications and services for distributed systems. After starting IG, a set of ready-to-use grid services and applications are available without any user interaction. These were carefully chosen and are aimed to be understandable for non-experts. All applications and services are deployed fully automated and preconfigured for demonstrations and work inside the local IG environment. Neither Internet connection nor global grid connectivity is required. The ready-to-use applications were presented in our previous works [28, 29, 30, 135, 188]. Here we just highlight the most important ones:

Distributed Rendering The *Persistence of Vision Raytracer* (POV-Ray) application creates photo-realistic 3D images using a rendering technique called raytracing. This is a CPU intensive task, but can easily be parallelized. POV-Ray is one of the applications that was chosen to demonstrate the dynamic resource management of a grid system. Each participating IG node renders a part of the original 3D image or one frame of an animation, which can be done simultaneously. Here, IG is used as a renderfarm for rendering complex scenes.

Portal The portal is the main user interface of the IG. It is based on an open-source portal framework (GridSphere[165]), which is compliant to the JSR-168 standard for portlet type web applications [66]. Developers can deploy third party portlets into the GridSphere portlet container. There are portlets of example applications. The workflow system can also be accessed by a portlet and the monitoring portlet displays monitoring information and resource information. In order to facilitate administrative tasks, we have implemented a portlet for account management that allows to create new accounts with one click and without any background knowledge of user management and the security infrastructure. The standard GridPortlets can be used to submit jobs, transfer data, and manage credentials.

Distributed Search Whenever a new storage resource is introduced into any of the nodes of the environment, the texts on the resource can be indexed. The index data is then accessible for other user that collaborate in the distributed research environment. In Instant-Grid, the GridSearch framework for building an index of text corpora is integrated.

Workflow Management System On top of the middlewares, IG employs a flexible workflow orchestration infrastructure called *Generic Workflow Execution Service* (GWES) [114]. The GWES fully automates the distributed execution of complex compound applications on distributed computing resources.

Collaborative Tools Certain applications and tools have been chosen and integrated in the environment to demonstrate the possibilities of collaborative works in distributed research environments. These include a realtime collaborative editor and a chat client with a built-in whiteboard:

- A realtime collaborative editor (Gobby) that communicates through encrypted channels and supports multiple documents in one session. Documents can be synchronized on request. For recognizability in this framework each user has its own changeable color and a name based on the hostname of the client. This way each contribution can be identified by others.
- A chat client with a built-in whiteboard based on a jabber server (Coccinella) establishes a multiuser chat environment together with an inbox for instant messaging. All Instant-Grid clients are configured to form a group inside the jabber server and are automatically logged in with their hostname. The user can easily draw pictures and instantly communicate them to others.

5.3. Automated Middleware Deployment

This section presents a range of new features, which are responsible for the fully automated deployment of the middlewares that provide the distributed computing capabilities. These features include:

- the dynamic configuration of the UNICORE middleware,
- the dynamic configuration of the Globus Toolkit middleware,
- the automated setup of the security environment, and
- the additional customization possibilities.

5.3.1. Dynamic Deployment of UNICORE 6

In addition to the configuration of basic services during the startup process as described in Section 5.2.1, certain dynamic configurations of UNICORE 6 components need to be applied in the startup on every nodes. They allow users to easily deploy the UNICORE environment without preexisting knowledge of UNICORE or distributed computing technologies. Figure 5.3 illustrates the UNICORE 6 architecture of the Instant-Grid.

As can be seen in Figure 5.3 the Instant-Grid server's task is to manage the nodes. This task includes the allocation of hostnames, signing and issuing host certificates, as well as ensuring an accurate setup of the different UNICORE components. We described our approach in [135]. In the following, we highlight the most important components.

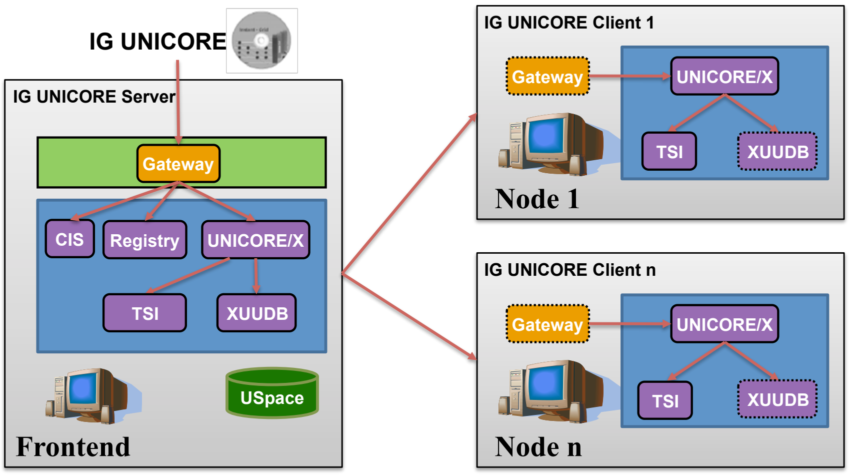


Figure 5.3.: Dynamically Configured UNICORE 6 Architecture of IG

Access management In order to provide access to services and to perform authentication decisions, a UNICORE Gateway is configured within the Instant-Grid environment. This Gateway is not started on the clients for performance reasons. However, the service Gateway is distributed to all clients, which enables services to contact the gateway. All UNICORE services that run on the client nodes are served by the Gateway that runs on the frontend.

Job and data management The central component for job and data management inside UNICORE 6 is UNICORE/X which is a container for the UNICORE 6 atomic services and also includes a local Registry service. UNICORE/X is started on all Instant-Grid nodes. However, the auto registration with an external registry is enabled for the services on the clients. They are able to contact the shared Registry on the frontend in order to publish and query information. Other UNICORE/X parameters are also configured automatically during the startup. For example, the hostname is used as the name of the UNICORE/X site and by the AdminDomain and ComputingService entities of the site description. The site description is used by the CIP which collects resource information from the UNICORE/X and constructs a GLUE v2.0 compliant XML document for the information service.

Information service The grid information service CIS is also dynamically configured and started on the Instant-Grid frontend. It fetches all connected Instant-Grid sites from the global Registry and periodically gathers static and dynamic resource information from all nodes. It aggregates and publishes the resource information in GLUE v2.0 format and provides XPath and XQuery standard query mechanisms. The CIS in Instant-Grid stores the resource information in a native XML database. The web interface of CIS is also set up to enable searching and browsing entities published via GLUE v2.0 documents.

Registry A global, shared Registry is started on the Instant-Grid frontend. The service address of the Registry is communicated to the other services in order to register all services. The UNICORE 6 services that run on the Instant-Grid nodes are automatically configured and registered in the UNICORE registry without any user intervention.

User database In addition to the Gateway which is responsible for the authentication, a *UNICORE User Database* (XUUDB) service which performs authorization in the Instant-Grid environment is executed. The grid certificates of the local Instant-Grid users are registered in the XUUDB database and mapped to a local Linux account automatically. Because of performance issues, the XUUDB service is not started on the clients. The service address of the XUUDB running on the frontend are dynamically configured on all clients during the boot process and the client nodes are served by that XUUDB service.

Clients Two UNICORE clients are installed in Instant-Grid: (a) a command-line client and (b) a graphical client. Both can be used to access the services after the grid environment is started. The UCC is a command-line tool which can be utilized in a shell or scripting environment. Additionally, the ucc extensions for CIS commands are preconfigured and the CIS libraries are deployed in the Instant-Grid UNICORE edition.

The *UNICORE Rich Client* (URC) is an Eclipse-based graphical client, which provides a graphical view of the local Instant-Grid environment. All registered UNICORE resources in a running Instant-Grid environment can be browsed. The resources can be used to execute grid applications utilizing these resources. Resource requirements, e.g., required main memory or number of processors can also be specified for jobs.

Other components The job directory of UNICORE, the USpace, exists in ramdisk only. It can be configured on a local harddisk on the frontend. The USpace is then shared via NFS and accessible to the clients.

The *Target System Interface* (TSI), which is an interface to batch systems and allows job submission into local resource management systems, is under development. However, the embedded TSI starts jobs as a simple unix fork job.

The services with dotted frame in Figure 5.3 do not run because of performance issues, but can be enabled for demonstration purposes. For example, the XUUDB service should be enabled on each node, to establish a multi-user and a multi-site IG environment.

5.3.2. Dynamic Deployment of Globus Toolkit

Here we give an overview of the automated deployment of the components of the Globus Toolkit middleware. For a more detailed overview we refer to our previous works [28, 29, 30, 135].

Job Management Several tools are available for job distribution in Instant-Grid. Two of them, GRAM (Grid Resource Allocation and Management) and WS GRAM (Web Services GRAM), are part of the Globus Toolkit. Both can start jobs at specified Instant-Grid clients, and manage the necessary authentication steps. Together with the data management services provided by GridFTP or RFT they can also execute data transfers before or after the job. Necessary information on the job can be provided on the command line and in a job specification file. In the case of GRAM this file uses a custom language, the *Resource Specification Language* (RSL). For WS GRAM the RSL has been translated into an XML schema.

The third job management tool in Instant-Grid is an *Message Passing Interface* (MPI) environment built on top of the Globus infrastructure. It allows for starting parallel tasks on an arbitrary number of clients. This can be done both for non-MPI programs without interprocess communication, and for programs with inter-process communication, if they use MPI and are linked to the provided MPI libraries.

Data management There are two data management services provided by the Globus Toolkit. We depicted these components on Figure 5.4. One is GridFTP, which is a remote copy mechanism, based on FTP. A GridFTP server is automatically started during boot time on all nodes.

The other data management service is the Reliable File Transfer (RFT) service. It is a web service to monitor and steer gridwide file transfers. RFT requires GridFTP for the actual file transfers and an SQL database for storing information about them.

Because of the automatically generated and distributed credentials, the user can use the RFT and GridFTP mechanisms, without any configuration and additional authentication,

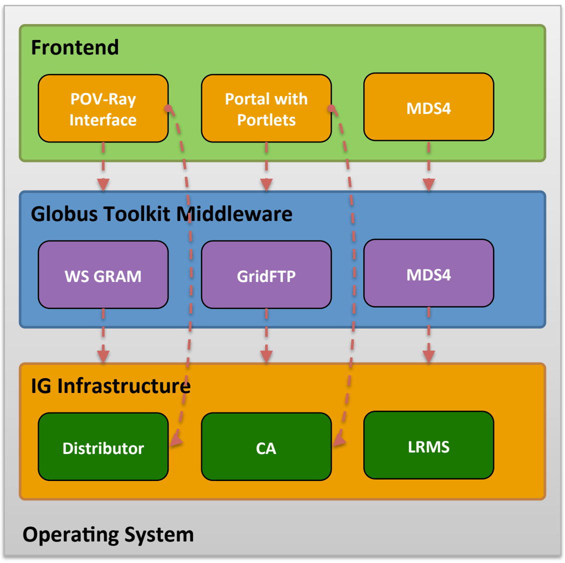


Figure 5.4.: Components of the IG Globus Toolkit Edition

by either the command line tools (globus-url-copy for GridFTP or rft for RFT) or by a file browser portlet in our GridSphere based portal.

Information Services The dynamic character of a grid - as opposed to the static character of a cluster - necessitates the availability of an information service, which can provide current data on status, resources, and services of the grid. In Instant-Grid the hardware resources are monitored using Ganglia[153], while the web service based Monitoring and Discovery Service (MDS4) of the GlobusToolkit handles information about available services. Both Ganglia and MDS4 are started and configured at boot time, without user involvement. Additionally, Instant-Grid provides test routines to ensure correct behavior of the registered resources and services. The availability and load information obtained by Ganglia can also be seen in a portlet, which reads these data from a resource database [230].

Workflow Management System On top of the basic GT4 middleware, Instant-Grid employs a flexible workflow orchestration infrastructure called *Generic Workflow Execution Service* (GWES) [114]. The GWES is a development of the Fraunhofer-Institute for Computer Architecture and Software Technology (FIRST) and is also used in several other European Grid projects, e.g. K-Wf Grid, Core Grid, and MediGRID. It fully automates the distributed execution of complex compound applications on distributed computing resources.

The application workflows consist of several subsequent program executions and intermediate data transfers. They also allow the modeling of program dependencies. The *Grid Workflow Description Language* (GWorkflowDL), a Petri net based *eXtensible Markup Language* (XML) language for process flows, is used to describe workflows on several levels of abstraction. This ranges from simple control tasks to the specification of complete executable grid operations.

5.3.3. Security Environment

Fast and easy changes to the user base and resource pool of a distributed computing environment are instrumental to its accessibility. Additionally, users can expect that the security of their data is not unduly compromised. Where these two goals are in conflict, we had to find a workable middle ground.

All nodes in the Instant-Grid use the same operating system environment, and therefore the same security configuration. For example, the user base and the operating system service configuration need not be individualized for each client. System-wide changes to setup files like "/etc/passwd" are distributed over the infrastructure by a distribution mechanism, which makes client management much easier. As it is common practice with cloud environments, a default user account is provisioned. In order to prevent unauthorized access, external login into this account is disabled by default.

Services like the middleware components run under the special system accounts ("globus" and "unicore", respectively), which are not available for login. The Instant-Grid clients use only a private network, and therefore are accessible from outside exclusively via the frontend, which in turn is protected by a firewall. Outside communication originating from the clients is handled by NAT on the frontend. (We described the technical details of the NAT setup in Section 5.2.1). Therefore, Instant-Grid can be both an isolated test environment and an accessible distributed research infrastructure site.

UNICORE The UNICORE security environment is based on the X.509 public key infrastructure. Therefore, an Instant-Grid certificate authority is initialized on the frontend as a part of the boot process. It is based on the Open Source OpenSSL toolkit and issues user and server certificates. Certificates identifying trusted parties are stored in a truststore. UNICORE 6 requires truststores in *Java Key Store* (JKS) format. The Instant-Grid certificate authority certificate generated during the startup process is included in every truststore. Private keys belonging to users or services are stored in a keystore, which can be *Public-Key Cryptography Standards #12* (PKCS12) or JKS format. The necessary key and trust stores are automatically created if a new user account is required by a new user, or a new client requires a host or service certificate.

Globus Toolkit The security layer of the Globus Toolkit middleware is provided by *Grid Security Infrastructure* (GSI) [225]. The GSI has become an accepted security infrastructure by *Open Grid Forum* (OGF) [169] and the *Internet Engineering Task Force* (IETF). When new end entities, be it users, hosts, or services, enter the grid, they automatically receive an X.509 end entity certificate, issued and signed by the Instant-Grid Certificate Authority (CA). New users also receive proxy credentials, which are stored in a MyProxy repository automatically.

When a new client boots up or when a new test user is generated, not only the hostfile and user database is updated, but also the new client or user gets its own X.509 certificate issued automatically by the Instant-Grid CA. This is made without any user interaction and fully automated (the certificate management works with OpenSSL CA functionality).

5.3.4. Customization Options

Configurations applied in a running Linux Live CD system are not persistent. Usually the system starts with default settings and the data is not kept after restarting the instance. For specific conditions such as in case for teaching, a customized Instant-Grid is required. Therefore, Instant-Grid supports two types of persistent configurations that are needed, e.g., for user or software management. These are described in the following.

5.3.4.1. Configuration in the Image Source

An Instang-Grid image can be customized by building a modified version of it.

Building and remastering images Changes are directly applied in the source from which the image is built. This process, also called remastering takes place every night because it is CPU and time intensive. A new version with the changes applied by the community in the day before is then produced automatically. It is also released to the developers as a test version every night in order that the modifications can be verified. We use the advantage that

IG is a Knoppix [101, 185] derivate and it uses the specific Cloop compressed filesystem. In contrast to usual images, Cloop makes it possible to integrate about 2 *Gigabyte* (GB) software on a 700 *Megagabyte* (MB) image. Near cloud environments, such an image can also be used as a traditional Live CD to boot up hadware machines easily.

Packaging software components In addition, binary Debian software packages are built to automate the management of additionally required software components. To create a debian package, the data part is downloaded (cached) directly from the software page on demand, and the control information is created by the Instant-Grid developers. Several legacy Debian packages have been built for the UNICORE middleware (UNICORE software components such as the UNICORE 6 Server, CIS, UCC, and RichClient), as well as for the Globus Toolkit middleware. We also build configuration packages for the services we utilize in Instant-Grid. We refer exemplary Coccinella, Ganglia, Gobby, GridSearch, Jabber, PostgreSQL, POV-Ray, Tomcat, as well as the components of the *Portable Batch System* (PBS) and *Message Passing Interface* (MPI) environment. For a consistent setup within Instant-Grid, an additional configuration package is built, which contains the Instant-Grid components.

The source code of these packages is under version control and available for developers. Pushing and pulling changes in the configuration is available via the source code repository. Finding, managing, pushing and pulling binary packages are possible by using the software repository.

Software repository When changes are made to the source code repository, the build workflows can be triggered automatically by hooks. A hook or a hook script is a process, which is usually triggered by a repository event, such as the modification of the source code. The output of the build workflows, the binary software packages, are then copied to the Instant-Grid software repository. This allows to find, manage, and push and pull packages.

The package management system of the operating system evaluates the meta-information to allow package searches, to perform an automatic installation, or to check that all dependencies of a package are fulfilled. The selected packages and their dependencies are installed automatically into Instant-Grid during the image build process.

The Instant-Grid image does not contain any configurations, e.g. hostname or certificates, because these are dynamically configured during the boot process.

5.3.4.2. Fast Adaptable Persistent Setup

Building a new image is a time, storage, and CPU intensive process. Therefore, we suggest to store specific and customized configuration data and scripts in a directory on the filesystem. This directory is mounted during the boot process and parsed for scripts which are then executed to apply certain configurations. This method enables a fast, local, and persistent configuration. However, it can only work on the ramdisk and is, therefore, quite expensive. Also, the data is not written permanently. As depicted in Figure 5.5, we determined two configuration phases: the pre-setup phase and the post-setup phase which are integrated in addition to the fully automated setup as presented in Section 5.2.1.

In the following, we describe the (1) pre-setup, (2) post-setup, and the (3) local persistent filesystem options.

Pre-setup phase After the hardware components are fully configured and all necessary disks are mounted, the pre-setup phase is invoked. This phase needs to be executed before services are started by the init-process. The pre-setup is suitable to change the default service configuration of Instant-Grid. For example, X.509 certificates used by the different services

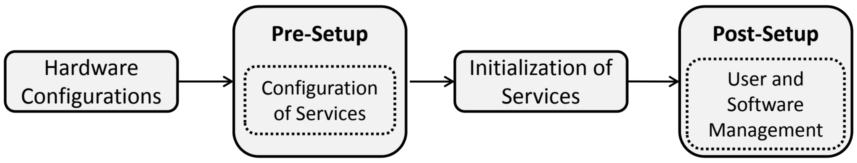

Hardware Configurations
Pre-Setup
Configuration of Services
Initialization of Services
Post-Setup
User and Software Management

Figure 5.5.: New Setup Phases During the Boot Process of the IG UNICORE Edition

are generated during every boot process. The pre-setup stage can be used to overwrite the default behavior for its generation and use specific service certificates issued by other certificate authorities. Another example is to change specific service ports in case they are not suitable for the local network environment or its administrative restrictions.

Post-setup phase The post-setup phase is invoked after all components are set up and all services are started. It can be used amongst others to add further grid users or install software packages that are not included in the Instant-Grid image.

Local persistent filesystem User applications or data can also be integrated by using a specific local and persistent filesystem. The filesystem is available on each computing node via the NFS, and can therefore be used by the applications. Within such a local trustful environment, multiple users are able to start, build, and use an experimental distributed computing system at any time without the restrictions that are associated with a remote environment.

5.4. Use Case: Practical Course

These customization features of the Instant-Grid allows to be utilized for education purposes, offering a multi site and multi user distributed computing environment. A practical course held at the University of Göttingen proved Instant-Grid to be a suitable tool for the course's technical support [188]. Being an adjustable and scalable instrument for ad-hoc distributed computing infrastructures, Instant-Grid enables extensions like specific portals or development environments to be included for short-term development. That way, students are able to utilize a distributed computing system on their own computers without the restrictions that are typically associated with a remote production environment.

Instant-Grid supports several demonstration modes, e.g. cluster computing, grid computing, platform-as-a-service, and software-as-a-service environments.

In Figure 5.6 we depicted the most common scenario of our supervised courses, where one Instant-Grid server can span several Instant-Grid clients. These clients can be utilized as a UNICORE or Globus Toolkit computational resource themselves. The Instant-Grid server used in the practical course is configured to utilize a local resource manager, i.e the PBS that employs lab computers from the University of Göttingen as a computing cluster. This configuration enables students to gain knowledge about cluster computing, grid computing and software-as-a-service using a bottom-up approach. The details of our gathered experiences we published in [188].

5.5. Connectivity with Research Infrastructures

The use case we mentioned in the previous Section modeled a multi site and multi user distributed computing environment, where jobs and workflows are run inside the IG environment. In contrast to that, IG can also be connected to existing productional distributed research infrastructures. In the following, we describe how IG can be utilized (1) to be a secure submission client of productional infrastructures and (2) to build a site which accepts workload from productional research infrastructures.

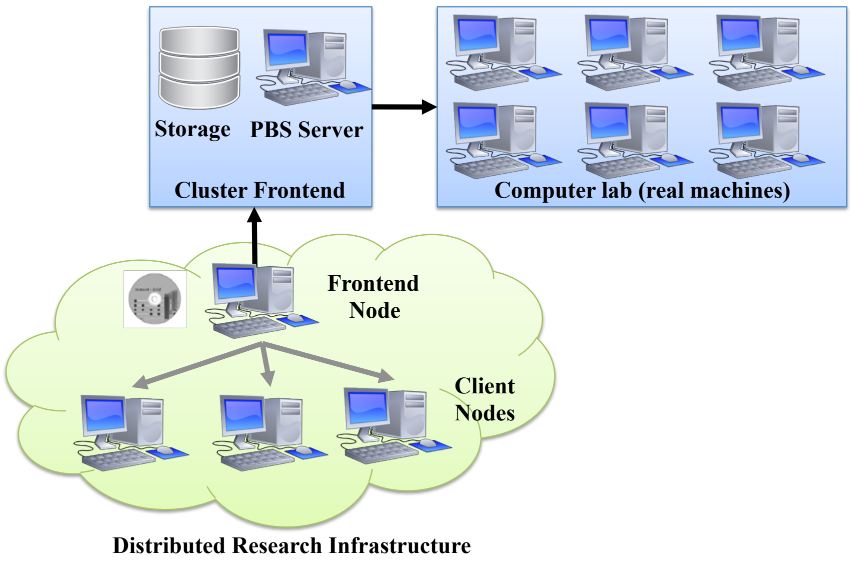

Storage
PBS Server
Cluster Frontend
Computer lab (real machines)
Frontend Node
Client Nodes
Distributed Research Infrastructure

Figure 5.6.: IG Environment Deployed in the Practical Course

5.5.1. Secure Submission Client of Productional Research Infrastructures

Using IG, the end users can run test jobs or workflows locally, before submitting them into a real distributed computing environment. A real production environment can be used, if more resources are required than available in the local IG, for example if no or not enough Instant-Grid nodes are available. The resources of the German e-Science Initiative [164] , accessible with valid user credentials, are supported by default. Instant-Grid was designed to help users with the creation of grid proxy credentials, which is the first step in accessing a grid infrastructure. The credentials can be stored in a local online repository. This prevents the private key from leaving the user's computer. There are general requirements for using resources of an operational research infrastructure. First, one needs proper network settings. The Instant-Grid server must not be behind a NAT router and must have a DNS resolvable fully qualified domain name and open ports. The user needs valid credentials in one of the accepted formats, which are usually PKCS12, X.509 PEM, or MyProxy online credential repository. The user needs to be authorized to use the production resources.

5.5.2. Accept Workload from Productional Research Infrastructures

The IG can operate as a site of a distributed computing and storage infrastructure, making it possible to share local resources. Booting IG in this mode starts a local batch scheduler (Torque) using the IG nodes in cluster mode. The distributed computing middleware of the frontend is configured to interoperate with the local scheduler and the job execution handled through the IG-server. The jobs can be submitted using the standard interfaces provided by the middlewares or using the Instant-Grid Portal. Providing such an external functionality is not enabled by default, but can be configured. Note that there are security and configuration issues related to the usage of a cloud environment or a live-CD. The general requirements are the same as for submission into a production infrastructure, detailed above.

The security infrastructure in IG accepts trusted host and user certificates only. If the site contacts other sites or accepts user certificates issued by other parties, the chain of trust from the issuer CA to the certificate must be established. The trusted CAs must be configured and the external interface of the IG-server needs to be certified by one of these CAs. The remote users should be mapped to a local user account, which can be done on the IG Portal with the User-Administration portlet. Data management like file stage-in, file stage-out, and file transfers between the remote client and the IG frontend is handled by the middlewares, which interacts with the batch system to provide for data staging further to the worker nodes.

There are different ways to integrate the IG monitoring and information services into external ones. Data from the information providers are collected on the frontend and can be forwarded to a remote index service, for example by upstreaming to an other MDS4 or CIS Index.

