UNIVERSITÁ DI PISA
DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

TRANSACTIONAL REST INFORMATION SYSTEM FOR FEDERATED RESEARCH INFRASTRUCTURES ENABLING VIRTUAL RESEARCH ENVIRONMENTS

DOCTORAL THESIS

Author
Luca Frosini

Tutor (s)
Prof. Cinzia Bernardeschi, Dr. Pasquale Pagano, Prof. Alessio Vecchio

Reviewer (s)
Prof. Hussein Suleman, Prof. Yannis Tzitzikas

The Coordinator of the PhD Program
Prof. Marco Luise

Pisa, October 2018

Cycle XXXI

This thesis is dedicated to Alessio and Manuela

"The difficulty lies not so much in developing new ideas
as in escaping from old ones."
John Maynard Keynes

Summary

THIS current PhD thesis aims at researching solutions to the long-standing gap in the research infrastructure field of formal modelling and discovery of information.

Research Infrastructures (RIs) are facilities, resources and services used by the research community to conduct research and promote innovation. They provide access to tools, data and algorithms, pooled on working environments named Virtual Research Environments (VREs). Due to their nature, Research Infrastructures (RIs) are a heterogeneous landscape of resource types and diverse, often redundant, models for describing the same resource entities. Such diversity drastically diminishes the capabilities of a consistent fruition of the knowledge and skills available within an infrastructure, and ultimately limits the creation of federations of RIs. The current state of knowledge in the field is even more complex due to the emerging concept of Hybrid Data Infrastructures (HDIs), which combine on-premises resources with resources provisioned by cloud and/or grid computing infrastructures.

The focus of this study is to address the critical aspects of this heterogeneity and provides to the research infrastructure field both the conceptual and analytical instruments for expanding the scale at which the infrastructures currently operate.

The following research questions are investigated in this study:

Research question 1: Is it possible to formalise a uniform resource model to seamlessly federate research infrastructures?

Research question 2: To what extent do state-of-the-art discovery algorithms need to be amended to properly handle VREs across Hybrid Data Infrastructures?

Research question 3: To what extent does the interface of a discovery service can support the new model and algorithms?

First objective is attained with the definition of a resource model open by design. An accurate mathematical notation presents a minimal but powerful model consisting of few constructs. This is an abstract model, which can be instantiated to accommodate almost every possible scenario. Second objective is confronted with the design of a novel communication model based on the concept of *transaction*. Transactional models

are not a new idea; rather they have been extensively described in the literature since the 80s. However, this work identifies a misapplication of these models in the context of research infrastructure and proposes changes to support the emerging requirements in the field. Lastly, the third objective refers to the impact that the first two questions have on architectural aspects of the services devoted to distribute the information in a research infrastructure. This point is deeply investigated with a detailed design of a production-ready solution to prove the validity and accuracy of the theoretical aspects of this thesis.

The originality of the proposed thesis consists also in its methodology. In fact, this dissertation offers an innovative analytical and methodological approach to address these research questions. A key-enabling component of modern Hybrid Data Infrastructures, the Information System (IS), is used as platform to validate this work. An IS is central to all the themes tackled in this thesis. It exploits the resource model, plays the role of registry inside the infrastructure, deals with an intensive communication with all the other actors and works with the heterogeneity of such a context. Its ultimate goal is to provide uniform views of the resources assigned to each VRE, regardless of their origin.

The study presented in this thesis has been conducted within the InfraScience group, the research group pioneering the Enabling Scientific Data Infrastructures field in the past two decades. The group operates at the Institute of Information Science and Technologies (ISTI), an institute of the Italian National Research Council (CNR). This invaluable environment provided a unique ground to develop and validate the theories illustrated in the thesis. Focusing on the validation, the abstract resource model has been instantiated in the gCube model of the D4Science HDI, while the transactional model has been translated in the design of the Information System (IS), successfully implemented and adopted by the same HDI.

The dissertation carries innovative theoretical, methodological and analytical value in the research infrastructure field and especially in the emerging Hybrid Data Infrastructures. Importantly, this work has been validated in one of the most challenging contexts available. The thesis is organised in chapters: Chapters 1 and 2 describe the problem statement; Chapter 3 provides a state-of-the-art overview of the problem; Chapter 4 develops the theoretical framework by describing both the abstract and gCube resource models; Chapters 5 presents transaction model, how it can be used, and why it focuses on the transaction technique; Chapter 6 offers the core methodological support with the description of the IS to support the abstract resource model and the transaction model and; Chapters 7 and 8 conclude with the validation of proposed study and the contribution to the literature it provides.

Sommario

QUESTA tesi di dottorato si propone di ricercare soluzioni alle lacune nel campo delle infrastrutture di ricerca riguardo la loro modellazione formale e la scoperta delle informazioni.

Le infrastrutture di ricerca sono un insieme di risorse e servizi utilizzati da una comunità scientifica per condurre ricerche e promuovere l'innovazione. Forniscono accesso a strumenti, dati e algoritmi, riuniti in ambienti di lavoro denominati ambienti di ricerca virtuali - Virtual Research Environment (VRE). A causa della loro natura, le infrastrutture di ricerca sono un insieme eterogeneo di tipi di risorse e modelli diversi, spesso ridondanti, per descrivere le stesse entità. Tale diversità riduce drasticamente le capacità di una fruizione coerente delle conoscenze e delle competenze disponibili all'interno di un'infrastruttura e, in ultima analisi, limita la creazione di federazioni. Lo stato attuale delle conoscenze sul campo è ancora più complesso a causa del concetto emergente di Hybrid Data Infrastructure (HDI), che combina risorse locali con risorse fornite da altre infrastrutture.

L'obiettivo di questo studio è quello di affrontare gli aspetti critici di questa eterogeneità e fornire al campo delle infrastrutture di ricerca sia gli strumenti concettuali che analitici per espandere la scala a cui le infrastrutture operano attualmente.

Le seguenti domande di ricerca sono state investigate in questo studio:

1. È possibile formalizzare un modello di risorse uniformi per federare in modo trasparente le infrastrutture di ricerca?

2. In che misura è necessario modificare gli algoritmi di scoperta delle informazioni per gestire correttamente i VRE attraverso le HDI?

3. In che misura l'interfaccia di un servizio di scoperta può supportare il nuovo modello e gli algoritmi?

Il primo obiettivo è stato raggiunto con la definizione di un modello di risorse aperto per progetto. Un'accurata notazione matematica presenta un modello minimale ma potente costituito da pochi costrutti. Questo è un modello astratto, che può essere istanziato per adattarsi a quasi tutti gli scenari possibili. Il secondo obiettivo è confrontato con

la progettazione di un nuovo modello di comunicazione basato sul concetto di *transazioni*. I modelli transazionali non sono una nuova idea; piuttosto sono stati ampiamente descritti in letteratura dagli anni '80. Tuttavia, questo lavoro identifica un'applicazione errata di questi modelli nel contesto delle infrastrutture di ricerca e propone modifiche per supportare i requisiti emergenti nel campo. Infine, il terzo obiettivo si riferisce all'impatto che le prime due domande hanno sugli aspetti architettonici dei servizi dedicati alla distribuzione delle informazioni di un'infrastruttura di ricerca. Questo punto è stato approfondito con una progettazione dettagliata di una soluzione pronta per la produzione per dimostrare la validità e l'accuratezza degli aspetti teorici di questa tesi.

L'originalità della tesi proposta consiste anche nella sua metodologia. In realtà, questa tesi offre un approccio analitico e metodologico innovativo per rispondere a queste domande di ricerca. Un componente chiave per l'abilitazione delle moderne HDI, il sistema informativo (Information System (IS)), viene utilizzato come piattaforma per convalidare questo lavoro. Un IS è centrale in tutti i temi affrontati in questa tesi. Sfrutta il modello di risorse, svolge il ruolo di registro all'interno dell'infrastruttura, gestisce una comunicazione intensiva con tutti gli altri attori e lavora con l'eterogeneità di tale contesto. Il suo obiettivo finale è fornire una visione uniforme delle risorse assegnate a ciascun VRE, indipendentemente dalla loro origine.

Lo studio presentato in questa tesi è stato condotto all'interno del gruppo InfraScience, il gruppo di ricerca pioniere nel campo delle infrastrutture di ricerca scientifica negli ultimi due decenni. Il gruppo opera presso l'Istituto di Scienze e Tecnologie dell'Informazione (ISTI), un istituto del Consiglio Nazionale delle Ricerche (CNR). Questo ambiente inestimabile ha fornito un punto di partenza unico per sviluppare e convalidare le teorie illustrate nella tesi. Concentrandosi sulla validazione, il modello di risorse astratte è stato istanziato nel modello gCube di D4Science HDI, mentre il modello transazionale è stato tradotto nella progettazione del Sistema Informativo, implementato con successo e adottato dalla stessa HDI.

La tesi presenta un valore innovativo teorico, metodologico e analitico nel campo delle infrastrutture di ricerca e in particolare nelle emergenti infrastrutture ibride di dati. È importante sottolineare che questo lavoro è stato convalidato in uno dei contesti più impegnativi disponibili. La tesi è organizzata in capitoli: i capitoli 1 e 2 descrivono la definizione del problema; il capitolo 3 fornisce una panoramica sullo stato dell'arte; il capitolo 4 sviluppa il quadro teorico descrivendo sia i modelli astratti che i modelli di risorse gCube; il capitolo 5 presenta il modello transazionale, come può essere utilizzato e perché si concentra sulla tecnica della transazione; il capitolo 6 offre il supporto metodologico principale con la descrizione dell'IS per supportare il modello di risorsa astratto e il modello delle transazioni; infine i capitoli 7 e 8 forniscono la validazione dello studio proposto e il contributo di ricerca.

List of publications

International Journals

1. Assante, M., Candela, L., Castelli, D., Cirillo, R., Coro, G., Frosini, L., Lelii, L., Mangiacrapa, F., Marioli, V., Pagano, P., Panichi, G., Perciante, C., Sinibaldi, F. (2018). The gCube System: Delivering Virtual Research Environments as-a-Service. *Future Generation Computer Systems*. (In Press, Accepted Manuscript, https://doi.org/10.1016/j.future.2018.10.035).

2. Frosini, L., Pagano, P. (2018). A Facet-based Open and Extensible Resource Model for Research Data Infrastructures. *Grey Journal*. (Vol. 14, pp. 69-76).

3. Frosini, L., Bardi, A., Manghi, P., Pagano, P. (2018). An Aggregation Framework for Digital Humanities Infrastructures - The PARTHENOS Experience. *SCIRES-IT - SCIentific RESearch and Information Technology*. (Vol. 8(1), pp. 33–50).