We also extended the GRDB [230] daemon. The original goal of GRDB is to store the local IG's hardware and software information periodically. We designed several enhancements which allow to use new information providers for collecting information about remote resources of productional infrastructures. We summarized our contributions (RGID and RGRDB) in [128, 127].

5.6. Related Work

In this Section we outline related work by introducing and discussing (1) our previous works on the topic, (2) the demonstration tools of the middlewares, and (3) the automated deployment of software components and services.

In contrast to our previous results on the topic [28, 29, 30], the current work can support several demonstration modes, e.g. it can act as a cluster, a grid, a platform-as-a-service, or a software-as-a-service environment. Our current approach can also be connected to existing productional research infrastructures. The principle of the automatically configured network remains the same in both editions. However, the differences are revealed in the deployment of the security environment and the different middleware services, which rely on different architectures. The new persistent setup and customization features described in this Chapter are also completely missing from the previous versions.

Regarding the demonstration tools of the middlewares, we briefly introduce and discuss the approaches of the middleware providers, namely (a) the UNICORE 6 Live CD, (b) the UNICORE 6 Testgrid, and (c) the Globus approach.

The UNICORE 6 Live CD [217] provided by the UNICORE community is a Live Linux system with all the UNICORE components already installed. UNICORE 6 Live CD contains preinstalled key- and truststores which remain the same in the CD image for each user. In the development of the UNICORE Live CD, all services are statically preconfigured. The UNICORE 6 Live CD is primarily designed for a single user on a single computer. Compared with it, IG allows persistent multi user configuration and automated, multi hosts deployment of the UNICORE middleware. Within this environment, it is also possible to obtain administrative access to configure the grid computing environment according to specific user requirements.

The UNICORE 6 Testgrid [216] comprises the latest version of all UNICORE 6 components. It is provided by the UNICORE community as a preconfigured bundle and enables potential new clients to easily access a UNICORE 6 demo site and test grid systems based in the UNICORE 6 edition. Users need to register in order to obtain access to the demo site's specific resources. Access is granted with a newly registered certificate and entitles users to submit jobs with limited requirements, i.e. jobs with low resource consumption or a short duration. Resources are in another administrative domain and need to be trusted. In contrast to the Testgrid users, IG users are not restricted in terms of the resource consumption or duration of a job. The IG resources are furthermore located in the home administrative domain and can therefore be fully trusted.

The latest developments of the Globus project provide a similar approach, which utilizes portal and science gateway components. It aims to lower the barriers to demonstrate distributed computing and data management possibilities by using just a web browser [86]. After signing-up for an account, Globus services can be tried out, files can be transferred and published reliably and securely directly from both cloud storage and locally owned storage. That way, one can become familiar with the basic concepts and interfaces of Globus. It helps to develop web applications or science gateways using the Globus *Platform as a Service* (PaaS) [6] as well as demonstrates how the *Software as a Service* (SaaS) paradigm provides advantages as a sustainable delivery method for scientific software [43]. Compared to our approach, Globus requires signing-up and offers online resources, while IG enables the utilization of secure, local resources.

Various concepts exist for the automated deployment of software components, as well as for the automated configuration of services. There are contextualization solutions, which are (a) de-facto standards and made available as operating system packages, or (b) unsupported tools that provide a way to customize preconfigured operating system components.

The first group includes industry standard tools, which bootstrap the startup of a system. An example is CloudInit[25], which allows to handle the early initialization of virtualized instances. Configuration management tools, like Ansible [110], Chef [151], Puppet [145], or SaltStack [103], help the deployment of software components and provides support for the automated configuration of services. Several ready-made modules are available in own repositories. These standard tools are supported by default in most modern Linux distributions.

At the other end of the spectrum, there are several unsupported tools that provide a way to customize preconfigured, read-only firmwares of embedded devices. These tools utilize undocumented hooks of the firmware or escalate privileges on various ways to allow the user to start additional programs on *Network Attached Storage* (NAS) devices or routers. We refer exemplary to FFP[26], Optware[27], and Entware-Next-Generation[28].

To create a suitable system design, it is important to understand where these concepts fit into the provisioning ecosystem. While such tools usually require additional steps for their installation, the IG provides several built-in phases for customization. We described the (1) pre-setup, (2) post-setup, and the (3) local persistent filesystem options in Section 5.3.4. That way IG combines the advantages of contextualization tools and Live-CD environments.

5.7. Analysis of Results

The goal of this work is neither delivering community-specific solutions, nor designing systems related to the recent technology trends. Much rather we address the common challenges of the competing computing paradigms due to identifying several similarities and the research progresses behind these paradigms. In this Chapter, we presented an extended edition of *Instant-Grid* (IG) that allows to build a multi site and multi user distributed research infrastructure. It can support several demonstration modes, e.g. it can act as a cluster, a grid, a platform-as-a-service, or a software-as-a-service environment.

The main benefit of IG is that it allows to establish a distributed computing environment based on well-known middlewares without any pre-existing knowledge of distributed computing technologies or the middleware systems. In this Chapter, we described the main technical realization behind such an environment. This includes a description of the automatic configuration setup, the customization options, the ready-to-use features of such an environment, as well as the demo applications of IG.

We highlighted the system design and the technical concepts in Section 5.2 and summarized our work on the topic of automated middleware deployment [28, 29, 30], as well as our results on the topic of self-configured demonstration and testing environments [135, 188].

The IG can be used for demonstration, education, and testing purposes in distributed computing systems. As a use case, we described in Section 5.4 a practical course of distributed computing where IG builds the technical base. Within such an environment, students are able to develop distributed services and applications, but also to establish, configure, and administrate their own computing system without interrupting any production environment. In addition, it provides a closed testing environment for distributed computing applications and systems [190]. It can be

[25]CloudInit, Online, `https://help.ubuntu.com/community/CloudInit`
[26]Fonz fun_plug (FFP), Online, `http://dns323.kood.org/howto:ffp`
[27]Optware, Online, `http://www.nslu2-linux.org/wiki/Optware/HomePage/`
[28]Entware-Next-Generation (Entware-NG), Online, `https://github.com/Entware-ng/Entware-ng/wiki`

used and deployed for the demonstration of distributed computing technology without knowing the complex constrains of such an environment. It is easy deployable into cloud environments or without impact on the local machines.

Furthermore, we considered to integrate a mechanism for handling overload capacity by providing a solution to integrate external resource management systems. In addition, global connection to other distributed computing sites can be set up in order to allow the use of additional external resources (Section 5.4).

In Section 5.4 we outlined related work by introducing and discussing (1) our previous works on the topic, (2) the available demonstration tools of the middleware providers, and (3) automated deployment of software components and services. For (1) we found that the current work, in contrast to our previous results [28, 29, 30], can support several demonstration modes like cluster, grid, platform-as-a-service, or software-as-a-service environment. The new persistent setup and customization features described in this paper are also completely missing from the previous versions.

Regarding (2), we briefly introduced and discussed the demonstration tools of the middleware providers. We found that (a) the demonstration environments are primarily designed for a single user on a single computer, (b) the users have to register in order to obtain access, (c) the users submit jobs with limited requirements (low resource consumption or a short duration), and (d) the resources need to be trusted. Compared with it, (a) the IG allows persistent multi user configuration and automated, multi hosts deployment, (b) no registration is required, (c) the IG users are not restricted in terms of the resource consumption or duration of a job, (d) the IG resources are furthermore located in the home administrative domain and can therefore be fully trusted.

For (3) we grouped the contextualization solutions in (a) de-facto standards and made available as operating system packages, and (b) unsupported tools that provide a way to "customize" preconfigured operating system components. For a suitable system design, it is important to understand where these concepts fit into the provisioning ecosystem. The IG provides several built-in phases for customization. We described the (a) pre-setup, (b) post-setup, and the (c) local persistent filesystem options. That way IG combines the advantages of contextualization tools and Live-CD environments.

As future work, we consider to install more demo applications, and implement concepts for supporting further computing paradigms, e.g. *Infrastructure as a Service* (IaaS). The demo applications should be configured automatically to use computing and storage resources inside the IG - independently from the applied computing paradigm. Tools for developers also should be integrated. That way IG should provide a suitable environment to develop applications. In addition, a meta scheduler is required that schedules jobs automatically to IG resources. Furthermore, the various editions of IG (e.g. the IG Globus Toolkit edition and the IG UNICORE edition) will be combined, so that the user can choose the middleware.

Recent technology trends shift the standardized unit both for software development as well as for deployment of distributed applications. For example, various open platforms achieve packaging, deployment, and shipment of applications by defining containers. The *Container as a Service* (CaaS) paradigm enables to ship applications together with their dependencies in a container, which is the standardized unit for system designers, developers and system administrators. Because IG utilizes similar technologies, e.g. UnionFS layered file systems, early initialization of instances, etc., we continue to monitor the CaaS standardization processes. We plan to evaluate several container system variants including Docker[29], LXC[30], and make use of several standardized components of the CaaS design.

[29]Docker, Online, `http://www.docker.com/`

[30]Linux Containers, Online, `https://linuxcontainers.org`

The contribution, we described in this Chapter, was a solution for automated system deployment, which allows to set up a self-configured and independent multi site and multi user distributed research infrastructure. We investigated in the technical concepts including the automatic configuration, presented ready-to-use features, and applications. We connected this environment with productional distributed research infrastructures and applied this environment as our simulation framework. We showed that it is also suitable for demonstrating, developing, and testing purposes of distributed computing environments.

6. Describing, Monitoring and Publishing Resource Quality in Distributed Research Infrastructures

Here we focus on describing, monitoring and publishing the quality and performance of resources in distributed research infrastructures. To demonstrate the practical relevance of our work, we chose the german life science communities as a use case. First, we describe the state of their quality metrics, afterwards we identify further key performance indicators that can be used by the life science communities for defining service levels that can be agreed on. We discuss how information systems can publish and exchange the quality information, as well as how the information model can handle them. Finally, we present two approaches to measure and monitor the quality metrics in multi-middleware environments: an external benchmarking system and traditional monitoring system.

This chapter summarizes our work on the topic of describing, measuring, monitoring and publishing resource quality in distributed research infrastructures ([129, 132], as well as [127, 128, 130, 131, 148]).

Previously in this work, we described an information modeling process for exchanging resource descriptions in distributed research infrastructures (Chapter 3). We first identified the main actors and their information demand, outlined generic entities of a theoretical information model, and presented a theoretical approach for a schema mediation process for distributed research infrastructures. That way, we gave a solution to describe the main characteristics of heterogeneous, distributed research infrastructures in a standardized way. We followed in Chapter 4 with using the results of our theoretical analysis and designed an automated resource description exchange process and a respective generic monitoring architecture supporting it. We also developed a proof of concept for grid environments.

Based on our results presented in the previous chapters, it is possible to discover monitoring- and information services of distributed research infrastructures that are the entry points to computing and storage services of such infrastructures. Furthermore, the presented concept is also capable to extract the source monitoring data and can transform it into a generic data model required for exchanging resource information in environments, which utilize different middlewares.

In this Chapter, we focus on managing distributed research infrastructures, which comprises various activities. Measuring, describing, monitoring and publishing the quality and performance of resources enables to perform essential management activities. To demonstrate the practical relevance of the problem, we chose the german life science communities as a use case. The communities aim to employ Service Level Agreements (SLAs) with their resource providers to ensure the delivery of services. For this, it is important for both the communities and their providers to understand and quantify the performance and service quality of different resources. However, measuring service quality in distributed infrastructures utilizing different middlewares, is a complex problem. We describe the state of quality metrics which are currently used by the German life science communities MediGRID, Services@MediGRID and PneumoGrid. We also identify further quality metrics for defining, describing, measuring and monitoring the resource quality. It is important to publish and exchange the quality information by information systems, which are the entry points to the services of distributed research infrastructures. Therefore, we also present how quality information can be handled by the GLUE v2.0 Schema, which is our generic data model. For measuring and monitoring the quality metrics in multi-middleware environments two approaches are discussed. The first approach extracts quality information from an external benchmarking system and loads it to the information systems. The second solution targets life science communities that do not utilize legacy benchmarking systems, but operate traditional monitoring systems, like Nagios.

The prerequisites to understand this Chapter includes knowledge about the computing paradigms utilized to set up distributed research infrastructures (Chapter 2), as well as our examinations of the information and data models (Section 3.5.3), the information and monitoring systems (Section 3.5.2), and the middlewares (Section 3.5.1). We outlined the generic entities of a theoretical information model that are capable to describe the main characteristics of heterogeneous, distributed research infrastructures in Section 3.4. We also need to recall our analysis regarding to the determination of generic models at a conceptual level (information models) as well as at a lower level of abstraction (data models) (Section 3.6). Furthermore, we refer to our system design of a distributed monitoring architecture for an interoperable and integrated information service (Section 4.3), as well as its implementation within the distributed research infrastructure set up by the German e-Science initiative (Section 4.4). We also build upon the three distributed infrastructure scenarios we described in great details in Section 4.1. Finally, we refer to research works made on the field of *Service Level Agreement*s (SLAs), *Key Performance Indicator*s (KPIs), and the management of distributed infrastructures.

This Chapter is structured as follows: we introduce our motivation for measuring, describing, monitoring, and publishing resource quality in distributed research infrastructures in Section 6.1.2. We give an overview of the quality metrics currently used by the german life science communities and we suggest further metrics and key performance indicators in Section 6.2. The capabilities of the generic information schema is then analyzed in Section 6.3. We also describe the attributes and entities of the generic model that can be applied to exchange resource quality information. Following the analysis of the information schema, we shift our focus to data sources which can provide quality information (Section 6.4). In Section 6.5 and 6.6 we present two case studies to evaluate our concept. First we extract benchmark scores from an external system and in our second case study we extend a traditional monitoring framework to monitor stateful grid services efficiently. Finally, we consider related work (Section 6.7), and conclude with the analysis of our results (Section 6.8).

This chapter is partly adapted from [129, 132], as well as from our presentations [127, 128, 130, 131, 148].

6.1. Managing Distributed Research Infrastructures

Managing distributed research infrastructures comprises various activities [98].These activities are related to each other and should be harmonized with the aim to maximize the effectiveness and so the return on investment. Describing, monitoring and publishing resource quality are the most important aspects that form the basis of the management of distributed research infrastructures. Depending on such information, essential management activities can be performed.

6.1.1. Activities of Managing Distributed Research Infrastructures

For this work we defined 5 activities, which we use for the management of distributed research infrastructures. We depicted these activities on Figure 6.1.

The five activities of our management process are defined as follows:

1. Services and Resources:
   The first step of our modeling process is to identify the main services and resources, which we aim to manage in the distributed research infrastructure. This step is done in a way that

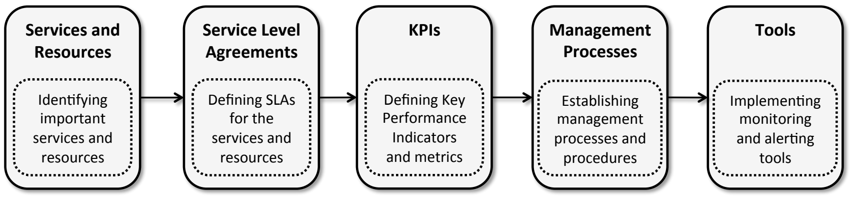

Services and Resources
Identifying important services and resources
Service Level Agreements
Defining SLAs for the services and resources
KPIs
Defining Key Performance Indicators and metrics
Management Processes
Establishing management processes and procedures
Tools
Implementing monitoring and alerting tools

Figure 6.1.: Activities of Managing Distributed Research Infrastructures

leads to a solution which is independent from the computing paradigm utilized to set up the distributed research infrastructure.

2. Service Levels:
   Defining service levels is the next step of our management model. For the products and services, which were identified in the previous step, investigations on their scope, criticality, and severity have to be made.

3. KPIs:
   The service levels are usually defined with indicators and metrics, which can be monitored and measured in distributed research environments. Such indicators and metrics are called in this work *Key Performance Indicator*s (KPIs).

4. Management Processes:
   The next step is to create management processes and establish the related procedures. Such procedures are of particular importance, when it comes to handle incidents in a distributed research infrastructure.

5. Tools:
   Proper tools are needed to support the management activities and processes. Not just individual entities, but complex components have to be managed to meet service levels.

Managing distributed research infrastructures, which utilize different middlewares, is a complex problem. We continue by describing the scenario of the German life-science communities, and their most important aspects that form the basis for managing their distributed research infrastructure.

6.1.2. Monitoring Resource Quality

Today's distributed research infrastructures are complex environments and utilize several different middlewares. We refer to Section 4.1, where we chose three distributed infrastructure scenarios and described them in great details. Here we just recall that the German e-Science initiative [164], which is a research program to establish a national research infrastructure for research and industry, supports three middleware systems [2], namely gLite Lightweight Middleware for Grid Computing (gLite) [97], Globus Toolkit 4 (GT4) [79], and Uniform Interface to Computing Resources (UNICORE) [221]. Another example for utilizing various middlewares is the *European Grid Infrastructure* (EGI). A collaboration, named *European Middleware Initiative* (EMI), of the major european middleware providers with the mission to deliver a consolidated set of middleware components for deployment in EGI as part of the *Unified Middleware Distribution* (UMD).

Life science communities, like MediGRID [143], Services@MediGRID [67] and PneumoGrid [144], have to agree on Service Level Agreements (SLAs) with their resource providers. Measuring service quality in distributed research infrastructures utilizing different middlewares, like the German e-Science Initiative, is a difficult problem. The different kind of life science applications of the communities perform varyingly in different environments. For instance, the application AUGUSTUS from the bioinformatics has no such requirements of stable environments as image processing applications. Another example is life science applications that have to transfer input and output data during job execution, but require less computing power. In order to consume infrastructure services with appropriate characteristics for their different applications, the life scientists require detailed information about the service quality and performance of the resources.

At the same time, the providers aim to offer stable, sustainable, and performant distributed infrastructures. If the service availability and reliability were not monitored, the resource-providing organizations are not in a position to deliver the service quality agreed in *Service Level Agreements* (SLAs). In order to continue to provide even more performant services expected by the users, the performance of different infrastructure components need to be evaluated.

Measuring service quality in distributed research infrastructures utilizing different middlewares is a complex problem. Neither a standard evaluation schema, nor quality assessments for the resources exist. Many traditional tools measure only one property, for example, CPU speed (Flops), network bandwith (Mb/s). However, a single number cannot express the quality of the resources.

Complex benchmarking tools were developed, which can test distributed services, but they do not directly publish this information to the research infrastructure's information systems. Therefore the benchmark information is not accessible for research infrastructure's tools, like metaschedulers. A way has to be found for extracting the measured values from the external benchmarking systems.

Data with information about resource quality still has to be exchanged between providers, users, and services. Therefore, the quality information has to be described and published in a standardized form. For that purpose, abstract data models are used, which determine the structure of monitoring data in information systems. However, many data models utilized by current research infrastructure's information systems were not designed to handle quality information. Therefore, the standard data models have to be examined and extended if needed.

A further problem is that the benchmark tools can not handle the stateful infrastructure services correctly.

The following sections of this paper address the following questions:

- What metrics are important for measuring resource quality in distributed research infrastructures?
- How can the resource quality information be published and exchanged?
- From what sources can the resource quality information be collected in distributed research infrastructures?
- How can the quality information be modeled with the generic data schema?

6.2. Quality Metrics and Key Performance Indicators

On top of the basic GT4 middleware, the German life science communities employ further specialized services. One of the most important services is the GWES [114], which is an advanced workflow system for orchestrating the distributed execution of applications on the resources of a research infrastructure. In this section we describe the state of the art of measuring resource quality by the GWES scheduling system. Following the analysis of the currently utilized metrics, we shift our focus to further quality metrics which we suggest for monitoring the resource quality.

For describing the performance or checking the quality of services, KPIs are widely accepted mechanisms in the industry. KPIs are reportable metrics in order to measure the health or performance of the service. The KPI is defined in a measurable way, therefore it can be checked whether the measured value meets the expectations or not.

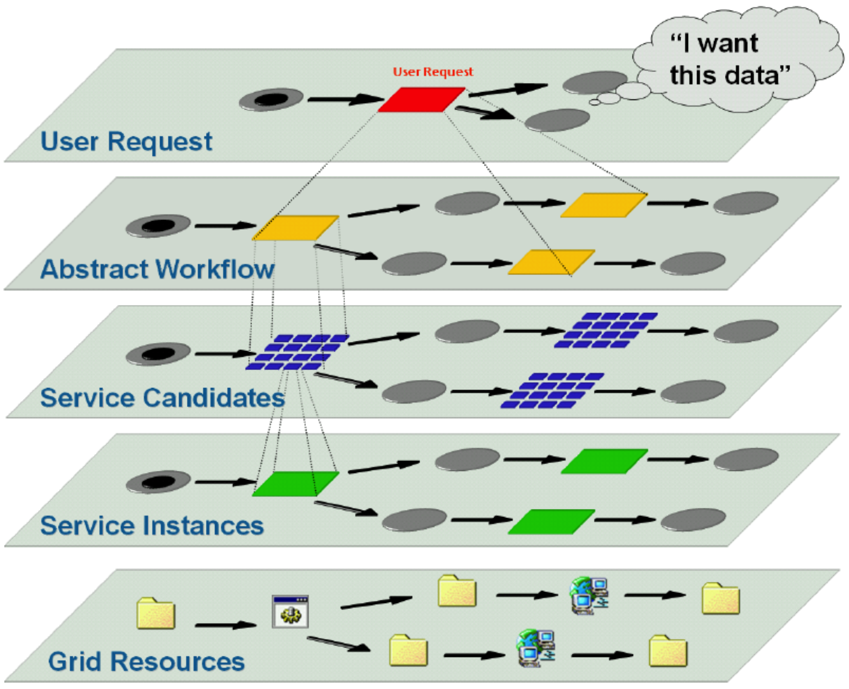


Figure 6.2.: Resource Matching by GWES (source: GWES User Manual[31])

6.2.1. State of the Art Measuring Resource Quality

The computational activities of the german life science communities MediGRID, Services@MediGRID, [redacted] PneumoGrid are grouped into workflows and delivered to the resources by the GWES. For program executions, GWES does a resource matching, as it is shown in Figure 6.2. First, it searches for suitable resources where the program has already been installed. For this, every life science community maintains a resource database that provides information about the software installed on their resources. The resource database contains resource descriptions in XML format suitable for the D-GRDL. We discuss the D-GRDL data model in the following section. The resource descriptions are periodically updated by a resource monitoring daemon. A description contains static resource information and utilization data, which is usually very dynamic.

- Utilization value
  The utilization value is calculated by the scheduling algorithm and it is between 0 (busy) and 1 (idle) for all resources. Thus the utilization value u, i.e. metric, is

$$u \in \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}.$$

GWES selects the resources based on the current utilization of the resources, but simple quality metrics are also considered during resource matching. The ResourceMatcher component of GWES considers the following quality metrics:

- Current status
  The current status of a service is very important for scheduling jobs to those services only, which are working correctly. The resource monitoring daemon periodically updates the status information.
- Success-Failure ratio
  The GWES workflow system accounts the successful executions of job activities and up-

[31]GWES User Manual - The Generic Workflow Execution Service `http://www.gridworkflow.org/kwfgrid/gwes-web/KWF-WP2-D2-FIRST-GWESUserManual.pdf`, Online, last accessed on December 12, 2016

Algorithm 1: The Calculation of the 'Success-Failure' Quality Metric

```
Initialize ratio = 0;
while there are jobs to schedule do
    schedule the job;
    check return value;
    if successful execution then
        ratio = +1;
    else
        ratio = -1;
    end
end
```

dates the Success-Failure quality metric of the resource description. Algorithm 1 explains that the value is increased for every successful activity and decreased after failed executions.

6.2.2. Suggested Key Performance Indicators and Quality Metrics

The currently used quality metrics can be measured and calculate in a simple way. However, the current metrics cannot handle complex scenarios. For example, if a resource went offline for a certain time, but it processes the jobs without errors, then it gets a good score for its Success-Failure ratio, even if the availability of the resource was not good.

We identified further quality metrics which we suggest for monitoring the resource quality in distributed research infrastructures:

- Resource availability
  Monitoring systems of distributed infrastructures check periodically whether a resource is accessible and works correctly. If the resource was not accessible or did not deliver the expected job output, then the resource is not available for the life science community. The practical meaning is: how long the resource is available and accessible for the life science community. Mathematically, the resource availability is expressed as a ratio of the value of the uptime of the resource to the aggregate of the values of its uptime and downtime:

  $$Availability = \frac{Accessible}{Totaltime} = \frac{Uptime}{Uptime + Downtime}$$

  The availability is a number between 0 and 1. Thus, the availability value a is

  $$a \in \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}.$$

  For example, if a resource was accessible for 1 hour and had a scheduled downtime for 1 hour, then its availability is:

  $$Availability = \frac{Accessible}{Totaltime} = \frac{1hour}{1hour + 1hour} = \frac{1}{2}$$

  We point out that the availability can be expressed as a direct proportion (for example, $\frac{1}{2}$) or as a percentage (for example, 50%), since it is a number between 0 and 1.
- Resource reliability
  Reliability is the ratio of the uptime of a service to the time the service was scheduled to be accessible. In other words, if a resource was scheduled to not be accessible, then it does not get punished.

$$Reliability = \frac{Accessible}{Totaltime - Scheduleddowntime}$$

The reliability is a number between 0 and 1. Thus, the reliability value r is

$$r \in \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}.$$

For example, if a resource was accessible for 1 hour and had a scheduled downtime for 1 hour, then its reliability is:

$$Reliability = \frac{Accessible}{Totaltime - Scheduleddowntime} = \frac{1hour}{2hour - 1hour} = \frac{1}{1} = 1$$

We point out that the availability can be expressed as a direct proportion (for example, $\frac{1}{2}$) or as a percentage (for example, 50%).

- Job turnaround time
  The job turnaround time is the time required for the execution of a job. Submitting multiple jobs or simulating job workflows refine the metric.
- Data turnaround time
  During job execution it is a common scenario that input data and output data have to be transferred. The metric is the time needed for the data transfer. Transferring multiple files or transferring data with different size refine the metric.
- Information query turnaround time
  The information services are very important for successful job execution, because they provide descriptions of the infrastructure's resources. The information query turnaround time is the time required to collect the necessary resource information from an information service.
- Services throughput
  The resources of a distributed research infrastructure should be robust. Overloading the services can have different characteristics. The number of requests attended per a unit of time represents the throughput of the service.
- Licenses
  The utilizing of commercial licenses in distributed research environments is a complex problem and it was also addressed outside of the life science communities. The number of licenses that can be provided per a unit of time represents the maximum number of licensed applications that can be run by the service.

6.3. Service Quality in the Data Model

The provider, user, and services have to exchange data with resource quality information. Therefore the quality information has to be described and published in a standardized form. Some of the data models utilized by current grid information systems were not designed to handle quality data, other models have limited capabilities only. In this Section, we discuss the D-GRDL [232], which is the current data model of the GWES workflow scheduling system utilized by the german life science communities. Then we shift our focus to the GLUE v2.0 Schema [70], which is the generic information model for exchanging monitoring data of the D-MON [21]. The D-MON project has a mission to provide an infrastructure-wide monitoring system that works with various middleware environments. This is essential for monitoring the status of the resources and services that are used by various communities. This is as relevant for the life science communities as it is for resource providers. The D-MON Project utilizes the GLUE v2.0 information schema, which is the latest published version of a standard effort running within the OGF [169]. The purpose of the GLUE v2.0 Schema is to define a standard information model for defining common entities in

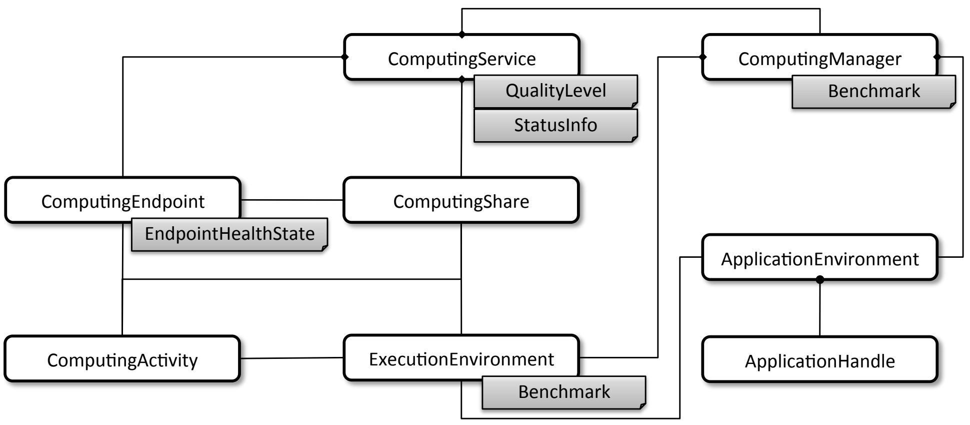


Figure 6.3.: Entities of the GLUE v2.0 Schema and Their Attributes Related to Describe Quality of Computing Resources

various distributed infrastructure environments. In the following we discuss the quality attributes both of the currently used D-GRDL data model and the GLUE v2.0 Schema.

6.3.1. The Currently Used D-GRDL Data Model

The *D-Grid Resource Description Language* was developed within the German DGI project for describing sets of grid resources. The D-GRDL is defined by an XML-Schema, which offers a framework for the description of data to all kinds of resources. The resources have properties and the properties can be extended easily. This way quality metrics and key performance indicators can be represented easily. The Status and the Success-Failure Ratio quality metrics are currently represented as resource properties.

6.3.2. GLUE v2.0 Data Model for Computing Resources

The GLUE v2.0 Schema defines entities for computing- and storage resources. In this subsection we discuss the computing resource entity. The main GLUE v2.0 Schema entities are depicted in Figure 6.3. We analyzed the GLUE v2.0 information model and found the following attributes related to describe quality of computing resources:

- QualityLevel
- EndpointHealthState
- StatusInfo
- Benchmark

The attributes are shown in Figure 6.3 on the bottom of their entities. In the following, we shortly describe these attributes.

6.3.2.1. QualityLevel Attribute of a Computing Service

The GLUE v2.0 Schema uses the Computing Service entity as an abstract entity for modeling computing capacity in grids. The conceptual model of the Computing Service entity enables to define the maturity of the computing services in terms of quality of the software components. A computing service can exactly have one quality level. The possible service quality levels are described by the QualityLevel_t type in the GLUE v2.0 Schema. Table 6.1 summarizes the quality levels.

Value	**Description**
development	The component is under active development both in functionalities and interfaces.
pre-production	The component has completed the development and passed the testing phase; it is being used in real world scenarios.
production	The component completed the development and is considered stable for real world scenarios.
testing	The component has completed the development phase and is under testing.

Table 6.1.: The Quality Levels of a Computing Service Defined by the GLUE v2.0 Schema

Value	**Description**
critical	It was possible to check the state of the endpoint and either it was not running or it was above some critical threshold.
ok	It was possible to check the state of the endpoint and it appeared to be functioning properly.
other	It was possible to check the state of the endpoint, but this is not covered by the defined states.
unknown	It was not possible to check the state of the endpoint.
warning	It was possible to check the state of the endpoint, but it appeared to be above some warning threshold or did not appear to be working properly.

Table 6.2.: The Health States Defined by the GLUE v2.0 Schema

6.3.2.2. EndpointHealthState Attribute of a Computing Endpoint

The conceptual model uses the Computing Endpoint entity to describe an interface for job submission, management, and monitoring. The model of the Computing Endpoint entity enables to define a state representing the health of the endpoint in terms of its capability of properly delivering the functionalities. A computing endpoint can be in exactly one state. The possible health states are defined by the EndpointHealthState_t type in the GLUE v2.0 Schema. Table 6.2 summarizes the health states.

6.3.2.3. Benchmark Attribute of Execution Environments and Computing Managers

An Execution Environment entity in the GLUE v2.0 model is a type of environment providing computing capacity to a grid job. The environment is described in terms of hardware, operating system and network characteristics. The Computing Manager entity is a software component locally managing one or more Execution Environments. The computing manager is also known as a *Local Resource Management System* (LRMS). The conceptual model of the Computing Manager and Execution Environment entities enables to define zero or more associated benchmarks. The

Value	**Description**
bogomips	BogoMips
cfp2006	SPEC CFP 2006 floating point benchmark
cint2006	SPEC CINT 2006 integer benchmark
linpack	LINPACK benchmark
specfp2000	SPECfp2000 floating point benchmark
specint2000	SPECint2000 integer benchmark

Table 6.3.: The Benchmark Types Defined by the GLUE v2.0 Schema

possible benchmark types are defined by the Benchmark_t type in the GLUE v2.0 Schema and are summarized in Table 6.3.

6.3.2.4. StatusInfo Attribute of a Computing Service

The StatusInfo attribute of a Computing Service entity holds an *Uniform Resource Identifier* (URI) of a web page providing additional monitoring information. The structure and format of the web page is not specified. A computing service has zero or more associated status information web pages.

6.3.3. GLUE v2.0 Data Model for Storage Resources

The GLUE v2.0 Schema defines entities for computing- and storage resources. In this subsection the storage resource entity is discussed. We analyzed the GLUE v2.0 information model and found the following attributes related to describe quality of storage resources:

- QualityLevel
- EndpointHealthState
- StatusInfo
- *the Benchmark attribute is not foreseen in the GLUE v2.0 model*

In the following we shortly describe these attributes and we introduce an extension to the GLUE v2.0 schema containing Benchmark attributes for storage services.

6.3.3.1. QualityLevel Attribute of a Storage Service

Analogous to the QualityLevel attribute of a Computing Service (described in Section 6.3.2.1).

6.3.3.2. EndpointHealthState Attribute of a Storage Endpoint

Analoguous to the EndpointHealthState attribute of a Computing Endpoint (described in Section 6.3.2.2).

6.3.3.3. StatusInfo Attribute of a Storage Service

Analoguous to the StatusInfo attribute of a Computing Service (described in Section 6.3.2.4).

Association End	**Multiplicity**	**Description**
StorageService.ID *(existing)*	1	A storage manager participates in a storage service.
DataStore.ID *(existing)*	0..*	A storage manager manages zero or more data stores.
Benchmark.ID *(NEW)*	**0..***	**A storage manager has zero or more associated benchmarks.**

Table 6.4.: Benchmark.ID is a Suggested New Attribute to the GLUE v2.0 Schema

6.3.3.4. Suggested Attribute for Storage Manager

Storage Managers are software components that locally manage data stores. The Storage Service conceptual model of the GLUE v2.0 Schema does not foresee any benchmark attribute for Storage Managers. We suggest the extension of the Storage Manager conceptual model with a new Benchmark.ID attribute. The new Benchmark.ID association endpoint has 0..* multiplicity and defines zero or more benchmarks associated with the storage manager.

6.4. Data Sources of Quality Information

In the previous subsections we discussed the metrics required for monitoring service quality and performance in distributed research infrastructures. We also described the attributes of the data models that can represent quality-related data in the information systems. In Chapter 4 we described an ETL method to provide interoperable monitoring with different source systems. With ETL one can extract, transform, and load service quality data. In this section we discuss data sources that contain information related to quality metrics.

Extracting Data from External Benchmarking Systems This approach aims to extract benchmark data from scoring systems that do not provide interfaces for the monitoring systems. Our first case study is an extractor for Jawari Grid Benchmarking and it gets the calculated scores from an external benchmarking system. Our solution transforms the benchmark scores into a standardized form and loads them to the information systems. We discuss the approach in 6.5

Extending Traditional Monitoring Systems for Collecting Quality Metrics Information This approach targets life science communities that do not utilize legacy benchmarking systems, but operate traditional monitoring systems, like Nagios. Our solution extends Nagios with new plugins that collect test execution results during monitoring. From the results quality scores can be calculated. The approach is discussed in 6.6

6.5. Case Study 1: Extractor for the Jawari Benchmarking System

In [127] an approach is discussed how existing legacy benchmarking services could be accessed by an integrated, interoperable monitoring and information system. For this, an ETL adaptor has to be developed which can extract the benchmark and measurement data from the Jawari Benchmarking Toolkit, which aims to provide benchmarks for different middlewares.

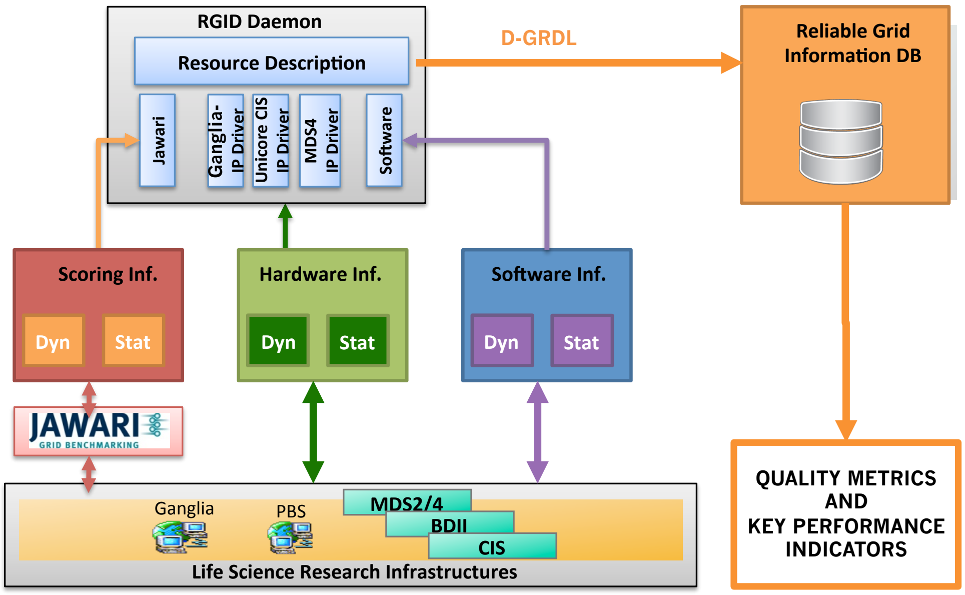


Figure 6.4.: RGID Interface for the Jawari Grid Benchmarking Service

6.5.1. Jawari Benchmarking Toolkit

The Jawari Benchmarking Toolkit [123] provides benchmarks for different middlewares. It utilizes a scoring algorithm based on the Human Development Index [5] and variable normalization. The benchmark suite is currently composed of 20 benchmarks focused on different components and aspects of the distributed research infrastructures. However, Jawari is under development and with the help of the community the number of benchmarks is increasing. The current Jawari benchmarks are: Three Node Probe, Circle Probe, Gather Probe, Data Job, Error Prone Task, Conditional Task, Discovery Overload, Job Submission Overload, File Transfer Overload, Single File Transfer, Single Discovery, Static Single Task, Static Bag of Tasks, Static Long Pipe, Static Compound Pipe, Static Mixed Bag, Dynamic Single Task, Dynamic Bag of Tasks, Dynamic Long Pipe, Dynamic Compound Pipe, Dynamic Mixed Bag, Single Connection. The current benchmark suite focuses on job submissions and data transfers. The job execution benchmarks represent simple jobs, independent tasks, interconnected tasks, dependent tasks, and interdependent tasks. The data exchange benchmarks provides tests from simple data transfers to complex scenarios: Jawari can benchmark a data exchange between two hosts, but also can probe the robustness of services by transferring multiple files in parallel.

6.5.2. Extracting the Results of the Static Single Task Benchmark

Jawari has only one default formula for calculating the benchmarks. The Jawari score calculation adopts an approach based on variables normalization, also found in other composite indexes such as the Human Development Index. The overall performance and quality metric of a resource is the result of the performance of several features of the environment. In scenarios when custom quality metrics are calculated by communities or users, benchmark scores should not be made by the default Jawari formula. Users or services require more information than a score from the benchmark system.

In the following, we present a prototype for extracting the results of Jawari probes. Figure 6.4 depicts the architecture of the prototype. The extractor is utilized by the RGID [128]. The Static Single Task benchmark simulates the simplest distributed application. This is our use case for extracting benchmark scores from Jawari.

6.5.2.1. Jawari Authentification

The Jawari-Extractor uses the Jawari client Java-API. The extractor has to be registered at the Jawari benchmarking service and it has to authenticate itself with the registered loginname and password as listed in Listing 6.1:

```
/**
  * Jawari client API requires that the extractor is
  * registed at the Jawari Benchmarking Service and
  * it authenticates itself with loginname and password
  * @param loginname the registered username
  * @param passwd the registered password
  * @return the status of authentication
*/

public boolean auth(String loginname, String passwd) {
  interface.client = JawariClient.newInstance();
  // log in to the Jawari Service
  client.login(loginname, passwd);
  // do not forget to log off from the Jawari Service
  // with: client.logoff();
  return client.authenticated();
}
```

Listing 6.1: Authentification to the Jawari Benchmarking Service

We describe the two most important functions of the Jawari extractor:

6.5.2.2. Timestamp of the Latest Service Benchmark

The function, which gets the latest timestamp of benchmark for a given service running by the grid site, is listed in Listing 6.2:

```
/**
  * Returns the latest timestamp when the required
  * benchmark type was executed for the service.
  * @param uri the service URI.
  * @param benchmarkType the benchmark type.
  * @return the execution timestamp.
*/

public Date getLatestTimestamp(String uriService,
  BenchmarkType benchmarkType) {
  EvaluationResult evaluationResult =
    client.getLatestEvaluationResult(scheduleName);
  Evaluation evaluation =
    evaluationResult.getEvaluation();
  return evaluation.getLatestUpdate();
}
```

Listing 6.2: Get the Timestamp of the Latest Service Benchmark

6.5.2.3. Latest Result of a Service Benchmark

The function, which delivers the last result of a service test, is listed in Listing 6.3:

```
/**
  * This returns with the latest test result for one host
  * (execution status of a service benchmark)
  * @param uri the service URI.
  * @param benchmarkType the benchmark type.
  * @return the value if the benchmark execution was
  * successful; and the error message otherwise..
 */

 public String getLatestTestResult(String uri,
  BenchmarkType benchmarkType) {
  Set<BenchmarkResult> results =
    getLatestBenchmarkResults(uri, benchmarkType);
  BenchmarkResult latest = results.iterator().next();
  if (latest instanceof FailureResult) {
    FailureResult failure = (FailureResult) latest;
    return failure.getErrorMessage();
  } else {
    return latest.getValue();
  }
}
```

Listing 6.3: Get the Latest Result of a Service Benchmark

The functions getLatestTimestamp() and getLatestTestResult() enable to extract the time and the result of service tests executed by Jawari.

6.6. Case Study 2: Extending a Traditional Monitoring Framework for Extracting Quality Information

The benchmarking systems are not commonly utilized by the communities, but traditional monitoring systems are widely operated. Our second approach uses a traditional monitoring system (Nagios) for checking stateful services of distributed infrastructures and it extends Nagios with new plugins that collect test execution results during monitoring. As a byproduct, quality scores can be calculated from the test results. In this subsection, we describe the system developed at the GoeGrid resource center and its implementation within the research infrastructure set up by the German e-Science initiative.

The German life science communities can choose between the GT4, gLite and UNICORE6 middlewares. This subsection provides an overview of a flexible monitoring framework developed to monitor heterogeneous distributed research infrastructures that employ stateful services, like grid services. Stateful grid services use resources and resource properties in order to enable the storage and retrieval of data even if service consumers disappear. For this purpose, persistent data is generated in the service container, and this data should be cleaned up properly in every case. The clean-up process applies analogously to the different kinds of tests. To monitor such services in an efficient way, we implemented mechanisms to correctly handle the Web Service Resource Framework. Even if a failure occurs during the execution of a test, the monitoring tool ensures the proper deletion of the resources and resource properties. Therefore, the performance consequences of a large number of non-usable resource instances is prevented. Stateful grid services enable web services to access states in a consistent manner by generating persistent data in the service container. To monitor stateful services in an efficient way, the persistent data should be cleaned up properly in every case. Currently, no homogeneous service monitoring exists that effectively observes stateful services offered by the heterogeneous middlewares. We present a homogeneous monitoring tool which is independent from the middleware and extensible for future distributed computing models, e.g. cloud implementations. In our solution, we utilize the monitoring system for collecting monitoring data that can be used for calculating key performance indicators.

6.6.1. Homogeneous Monitoring of Heterogeneous Stateful Services

In order to provide a stable and reliable infrastructure based on stateful services the security infrastructure, the job execution management, the data management, and the monitoring and information service components of the middleware should be monitored.

6.6.1.1. Stateful Services in Middlewares

A stateful service is a web service that supports management of the state through properties associated with the web service. The Web Service Resource Framework (WSRF) is a generic and open framework for modeling and accessing stateful resources (WS-Resource) using web

Middleware Component	Globus Toolkit v4.x	UNICORE6
Security Components	Container & CA certificates	Keystore, CA certificates
Job Execution Management	WS-GRAM (Fork,PBS,LSF)	UNICORE XNJS
Data Management	Reliable File Transfer	XNJS File Transfer Service
Information Service	MDS4	CIS

Table 6.5.: WSRF-based Services in GT4 and UNICORE6 for Remote Testing

services in grid environments. The state of a WS-Resource is supported by properties (WS-ResourceProperties). The stateful grid services generate persistent data in the service container. This approach enables the storage and retrieval of stateful data even if service consumers may disappear. There is no need for clients to remain online throughout the duration of long job executions. Nevertheless, the stateful data should be cleaned up properly. If the monitoring tool does not ensure the proper clean-up of the stateful resources after the execution, a large number of useless resource instances are stored in the service containers. The traditional stateless service monitoring interrupts the test, if a timeout or failure occured, and do not take care of stateful service data.

6.6.1.2. Use Cases: Globus Toolkit 4 and UNICORE6 WSRF Implementations

Several implementations of the WSRF standard exist. We chose the GT4 and UNICORE6 middlewares as use cases for our monitoring system. In the following, we briefly describe the most important services of the GT4 and UNICORE6 middlewares and their stateful behaviour relevant to service monitoring.

Globus WSRF The Globus Toolkit is an open source software toolkit used for building grid systems and applications. The Web Services Grid Resource Allocation and Management (WS-GRAM) component in GT4 includes some WSRF-compliant web services to submit, monitor, and cancel jobs on computing resources. The Reliable File Transfer (RFT) service in GT4 manages file transfers and GridFTP operations. When a client submits a data transfer request to the RFT service, the RFT service maintains the state of the data transfer in a relational database. The RFT service can handle data transfer problems and restart GridFTP operations if needed. The Monitoring and Discovery System (MDS4) is the monitoring component of the toolkit and allows users to discover resources and services on grids.

For GT4 we chose the WS-GRAM job management, the RFT data management, and the MDS4 information service components for monitoring because these stateful services are important for the production. These services are summarized in Table 6.5.

UNICORE6 WSRF The UNICORE6 grid services run inside the UNICORE/X container. The UNICORE Gateway is the entry point to a grid site running the UNICORE6 middleware. A UNICORE Gateway can serve several target systems behind it and performs the authentication of client requests. The job execution management component is the X Network Job Supervisor (XNJS) that provides job management services, storage resources, and file transfer services. The Common Information Service (CIS) is responsible for the monitoring. It collects the information from several UNICORE6 sites and publish the monitoring data using the GLUE2 Schema.

We chose the UNICORE Gateway, the XNJS job- and data management, and the CIS information service components of the UNICORE6 middleware for monitoring because these grid services are important for the grid production (see Table 6.5).

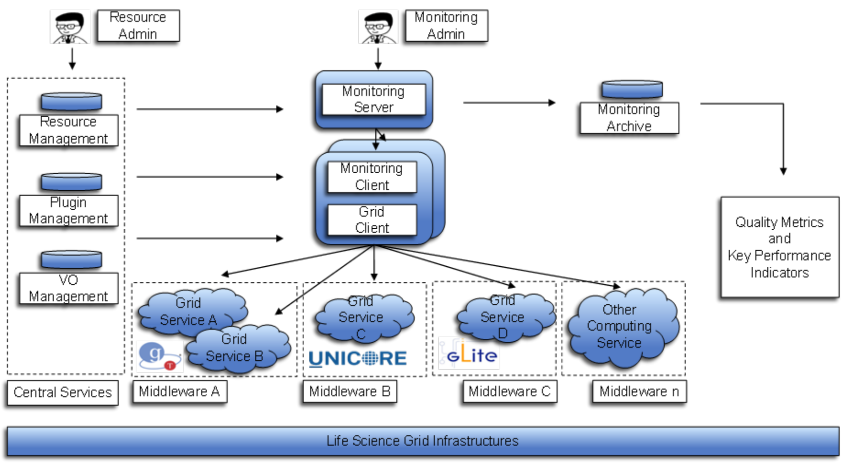


Figure 6.5.: Extending a Traditional Monitoring System for Collecting Information Related to Quality Metrics

6.6.2. Architecture and Implementation

The architecture of the monitoring system is depicted in Figure 6.5. On the lowest layer, there are various distributed infrastructure services provided by the GT4, UNICORE6, and gLite middlewares. The design of the monitoring system is extensible for distributed computing models of the future, e.g. cloud implementations.

The monitoring server schedules and coordinates the execution of the tests. It evaluates the test results and sends notifications if necessary. The monitoring client interacts with the server and the underlying components. Depending on the executed test, it uses a specific client to communicate with the service. It also transforms the test result and test output to a specific format required by the monitoring server. A monitoring archive stores the test results for analyzing trends or generating availability reports. The central services are shown in the left-hand column. These management components are part of the central research infrastructure.

6.6.2.1. Monitoring Server

In our approach, the Nagios monitoring framework was chosen for this task. Nagios is an open source monitoring framework which observes hosts, network devices and services. We defined a hierarchy between hosts or services to allow handling of relations between the monitored entities. Nagios schedules service checks depending on the service operation period, checking frequency, maximum number of retries and check timeout. The checks use external *plugins* which return status information to Nagios. Nagios sends notifications to administrative contacts of the research infrastructure resources when service or host problems occur and get resolved. The test results can be stored in the monitoring archive. In addition to the status of a test, various performance data (for example the job execution time, file transfer time) is archived.