International Conferences

1. Frosini, L., Pagano, P. (2017, October). A Facet-based Open and Extensible Resource Model for Research Data Infrastructures. *19th Conference in Grey Literature (GL 19) – Public Awareness and Access to Grey Literature*. (Vol. 19, pp. 113-120) TextRelease, 2018.

Chapter in Book

1. Frosini L., Paternò, F. (2016). Security in User Interfaces Distributed Amongst Dynamic Sets of Devices and Users. In: Anslow C., Campos P., Jorge J. (eds) Collaboration Meets Interactive Spaces. Springer, Cham.

Others

1. Bardi, A., Frosini, L. (2017,October). Building a federation of digital humanities infrastructures. *ERCIM News*. (Vol. 111, pp. 28-29) ERCIM News.

Technical Reports

1. Candela, L., Frosini, L., Pagano, P. (2016, September). The gCube Information System: Assessment of Enabling Technologies. ISTI Technical Report N. 2016-TR-32.
2. Candela, L., Frosini, L., Pagano, P. (2016, September). Devising a Resource Model for an Open and Heterogeneous Data Infrastructure: the gCube Approach. ISTI Technical Report N. 2016-TR-31.
3. Assante, M., Candela, L., Frosini, L., Lelii, L., Pagano, P., Pieve, A. (2016, September). The gCube Authentication, Authorization, and Accounting Infrastructure. ISTI Technical Report N. 2016-TR-30.

European Community Deliverable

1. Candela, L., Coro, G., Fabriani, P., Frosini, L., Galante, N. A., Giammatteo, G., Pavia, D., Kakaletris, G., Lelii, L., Simi, M., Sinibaldi, F., Koltisida, P. (2018, January) D10.2 - BlueBRIDGE Resources Federation Facilities: Revisited Version. Building Research environments for fostering Innovation, Decision making, Governance and Education to support Blue growth.
2. Frosini, L., Assante, M., Dell'Amico, A., Lelii, L., Candela, L., Pagano, P. (2017, November) D6.3 - Report on the implementation of the Joint Resource Registry (interim). Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization and Synergies.
3. Aloia, N., Candela, L., Debole, F., Frosini, L., Lorenzini, M., Pagano, P. (2017, April) D5.2 - Design of the Joint Resource Registry. Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization and Synergies.
4. Pagano, P., Candela, L., Assante, M., Frosini, L., Manghi, P., Bardi, A., Sinibaldi, F. (2016, October) D6.1 - PARTHENOS Cloud Infrastructure. Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization and Synergies.
5. Assante, M., Candela, L., Frosini, L., Lelii, L., Mangiacrapa, F., Pagano, P. (2016, August) D10.12 - VRE Specification and Software 1. SoBigData Research Infrastructure Social Mining & Big Data Ecosystem.
6. Assante, M., Candela, L., Frosini, L., Lelii, L., Mangiacrapa, F., Pagano, P. (2016, July) D10.5 - SoBigData e-Infrastructure software release 1. SoBigData Research Infrastructure Social Mining & Big Data Ecosystem.
7. Candela, L., Coro, G., Frosini, L., Galante, N. A., Giammatteo, G., Kakaletris, G., Lelii, L., Marioli, V., Sinibaldi, F. (2016, May) D10.1 - BlueBRIDGE Resources Federation Facilities. Building Research environments for fostering Innovation, Decision making, Governance and Education to support Blue growth.
8. Giammatteo, G., Frosini, L., Laskaris, N. (2015, October) D4.1 - Software Release Procedure and Tools. Building Research environments for fostering Innovation, Decision making, Governance and Education to support Blue growth.

Software with DOI

1. Frosini, L. gCube IS Exporter.
   `https://doi.org/10.5281/zenodo.1123117`;
2. Frosini, L. gCube IS Sweeper.
   `https://doi.org/10.5281/zenodo.1123213`;
3. Frosini, L. gCube Information System Model.
   `https://doi.org/10.5281/zenodo.1214231`;
4. Frosini, L. gCube Model.
   `https://doi.org/10.5281/zenodo.1122813`;
5. Frosini, L. gCube Resource Registry.
   `https://doi.org/10.5281/zenodo.1122645`;
6. Frosini, L. gCube Resource Registry API.
   `https://doi.org/10.5281/zenodo.1122957`;
7. Frosini, L. gCube Resource Registry Client.
   `https://doi.org/10.5281/zenodo.1123133`;
8. Frosini, L. gCube Resource Registry Publisher.
   `https://doi.org/10.5281/zenodo.1122781`;
9. Frosini, L. gCube Resource Registry Context Client.
   `https://doi.org/10.5281/zenodo.1122767`;
10. Frosini, L. gCube Resource Registry OrientDB Hooks.
    `https://doi.org/10.5281/zenodo.1122721`.