6.6.2.2. Monitoring Client and Grid Client

Nagios does not provide any plugin to check grid services, therefore we developed custom plugins for that purpose. The components, which interacts with the server and the underlying grid services are called plugins. A plugin is an executable that expects well known arguments (hostname, port, timeout, etc) and runs a specified test. The developed plugins transform the test results and produce output in a specific format required by the server. A plugin can return OK, Warning, Critical or Unknown states to the server. We used the Nagios::Plugin modules to create plugins that conform the Nagios Plugin guidelines [163]. The Nagios::Plugin modules provide a simple,

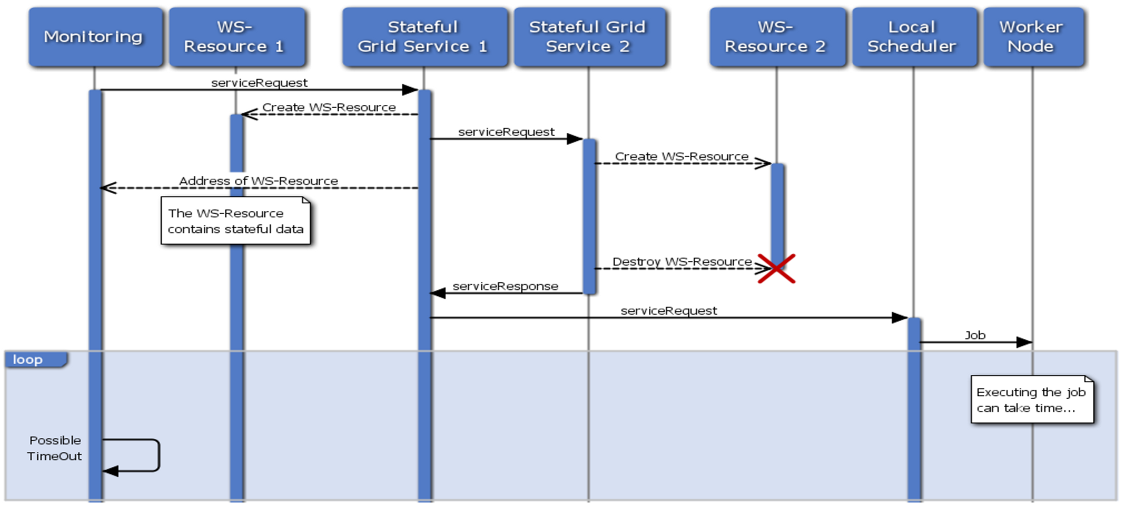


Figure 6.6.: Possible Monitoring Time-Out during Test Execution: Stateful Service Data Is Not Destroyed

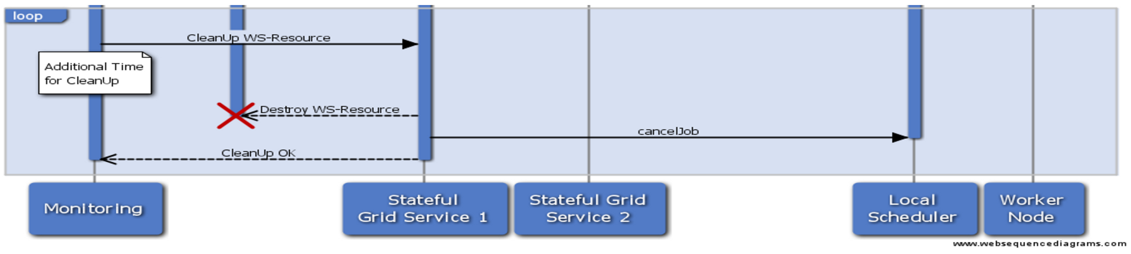


Figure 6.7.: Using Additional Time-Out for Proper Clean-Up in Case of a Monitoring Time-Out

object-oriented Perl interface for developers. For example, the *Nagios::Plugin::Getopt* module provides standardized argument processing for Nagios plugins.

6.6.2.3. Central management components

The central services are shown in the left-hand column. These management components are part of the central infrastructure. There are several components of a research infrastructure that are used by a monitoring system. For example, resource providers in D-Grid register their resources offered for the communities at a central Resource Registry Service (GRRS). The monitoring server can query this service to get information about the registered services and the resource administrators.

6.6.2.4. Handling Stateful Resources

In case of a time-out or failure, a traditional monitoring framework for stateless services only breaks the connection to prevent overloading of the monitoring service and does not take care of stateful service data (Figure 6.6). Our plugins utilize two different timeout durations in order to provide for the deletion of stateful resources in case of successful test executions or failures, respectively. The first timeout duration is the *normal* one used by Nagios, and we use it to define a termination time for the normal test execution part. Once this timeout period is reached, the test is aborted and an *additional* timeout period starts as shown in Figure 6.7. During this additional period, the stateful resources are destroyed. In case of an error during the clean-up process, the process could be automatically reiterated until the stateful resources are successfully destroyed. In our current implementation, the plugin returns with a critical error after a failed clean-up and then the monitoring administrator is notified.

6.7. Related Work

Monitoring of distributed research infrastructures has been a topic for many years. Several monitoring tools have been developed and utilized both by the communities and the resource providers. The existing monitoring infrastructures (like the one utilized in EGI [118] or the one deployed in GoeGrid [129]) are based on consistent user-level testing across resources to improve infrastructure stability (user-level monitoring) or they collect and aggregate monitoring data from existing tools (a system-level view of the resources).

One approach to monitor distributed research infrastructures is to provide user-level monitoring. An example of such a system is the Inca monitoring system [208]. Inca provides consistent user-level testing across resources to improve the stability and to support the operation. It makes possible to detect, solve, and analyze user-level failures from a generic, impartial user's perspective. The periodic, automated tests run under a standard user account and standard credentials are used for test execution. These executables, called reporters, measure some aspect of the system and output the result as XML. A Perl API is provided to create new reporters. The part of our work, which extends Nagios with new plugins, was inspired by the monitoring infrastructure of EGI, which was developed to observe the gLite middleware. The system is introduced in [118] and it is based on the open source monitoring framework Nagios. The focus of the research is the monitoring of resources in the distributed infrastructure for failure detection, notifications and automatic recovery. Sensors for various services, the sensor hierarchy and certificate based authorization on web interfaces are also described. These approaches aim to monitor traditional services and do not ensure the proper cleanup of stateful resources. The provided plugins are based on their custom-made developments and not necessarily conform to the Nagios Plug-in Development Guidelines [163].

Another monitoring approach is to collect and aggregate monitoring data from existing tools and provide a systems-level view of research infrastructure resources. GridICE [7] is such a solution. It is an open source distributed tool for distributed systems that provides both a summary and detailed view of the status of the resources, highlights a number of common failures and presents usage information.

The monitoring approaches described above focus on the current status, but not on the quality of the resurces. Benchmarking tools with various benchmark specifications already exist and are based on computationally intensive tests (the NAS Grid Benchmark (NGB) [89] provides metrics based on the job turnaround time) or apply both file-transfer and computing tasks (for developing benchmarks based on data-intensive applications, like [46]) or focus on low-level resource performance (like GridBench [220]). However, these scoring systems usually do not provide interfaces to the systems of distributed research infrastructures.

Our solution aims at research communities using or offering diverse WSRF middleware implementations. Our developed plugins also conform to the Nagios Plug-in Development Guideline [163]. Our work defines essential quality metrics to employ SLAs in distributed research infrastructures and utilizes both, monitoring and benchmarking systems as sources to extract quality information.

6.8. Analysis of Results

Today's distributed research infrastructures support the requirements of several communities simultaneously. To achieve this goal the German e-Science Initiative, for instance, utilize three middleware systems (Globus Toolkit, UNICORE, and gLite) for computing resources. Measuring, describing, monitoring and publishing the quality and performance of resources of such a diverse infrastructure is a complex task.

The different kind of applications of the communities perform varyingly in different environments. For employing *Service Level Agreement*s, it is essential for the research communities to understand and quantify the performance and service quality of different environments.

This chapter described the current quality metrics used by the German life science communities MediGRID, Services@MediGRID and PneumoGrid. We also identified new *Key Performance Indicator*s that can help to consume infrastructure services with appropriate characteristics for the various life science applications.

Publishing and exchanging quality information in a standardized form by the research infrastructure's information systems is an important step on the way to enable interoperability. Therefore, we presented how quality information can be represented by the currently employed D-GRDL data model and the GLUE v2.0 Schema.

For measuring and monitoring the quality metrics in multi-middleware environments we presented two solutions. The first case study, our extractor for Jawari, gets the calculated scores from an external benchmarking system. The benchmark scores are transformed into a standardized form and loaded to information systems of the distributed research infrastructure. The second case study targets life science communities that do not utilize legacy benchmarking systems, but operate traditional monitoring systems, like Nagios. Our solution extends Nagios with new plugins that collect test execution results during monitoring. From the results quality scores can be calculated.

As a future work, new key performance indicators could be defined and existing quality metrics could be refined. For example, simulating job workflows instead of submitting a single computing job, or data management involving more files parallelly instead of a single file transfer, could refine the existing metrics.

Further steps are necessary to define sets of key performance indicators for common life science applications or computational workflows. For achiving this goal, we plan to profile several life science applications. Then, it will be possible to identify typical workflow profiles and define suitable benchmark sets for them.

In this Chapter we focused on describing, monitoring and publishing the quality and performance of resources in distributed research infrastructures. The use case of the german life science communities demonstrated the practical relevance of our work. We started by investigating in their current quality metrics, afterwards we identified further key performance indicators that can be used by the life science communities to define service levels that can be agreed on. We discussed how the information model can handle such quality information, and how the information systems can publish and exchange them. We also presented two approaches to measure and monitor the quality metrics in multi-middleware environments: an external benchmarking system and traditional monitoring system.

7. Conclusion

In this chapter, we summarize the thesis, discuss its contributions and findings, and point out its limitations (Section 7.1 and 7.2). In addition to that, we outline possibilities to extend or refine the results and methods presented in this thesis, and we state directions for future work (Section 7.3).

7.1. Summary

In this dissertation, we focused on problems relevant to exchanging resource information in distributed research infrastructures. Our main motivation was to bring forward the interoperability of such infrastructures[32].

The first step to reach this vision is approaching interoperability between the basic components, for example security, job execution, data management, and information- and monitoring systems. We specifically addressed the information- and monitoring systems because of their high importance for all other components of a distributed research infrastructure.

To achieve interoperability of information- and monitoring systems, we focused on the information modeling through exchanging resource descriptions, discovering resources, as well as monitoring and publishing service quality. These tasks are essential for executing jobs or managing data in distributed research infrastructures.

The main objectives of this work are structured around enabling the exchange of resource descriptions based on a generic data model. We aim to present a generic concept for interoperable monitoring of our application domain, which is independent from the middlewares and extensible for future distributed computing models.

We neither designed monitoring systems related to the recent technology trends, nor did we aim to deliver solutions specific for communities. Instead, we addressed the common challenges of the competing computing paradigms by identifying several similarities and the research progresses behind these paradigms. The following section briefly reviews our main conclusions.

7.2. Thesis Conclusions

We group the results of our work into four categories: (1) generic information modeling, (2) generic architecture, implementation and proof of concept, (3) automated system deployment, and (4) additional metadata for resource descriptions. A detailed analysis of our results is presented in the respective chapters. In the following, we summarize the generic conclusions of the overall work.

7.2.1. Generic Information Modeling of Heterogeneous Resource Descriptions and Monitoring Data

In terms of generic information modeling, we described and analyzed the different computing paradigms that are utilized to set up distributed research infrastructures. Instead of providing individual solutions for each computing paradigm, we focused on the problems that all computing paradigms of our application domain have in common and provided generally valid solutions to these problems. The focus of our research was the exchange of resource information in distributed

[32]See Section 1.2 and Section 1.3 for a detailed list of our research contributions and their impacts.

research infrastructures. To assure an information exchange, a standardized process and a common understanding of how to describe the resources are required. The most crucial factor for success is the modeling of information.

Information Modeling Process of Heterogeneous Resource Data To provide a theoretical background for the information modeling process, we defined a 5-step process that is independent from the computing paradigm: (a) identifying the main actors of distributed research infrastructures, (b) analyzing the requirements for resource information from the view points of the various actors, (c) identifying a basic set of information which describes the main characteristics of an infrastructure, (d) determining models at a conceptual level (information models) as well as at a lower level of abstraction (data models), and finally, (e) investigating how the ETL data warehousing technique can be adapted to map existing information models onto a generic model. The respective steps are described in the sections 3.1-3.7. Here we just briefly summarize our results.

The main actors of distributed research infrastructures we identified based on an analysis of the different infrastructures and the definition of a high-level use-case model. These actors include (a) the community users, (b) the community administrators, (c) the community services, (d) the infrastructure administrators and resource providers, and (e) the infrastructure services. For each actor of the distributed research infrastructure, we examined a usage scenario and derived the actor-specific *requirements for resource information and monitoring data*.

This laid the foundation for identifying the required *generic entities* of a data model, which describes the following *main characteristics* of a distributed research infrastructure: (a) communities, (b) access policies and mapping policies, (c) allocation of resources and services to communities, (d) modeling resource and service scenarios, and (e) modeling resource providers. These characteristics serve as a basic set of abstract information, which is necessary for interoperable monitoring and a shareable resource description.

To find a model that fulfills these characteristics, we delivered a state of the art analysis of (a) the distributed research infrastructure middlewares of our application domain, (b) their information and monitoring systems, and (c) their information- and data models. The findings of our examination can be found in Section 3.5. As a result of our analysis, we could identifiy two models that best fulfill our requirements; namely the *Grid Laboratory Uniform Environment* (GLUE) Schema and the *D-Grid Resource Description Language* (D-GRDL) Schema. While the GLUE Schema is the most widely accepted information model according to our research, the D-GRDL is designed to be very generic and subsequently enables the definition of virtually all kind of objects of a distributed research infrastructure.

Finally, we investigated how the existing information models can be mapped onto the generic model. We adapted the *Extract, Transform, Load* (ETL) data warehousing technique to show how the generic model can be filled by the information models used by today's research infrastructures. We considered the interoperable information system as a data warehouse, which is why we can apply research results made in the field of data warehousing, ETL processes, and schema mapping.

7.2.2. Generic Architecture, Implementation and Proof of Concept

Regarding the architecture implementation and proof of concept, we concluded our work with 3 main considerations: (a) resource exchange is possible, (b) community-aware regulation of access brings new challenges, and (c) lossless transformation can be ensured.

Proof of concept and prototype We used the results of our theoretical analysis outlined in the previous section in order to provide a prototype as well as a proof of concept for our application domain. To provide the prototype, we designed an automated resource description exchange process and a respective generic monitoring architecture supporting it. Results of our work showed that the exchange of resource descriptions coming from the monitoring and information systems of different research infrastructures is possible.

Community-aware regulation of access to information In terms of access regulation, we designed a community-aware monitoring system, which uses the identity information to regulate access to resource information. Our research results indicate that the federated identity management solutions in the scenarios of our application domain use a rather complex security model. Various workarounds have been introduced in order to lower the complexity of the security management for non-experts[33]. A further harmonization of the established federated identity management systems in the future is likely. However, we point out that such a development brings new challenges, like (a) the sovereignty of policy decision points, (b) the detection of community memberships' expirations, (c) the self-registration possibilities for citizen scientists, as well as (d) the persistence of user identities and their uniqueness across the infrastructures. Thus, we stress that the translation between identity management systems results in the challenge to realize the same trust level in the different systems of distributed research infrastructures. We achieved the *technical* interoperability by our system design, but the managerial issues, e.g. *integrating the policies*, raise new research questions.

Lossless transformation During the examination of the information- and monitoring systems for our proof of concept, we found semantic differences. These differences might lead to information losses in the transformation process. Our implementation work focused on the sources MDS4, BDII, and CIS. We were able to show that important data for the relevant values can be gathered and transformed without loss of accuracy.

Many monitoring systems do not exclusively utilize one of the original standard data schemas. Instead, they have implemented changes and supplements according to their specific needs. Consequently, even systems that are based on the same schema, for instance on the same GLUE version, are not always completely compatible with each other. Hence, they cannot necessarily be translated into each other without losses.

7.2.3. Automated system deployment and productional infrastructures

To set up a self-configured and independent multi site and multi user distributed research infrastructure, we described a solution for an automated system deployment and investigated the technical concepts behind. We showed that the presented solution is also suitable for demonstration, development, and testing purposes of distributed computing environments.

We also connected this environment with productional distributed research infrastructures and applied this environment as our simulation framework. Our approach supports the automated integration of distributed infrastructures, however, our approach solely addresses *technical* solutions for the integration. New research questions are raised related to the *organizational* issues, which need to be addressed as well (e.g. only services not be behind a NAT router accepted, a DNS resolvable fully qualified domain name is required, and the user needs to be pre-authorized to use the resources).

7.2.4. Additional Metadata for Resource Descriptions

Concerning additional resource metadata, we focused on three main aspects: (a) the identification of data history by data provenance, (b) the description, monitoring and publication of the resource quality and performance, and (c), the necessity of naming standards and global unique identifiers.

[33]Various workarounds have been introduced, we refer exemplary to [88, 105, 166].

Data History by Data Provenance In an integrated monitoring system, monitoring data and resource information are gathered from heterogeneous middlewares. A documentation of the history and origin of the monitoring data is crucial, especially when the middlewares send diverging data sets belonging to the same resource. Our solution adds provenance information by extending the database schema of our implementation. Each transformation step and each data item is enriched with provenance information. For our scenarios, we defined (1) the source information or monitoring system, (2) the exact time and date of retrieval, and (3) the IP-address of the source system as necessary provenance information.

Description, monitoring and publication of the resource quality and performance Our generic information modeling concept and the monitoring architecture supporting it enable the exchange of resource information. A better picture of the service quality is needed for the definition of service levels that can be agreed on. Our research contributed to a better picture of the service reliability by describing, monitoring, and publishing the resource quality and performance. In order to demonstrate the practical relevance of our work, we chose the German life science communities as a use-case. We started by describing the state of the quality metrics and continued by identifying further key performance indicators that can be used by the life science communities. We then assessed how information systems can publish and exchange this information, and how information models can manage the information. Last but not least, we presented two approaches to measure and monitor the quality metrics in multi-middleware environments: an external benchmarking system and traditional monitoring system.

The necessity of naming standards and global unique identifiers Integrated, unified monitoring discloses the structural shortcomings of monitored infrastructures. This may lead to duplicate entries that describe the same underlying object. Monitoring data usually consists of information about resources, services, jobs, and activities that exist within the research infrastructure. Identifiers that are globally unique across infrastructures are needed for the consistency in tagging. That way, proper statistics, efficient searches, and scheduling can be guaranteed. The assignment of such identifiers, e.g. names and numbers, is an organizational issue. Similar issues have already been solved, for example by establishing the *Internet Assigned Numbers and Names Authority* (IANA) for the IP-addresses to autonomous systems in 1984, and by establishing the *Digital Object Numbering Authority* (DONA) for the *Digital Object Architecture* (DOA) [120] in 2014. Identifying resources in a unique, persistent way requires common naming standards or policies. These could be assigned at well-known policy decision points. For a deeper insight, we refer the interested reader to our work in that field [134].

7.3. Outlook

In this Section we outline possible research items, which extend the results and methods presented in this thesis. We state three interesting directions for future work: (1) automated system provisioning and service deployment, (2) evolution of paradigms, (3) extension of data and information modeling.

7.3.1. Automated System Provisioning and Service Deployment

For automated system provisioning and service deployment, it is important to understand where the various concepts fit into the provisioning ecosystem. We described our solution, which combines the advantages of contextualization tools and image-based Live-CD environments. Recent technology trends have transformed the standardized unit for both, the software development as well as the deployment of distributed applications. For instance, to attain an effective packaging, deployment, and shipment of applications, various platforms that define containers or utilize micro-services have been developed. One example is the *Container as a Service* (CaaS) paradigm, which enables system designers, developers, and administrators to ship applications together with their dependencies in a container. This new design philosophy has established *containers* as the standardized unit.

A rapid growth in container-based solutions has recently made the Docker container technologies and the Docker image format a de facto standard for many purposes. The solution presented in this thesis uses similar technologies to the major open source CaaS platforms, for example UnionFS layered file systems and the early initialization of instances. Nevertheless, we feel it desirable that open governance structures and more formal, open, industry specifications should be introduced. This would, however, lead to a transformation of the container-based solutions. We therefore continue to monitor the major CaaS initiatives and their standardization processes. The *Open Container Initiative* (OCI)[34], for instance, is based on an open governance structure and aims to create open industry standards around container formats and cloud runtime environments. An other example is the *Cloud Native Computing Foundation* (CNCF)[35], which is about to create a new set of common portable container technologies and to drive their adoption. We plan to evaluate the outcome of these activities and make use of several standardized components of the CaaS design.

7.3.2. Evolution of Paradigms

Over time, various computing paradigms have evolved with the goal to establish computing as a new utility and turn the promise of distributed resource sharing into reality. This evolution of paradigms also affects the supporting technology that backs the distributed research infrastructures. From the technical point of view, this implies that paradigms overlap and no clear border exists between them. Subsequently, the landscape of tools supporting these paradigms is changing rapidly, and the same tools are often utilized to implement different paradigms. One example for this development is the Globus Toolkit, which we examined as a grid middleware in this thesis, but is now also supporting PaaS and SaaS [36] in the same manner.

Besides that, other related aspects also need to be addressed. For instance, the concepts of AAI also evolve with the paradigms. To monitor such infrastructures and exchange resource descriptions, the various AAI concepts should be supported. Based on our research results, we propose to support new paradigms by developing new adaptors. Also, we can observe that besides computing, research data is becoming more important as well. Therefore, monitoring activities focus not only on computing infrastructures, but to an increasing extend also on distributed infrastructures which support distributed research data management. Research is increasingly conducted on such *Collaborative Data Infrastructure*s (CDIs), for example in the context of EUDAT[37], RDA[38], and ePIC[39].

The solutions we provided in this thesis are independent from the paradigms. We propose further research that investigates how these solutions can be applied to distributed data infrastructures.

7.3.3. Extension of Data and Information Modeling

Interoperability is based on the utilization of standardized components and application of standards. Several interoperability and standard initiatives for distributed computing and data systems exist, for example the *Institute of Electrical and Electronics Engineers* (IEEE) [117], the *Distributed Management Task Force, formerly "Desktop Management Task Force"* (DMTF) [68], the

[34]*Open Container Initiative* (OCI), Online, `https://www.opencontainers.org/about`

[35]*Cloud Native Computing Foundation* (CNCF), Online, `https://cncf.io/about`

[36]Globus allows to develop science gateways using the Globus Platform [6], as well as it demonstrates how the SaaS paradigm provides advantages as a sustainable delivery method for scientific software [43].

[37]EUDAT, Online, `https://www.eudat.eu/`

[38]*Research Data Alliance* (RDA), Online, `https://rd-alliance.org`

[39]*European Persistent Identifier Consortium* (ePIC), Online, `http://www.pidconsortium.eu`

Open Commons Consortium, formerly the Open Cloud Consortium (OCC) [167], and the *Open Grid Forum* (OGF) [117]. While the work of these initiatives has a strong focus on interoperability, their activities are still diverse and their outcome offers a very high degree of extensibility. With regard to our field of research, their information and data models are not necessarily compatible with each other.

In the context of our work, we delivered an information modeling process to provide a solution for the transformation of such models. Due to the complexity of the generic schema, e.g. the GLUE v2.0 Schema, we defined the mappings manually and our implementation work focused on the source systems MDS4, BDII, and CIS only. To support further models and to take into account the continuous changes in already existing models, we aim to to automate the schema mapping and schema matching approaches as much as possible. In this way, both the costs of the data integration work, as well as the possibility of the mapping errors would also be reduced.

The process of automatically identifying whether and how two schemas are semantically related is challenging. Further research in this field can draw on previous work on database transformations[40] as well as on the many schema and ontology matching methods have been proposed[41]. Due to the extensive research on matching algorithms, current matching technologies can deliver better performance compared with the early schema mapping systems[42] and the application of these strategies in the context of distributed research infrastructures is an interesting new research question. In the context of our work, we propose future work on hybrid matching techniques, which combine the label-based matching techniques with the structure-based matching algorithms to allow suitable and well-performing full or semi-automated matching systems.

[40]Various workarounds have been introduced, we refer exemplary to [58].

[41]We refer to studies that describe and assess the most relevant approaches [184, 25, 203, 204, 173].

[42]Compared with the early schema mapping systems recent algorithms can deliver *"dramatic change in the performance"* [31].

Bibliography

[1] G. Aceto, A. Botta, W. De Donato, and A. Pescapè. Cloud Monitoring: A Survey. *Computer Networks*, 57(9):2093–2115, June 2013.

[2] M. Alef, T. Fieseler, S. Freitag, A. Garcia, C. Grimm, W. Gürich, H. Mehammed, L. Schley, O. Schneider, and G. Volpato. Integration of Multiple Middlewares on a Single Computing Resource. *Future Generation Computer Systems*, 25(3):268–274, March 2009.

[3] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus Striped GridFTP Framework and Server. *SC Conference*, 0:54, 2005.

[4] Amazon Inc. Amazon Web Services. `http://aws.amazon.com/`. Online, last accessed on December 12, 2016.

[5] S. Anand and A. Sen. Human Development Index: Methodology and Measurement. Human Development Occasional Papers (1992-2007) HDOCPA-1994-02, Human Development Report Office (HDRO), United Nations Development Programme (UNDP), June 1994.

[6] R. Ananthakrishnan, K. Chard, I. Foster, and S. Tuecke. Globus Platform-as-a-Service for Collaborative Science Applications. *Concurrency and Computation: Practice and Experience*, 27(2):290–305, 2014.

[7] S. Andreozzi, C. Aiftimiei, G. Cuscela, S. D. Pra, G. Donvito, V. Dudhalkar, S. Fantinel, E. Fattibene, G. Maggi, G. Misurelli, and A. Pierro. Next Steps in the Evolution of GridICE: A Monitoring Tool for Grid Systems. *Journal of Physics: Conference Series*, 119(6):062010 (4pp), 2008.

[8] S. Andreozzi, S. Burke, L. Field, and B. Konya. Towards GLUE 2: Evolution of the Computing Element Information Model. *Journal of Physics: Conference Series*, 119(6):062009, 2008.

[9] S. Andreozzi. Information Modeling of Grid Resources: the OGF GLUE WG Approach. In *DMTF Symposium*, 2007.

[10] M. Antonioletti, M. P. Atkinson, R. Baxter, A. Borley, N. C. Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead. The Design and Implementation of Grid Database Services in OGSA-DAI. *Concurrency and Computation: Practice and Experience*, 17(2-4):357–376, February 2005.

[11] O. Appleton, A. Cook, and H. Hämmerle. EGEE. `http://egee-na2.web.cern.ch/egee-na2/files/material/EGEEin2007-final.pdf`. Online, last accessed on December 12, 2016.

[12] M. Araya, M. Solar, and J. Antognini. A Brief Survey on the Virtual Observatory. *New Astronomy*, 39:46–54, 2015.

[13] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[14] AstroGrid. The AstroGrid Project. `http://www.astrogrid.org/`. Online, last accessed on December 12, 2016.

[15] S. Azodolmolky, P. Wieder, and R. Yahyapour. Cloud Computing Networking: Challenges and Opportunities for Innovations. *IEEE Communications Magazine*, 51(7):54–62, 2013.

[16] H. Bauke and S. Mertens. *Cluster Computing*. Springer, 2006.

[17] T. Baur. Functional Analysis and Architecture for Interoperable and DVO-specific Grid Monitoring Services. In *Proceedings of the Fourth IEEE International Conference on eScience (eScience 2008)*. IEEE Computer Society Press, 2008.

[18] T. Baur, R. Breu, T. Kálmán, T. Lindinger, A. Milbert, G. Poghosyan, and M. Romberg. Adopting GLUE 2.0 for an Interoperable Grid Monitoring System. *Cracow Grid Workshop CGW08*, Mar 2009.

[19] T. Baur and S. Saad. Customer Service Management for Grid Monitoring and Accounting Data. In *Proceedings of the 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2007)*. Springer, 2007.

[20] T. Baur, O. v.d.g. Felde, and H. Reiser. Konzepte zum Monitoring im D-Grid (D-Grid Report). `http://www.d-grid.de/uploads/media/mab_monitoring_konzepte.pdf`, 2007. Online, last accessed on December 12, 2016.

[21] T. Baur, R. Breu, T. Kálmán, T. Lindinger, A. Milbert, G. Poghosyan, H. Reiser, and M. Romberg. An Interoperable Grid Information System for Integrated Resource Monitoring based on Virtual Organizations. *Journal of Grid Computing*, 7(3):319–333, Sep 2009.

[22] T. Baur, T. Kálmán, T. Lindinger, A. Milbert, G. Poghosyan, and M. Romberg. Middleware-übergreifendes Monitoring: Evaluierung und Auswahl von Komponenten, 2008. D-Grid Report, 2008.

[23] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. GENI: A federated testbed for innovative network experiments. *Computer Networks*, 61(0):5 – 23, 2014. Special issue on Future Internet Testbeds – Part I.

[24] D. Bernholdt, S. Bharathi, D. Brown, K. Chanchio, M. Chen, A. Chervenak, L. Cinquini, B. Drach, I. Foster, P. Fox, et al. The Earth System Grid: Supporting the Next Generation of Climate Modeling Research. *Proceedings of the IEEE*, 93(3):485–495, 2005.

[25] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic Schema Matching, Ten Years Later. *PVLDB*, 4(11):695–701, 2011.

[26] D. Berry, A. Luniewski, and M. Antonioletti. OGSA Data Architecture. Technical Report GDF.121, Global Grid Forum, December 2007.

[27] T. Blanke, C. Kristel, and L. Romary. Crowds for Clouds: Recent Trends in Humanities Research Infrastructures. In A. Benardou, E. Champion, C. Dallas, and L. Hughes, editors, *Cultural Heritage Digital Tools and Infrastructures*. 2016.

[28] C. Boehme, T. Ehlers, J. Engelhardt, A. Félix, O. Haan, T. Kálmán, B. Neumair, U. Schwardmann, and D. Sommerfeld. Instant-Grid: Fully Automated Middleware-Deployment Using a Live-CD. In *Proceedings in International Conference on Networking and Services (ICNS'06)*, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[29] C. Boehme, T. Ehlers, J. Engelhardt, A. Félix, O. Haan, T. Kálmán, U. Schwardmann, D. Sommerfeld, and A. Willner. Grid-Demonstration und Schulung mit Instant-Grid, 2007. 21. DFN-Jahrestagung.

[30] C. Boehme, T. Ehlers, J. Engelhardt, A. Félix, O. Haan, T. Kálmán, U. Schwardmann, D. Sommerfeld, and A. Willner. Instant-Grid - A Toolkit for Demonstration, Test and Development of Grid-Infrastructure. In *German e-Science Conference*, 2007.

[31] A. Bonifati, G. Mecca, P. Papotti, and Y. Velegrakis. Discovery and Correctness of Schema Mapping Transformations. In Z. Bellahsene, A. Bonifati, and E. Rahm, editors, *Schema Matching and Mapping*, Data-Centric Systems and Applications, pages 111–147. Springer, 2011.

[32] M. Breslin. Data Warehousing Battle of Giants: Comparing the Basics of the Kimball and Inmon Models. *Business Intelligence Journal*, 9(4), 2004.

[33] J. Buford, H. Yu, and E. K. Lua. *P2P Networking and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[34] S. Burke. EGI profile for the use of the GLUE 2.0 Information Schema. `http://go.egi.eu/glue2-profile`, 2013. Online, last accessed on December 12, 2016.

[35] J. Butler, J. Lambea, M. Nolan, W. Theilmann, F. Torelli, R. Yahyapour, A. Chiasera, and M. Pistore. SLAs Empowering Services in the Future Internet. pages 327–338. Springer-Verlag, Berlin, Heidelberg, 2011.

[36] S. Buxton, L. Fryman, R. H. Güting, T. A. Halpin, J. L. Harrington, W. H. Inmon, S. Lightstone, J. Melton, T. Morgan, T. P. Nadeau, B. O'Neil, E. J. O'Neil, P. E. O'Neil, M. S. 0001, G. Simsion, T. J. Teorey, and G. Witt. *Database Design - Know It All*. Morgan Kaufmann, 2008.

[37] R. Buyya, R. Ranjan, and R. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In C.-H. Hsu, L. Yang, J. Park, and S.-S. Yeo, editors, *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 13–31. Springer Berlin Heidelberg, 2010.

[38] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing As the 5th Utility. *Future Generation Computing Systems*, 25(6):599–616, June 2009.

[39] C3Grid. Collaborative Climate Community Data and Processing Grid. `https://verc.enes.org/c3web/`. Online, last accessed on December 12, 2016.

[40] caGrid. Cancer Grid. `http://www.cagrid.org/`. Online, last accessed on December 12, 2016.

[41] N. Carr. *The Big Switch: Rewiring the World, from Edison to Google*. W. W. Norton & Company, 2008.

[42] H. Casanova. Distributed Computing Research Issues in Grid Computing. *SIGACT News*, 33(3):50–70, September 2002.

[43] K. Chard, J. Pruyne, B. Blaiszik, R. Ananthakrishnan, S. Tuecke, and I. Foster. Globus Data Publication as a Service: Lowering Barriers to Reproducible Science. In *11th IEEE International Conference on eScience*, 2015.

[44] A. Chatr-aryamontri, B.-J. Breitkreutz, R. Oughtred, L. Boucher, S. Heinicke, D. Chen, C. Stark, A. Breitkreutz, N. Kolas, L. O'Donnell, T. Reguly, J. Nixon, L. Ramage, A. G. Winter, A. Sellam, C. Chang, J. E. Hirschman, C. L. Theesfeld, J. M. Rust, M. S. Livstone, K. Dolinski, and M. Tyers. The BioGRID Interaction Database: 2015 Update. *Nucleic Acids Research*, 43(Database-Issue):470–478, 2015.

[45] X. Chem, A. Khan, J. Ainsworth, S. Newhouse, and J. MacLaren. WSI Resource Usage Service (RUS) Core Specification. Technical report, Open Grid Forum, March 2007.

[46] G. Chun, H. Dail, H. Casanova, and A. Snavely. Benchmark Probes for Grid Assessment. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium - Workshop 17*, Santa Fe, New Mexico, April 2004. IEEE Computer Society Press. High-Performance Grid Computing Workshop.

[47] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero-Merino, L. M. Vaquero, K. Nagin, and B. Rochwerger. Monitoring Service Clouds in the Future Internet. In *Future Internet Assembly*, pages 115–126, 2010.

[48] S. Clayman, A. Galis, G. Toffetti, L. Vaquero, B. Rochwerger, and P. Massonet. Future Internet Monitoring Platform for Computing Clouds. In E. Nitto and R. Yahyapour, editors, *Towards a Service-Based Internet*, volume 6481 of *Lecture Notes in Computer Science*, pages 215–217. Springer Berlin Heidelberg, 2010.

[49] A. Cooke, A. J. G. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Byrom, L. Field, S. Hicks, J. Leake, M. Soni, A. Wilson, R. Cordenonsi, L. Cornwall, A. Djaoui, S. Fisher, N. Podhorszki, B. Coghlan, S. Kenny, and D. OrsquoCallaghan. R-GMA: An Information Integration System for Grid Monitoring. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *LNCS*, pages 462 – 481. Springer, October 2003.

[50] F. J. Corbató and V. A. Vyssotsky. Introduction and Overview of the Multics System. In *Fall Joint Computer Conference*, AFIPS'65, pages 185–196, New York, NY, USA, 1965. ACM.

[51] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed Systems: Concepts and Design (5th Edition)*. Addison Wesley, 2011.

[52] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627, IETF, July 2006.

[53] L. C. Crosswell and J. M. Thornton. ELIXIR: a Distributed Infrastructure for European Biological Data. *Trends in Biotechnology*, 30(5):241–242, 2012.

[54] D. Lorenz and S. Borovac and P. Buchholz and H. Eichenhardt and T. Harenberg and P. Mättig and M. Mechtel and R. Müller-Pfefferkorn and R. Neumann and K. Reeves and Ch. Uebing and W. Walkowiak and Th. William and R. Wismüller. Job monitoring and steering in D-Grid's High Energy Physics Community Grid. *Future Generation Computer Systems*, 25(3):308 – 314, 2009.

[55] D-MON Project. Horizontal Integration of Resource- and Service Monitoring in D-Grid. `http://www.d-grid.de/index.php?id=401&L=1`. Online, last accessed on December 12, 2016.

[56] DARIAH-DE Project. Authentication and Authorization Infrastructure. `https://wiki.de.dariah.eu/display/publicde/DARIAH+AAI+Documentation`. Online, last accessed on December 12, 2016.

[57] DataTAG. Research & Technological Development for a Data TransAtlantic Grid. `http://datatag.web.cern.ch/`. Online, last accessed on December 12, 2016.

[58] S. B. Davidson and A. Kosky. Specifying Database Transformations in WOL. *IEEE Data Eng. Bull.*, 22(1):25–30, 1999.

[59] DEISA. Distributed European Infrastructure for Supercomputing Applications. `http://www.deisa.eu/`. Online, last accessed on December 12, 2016.

[60] E. F. del Castillo, D. Scardaci, and Álvaro López García. The EGI Federated Cloud e-Infrastructure. *Procedia Computer Science*, 68:196 – 205, 2015. 1st International Conference on Cloud Forward: From Distributed to Complete Computing.

[61] Y. Demchenko, C. de Laat, O. Koeroo, and D. Groep. Re-thinking Grid Security Architecture. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 79–86, Dec 2008.

[62] Y. Demchenko, C. de Laat, and V. Ciaschini. VO-based dynamic security associations in collaborative grid environment. In *CTS '06: Proceedings of the International Symposium on Collaborative Technologies and Systems*, pages 38–47, Washington, DC, USA, 2006. IEEE Computer Society.

[63] Y. Demchenko, C. Ngo, C. de Laat, J. A. Garcia-Espin, S. Figuerola, J. Rodriguez, L. M. Contreras, G. Landi, and N. Ciulli. Intercloud Architecture Framework for Heterogeneous Cloud Based Infrastructure Services Provisioning On-Demand. *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, 0:777–784, 2013.

[64] I. Díaz, G. Fernández, P. González, M. J. Martín, and J. Touriño. Extending the Globus Information Service with the Common Information Model. In *IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 113–119, 2011.

[65] I. Díaz, G. Fernández, M. J. Martín, P. González, and J. Touriño. Integrating the Common Information Model with MDS4. In *GRID*, pages 298–303, 2008.

[66] O. Diaz and J. J. Rodriguez. Portlets as Web Components: an Introduction. *Journal of Universal Computer Science*, 10(4):454–472, 2004.

[67] F. Dickmann, J. Falkner, W. Gunia, J. Hampe, M. Hausmann, A. Herrmann, N. Kepper, T. A. Knoch, S. Lauterbach, J. Lippert, K. Peter, E. Schmitt, U. Schwardmann, J. Solodenko, D. Sommerfeld, T. Steinke, A. Weisbecker, and U. Sax. Solutions for Biomedical Grid Computing-Case Studies from the D-Grid Project Services@MediGRID. *Journal of Computational Science*, In Press, Accepted Manuscript, 2011.

[68] Distributed Management Task Force. Cloud Management Standards. `http://www.dmtf.org/standards/cloud`. Online, last accessed on December 12, 2016.

[69] A. Duarte, P. Nyczyk, A. Retico, and D. Vicinanza. Monitoring the EGEE/WLCG grid services. *Journal of Physics: Conference Series*, 119(5):052014 (10pp), 2008.

[70] S. A. (Ed.), S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, and J. Navarro. GLUE 2.0 Specification V2.0. Technical report, Global Grid Forum, May 2008.

[71] EDG. European Data Grid project. `http://eu-datagrid.web.cern.ch/`. Online, last accessed on December 12, 2016.

[72] EGI. European Grid Initiative. `http://www.egi.eu/`. Online, last accessed on December 12, 2016.

[73] M. Ellert, M. Gronager, A. Konstantinov, B. Konya, J. Lindemann, I. Livenson, J. Nielsen, M. Niinimaki, O. Smirnova, and A. Waananen. Advanced Resource Connector middleware for lightweight computational Grids. *Future Generation Computer Systems*, 23(2):219–240, February 2007.

[74] C. Ernemann and R. Yahyapour. Applying Economic Scheduling Methods to Grid Environments. In J. Nabrzyski, J. Schopf, and J. Węglarz, editors, *Grid Resource Management*, volume 64 of *International Series in Operations Research and Management Science*, pages 491–506. Springer US, 2004.

[75] Y. Etsion and D. Tsafrir. A Short Survey of Commercial Cluster Batch Schedulers. *School of Computer Science and Engineering, The Hebrew University of Jerusalem*, 44221:2005–13, 2005.

[76] FIA. Future Internet Assembly. `http://www.future-internet.eu/home/future-internet-assembly.html`. Online, last accessed on December 12, 2016.

[77] L. Field and M. Schulz. Grid Deployment Experiences: The Path to a Production Quality LDAP Based Grid Information System. In *Computing in High Energy Physics and Nuclear Physics. Proceedings, Conference, CHEP'04, Interlaken, Switzerland, September 27-October 1, 2004*, pages 723–726, 2005.

[78] T. Fieseler and W. Gürich. Development and Operation of the D-Grid Infrastructure. In *Production Grids in Asia*, pages 199–211. Springer US, 2009.

[79] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In H. Jin, D. A. Reed, and W. Jiang, editors, *IFIP International Conference on Network and Parallel Computing (NPC05)*, volume 3779 of *Lecture Notes in Computer Science*, pages 2 – 13. Springer, 2005.

[80] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Open Grid Service Infrastructure WG*. Global Grid Forum, 2002.

[81] I. Foster, T. Maguire, and D. Snelling. OGSA WSRF Basic Profile 1.0. Technical Report GDF.72, Global Grid Forum, May 2006.

[82] I. Foster. The Grid: A New Infrastructure for 21st Century Science. *Physics Today*, 55(2):42–47, 2002.

[83] I. Foster. What is the Grid? A Three Point Checklist. *Grid Today*, 1(6):22, 2002.

[84] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[85] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[86] I. Foster, V. Vasiliadis, and S. Tuecke. Software as a Service as a Path to Software Sustainability. Technical report, 2013.

[87] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Proceedings of Grid Computing Environments Workshop*, number 5, pages 1–10. IEEE, 2008.

[88] B. Fritzsch, J. Falkner, P. Gietz, S. Pinkernell, M. Haase, F. Dickmann, M. Quade, and F. Viezens. Nutzung von kurzlebigen Zertifikaten in portalbasierten Grids (GapSLC). *Zusammenfassender Abschlussbericht zum BMBF-Projekt, FKZ 01IG09003 AF*, 2011.

[89] M. Frumkin and R. F. Van der Wijngaart. NAS Grid Benchmarks: A Tool for Grid Space Exploration. *Cluster Computing*, 5:247–255, July 2002.

[90] P. Fuhrmann. dCache, the Overview. `http://www.dcache.org/manuals/dcache-whitepaper-light.pdf`, last accessed September 2008. White Paper, Online, last accessed on December 12, 2016.

[91] A. Galis, A. Gavras, F. Álvarez, A. Bassi, M. Bezzi, L. Ciavaglia, F. Cleary, P. Daras, H. D. Meer, P. Demestichas, J. Domingue, T. G. Kanter, S. Karnouskos, S. Krčo, L. Lefevre, J. Lentjes, M.-S. Li, P. Malone, A. Manzalini, V. Lotz, H. Müller, K. Oberle, N. E. O'Connor, N. Papanikolaou, D. Petcu, R. Rahmani, D. Raz, G. Richards, E. Salvadori, S. Sargento, H. Schaffers, J. Serrat, B. Stiller, A. F. Skarmeta, K. Tutschku, and T. Zahariadis, editors. *The Future Internet – Future Internet Assembly 2013: Validated Results and New Horizons*, volume 7858 of *Lecture Notes in Computer Science*. Springer, May 2013.

[92] Ganglia. The Ganglia Monitoring System Project. `http://ganglia.sourceforge.net/`. Online, last accessed on December 12, 2016.

[93] N. gentschen Felde, T. Baur, M. Garschammer, and H. Reiser. Anforderungen an das Monitoring, Ergebnisse aus den Erhebung bei den Communities und Ressourcenanbietern im D-Grid. In C.-P. Ruckemann, editor, *Ergebnisse der Studie und Anforderungsanalyse in den Fachgebieten Monitoring, Accounting, Billing bei den Communities und Ressourcenanbietern im D-Grid*, pages 45–63. Fachgebiete Monitoring, Accounting und Billing im D-Grid-Integrationsprojekt, 2006.

[94] W. Gentzsch, D. Girou, A. Kennedy, H. Lederer, J. Reetz, M. Riedel, A. Schott, A. Vanni, M. Vazquez, and J. Wolfrat. DEISA - Distributed European Infrastructure for Supercomputing Applications. *Journal of Grid Computing*, 9(2):259–277, 2011.

[95] GeoMaint. Informationprovider (Sensor) for Geolocation and Maintenance Data. `http://geomaint.sourceforge.net/`. Online, last accessed on December 12, 2016.

[96] P. Gietz, A. Aschenbrenner, S. Budenbender, F. Jannidis, M. W. Kuster, C. Ludwig, W. Pempe, T. Vitt, W. Wegstein, and A. Zielinski. TextGrid and eHumanities. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '06, pages 133–, Washington, DC, USA, 2006. IEEE Computer Society.

[97] gLite. Lightweight Middleware for Grid Computing. `http://glite.cern.ch/`, Online, last accessed on December 12, 2016.

[98] S. Gnanasundaram and A. Shrivastava, editors. *Information Storage and Management: Storing, Managing, and Protecting Digital Information in Classic, Virtualized, and Cloud Environments*. Wiley, Indianapolis, Indiana, USA, 2nd edition, 2012.

[99] GOCDB. Grid Operations Centre DataBase. `http://goc.egi.eu/`, Online, last accessed on December 12, 2016.

[100] Google Inc. Google App Engine. `http://appengine.google.com/`. Online, last accessed on December 12, 2016.

[101] S. Granneman. *Hacking Knoppix*. ExtremeTech. Wiley, 2006.

[102] H. Hahn, T. Kálmán, W. Kolbmann, T. Kollatz, M. Neuschäfer, S. Pielstroem, J. Puhl, J. Stiller, and D. Tonne. *Handbuch Digital Humanities*. DARIAH-DE, 1 edition, 2015.

[103] J. Hall. *Mastering SaltStack*. Community experience distilled. Packt Publishing Limited, Birmingham, UK, 2015.

[104] A. Hardisty and D. Roberts. A Decadal View of Biodiversity Informatics: Challenges and Priorities. *BMC Ecology*, 13(1):1–23, 2013.

[105] M. Hardt, A. Hayrapetyan, P. Millar, and S. Memon. Combining the X.509 and the SAML Federated Identity Management Systems. In G. Martínez Pérez, S. Thampi, R. Ko, and L. Shu, editors, *Recent Trends in Computer Networks and Distributed Systems Security*, volume 420 of *Communications in Computer and Information Science*, pages 404–415. Springer Berlin Heidelberg, 2014.

[106] H. Harmsen, T. Kálmán, and E. Wandl-Vogt. DARIAH meets EGI. *Inspired, 19*, page 8, April 2015.

[107] A. Harutyunyan, P. Buncic, T. Freeman, and K. Keahey. Dynamic virtual AliEn Grid sites on Nimbus with CernVM. *Journal of Physics: Conference Series*, 219(7):072036, 2010.

[108] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems: Concepts, Architectures, and Their Operational Application*. Morgan Kaufmann, 1999.

[109] O. Hinz, B. Skiera, R. Beck, and W. König. *Grid Computing in der Finanzindustrie: Ein Herausgeberband des D-Grid-Projekts FinGrid*. 2009.

[110] L. Hochstein. *Ansible: Up and Running*. O'Reilly Media, 2014.

[111] J. A. Hoffer, R. Venkataraman, and H. Topi. *Modern Database Management*. Prentice Hall Press, Upper Saddle River, NJ, USA, 11th edition, 2012.

[112] M. Högqvist, T. Röblitz, and A. Reinefeld. Stellaris: An RDF-based information service for AstroGrid-D. In *German e-Science Conference*, 2007.

[113] A. Hoheisel. Grid Workflow Execution Service - User Manual. Technical report, Fraunhofer FIRST/K-Wf Grid Project, 2005.

[114] A. Hoheisel. Grid Workflow Execution Service - Dynamic and Interactive Execution and Visualization of Distributed Workflows. In *Proceedings of the Cracow Grid Workshop*, volume 2, pages 13–24, 2006.

[115] K. Hwang, G. C. Fox, and J. J. Dongarra. *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann, 2012.

[116] A. Iamnitchi, P. Trunfio, J. Ledlie, and F. Schintke. Peer-to-Peer Computing. In P. D'Ambra, M. Guarracino, and D. Talia, editors, *Euro-Par 2010 - Parallel Processing*, volume 6271 of *Lecture Notes in Computer Science*, pages 444–445. Springer Berlin Heidelberg, 2010.

[117] IEEE Computer Society, Adaptive Management of Cloud Computing Environments Working Group. P2303 - Standard for Adaptive Management of Cloud Computing Environments. `http://standards.ieee.org/develop/project/2303.html`. Online, last accessed on December 12, 2016.

[118] E. Imamagic and D. Dobrenic. Grid infrastructure monitoring system based on Nagios. In *GMW '07: Proceedings of the 2007 workshop on Grid monitoring*, pages 23–28, New York, NY, USA, 2007. ACM.

[119] Instant-Grid. The Instant-Grid Project. `http://www.instant-grid.org/?q=en`. Online, last accessed on December 12, 2016.

[120] International Telecommunication Union. Recommendation X.1255: Framework for Discovery of Identity Management Information. Technical report, ITU-T, Sep 2013.

[121] ITU-T SG13. ITU-T Study Group 13 on Future networks including cloud computing, mobile and next-generation networks. `http://www.itu.int/en/ITU-T/about/groups/Pages/sg13.aspx`. Online, last accessed on December 12, 2016.