List of Abbreviations

A	
ABAC	Attribute-Based Access Control. 19
ACID	Atomicity, Isolation, Durability and Consistency. 53, 58, 68, 73, 79, 95, 103, 121
ADC	ATLAS Distributed Computing. 18
AGIS	ATLAS Grid Information System. 13, 18, 19, 21, 22
API	Application Programming Interface. 18, 57, 60, 61, 78, 81–83, 85–93, 96–99, 123
B	
BDII	Berkley Database Information Index. 13, 15–22, 87
BlueBRIDGE	Building Research environments fostering Innovation, Decision making, Governance and Education to support Blue growth. 32, 105, 110–112, 120, 122
BSD	Berkeley Software Distribution. 39
BSON	Binary JSON. 93
C	
CAP	Consistency, Availability, Partitionability. 79, 95
CC	Common Creative. 39
CC-BY	Creative Commons Attribution. 39
CC-BY-ND	Creative Commons Attribution No-Derivatives. 39
CENDARI	Collaborative European Digital Archive Infrastructure. 115
CERN	Conseil Européen pour la Recherche Nucléaire. 18

CLARIN	Common Language Resources and Technologies infrastructures. 115
CMDB	Configuration Management Database. 17
CNR	Consiglio Nazionale delle Ricerche. II, 113
CoL	Catalogue of Life. 111
CPU	Central Processing Unit. 38, 51
CRUD	Create, Read, Update, Delete. 56, 72, 78
CVE	Collaborative Virtual Environment. 6, 7
D	
DARIAH	Digital Research Infrastructure for Arts and Humanities. 115, 116
DDL	Data Definition Language. 78, 83, 85
DHI	Digital Human Infrastructure. 8, 106, 115, 116
DML	Data Manipulation Language. 78, 88
DNS	Domain Name System. 14
DOI	Digital Object Identifier. 30, 39
DTD	Document Type Definition. 33, 41
E	
EGEE	Enabling Grid for E-Science. 15, 18
EGI	European Grid Infrastructure. 8
EHRI	European Holocaust Research Infrastructure. 115, 116
EUPL	European Union Public Licence. 39, 48, 94, 105
F	
FAO	Food and Agriculture Organization. 111
FORTH	Foundation for Research and Technology-Hellas. 32, 116
G	
GBIF	Global Biodiversity Information Facility. 111
GLUE	Grid Laboratory Uniform Environment. 15
GOCDB	Grid Operations Centre Database. 13, 17–19, 21, 22, 94
GPL	GNU General Public License. 39, 48
GPS	Global Positioning System. 52
GSR	Global Service Registry. 18, 19, 21, 22
H	
HA	High-Availability. 14, 35, 79, 80, 103, 104, 121

HATEOAS	Hypermedia As The Engine Of Application State. 55, 68, 71
HDI	Hybrid Data Infrastructure. I–IV, 1–4, 8–13, 15–17, 19, 21–23, 32, 43, 48, 58, 79, 105, 119, 120, 122
HMA	Human Mobility Analytics. 112, 113
HS	Horizontal Scalability. 79, 80, 103, 104, 121
HTTP	HyperText Transfer Protocol. 34, 54, 56–58, 60, 61, 63–65, 67–73, 87, 88, 90, 99, 100, 102, 120
I	
ICES	International Council for the Exploration of the Sea. 112
ICT	Information and Communication Technology. 1, 14, 115
IETF	Internet Engineering Task Force. 41
IP	Internet Protocol. 41, 52
IRI	Internationalised Resource Identifier. 30, 39
IRMNG	Interim Register of Marine and Nonmarine Genera. 111
IS	Information System. II, IV, 2–7, 11–13, 15, 17–24, 26, 32, 51, 53, 54, 58, 76–81, 92, 94, 103–112, 114, 117, 119–123
ISTI	Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo". II
ITIS	Integrated Taxonomic Information System. 111
J	
JISC	Joint Information Systems Committee. 6
JMS	Java Messaging System. 102
JSON	JavaScript Object Notation. 41, 60, 83, 85, 87, 93
L	
LDAP	Lightweight Directory Access Protocol. 15, 17
LDIF	LDAP Data Interchange Format. 15
LGPL	GNU Lesser General Public License. 48
LHC	Large Hadron Collider. 1
LR	Language Resource. 115
LT	Language Technology. 115

N	
NDGF	Nordic Data Grid Facility. 18
O	
OBIS	Ocean Biogeographic Information System. 111
OIM	OSG Information Management System. 18, 19
OLAP	On-Line Analytical Processing. 94
OLTP	On-Line Transaction Processing. 94
OSG	Open Science Grid. 18
P	
PARTHENOS	Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization and Synergies. 32, 34, 43, 49–51, 105, 106, 115–117, 120, 122, 123
POS	Part Of Speech. 115
Q	
QoS	Quality of Service. 14
R	
RBAC	Role-Based Access Control. 95
RDF	Resource Description Framework. 93, 94
RDI	Research Data Infrastructure. 8
REST	REpresentational State Transfer. 18, 19, 54–59, 67–72, 76, 80, 81, 83, 87, 88, 90, 95–99, 103, 120, 121
REST+D	REST with Delegation. 68
RFC	Request For Comments. 56, 57
RI	Research Infrastructure. I, 1, 3, 4, 7, 8, 22, 112, 119, 120
ROA	Resource Oriented Architecture. 54, 56–59, 70, 72, 81, 85, 120
S	
SCI	Scholarly Communication Infrastructure. 8
SD	Social Data. 112, 113
SG	Science Gateway. 6, 7
SNA	Social Network Analysis. 112, 113
SOAP	Simple Object Access Protocol. 19, 20
SoBigData	Social Mining & Big Data Ecosystem. 32, 105, 112–114, 120, 122
SoS	System of Systems. 10, 11, 77, 119
SQL	Structured Query Language. 92, 102