[122] iVDGL. international Virtual Data Grid Laboratory. `http://igoc.ivdgl.indiana.edu/`. Online, last accessed on December 12, 2016.

[123] Jawari. Grid Benchmarking. `http://www.jawari.net/`. Online, last accessed on December 12, 2016.

[124] K. G. Jeffery and D. Bailo. EPOS: Using Metadata in Geoscience. In S. Closs, R. Studer, E. Garoufallou, and M.-A. Sicilia, editors, *Metadata and Semantics Research: 8th Research Conference*, pages 170–184, Cham, 2014. Springer International Publishing.

[125] K. G. Jeffery, D. Kyriazis, and L. Schubert. Beyond Cloud Computing: Towards Complete Computing. `http://www.holacloud.eu/wp-content/uploads/2016/03/HOLACloud_Roadmap2015.pdf`, 2015. Online, last accessed on December 12, 2016.

[126] X. Jin and J. Liu. From Individual Based Modeling to Autonomy Oriented Computation. In *Agents and Computational Autonomy*, pages 151–169. Springer, 2004.

[127] T. Kálmán. Reliable Grid Information Database (RGID) and the Jawari Interface, May 2008. 3rd D-Grid Monitoring Workshop.

[128] T. Kálmán. [D]-Grid Resource Definition Language (D-GRDL) and the Reliable Grid Resource Database, Nov 2007. 2nd D-Grid Monitoring Workshop.

[129] T. Kálmán. Extension of a Traditional Monitoring Framework to Support Efficient Monitoring of Stateful Grid Services, International Conference on Computing in High Energy and Nuclear Physics (short paper). Oct 18-22 2010, Taipei.

[130] T. Kálmán. Überwachung von Globus Toolkit v4.x Diensten mit Nagios, Nov 2009. 6th D-Grid Monitoring Workshop, Nov 2009.

[131] T. Kálmán. Information about the Grid Environment at GoeGrid, Mar 2010. D-Grid All-Hands Meeting, Mar 2010.

[132] T. Kálmán. Assessment of Resource Quality for Service Level Agreements in Life Science Grids. In *7th IEEE International Conference on e-Science: Workshop on Computing Advances in Life Science (CALS2011)*, pages 170–177, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[133] T. Kálmán, X. Kong, and U. Schwardmann. Die digitale Forschungsinfrastruktur DARIAH-DE: Angebotspalette für die Geistes- und Kulturwissenschaften. *Bibliothek Forschung und Praxis*, 40(2):234–243, July 2016.

[134] T. Kálmán, D. Kurzawe, and U. Schwardmann. European Persistent Identifier Consortium - PIDs für die Wissenschaft. In R. Altenhöner and C. Oellers, editors, *Langzeitarchivierung von Forschungsdaten - Standards und disziplinspezifische Lösungen*, pages 151–168, Berlin, 2012. SCIVERO Verlag.

[135] T. Kálmán and T. Rings. A UNICORE-based Multi Site and Multi User Grid Environment for Demonstration, Education, and Testing Purposes. In *Proceedings of the UNICORE Summit 2010*, volume 5, pages 1–10. IAS Series, May 2010.

[136] T. Kálmán, D. Tonne, and O. Schmitt. Sustainable Preservation for the Arts and Humanities. *New Review of Information Networking*, 20(1-2):123–136, 2015.

[137] T. Kálmán and E. Wandl-Vogt. DARIAH-ERIC: Towards a Sustainable, Social and Technical European e-Research Infrastructure for the Arts and Humanities. [Abstract], e-Infrastructure Reflection Group (e-IRG) Workshop, November 2014, Rome, Italy.

[138] G. Katsaros, R. Kubert, and G. Gallizo. Building a Service-Oriented Monitoring Framework with REST and Nagios. *2013 IEEE International Conference on Services Computing*, pages 426–431, 2011.

[139] K. Keahey and T. Freeman. Contextualization: Providing One-Click Virtual Clusters. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ESCIENCE '08, pages 301–308, Washington, DC, USA, 2008. IEEE Computer Society.

[140] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes. Sky Computing. *IEEE Internet Computing*, 13(5):43–51, September 2009.

[141] R. Kettimuthu, L. Wantao, F. Siebenlist, and I. Foster. Communicating Security Assertions over the GridFTP Control Channel. *IEEE 9th International Conference on e-Science*, 0:426–427, 2008.

[142] L. Kleinrock. An Internet Vision: The Invisible Global Infrastructure. *Ad Hoc Networks*, 1(1):3–11, 2003.

[143] D. Krefting, J. Bart, K. Beronov, O. Dzhimova, J. Falkner, M. Hartung, A. Hoheisel, T. A. Knoch, T. Lingner, Y. Mohammed, K. Peter, E. Rahm, U. Sax, D. Sommerfeld, T. Steinke, T. Tolxdorff, M. Vossberg, F. Viezens, and A. Weisbecker. MediGRID: Towards a User Friendly Secured Grid Infrastructure. In T. Solomonides, I. Blanquer, V. Breton, T. Glatard, and Y. Legre, editors, *Healthgrid Applications and Core Technologies: Proceedings of HealthGrid 2010*. IOS Press, 2010.

[144] D. Krefting, S. Canisius, A. Hoheisel, H. Loose, T. Tolxdorff, and T. Penzel. Grid Based Sleep Research - Analysis of Polysomnographies Using a Grid Infrastructure. *Future Gener. Comput. Syst.*, 29(7):1671–1679, September 2013.

[145] S. Krum, W. V. Hevelingen, B. Kero, J. Turnbull, and J. McCune. *Pro Puppet*. Apress, Berkely, CA, USA, 2nd edition, 2013.

[146] M. Kunze and G. Pogoshyan. Monitoring for Multi-Middleware Grid. In *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008*. IEEE Computer Society Press, April 2008.

[147] M. S. L. Field. Grid Interoperability: The Interoperations Cookbook. *Journal of Physics: Conference Series*, 119(1), 2008.

[148] T. Lindinger and T. Kálmán. D-MON: Site und System Monitoring im D-Grid, May 2009. 5th D-Grid Monitoring Workshop.

[149] R. Mach, R. Lepro-Metz, and S. Jackson. Usage Record Format Recommendation. Technical report, Open Grid Forum, September 2006.

[150] J. Mancini. The Challenge of Information Chaos: 34 Immediate Actions to Meet the Business Challenge of the Century. *Association for Information and Image Management (AIIM) White Paper*, 2014.

[151] M. Marschall. *Chef Infrastructure Automation Cookbook - Second Edition*. Quick answers to common problems. Packt Publishing, 2015.

[152] M. Marzolla, P. Andreetto, V. Venturi, A. Ferraro, A. S. Memon, M. S. Memon, B. Twedell, M. Riedel, D. Mallmann, A. Streit, S. van de Berghe, V. Li, D. Snelling, K. Stamou, Z. A. Shah, and F. Hedman. Open Standards-Based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 592–601, Washington, DC, USA, 2007. IEEE Computer Society.

[153] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation And Experience. *Parallel Computing*, 30(7), July 2004.

[154] G. Mathieu, D. A. Richards, D. J. Gordon, C. D. C. Novales, P. Colclough, and M. Viljoen. GOCDB, a Topology Repository for a Worldwide Grid Infrastructure. *Journal of Physics: Conference Series*, 219(6):062021, 2010.

[155] P. Mell and T. Grance. The NIST Definition of Cloud Computing. In *National Institute of Standards and Technology (NIST)*, volume 53, page 50, 2009.

[156] A. S. Memon, M. S. Memon, P. Wieder, and B. Schuller. CIS: An Information Service based on the Common Information Model. In *Proceedings of 3rd IEEE International Conference on e-Science and Grid Computing*, pages 465 – 472. IEEE Computer Society, December 2007.

[157] M. S. Memon. *A Reliable Grid Information Service Using a Unified Information Model*. Diplom (fh), Techn. Hochsch. Aachen, Jülich, 2008.

[158] F. Michel. Using VAPOR to Improve VO Administration and Operations, 2014. EGI Community Forum 2014.

[159] Microsoft Corporation. Office 365. `http://office.microsoft.com/`. Online, last accessed on December 12, 2016.

[160] Microsoft Corporation. Windows Azure. `http://www.microsoft.com/windowsazure/`. Online, last accessed on December 12, 2016.

[161] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical report, 2002.

[162] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Key Challenges in Cloud Computing: Enabling the Future Internet of Services. *IEEE Internet Computing*, 17(4):18–25, 2013.

[163] Nagios. Plug-in Development Guidelines. `http://nagiosplug.sourceforge.net/developer-guidelines.html`. Online, last accessed on December 12, 2016.

[164] H. Neuroth, M. Kerzel, and W. Gentzsch, editors. *German Grid Initiative D-Grid*. Universitätsverlag Göttingen, 2007.

[165] J. Novotny, M. Russell, and O. Wehrens. GridSphere: A Portal Framework for Building Collaborations. *Concurrency And Computation-Practice & Experience*, 16(5):503 – 513, 2004.

[166] J. Novotny, S. Tuecke, and V. Welch. An Online Credential Repository for the Grid: MyProxy. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 104–111, 2001.

[167] OCC. Open Commons Consortium. `http://occ-data.org/`. Online, last accessed on December 12, 2016.

[168] M. T. Oezsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer, New York, USA, 3rd edition, 2011.

[169] OGF. Open Grid Forum. `http://www.ogf.org`. Online, last accessed on December 12, 2016.

[170] Open Grid Forum. GLUE Working Group. `http://redmine.ogf.org/dmsf/glue-wg`. Online, last accessed on December 12, 2016.

[171] Open Grid Forum. Open Cloud Computing Interface. `http://occ-data.org/`. Online, last accessed on December 12, 2016.

[172] OSG. Open Science Grid. `http://www.opensciencegrid.org/`. Online, last accessed on December 12, 2016.

[173] L. Otero-Cerdeira, F. J. Rodríguez-Martínez, and A. Gómez-Rodríguez. Ontology Matching: A Literature Review. *Expert Systems with Applications*, 42(2):949–971, 2015.

[174] M. P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, WISE '03, pages 3–12, Washington, DC, USA, 2003. IEEE Computer Society.

[175] D. F. Parkhill. *The challenge of the computer utility*. Addison-Wesley Pub. Co Reading, Mass, 1966.

[176] D. Petcu, C. Craciun, M. Neagul, S. Panica, B. D. Martino, S. Venticinque, M. Rak, and R. Aversa. Architecturing a Sky Computing Platform. volume 6569 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2010.

[177] G. Pfister. *In Search of Clusters*. Parallel Programming Computer Architecture. Prentice Hall PTR, 1998.

[178] G. Poghosyan and M. Kunze. Monitoring for Multi-Middleware Grid. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, 14-18 April*, Miami, Florida, USA, 2008. IEEE.

[179] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, and L. Foschini. DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds. *Future Generation Computer Systems*, 29(8):2041 – 2056, 2013.

[180] PRACE. Partnership for Advanced Computing in Europe. `http://www.prace-ri.eu/`. Online, last accessed on December 12, 2016.

[181] A. Pras and J. Schönwälder. On the Difference between Information Models and Data Models. Internet RFC 3444, January 2003.

[182] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, IRVINE, 2000.

[183] Rackspace Inc. Rackspace. `http://www.rackspace.com/`. Online, last accessed on December 12, 2016.

[184] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The Very Large Data Bases Journal (VLDB)*, 10(4):334–350, 2001.

[185] K. Rankin. *Knoppix Hacks: Tips and Tools for Hacking, Repairing, and Enjoying Your PC*. O'Reilly Media, 2007.

[186] J. Rao and X. Su. A Survey of Automated Web Service Composition Methods. In *LNCS*, volume 3387/2005, pages 43–54. Springer, 2005.

[187] B. P. Rimal, E. Choi, and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. In *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, NCM '09, pages 44–51, Washington, DC, USA, 2009. IEEE Computer Society.

[188] T. Rings, A. Aschenbrenner, J. Grabowski, T. Kálmán, G. Lauer, J. Meyer, A. Quadt, U. Sax, and F. Viezens. An Interdisciplinary Practical Course on the Application of Grid Computing. In *Proceedings of the 1st Annual IEEE Engineering Education Conference (EDUCON 2010)*, pages 149–155, 2010. April 14-16 2010, Madrid, Spain.

[189] T. Rings and J. Grabowski. Pragmatic Integration of Cloud and Grid Computing Infrastructures. In *Proceedings of the 5th International Conference on Cloud Computing (CLOUD 2012), June 24-29 2012 , Honolulu, Hawaii, USA*, pages 710–717. IEEE, June 2012.

[190] T. Rings, H. Neukirchen, and J. Grabowski. Testing Grid Application Workflows Using TTCN-3. In *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation (ICST 2008). Lillehammer, Norway*, pages 210–219. IEEE Computer Society, 2008.

[191] P. Riteau. Building Dynamic Computing Infrastructures over Distributed Clouds. In *IEEE First Symposium on Network/Cloud Computing and Applications (NCCA 2011)*, pages 127–130, 2011.

[192] L. Robertson. *From the Web to the Grid and Beyond: Computing Paradigms Driven by High-Energy Physics*, chapter Computing Services for LHC: From Clusters to Grids, pages 69–89. The Frontiers Collection. Springer Berlin Heidelberg, 2012.

[193] B. Rochwerger, C. Vazquez, D. Breitgand, D. Hadas, M. Villari, P. Massonet, E. Levy, A. Galis, I. Llorente, R. Montero, Y. Wolfsthal, K. Nagin, L. Larsson, and F. Galan. An Architecture for Federated Cloud Computing. In *Cloud Computing: Principles and Paradigms*, chapter 15, pages 393–411. Wiley, February 2011.

[194] M. Russell, P. Dziubecki, P. Grabowski, M. Krysinśki, T. Kuczyński, D. Szjenfeld, D. Tarnawczyk, G. Wolniewicz, and J. Nabrzyski. The Vine Toolkit: A Java Framework for Developing Grid Applications. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 4967 of *Lecture Notes in Computer Science*, pages 331–340. Springer Berlin Heidelberg, 2008.

[195] N. Sadashiv and S. M. D. Kumar. Cluster, Grid and Cloud Computing: A Detailed Comparison. *Proceedings of the 6th International Conference on Computer Science Education (ICCSE)*, pages 477–482, 2011.

[196] Salesforce Corporation. Heroku. `https://www.heroku.com/`. Online, last accessed on December 12, 2016.

[197] L. F. Sarmenta. Bayanihan: Web-Based Volunteer Computing Using Java. In *Second International Conference on World-Wide Computing and its Applications*, pages 444–461, 1998.

[198] S. Scholz. *Geschäftsmodelle für Grid Computing in der Medizin und der Biomedizin*. PhD thesis, University of Hanover, 2010.

[199] J. Schopf, I. Raicu, L. Perlman, N. Miller, C. Kesselman, I. Foster, and M. D'Arcy. Monitoring and Discovery in a Web Services Framework: Functionality and Performance of Globus Toolkit MDS4. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, Los Alamitos, 2006. IEEE Computer Society Press.

[200] J. M. Schopf, L. Pearlman, N. Miller, C. Kesselman, I. Foster, M. D'Arcy, and A. Chervenak. Monitoring the Grid with the Globus Toolkit MDS4. *Journal of Physics Conference Series*, 46:521–525, September 2006.

[201] F. J. Seinstra, J. Maassen, R. V. van Nieuwpoort, N. Drost, T. van Kessel, B. van Werkhoven, J. Urbani, C. Jacobs, T. Kielmann, and H. E. Bal. Jungle Computing: Distributed Supercomputing Beyond Clusters, Grids, and Clouds. In M. Cafaro and G. Aloisio, editors, *Grids, Clouds and Virtualization*, page 167, 2011.

[202] X. U. Shukla, P. Parsania, and K. C. Kamani. Applications of Grid Computing in Agriculture: An Indian Scenario. *Gujarat Journal of Extension Education*, page 58, 2014.

[203] P. Shvaiko and J. Euzenat. A Survey of Schema-Based Matching Approaches. *Journal on Data Semantics*, 4:146–171, 2005.

[204] P. Shvaiko and J. Euzenat. Ontology Matching: State of the Art and Future Challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):158–176, 2013.

[205] S. Sild, U. Maran, A. Lomaka, and M. Karelson. Open Computing Grid for Molecular Science and Engineering. *Journal of Chemical Information and Modeling*, 46(3):953–959, 2006.

[206] Y.-L. Simmhan, B. Plale, and D. Gannon. A Survey of Data Provenance Techniques. Technical Report TR-618, Computer Science Department, Indiana University, Bloomington, 2005.

[207] G. C. Simsion and G. C. Witt. *Data Modeling Essentials*. Amsterdam; Boston, 3rd edition, 2005.

[208] S. Smallen, K. Ericson, J. Hayes, and C. Olschanowsky. User-level Grid Monitoring with Inca 2. In *Proceedings of the 2007 workshop on Grid monitoring*, GMW '07, pages 29–38, New York, NY, USA, 2007. ACM, ACM.

[209] L. Smarr and C. E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.

[210] D. Sommerfeld, T. Lingner, M. Stanke, B. Morgenstern, and H. Richter. AUGUSTUS at MediGRID: Adaption of a bioinformatics application to grid computing for efficient genome analysis. *Future Generation Computer Systems*, 25(3):337 – 345, 2009.

[211] D. Sommerfeld and H. Richter. A Two-Tier Approach to Grid Workflow Scheduling. In *2009 2nd IEEE International Conference on Computer Science and its Applications*, pages 267–273, Jeju, Korea (South), 2009.

[212] D. Sommerfeld and H. Richter. Problems and Approaches of Workflow Scheduling in MediGRID. In *2009 Fifth IEEE International Conference on e-Science*, pages 223–230, Oxford, United Kingdom, 2009.

[213] A. S. Tanenbaum and M. Van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice Hall, October 2006.

[214] TeraGrid. Open scientific discovery infrastructure. http://www.teragrid.org/. Online, last accessed on December 12, 2016.

[215] TextGrid. The TextGrid Project. `http://www.textgrid.de/`. Online, last accessed on December 12, 2016.

[216] The UNICORE Project. The UNICORE 6 Testgrid. `http://www.unicore.eu/testgrid/`. Online, last accessed on December 12, 2016.

[217] The UNICORE Project. UNICORE 6 Live CD. `https://sourceforge.net/projects/unicore/files/Tools/LiveCD/`. Online, last accessed on December 12, 2016.

[218] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid Monitoring Architecture. Technical Report GDF.7, Global Grid Forum, January 2002.

[219] D. Tonne, R. Stotzka, T. Jejkal, V. Hartmann, H. Pasic, A. Rapp, P. Vanscheidt, B. Neumair, A. Streit, A. Garcia, D. Kurzawe, T. Kálmán, B. S. Bribian, and J. Rybicki. A Federated Data Zone for the Arts and Humanities. In *Proc.of the 20th International Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, pages 189 – 207, 2012.

[220] G. Tsouloupas and M. D. Dikaiakos. GridBench: A Tool for the Interactive Performance Exploration of Grid Infrastructures. *J. Parallel Distrib. Comput.*, 67(9):1029–1045, 2007.

[221] UNICORE. Uniform Interface to Computer Resources. `http://www.unicore.eu/`, Online, last accessed on December 12, 2016.

[222] L. Vaquero, S. S. Lor, J. Alcaraz-Calero, D. Niyato, S. Clayman, D. Audsin, and R. Nadarajan. On Measuring Disturbances in the Force: Advanced Cloud Monitoring Systems. *Future Generation Computer Systems*, 29(8):2007 – 2008, 2013.

[223] O. v.d.g. Felde, T. Baur, M. Garschhammer, and H. Reiser. Anforderungen an das Monitoring, Ergebnisse aus der Erhebung bei den Communties und Ressourcenanbietern im D-Grid. In: Rückemann, C.-P. (ed.) Ergebnisse der Studie und Anforderungsanalyse in den Fachgebieten Monitoring, Accounting, Billing bei den Communities und Ressourcenanbietern im D-Grid (D-Grid Report), 2006.

[224] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl. Scientific Cloud Computing: Early Definition and Experience. In *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, HPCC '08, pages 825–830, Washington, DC, USA, 2008. IEEE Computer Society.

[225] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for Grid Services. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 48–57. IEEE, 2003.

[226] M. P. Wellman. A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[227] M. West. *Developing High Quality Data Models*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011.

[228] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. Terminology for Policy-Based Management. Internet RFC 3198, November 2001.

[229] J. L. Whitten and L. D. Bentley. *Systems Analysis and Design Methods*. McGraw-Hill, Inc., New York, NY, USA, 7 edition, 2007.

[230] A. Willner. Entwurf und Implementierung einer Ressourcen-Datenbank für das Instant-Grid-Projekt der GWDG. Masters's thesis, Georg-August-Universität Göttingen, 2006.

[231] WLCG. Worldwide LHC Computing Grid. `http://lcg.web.cern.ch/lcg/`. Online, last accessed on December 12, 2016.

[232] A. Wolf. Spezifikation der D-Grid-Ressourcenbeschreibungssprache DGRDL. Technical report, Fraunhofer FIRST, 2007.

[233] R. Yahyapour and P. Wieder. Heiter bis wolkig. Cloud-Dienste für die Wissenschaft. *Wissenschaftsmanagement*, 19(3):22–25, 2013.

[234] P. Zadrozny, P. Aston, and T. Osborne. *J2EE Performance Testing with BEA WebLogic Server*. Apress, 2003.

[235] S. Zanikolas and R. Sakellariou. A Taxonomy of Grid Monitoring Systems. *Future Gener. Comput. Syst.*, 21(1):163–188, 2005.

A. Appendix: Size and Structure of the Monitoring Data

To design an automated resource description exchange process and a respective generic monitoring architecture supporting it, it is important to understand how the size and the structure of the monitoring data vary. Here we answer the following questions: (1) How many resources should be monitored in a distributed research infrastructure? (2) How much monitoring data does this mean? We therefore analyze the monitoring data coming from the source systems. In our application scenario the source systems are MDS4, BDII, and CIS, which are the monitoring and information systems of the middlewares GT4, gLite, and UNICORE6, respectively. Table A.1 summarizes the interfaces, clients, and extraction formats we utilize.

Information Service	**MDS4**	**BDII**	**CIS**
Middleware	Globus Toolkit v4.x	gLite	UNICORE6
Interface/Client	wsrf-query	LDAP query	ucc
Output	XML	LDIF	XML

Table A.1.: MDS4, BDII, CIS Sources for Monitoring Data

For several years we have extracted the raw monitoring data from the information systems of a productional research infrastructure 4 times a day. We archived the original monitoring data that enabled us to replay ETL transformations and make statistical analysis of the data itself as well as to examine the ETL processes. We therefore designed Algorithm 2 to prepare the archived monitoring data. Our process retrieves the compressed raw data from the archive, uncompresses it and counts the file size.