SSH	Social Sciences and Humanities. 115
T	
TGIR	Très Grande Infrastructure de Recherch. 115, 116
TSMM	Text and Social Media Mining. 112, 113
U	
UDDI	Universal Description, Discovery, and Integration. 13, 17, 19, 21, 22
URI	Uniform Resource Identifier. 30, 33, 34, 39, 56–58, 60–65, 67, 70, 72, 73, 81–83, 86–88, 96–101
URL	Uniform Resource Locator. 30, 39, 57, 70, 81, 89–91
URN	Uniform Resource Name. 30, 39
UUID	Universally Unique Identifier. 30, 39, 86, 87, 89–92, 123
V	
VA	Visual Analytics. 112, 113
VM	Virtual Machine. 50, 52, 53
VO	Virtual Organisation. 6, 7, 15, 17, 19, 20, 106, 107, 109–111, 123
VRC	Virtual Research Community. 6, 7
VRE	Virtual Research Environment. III, IV, 2–4, 6–14, 20, 21, 23, 26, 28, 50, 51, 53, 58, 77, 81, 105–107, 110–112, 114, 116, 119, 120, 123
W	
WA	Web Analytics. 112, 113
WADL	Web Application Description Language. 19, 108
WebDAV	Web-based Distributed Authoring and Versioning. 57, 73
WLCG	Worldwide LHC Computing Grid. 19
WMS	Web Map Service. 34
WoRMS	World Register of Marine Species. 111
WSDL	Web Services Description Language. 19, 108
X	
XML	eXtensible Markup Language. 20, 33, 41, 56, 60, 93, 106
XSD	XML Schema Definition. 20, 33, 41

Definitions

Symbols

Collaborative Virtual Environment (CVE)	Collaborative Virtual Environment (CVE) a computer-based, distributed, virtual space or set of places. In such places, people can meet and interact with others, with agents or with virtual objects [113]. 6, 7
Hybrid Data Infrastructure (HDI)	Hybrid Data Infrastructure (HDI) is an e-Infrastructure which leverage external systems (e.g., data, storage services, computational resources, grid and cloud computing infrastructures) by exposing them as a common unified space of resources to serve diverse community of researcher by providing them data, services and sharing tools through a dedicated, flexible, web-based and on-demand environments called Virtual Research Environment (VRE). I–IV, 1–4, 8–13, 15–17, 19, 21–23, 32, 43, 48, 58, 79, 105, 119, 120, 122

Information System (IS)	The Information System (IS) is the key service of HDI. It acts as a registry of the infrastructure, and it offers through either query answering or notifications [13] a global view (infrastructure level) and a partial view (VRE level) of: • its resources (e.g. computing, storage, services, software, datasets); • their current status (e.g. available, down, not responding); • their relationships with other resources; • the policies governing their exploitation. . II, IV, 2–7, 11–13, 15, 17–24, 26, 32, 51, 53, 54, 58, 76–81, 92, 94, 103–112, 114, 117, 119–123
Research Infrastructure (RI)	Research Infrastructures (RIs) are facilities, resources and services used by the science community to conduct research and foster innovation [39]. I, 1, 3, 4, 7, 8, 22, 112, 119, 120
Science Gateway (SG)	Science Gateway (SG) is a community-developed set of tools, applications, and data that are integrated via a portal or a suite of applications, usually in a graphical user interface, that is further customised to meet the needs of a specific community. Gateways enable entire communities of users associated with a common discipline to use national resources through a standard interface that is configured for optimal use. Researchers can focus on their scientific goals and less on assembling the cyberinfrastructure they require. Gateways can also foster collaborations and the exchange of ideas among researchers [119]. 6, 7
Virtual Organisation (VO)	Virtual Organisation (VO) is a group of users who collaborate to solve a problem [61]. 6, 7, 15, 17, 19, 20, 106, 107, 109–111, 123

Virtual Research Community (VRC)	Virtual Research Community (VRC) is a group of researchers, possibly widely dispersed, working together effectively through the use of information and communications technology. Within the community, researchers can collaborate, communicate, share resources, access remote equipment or computers and produce results as effectively as if they, and the resources they require, were physically co-located [23]. 6, 7
Virtual Research Environment (VRE)	Virtual Research Environment (VRE) is a web-based working environments where groups of scientists, possibly geographically distant from each other, have user-friendly, transparent and seamless access to the flexible and shared set of remote resources (data, services and computing capabilities) needed to perform their work collaboratively [35]. III, IV, 2–4, 6–14, 20, 21, 23, 26, 28, 50, 51, 53, 58, 77, 81, 105–107, 110–112, 114, 116, 119, 120, 123
C	
Collaboratory	Collaboratory is an organisational entity that spans distance, supports rich and recurring human interaction oriented to a common research area, and fosters contact between researchers who are both known and unknown to each other, and provides access to data sources, artifacts and tools required to accomplish research tasks [24]. 7
E	
e-Infrastructure	E-Infrastructure is a RI offering digital technologies for data management, computing and networking. 1–3, 8, 9

Contents

CHAPTER 1

Introduction

Research data play a key role in our society. They are used in every scientific discipline to produce research results, support the definition of new theories, extract information, create visual representations, and much more.