Algorithm 2: Algorithm for the Preparation and Transformation of Archived Monitoring Data

```
Initialize mean = 0; MonitoringDataSet = empty;
while there is archived monitoring data do
    LoadFromArchive(monitoring data);
    rawdata = Uncompress(archived monitoring data);
    switch the type of rawdata do
        case MDS4
            xmldata = rawdata;
        case BDII
            glue2data = TransformToXml(rawdata, BDIIadaptor);
        case CIS
            xmldata = rawdata;
    endsw
    if FileSize (xmldata) > 0 then
        append(MonitoringDataSet,xmldata);
    end
end
```

A.1. Size of Monitoring Data

A.1.1. Filesize MDS4: National RI

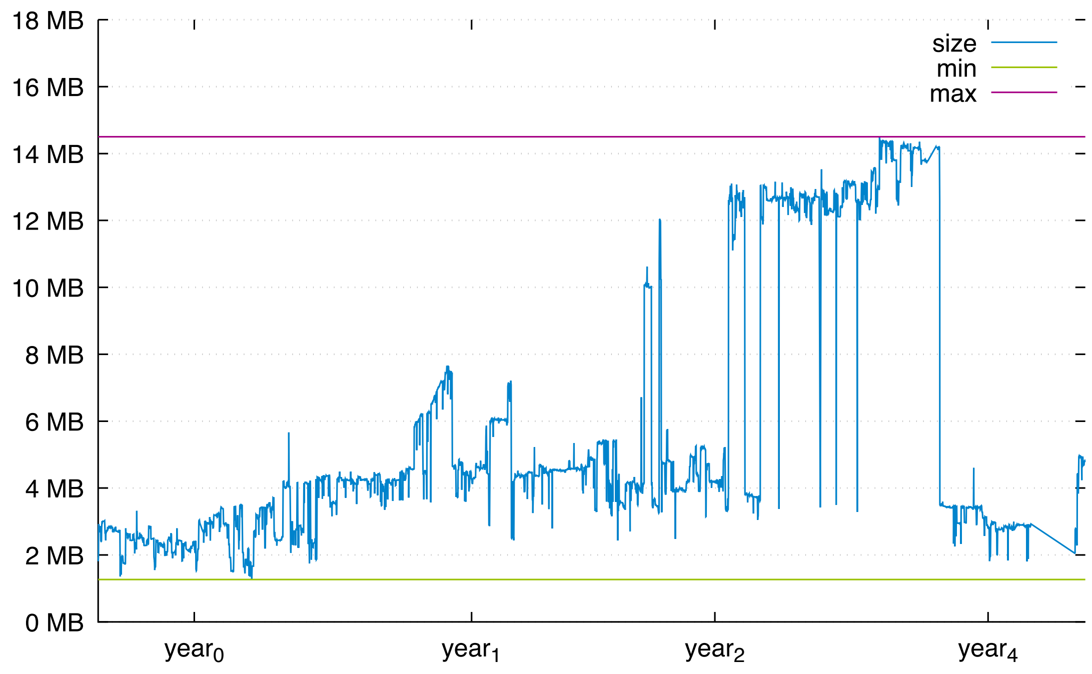


Figure A.1.: Size of MDS4 Monitoring Data in a National Research Infrastructure

A.1.2. Filesize MDS4: Community RI

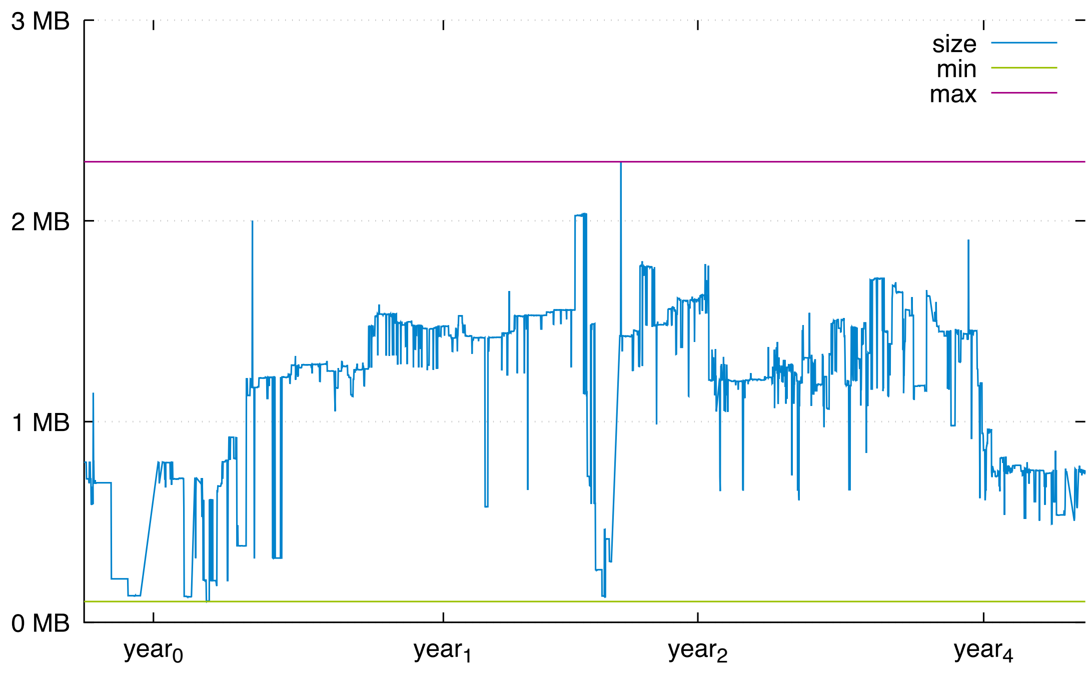


Figure A.2.: Size of MDS4 Monitoring Data in a Research Infrastructure Serving a Community

A.1.3. Filesize BDII: National RI

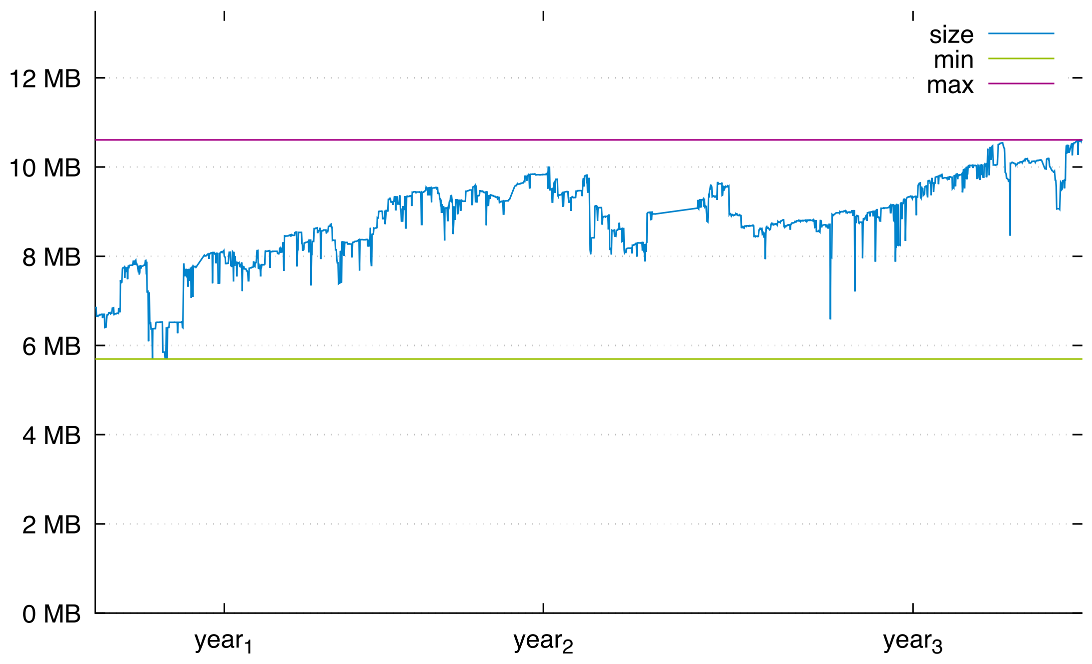


Figure A.3.: Size of BDII Monitoring Data in a National Research Infrastructure

A.1.4. Filesize BDII: RI Site

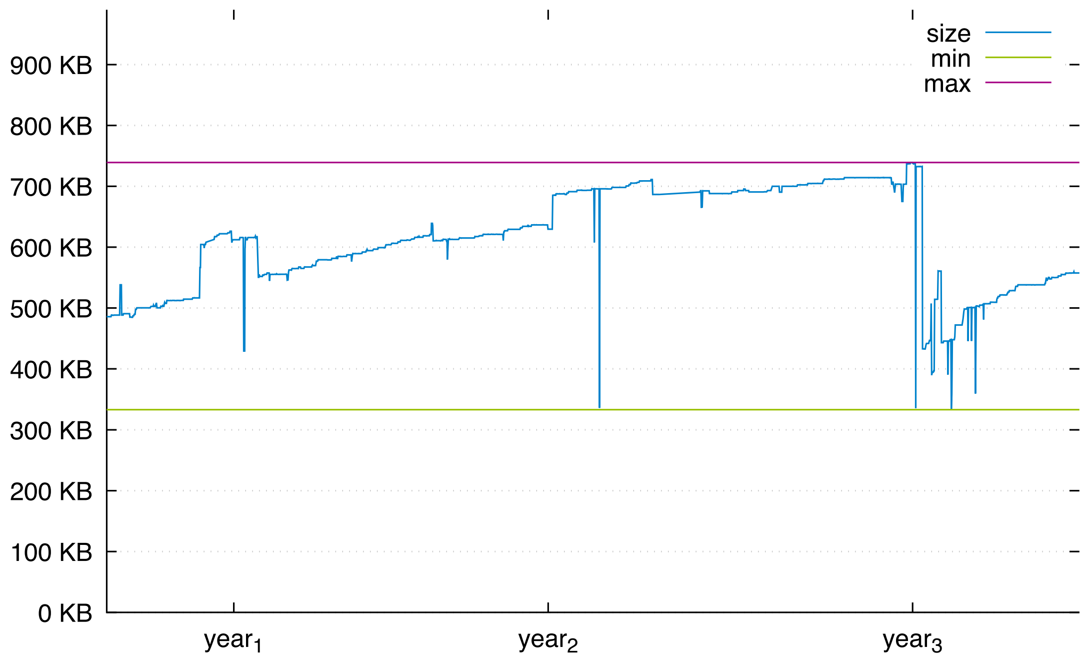


Figure A.4.: Size of BDII Monitoring Data in a Research Infrastructure Serving a Community

A.2. Structure of Monitoring Data

To transform resource descriptions into the generic schema efficiently, we need to understand how many records and what kind of records are to be transformed and stored. We designed Algorithm 3 for that purpose. The process first retrieves the raw data from the archive and uncompress it, as it was the case to analyze the size of the source monitoring data. The algorithm continues by transforming the source data into the GLUE v2.0 format and loading it into the target database. It counts the records in the database by grouping them by the GLUE-entities and timeframes.

Algorithm 3: Algorithm for the Calculation of the Arithmetic Mean of GLUE v2.0 Records

```
Initialize loaded = 0; bad data = 0;
Initialize Entities = ["AdminDomain", "AdminDomainLocation", ..., "StorageEndpoint"];
foreach element rawdata of the monitoring data set MonitoringDataSet do
    switch the type of sourcedata do
        case MDS4
            glue2data = ProcessXslt(sourcedata, MDS4schema);
        case BDII
            glue2data = ProcessXslt(sourcedata, BDIIschema);
        case CIS
            glue2data = ProcessXslt(sourcedata, CISschema);
    endsw
    clean up temporary files and raw data;
    if successful parse then
        InitializeDatabase();
        LoadToDatabase(glue2data);
        if successful load then
            loaded = +1;
            group the records by GLUE-entities and timeframes;
            count the GLUE2 records by entity and year;
        else
            baddata = +1;
        end
    end
    clean up glue2data;
end
foreach element entity of the Entities do
    foreach element year of the Years do
        $mean_{ij}$ = CalcArithmeticMean(entity, year);
        round arithmetic mean;
    end
end
```

A.2.1. GLUE2 Records: National RI with MDS4

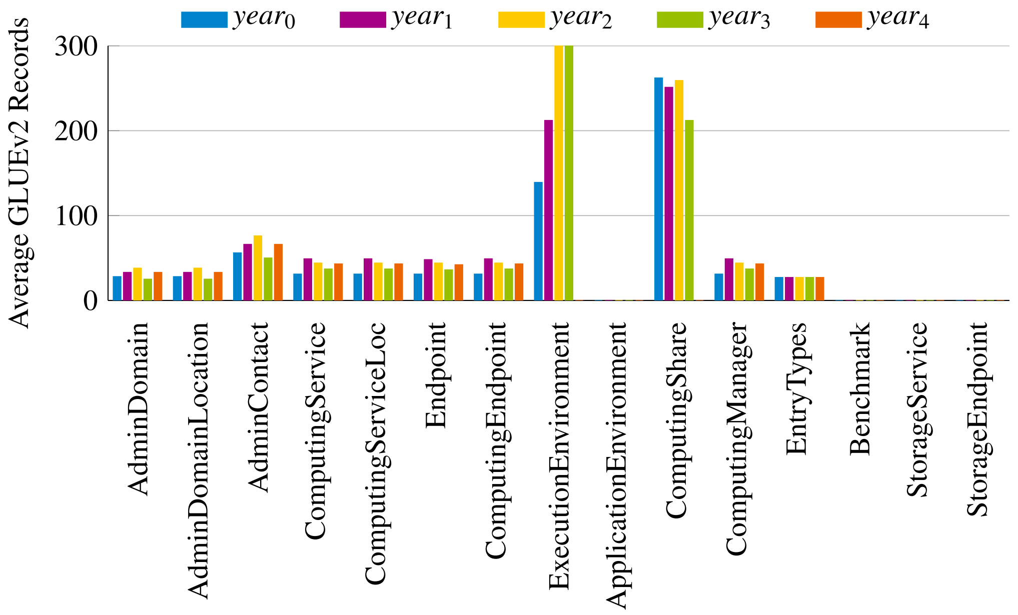


Figure A.5.: Mean of GLUE2 Records: National Research Infrastructure with MDS4

Entity	$Year_0$	$Year_1$	$Year_2$	$Year_3$	$Year_4$
AdminDomain	28	33	38	25	33
AdminDomainLocation	28	33	38	25	33
AdminContact	56	66	76	50	66
ComputingService	31	49	44	37	43
ComputingServiceLoc	31	49	44	37	43
Endpoint	31	48	44	36	42
ComputingEndpoint	31	49	44	37	43
ExecutionEnvironment	139	212	554	548	0
ApplicationEnvironment	0	0	0	0	0
ComputingShare	262	251	259	212	0
ComputingManager	31	49	44	37	43
EntryTypes	27	27	27	27	27
Benchmark	0	0	0	0	0
StorageService	0	0	0	0	0
StorageEndpoint	0	0	0	0	0

Table A.2.: Mean of GLUE2 Records: National Research Infrastructure with MDS4

A.2.2. GLUE2 Records: Community RI with MDS4

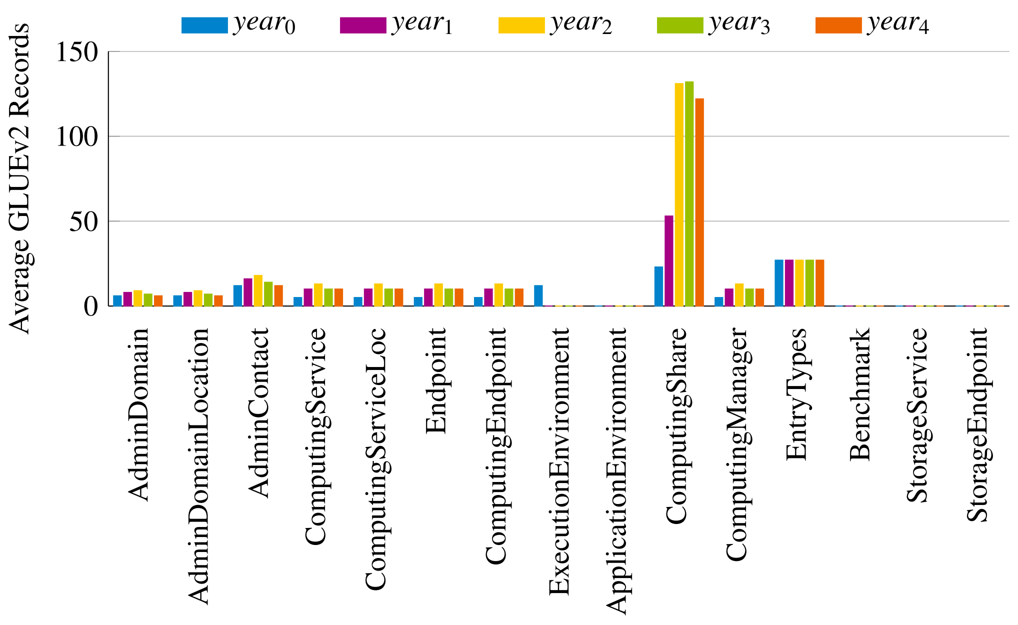


Figure A.6.: Mean of GLUE2 Records: Community Research Infrastructure with MDS4

Entity	$Year_0$	$Year_1$	$Year_2$	$Year_3$	$Year_4$
AdminDomain	6	8	9	7	6
AdminDomainLocation	6	8	9	7	6
AdminContact	12	16	18	14	12
ComputingService	5	10	13	10	10
ComputingServiceLoc	5	10	13	10	10
Endpoint	5	10	13	10	10
ComputingEndpoint	5	10	13	10	10
ExecutionEnvironment	12	0	0	0	0
ApplicationEnvironment	0	0	0	0	0
ComputingShare	23	53	131	132	122
ComputingManager	5	10	13	10	10
EntryTypes	27	27	27	27	27
Benchmark	0	0	0	0	0
StorageService	0	0	0	0	0
StorageEndpoint	0	0	0	0	0

Table A.3.: Mean of GLUE2 Records: Community Research Infrastructure with MDS4

A.2.3. GLUE2 Records: National RI with BDII

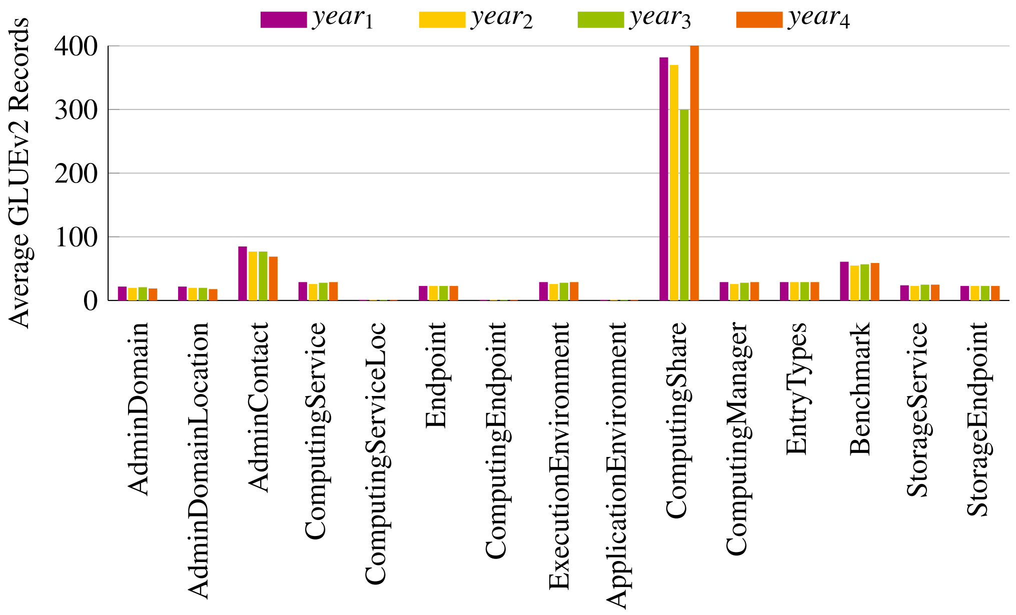


Figure A.7.: Mean of GLUE2 Records: National Research Infrastructure with BDII

Entity	*Year*$_1$	*Year*$_2$	*Year*$_3$	*Year*$_4$
AdminDomain	21	19	20	18
AdminDomainLocation	21	19	19	17
AdminContact	84	76	76	68
ComputingService	28	25	27	28
ComputingServiceLoc	0	0	0	0
Endpoint	22	22	22	22
ComputingEndpoint	0	0	0	0
ExecutionEnvironment	28	25	27	28
ApplicationEnvironment	0	0	0	0
ComputingShare	381	369	299	406
ComputingManager	28	25	27	28
EntryTypes	28	28	28	28
Benchmark	60	54	56	58
StorageService	23	22	24	24
StorageEndpoint	22	22	22	22

Table A.4.: Mean of GLUE2 Records: National Research Infrastructure with BDII

A.2.4. GLUE2 Records: RI Site with BDII

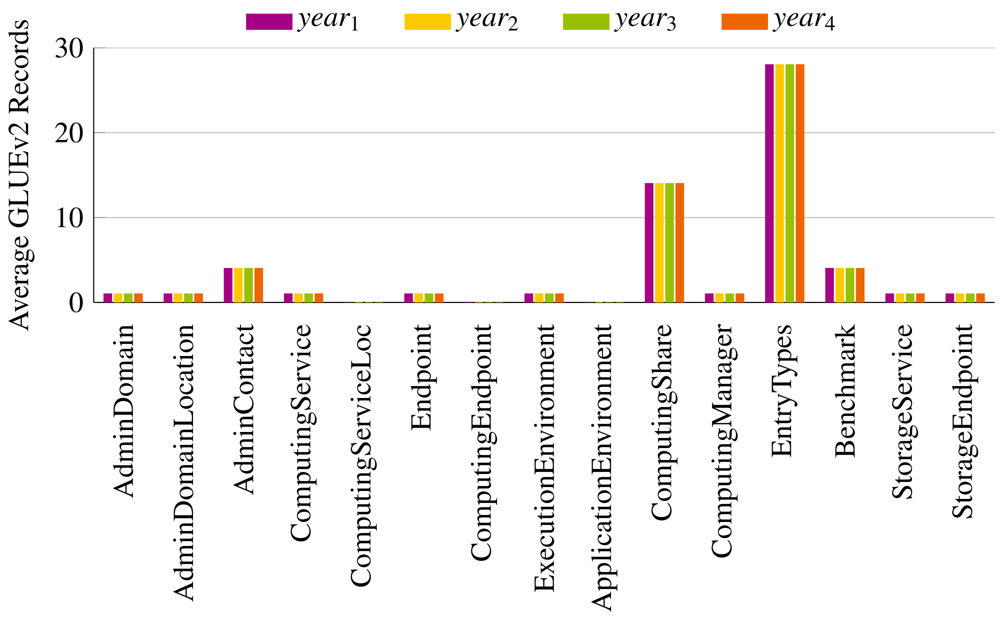


Figure A.8.: Mean of GLUE2 Records: Research Infrastructure Site with BDII

Entity	$Year_1$	$Year_2$	$Year_3$	$Year_4$
AdminDomain	1	1	1	1
AdminDomainLocation	1	1	1	1
AdminContact	4	4	4	4
ComputingService	1	1	1	1
ComputingServiceLoc	0	0	0	0
Endpoint	1	1	1	1
ComputingEndpoint	0	0	0	0
ExecutionEnvironment	1	1	1	1
ApplicationEnvironment	0	0	0	0
ComputingShare	14	14	14	14
ComputingManager	1	1	1	1
EntryTypes	28	28	28	28
Benchmark	4	4	4	4
StorageService	1	1	1	1
StorageEndpoint	1	1	1	1

Table A.5.: Mean of GLUE2 Records: Research Infrastructure Site with BDII

A.2.5. GLUE2 Records: National RI with CIS (Test-Instance)

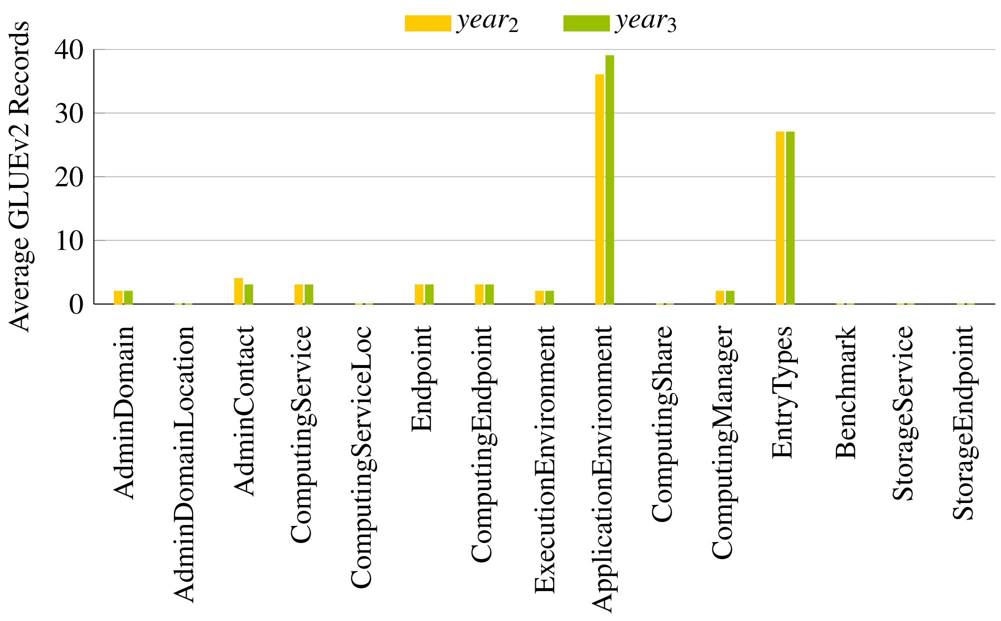


Figure A.9.: Mean of GLUE2 Records in CIS: Testbed for a National Research Infrastructure

Entity	*Year*$_2$	*Year*$_3$
AdminDomain	2	2
AdminDomainLocation	0	0
AdminContact	4	3
ComputingService	3	3
ComputingServiceLoc	0	0
Endpoint	3	3
ComputingEndpoint	3	3
ExecutionEnvironment	2	2
ApplicationEnvironment	36	39
ComputingShare	0	0
ComputingManager	2	2
EntryTypes	27	27
Benchmark	0	0
StorageService	0	0
StorageEndpoint	0	0

Table A.6.: Mean of GLUE2 Records in CIS: Testbed for a National Research Infrastructure

B. Appendix: Selected Mapping Tables

This Section highlights some mapping tables we used for our thesis. The mapping tables B.1, B.3, B.2 show, how the Site entity of the MDS4 schema is mapped to the entities of the generic GLUE v2.0 Schema. The *Site* entity describes in MDS4 the resource and service providers. However, the GLUE v2.0 information model does not have such a Site entity. The GLUE v2.0 schema describes resource providers by defining the *AdminDomain*, *AdminDomainLocation*, and *AdminContact* entities. Therefore, our mapping splits the Site entity into three new entities. We point out that the mapping transforms the name and description of a provider into the AdminDomain entity, while the location information of a provider is transformed into the AdminDomainLocation entity. The AdminContact entity stores various contact addresses of the provider (email address, website). Note that we transform the unique identifier of the site into the three new entities, because the Location and Contact entities use the unique identifier of the Domain as a foreign key.

The mapping tables B.4 and B.5 define the mappings for the ComputingService and ComputingShare entities, respectively.

Original entity	Attributes	comments	Mapped entities	Attributes	comments
MDS4.Site	UniqueID		GLUE20.AdminDomain	ID	
MDS4.Site	Name		GLUE20.AdminDomain	Name	
MDS4.Site	Description		GLUE20.AdminDomain	Description	

Table B.1.: Example Mapping for the AdminDomain Entity of the GLUE v2.0 Schema

Original entity	Attributes	comments	Mapped entities	Attributes	comments
MDS4.Site	Web		GLUE20.AdminContact	adminDomainID	
			GLUE20.AdminContact	URI	
		fixed	GLUE20.AdminContact	Type	"web", "email"

Table B.2.: Example Mapping for the AdminContact Entity of the GLUE v2.0 Schema

Original entity	Attributes	comments	Mapped entities	Attributes	comments
			GLUE20.AdminDomainLocation	adminDomainID	
			GLUE20.AdminDomainLocation	Name	
MDS4.Site	Location	(splitted)	GLUE20.AdminDomainLocation	Place	
		(splitted)	GLUE20.AdminDomainLocation	Address	
		(splitted)	GLUE20.AdminDomainLocation	Postcode	
MDS4.Site	Longitude		GLUE20.AdminDomainLocation	Longitude	
MDS4.Site	Latitude		GLUE20.AdminDomainLocation	Latitude	

Table B.3.: Example Mapping for the AdminDomainLocation Entity of the GLUE v2.0 Schema

ComputingService

GLUE v2.0	BDII	CIS	MDS4
ID	GlueClusterTop.GlueClusterUniqueID	ComputingService.ID	ComputingElement.Info.HostName
name	GlueClusterTop.GlueClusterName	ComputingService.Name	ComputingElement.Info.HostName
type	fix	ComputingService.Type	fix
qualityLevel			
complexity			
totalJobs	SUM GlueCETop.GlueCEStateTotalJobs	ComputingService.TotalJobs	SUM ComputingElement.State.TotalJobs
runningJobs	SUM GlueCETop.GlueCEStateRunningJobs	ComputingService.RunningJobs	SUM ComputingElement.State.RunningJobs
waitingJobs	SUM GlueCETop.GlueCEStateWaitingJobs	ComputingService.WaitingJobs	SUM ComputingElement.State.WaitingJobs
stagingJobs			
suspendedJobs			
preLRMSWaitingJobs			
domainID			Site.UniqueID

Table B.4.: Example Mapping for the ComputingService Entity of the GLUE v2.0 Schema

ComputingShare

GLUE v2.0	BDII	CIS	MDS4
serviceID	GlueCETop.	*	ComputingElement.Info.HostName
localID	GlueCETop.GlueCEUniqueID	*	ComputingElement.UniqueID
name			
description			
mappingQueue	GlueCETop.GlueCEUniqueID	*	ComputingElement.UniqueID
maxWallTime	GlueCETop.GlueCEPolicyMaxWallClockTime		!ComputingElement.Policy.MaxWallClockTime
maxTotalWallTime			
minWallTime			
defaultWallTime			
maxCPUTime	GlueCETop.GlueCEPolicyMaxCPUTime		!ComputingElement.Policy.MaxCPUTime
maxTotalCPUTim			
minCPUTime			
defaultCPUtime			
maxTotalJobs	GlueCETop.GlueCEPolicyMaxTotalJobs	*	!ComputingElement.Policy.MaxTotalJobs
maxRunningJobs	GlueCETop.GlueCEPolicyMaxRunningJobs	*	!ComputingElement.Policy.MaxRunningJobs
maxWaitingJobs			
maxPreLRMSWaitingJobs			
maxUserRunningJobs			
maxSlotsPerJobs			
maxStageInStreams			
maxStageOutStreams			
schedulingPolicy	GlueCETop.GlueCEPolicyPriority		
maxMemory			
maxDiskSpace			
preemption			
servingstate			
totalJobs	GlueCETop.GlueCEStateTotalJobs	*	ComputingElement.State.TotalJobs
runningJobs	GlueCETop.GlueCEStateRunningJobs	*	ComputingElement.State.RunningJobs
localRunningJobs			
waitingJobs	GlueCETop.GlueCEStateWaitingJobs	*	ComputingElement.State.WaitingJobs
localWaitingJobs			
stagingJobs			
suspendedJobs			
preLRMSWaitingJobs			
estimatedAverageWaitingTime	GlueCETop.GlueCEStateEstimatedResponseTime		
estimatedWorstWaitingTime	GlueCETop.GlueCEStateWorstResponseTime		
freeSlots	GlueCETop.GlueCEStateFreeJobSlots		
usedSlots			
requestedSlots			

* : planned in the future   ! : Not used by the middleware

Table B.5.: Example Mapping for the ComputingShare Entity of the GLUE v2.0 Schema

C. Appendix: Performance and Stress Tests with Grinder

C.1. Stress Test Tool

For our performance and stress tests we required a testing framework, which is able to run test suites on a distributed manner and allows to develop extensions. We found that the Grinder [234] distributed testing framework, which is based on open source technologies, can perform distributed tests utilizing load-injector machines and allows any test code to be encapsulated as a test. Its open source license enables to adapt it for distributed research infrastructure environments. We used Grinder to (1) prove the correct behavior of an application (functional testing), (2) to determine whether our integrated system can serve a high load with acceptable response times (load testing), and (3) to determines if the system behaves stable and reliable for a specified time under a specified load (stress testing).

Three types of processes consist the Grinder architecture: the worker(s), the agent(s), and the console. The worker processes interpret the test suites, load the test codes and perform the tests in parallel by utilizing threads. The agent processes manage, start and stop the workers on demand. Several agents can be started on the load-injector machines. The console is responsible for coordinating the other processes. It collects, calculates and displays test statistics. Figure C.1 depicts these components.

For our tests we started a Grinder agent process on each of our load-injector client machines. These Grinder agents were controlled by the Grinder console. We also used the console to collect and accumulate basic statistics, like the mean time, the transactions per seconds, and the standard deviation[43].

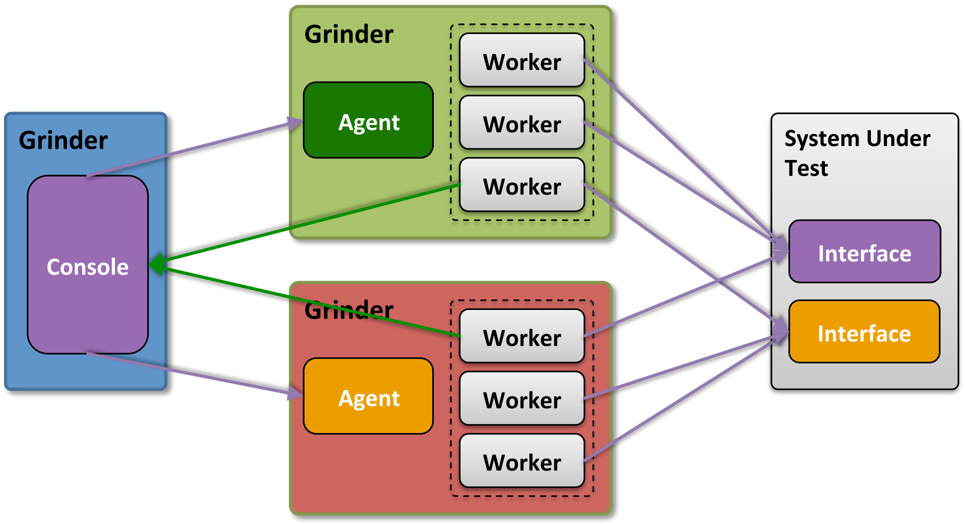
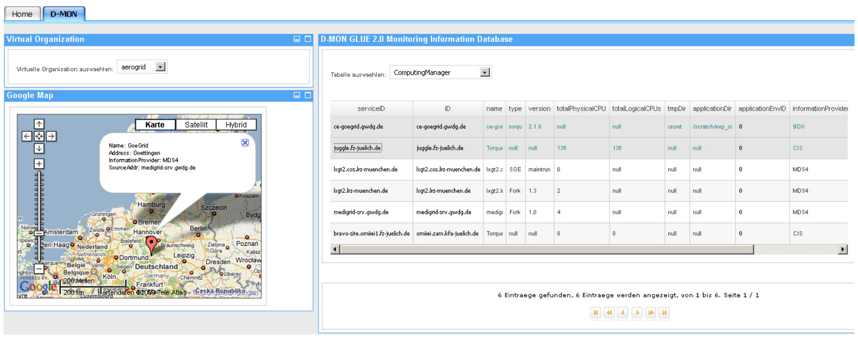


Figure C.1.: The Architecture of Grinder: Console, Workers, and Agents

[43]Grinder Statistics: `http://grinder.sourceforge.net/faq.html#calculation`, Online, last accessed on December 12, 2016

C.2. Portal Interface for D-MON

The D-MON project developed a higher level user interface, which uses the integrated monitoring system to access resource descriptions and monitoring data. Figure C.2 shows three screenshots of the D-MON web portlets.



Virtuelle Organisation auswaehlen: aerogrid

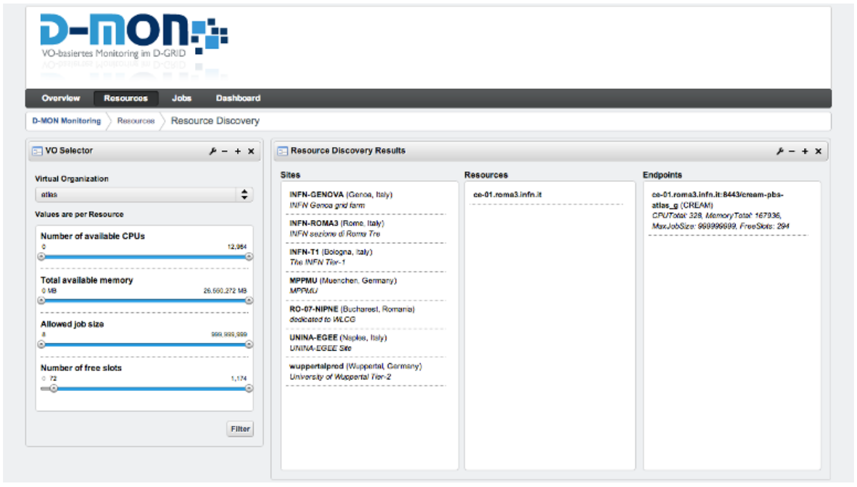
Name: GoeGrid
Address: Goettingen
InformationProvider: MDS4
SourceAddr: medigrid-srv.gwdg.de

Tabelle auswaehlen: ComputingManager

serviceID	ID	name	type	version	totalPhysicalCPU	totalLogicalCPUs	tmpDir	applicationDir	applicationEnvID	informationProvide
ce-goegrid.gwdg.de	ce-goegrid.gwdg.de	ce-goe	torqu	2.1.8	null	null	unset	/scratch/exp_s	0	BDII
juggle.fz-juelich.de	juggle.fz-juelich.de	Torque	null	null	136	136	null	null	0	CIS
lxgt2.cos.lrz-muenchen.de	lxgt2.cos.lrz-muenchen.de	lxgt2.c	SGE	maintrun	0	null	null	null	0	MDS4
lxgt2.lrz-muenchen.de	lxgt2.lrz-muenchen.de	lxgt2.l	Fork	1.3	2	null	null	null	0	MDS4
medigrid-srv.gwdg.de	medigrid-srv.gwdg.de	medigi	Fork	1.0	4	null	null	null	0	MDS4
bravo-site.omiiei1.fz-juelich.de	omiiei.zam.kfa-juelich.de	Torque	null	null	8	8	null	null	0	CIS

6 Eintraege gefunden, 6 Eintraege werden angezeigt, von 1 bis 6. Seite 1 / 1

(a) Screenshot: D-MON User Interface Based on JSR-168 Portlets



Virtual Organization: atlas

Values are per Resource

Number of available CPUs: 0 – 12,984

Total available memory: 0 MB – 26,550,272 MB

Allowed job size: 8 – 999,999,999

Number of free slots: 72 – 1,174

Sites

INFN-GENOVA (Genoa, Italy)
INFN Genoa grid farm

INFN-ROMA3 (Rome, Italy)
INFN sezione di Roma Tre

INFN-T1 (Bologna, Italy)
The INFN Tier-1

MPPMU (Muenchen, Germany)
MPPMU

RO-07-NIPNE (Bucharest, Romania)
dedicated to WLCG

UNINA-EGEE (Naples, Italy)
UNINA-EGEE Site

wuppertalprod (Wuppertal, Germany)
University of Wuppertal Tier-2

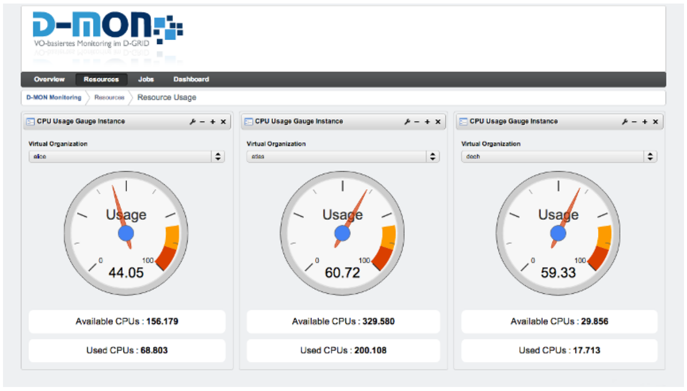
Resources

ce-01.roma3.infn.it

Endpoints

ce-01.roma3.infn.it:8443/cream-pbs-atlas_g (CREAM)
CPUTotal: 328, MemoryTotal: 167936, MaxJobSize: 999999999, FreeSlots: 294

(b) Screenshot: D-MON Resource Discovery According to Community Membership



Virtual Organization	Usage	Available CPUs	Used CPUs
alice	44.05	156.179	68.803
atlas	60.72	329.580	200.108
dech	59.33	29.856	17.713

(c) Screenshot: D-MON Resource Usage

Figure C.2.: The D-MON Portal Interface (Source: D-MON[44])

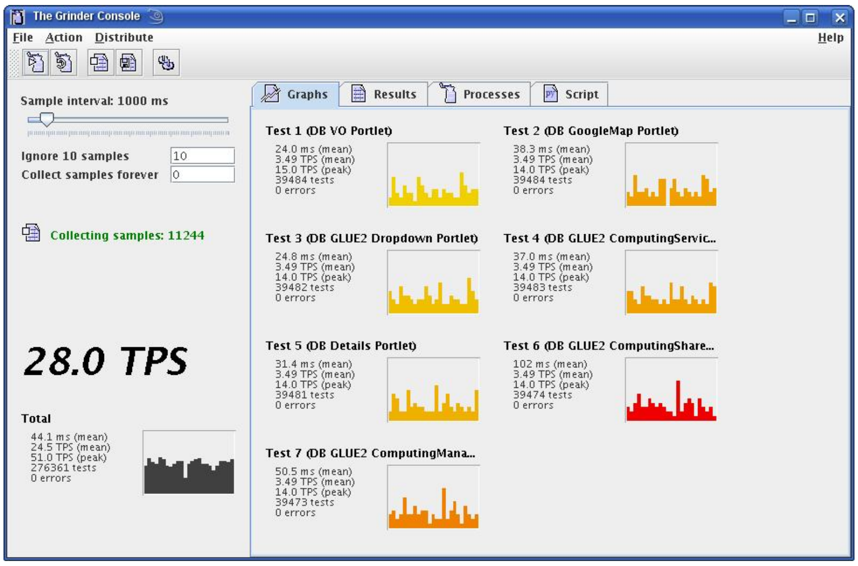
[44]D-MON Portal `https://www.lrz.de/services/compute/grid_en/software_en/dmon_en/`, Online, last accessed on December 12, 2016

C.3. Grinder Results

To simulate the behavior of portal users and to make load on the integrated monitoring system, we created suitable test cases for Grinder. This section highlights some results and shows some screenshots of Grinder running our test cases. Figure C.3 shows seven tests, which simulate the behavior of portal users. The test cases are controlled and monitored by the Grinder Console.

The Grinder Console also collects and aggregates the test results, as well as generates basic statistics. The screenshot in Figure C.4 shows some test results and statistics after 275 000 successful test runs. Figure C.5 depicts the Grinder Console while it coordinates the Grinder Agents that run 40 Workers and 40 Threads.

Table C.1 shows the statistics of Grinder test cases, which simulate portal users. The Grinder test results for ETL-Adaptors are shown on Table C.2.



Sample interval: 1000 ms
Ignore 10 samples 10
Collect samples forever 0
Collecting samples: 11244

28.0 TPS

Total
44.1 ms (mean)
24.5 TPS (mean)
51.0 TPS (peak)
276361 tests
0 errors

Test 1 (DB VO Portlet)
24.0 ms (mean)
3.49 TPS (mean)
15.0 TPS (peak)
39484 tests
0 errors

Test 2 (DB GoogleMap Portlet)
38.3 ms (mean)
3.49 TPS (mean)
14.0 TPS (peak)
39484 tests
0 errors

Test 3 (DB GLUE2 Dropdown Portlet)
24.8 ms (mean)
3.49 TPS (mean)
14.0 TPS (peak)
39482 tests
0 errors

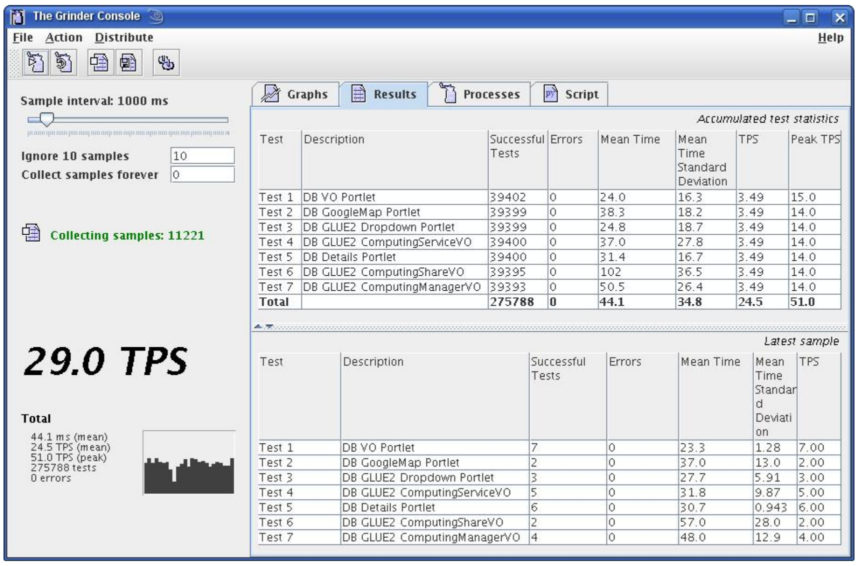
Test 4 (DB GLUE2 ComputingServic...
37.0 ms (mean)
3.49 TPS (mean)
14.0 TPS (peak)
39483 tests
0 errors

Test 5 (DB Details Portlet)
31.4 ms (mean)
3.49 TPS (mean)
14.0 TPS (peak)
39481 tests
0 errors

Test 6 (DB GLUE2 ComputingShare...
102 ms (mean)
3.49 TPS (mean)
14.0 TPS (peak)
39474 tests
0 errors

Test 7 (DB GLUE2 ComputingMana...
50.5 ms (mean)
3.49 TPS (mean)
14.0 TPS (peak)
39473 tests
0 errors

Figure C.3.: Stress Test Controlled and Monitored by the Grinder Console



Sample interval: 1000 ms
Ignore 10 samples 10
Collect samples forever 0
Collecting samples: 11221

29.0 TPS

Total
44.1 ms (mean)
24.5 TPS (mean)
51.0 TPS (peak)
275788 tests
0 errors

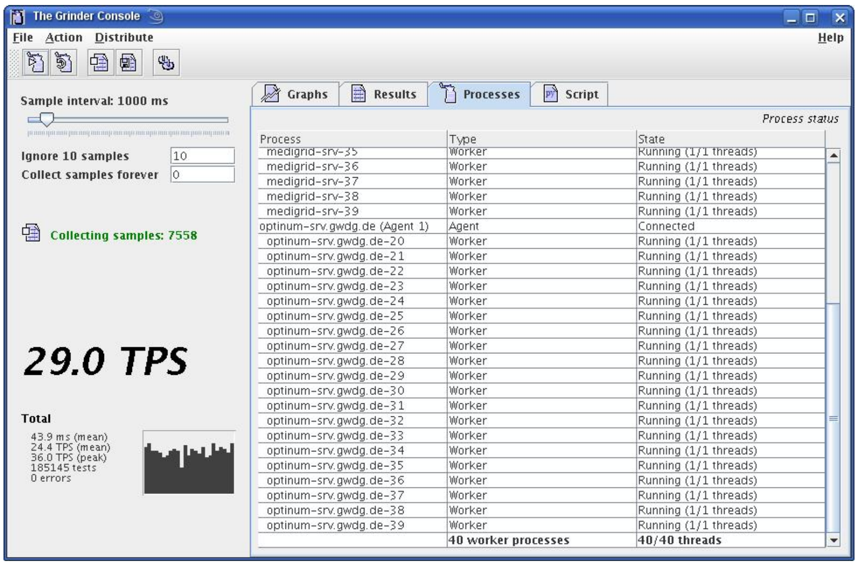
Accumulated test statistics

Test	Description	Successful Tests	Errors	Mean Time	Mean Time Standard Deviation	TPS	Peak TPS
Test 1	DB VO Portlet	39402	0	24.0	16.3	3.49	15.0
Test 2	DB GoogleMap Portlet	39399	0	38.3	18.2	3.49	14.0
Test 3	DB GLUE2 Dropdown Portlet	39399	0	24.8	18.7	3.49	14.0
Test 4	DB GLUE2 ComputingServiceVO	39400	0	37.0	27.8	3.49	14.0
Test 5	DB Details Portlet	39400	0	31.4	16.7	3.49	14.0
Test 6	DB GLUE2 ComputingShareVO	39395	0	102	36.5	3.49	14.0
Test 7	DB GLUE2 ComputingManagerVO	39393	0	50.5	26.4	3.49	14.0
Total		**275788**	**0**	**44.1**	**34.8**	**24.5**	**51.0**

Latest sample

Test	Description	Successful Tests	Errors	Mean Time	Mean Time Standard Deviation	TPS
Test 1	DB VO Portlet	7	0	23.3	1.28	7.00
Test 2	DB GoogleMap Portlet	2	0	37.0	13.0	2.00
Test 3	DB GLUE2 Dropdown Portlet	3	0	27.7	5.91	3.00
Test 4	DB GLUE2 ComputingServiceVO	5	0	31.8	9.87	5.00
Test 5	DB Details Portlet	6	0	30.7	0.943	6.00
Test 6	DB GLUE2 ComputingShareVO	2	0	57.0	28.0	2.00
Test 7	DB GLUE2 ComputingManagerVO	4	0	48.0	12.9	4.00

Figure C.4.: Test Results and Statistics Shown by the Grinder Console



Sample interval: 1000 ms

Ignore 10 samples: 10
Collect samples forever: 0

Collecting samples: 7558

29.0 TPS

Total
43.9 ms (mean)
24.4 TPS (mean)
36.0 TPS (peak)
185145 tests
0 errors

Process status

Process	Type	State
medigrid-srv-35	Worker	Running (1/1 threads)
medigrid-srv-36	Worker	Running (1/1 threads)
medigrid-srv-37	Worker	Running (1/1 threads)
medigrid-srv-38	Worker	Running (1/1 threads)
medigrid-srv-39	Worker	Running (1/1 threads)
optinum-srv.gwdg.de (Agent 1)	Agent	Connected
optinum-srv.gwdg.de-20	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-21	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-22	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-23	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-24	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-25	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-26	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-27	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-28	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-29	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-30	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-31	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-32	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-33	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-34	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-35	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-36	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-37	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-38	Worker	Running (1/1 threads)
optinum-srv.gwdg.de-39	Worker	Running (1/1 threads)
	40 worker processes	**40/40 threads**

Figure C.5.: Grinder Console Showing the Grinder Agents Running 40 Workers and 40 Threads

Test Nr.	Test Case Description	Successful Tests	Errors	Mean Time	Standard Deviation
1	DB VO Portlet	40000	0	24,3	4,037325848
2	DB GoogleMap Portlet	40000	0	38,2	4,266145802
3	DB GLUE2 Dropdown Portlet	40000	0	24,6	4,312771731
4	DB GLUE2 ComputingServiceVO	40000	0	37,1	5,282045058
5	DB Details Portlet	40000	0	31,4	4,086563348
6	DB GLUE2 ComputingShareVO	40000	0	102	6,033241252
7	DB GLUE2 ComputingManagerVO	40000	0	50,6	5,138093031

Table C.1.: Grinder Results of Test Cases Simulating Portal Users

Test Nr.	Test Case Description	Successful Tests	Errors	Mean Time	Standard Deviation
1	DB Update MDS4	10000	0	421	37,53664876
2	DB Update CIS	10000	0	240	29,61418579
3	DB Update BDII	10000	0	303	44,83302354

Table C.2.: Grinder Results of Test Cases Simulating ETL-Adaptors