Research Infrastructures (RIs) have been created to provide scientists with a research environment containing all the facilities and resources they require to do their job. The Research and Innovation unit of the European Commission defines RIs as "facilities, resources and services used by the science community to conduct research and foster innovation" [39].

In recent years, every research discipline has collected an increasing amount of data originated from the most disparate sources (e.g., electronic sensors, the Large Hadron Collider (LHC), citizen science) which require digital services to store, organise, manage, discovery, access, analyse, and visualise them. RIs providing digital services are referred as *e-Infrastructures* [76].

The explosion of telecommunication facilities and computing e-Infrastructures in 2000s provided a solid ground for the management of a considerable amount of data. Grid and Cloud computing are successful examples of e-Infrastructures. Unfortunately, the complexity of using and managing such e-Infrastructures limited their adoption among computer scientists and members of the Information and Communication Technology (ICT) community. Moreover, heterogeneous and relatively small groups of researchers of other disciplines built their own tailored solutions with the aim to realise a more comfortable environment to use.

During the past 10 years, the novel concept of *Hybrid Data Infrastructure (HDI)* has emerged to describe the richness of the several existing distributed solutions while promoting a more efficient exploitation of the available resources [33]. HDI is an e-Infrastructure which federates different typologies of ICT systems (e.g., data, storage

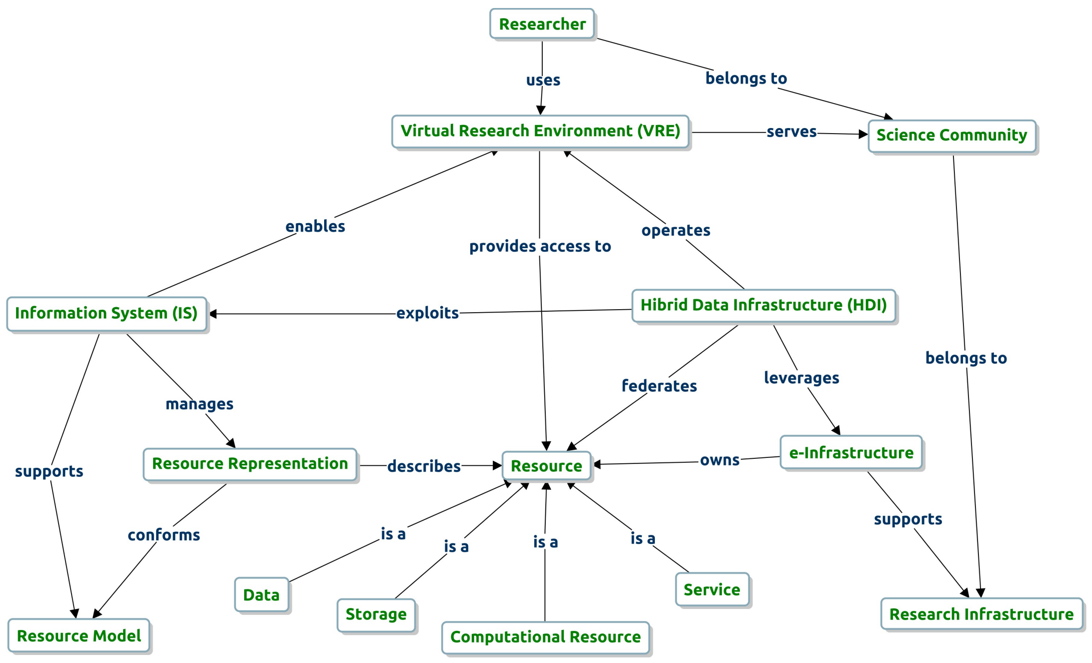


Figure 1.1: *Map of the main concepts involved in the dissertation.*

services, computational resources, grid and cloud computing infrastructures) to offer to scientists, of the most disparate disciplines, web-based environments called *Virtual Research Environments (VREs)* [35].

Nowadays, science is an interdisciplinary and collaborative effort and researchers must be able to work together across physical and administrative borders on a daily basis. For this reason, VREs not only offer efficient access to resources operated by different providers, they also integrate user-friendly facilities enabling scientific collaboration among colleagues [15,26]. By offering features like distributed data storage, replication and preservation, application deployment and scalability are provided as-service by the HDI, allowing researchers to focus on their scientific problems and data analysis activities [16].

To properly operate and provision all the federated services and technologies, the HDI needs an inventory of all the resources. The inventory first requires a description of the resources provided by the different systems. The set of all the resource descriptions creates a "common unified space of resources" managed through a software component named Information System (IS). The IS plays a key role in the e-Infrastructure supporting the creation, discovery, access, and management of all the resources and the relations created within the e-Infrastructure [14].

These resources are exposed to different VREs according to the requirements and constraints expressed by the VRE community.

Figure 1.1 illustrates the map of concepts object of my research activities by highlighting the main relationships among them[1].

[1]A concept map is an informal graphical way to illustrate key concepts and relationships among them. A concept identifies a class of objects that we expect to be able to identify in the proposed context. The precise “form” of concept may be different in diverse implementations, but the presence of the concept tells us what to look for in a given concrete scenario rather than prescribing its precise form. This formalism offers a practical method to represent complex information in a compact and easy-to-understand way [102].

It is important to remark that the HDI is not a static federation of systems: the providers of such systems, the description of the resources, their sharing policies (among the others) can change over time. The IS must support dynamic changes to the unified space of resources to respond to the evolution of the federation.

The analysis of the state-of-the-art ISs revealed open issues related to their exploitation in HDIs: (i) the support to the evolution of the representation of the resources; (ii) the capabilities of providing different coherent and consistent views on the common unified space of resources; (iii) the scalability of the software system.

The work presented here contributes to address these issues and proposes solutions by designing an IS able to adapt to the evolving needs of the e-Infrastructure. This research proposal includes:

- the definition of a Resource Model to represent the resources of a federated RIs (see chapter 4). Such a model is open by design and consists of versatile constructs that can be instantiated to serve the needs of actual scenarios. A concrete instance this model (named *gCube Model*) is also presented as proof of concept. The gCube Model has been validated within three different operational environments (see chapter 7);

- a complete design of an IS managing the defined Resource Model and supporting the creation and management of VREs in federated e-Infrastructures. The IS is based on fully-compliant REST (i.e., RESTful) services that enable simplified access to the resources from any (no matter the programming language) service, tool, or application joining the federation (see chapter 6);

- a RESTful Transaction Model to keep the IS service stateless while providing consistency, atomicity and isolation across operations made by concurrent actors (see chapter 5).

This dissertation is organised in eight chapters.

Chapter 1 (this chapter) introduces the dissertation by outlining the motivations.

Chapter 2 presents the context of study by analysing three research contexts, i.e., Information Systems, Research Infrastructures, Virtual Research Environments.

Chapter 3 reviews the state-of-the-art highlighting the current limitations in our specific context. The chapter introduces the challenges, the characteristics of the data model and the set of functionalities that guided the solutions presented in this work.

Chapter 4 defines a resource model as a fundamental part of the design of the Information System and an instance of the model used and validated in concrete scenarios.

Chapter 5 describes a resilient RESTful Transaction Model based on two-phase locking approach.

Chapter 6 delivers a complete design for the Information System, which apart the use of the proposed Resource Model provides (i) a multi-tenant approach with cross-context consistency to support the management of VREs; (ii) a REST compliant architecture; (iii) a specialisation of the RESTful Transaction Model to support the multi-context approach.

Chapter 7 discusses real scenarios used to design and validate this research proposal.

Finally, Chapter 8 summarises the dissertation by reporting the most relevant achievements of this work.

CHAPTER 2

Context of Study

The goal of this study is to describe an information system enabling the operation of any research infrastructure supporting virtual research environments. Preliminary for this work was a clear and deep understanding of each research context in order to define the characteristics and the impact of composing and aggregating such contexts.

To clarify our boundaries, we first discuss the main characteristics of such a system through the analysis of its definitions; then, we contextualise the exploitation of the Information System (IS) in Virtual Research Environments (VREs) operated by federated infrastructures.

This chapter is organised as follows. Section 2.1 presents the general definitions and characteristics of IS. Section 2.2 describes the concept and the characteristics of VRE. Section 2.3 introduces the characteristics of the Research Infrastructure (RI) and presents how they can support the VRE. Section 2.4 focuses on the federation of RI by defining the Hybrid Data Infrastructure (HDI). Finally, section 2.5 presents the characteristics of an IS for HDIs highlighting the specific challenges and constraints to design it.

2.1 Information Systems

The earliest definitions of Information System (IS) go back to 1970s. Carvalho [37] proposed a survey of the definitions which allows highlighting the meaning of the IS and what are the concepts behind it. Nowadays, we can find similar definitions to the one proposed in the literature in dictionaries and online encyclopedias. Table 2.1 reports the most relevant definitions for the context of this dissertation.

Author(s)	Definition
Literature	
Davis [46] (1986)	"an integrated man/machine system for providing information to support the operations, management and decision making functions in an organisation. The system uses computer hardware, software, manual procedures, management and decision models and a data"
Avison and Wood-Harper [18] (1986)	"a system to collect, process, store, transmit, and display"
Buckingham et al. [31] (1986)	"a system which assembles, stores, processes and delivers information relevant to an organisation (or to society), in such a way that the information is accessible and useful to those who wish to use it"
Hirschheim et al. [83] (1995)	"a collection of people, processes, data, models, technology and partly formalised language, forming a cohesive structure which serves some organisational purpose or function"
Alter [5] (1996)	"a system that uses information technology to capture, transmit, store, retrieve, manipulate, or display information used in one or more business processes"
Falkenberg et al. [57] (1998)	"an organisational system, being normally dynamic, active and open, and comprising the conception of how an organisation is composed (i.e. of specific actors and actands) and how it operates (i.e. performing specific actions in pursuit of organisational goals, guided by organisational rules and informed by internal and external communication), where its systemic properties are that it responds to (certain kinds of) changes caused by the system environment and, itself, causes (certain kinds of) changes in the system environment"
Dictionaries and Encyclopedias	
Business Dictionary [49] (2018)	"a combination of hardware, software, infrastructure and trained personnel organised to facilitate planning, control, coordination, and decision making in an organisation"
Encyclopedia Britannica [28] (2018)	"an integrated set of components for collecting, storing, and processing data and for providing information, knowledge, and digital products"
Wikipedia [40] (2018)	"any organised system for the collection, organisation, storage and communication of information"

Table 2.1: *Information System definitions*

As the term IS is composed of two words, it is essential to derive the meaning of each of the words from the reported definitions.

All the definitions convey on the characteristics of *information*. Information consists of data that

- is accurate and timely;
- is specific and organised for a purpose;
- is available within a context that gives it meaning and relevance;
- can increase understanding and decrease uncertainty.

While the term *information* provides a set of quality properties, the term *system* tells which are the aspects enabling the support to these qualities:

- human or machine resources;

- procedures for using, operating, and maintaining the system;
- set of basic principles and associated guidelines (e.g., policies) formulated and enforced to direct and limit actions in pursuit of long-term goals.

In our context, the *system* is a specialised software system [57] which has the following characteristics derived from Carvalho categorisations [37]:

- capture, transmit, store, retrieve, and manipulate information produced by other software systems;
- provide access to information organised for a purpose and within a contextual domain;
- preserve information according to well-known procedures operated under the limit of the (evolving) organisation policies;
- support people within an organisation and other software systems.

The next section describes the concept and the characteristics of Virtual Research Environment (VRE).

2.2 Virtual Research Environments

Differently from the Information System (IS), the term Virtual Research Environment (VRE) appears in the literature starting from 2000s.

Candela et al. [35] defined VREs as

> web-based working environments where groups of scientists, possibly geographically distant from each other, have user-friendly, transparent and seamless access to the flexible and shared set of remote resources (data, services and computing capabilities) needed to perform their work collaboratively.

The definition is similar to the description provided by Joint Information Systems Committees (JISCs) [1].

> The purpose of a VRE is to help researchers from all disciplines to work collaboratively by managing the increasingly complex range of tasks involved in carrying out research on both small and large scales. The concept of a VRE is evolving. The term VRE is now best thought of as shorthand for the tools and technologies needed by researchers to do their research, interact with other researchers (who may come from different disciplines, institutions or even countries) and to make use of resources and technical infrastructures available both locally and nationally. The term VRE also incorporates the context in which those tools and technologies are used. The detailed design of a VRE will depend on many factors including discipline, context, and security requirements [87].

Carusi and Reimer [36] evidenced the existence of other terms referring to an online collaborative environment for research such as e-Infrastructure/cyberinfrastructure, Collaborative e-Research Communities, Collaborative Virtual Environment (CVE), Collaboratory (formed by the union of the words 'collaborate' and 'laboratory'), Science Gateway (SG), Virtual Organisation (VO), Virtual Research Community (VRC).

[1]`https://www.jisc.ac.uk/`

Name	Definition
CVE - Snowdon et al. [113] (2001)	a computer-based, distributed, virtual space or set of places. In such places, people can meet and interact with others, with agents or with virtual objects
Collaboratory - Bos et al. [24] (2007)	an organisational entity that spans distance, supports rich and recurring human interaction oriented to a common research area, and fosters contact between researchers who are both known and unknown to each other, and provides access to data sources, artifacts and tools required to accomplish research tasks
Collaboratory - Vaart et al. [116] (2010)	a Collaboratory, or virtual research workplace/environment, is a web-based collaboration environment for researchers
SG XSEDE organisation [119] (2014)	a Science Gateway is a community-developed set of tools, applications, and data that are integrated via a portal or a suite of applications, usually in a graphical user interface, that is further customised to meet the needs of a specific community. Gateways enable entire communities of users associated with a common discipline to use national resources through a standard interface that is configured for optimal use. Researchers can focus on their scientific goals and less on assembling the cyberinfrastructure they require. Gateways can also foster collaborations and the exchange of ideas among researchers
VO - Field et al. [61] (2009)	a group of users who collaborate to solve a problem
VRC - Borda et al. [23] (2006)	a group of researchers, possibly widely dispersed, working together effectively through the use of information and communications technology. Within the community, researchers can collaborate, communicate, share resources, access remote equipment or computers and produce results as effectively as if they, and the resources they require, were physically co-located.

Table 2.2: *VRE synonyms*

Table 2.2 shows some of the synonyms of VRE and their definitions. Even if each of them tries to evidence specific aspects, all of them converge to a common set of features [36] which allow defining a VRE as:

a web-based environment supporting and promoting cooperation and collaboration between researchers of the same or different institutions at any level of the institutions hierarchy (e.g., research associate, associate professors, full professor), and providing facilities to preserve data and output[2].

From the analysis of the VRE, it is possible to identify the information to be captured by the IS to enable the support of VREs:

- data;
- researchers and their organisations;
- electronic services (to provide web access, to analyse and preserve data, to enable the researchers to collaborate and communicate with each other).

The next section clarifies what a Research Infrastructure (RI) is, and what are the relations between RIs and VREs.

[2]An output is a form of data, but it has been explicitly added to highlight that the outcome of any research activity becomes an asset of the VRE that has to be preserved and maintained as the primary data.

2.3 Research Infrastructures

The Research and Innovation unit of the European Commission defines a Research Infrastructures (RIs) as “facilities, resources and services used by the science community to conduct research and foster innovation [39]”.

RIs offering digital technologies for data management, computing and networking are called e-Infrastructures. In the rest of this dissertation the terms RI and e-Infrastructure are used interchangeably. Relevant, but not exhaustive, examples are high speed connectivity infrastructures (e.g., GÈANT[3]), grid and cloud computing infrastructures (e.g., European Grid Infrastructure (EGI))[4]), Scholarly Communication Infrastructures (SCIs) (e.g., OpenAIRE[5]), Digital Human Infrastructures (DHIs) (e.g., PARTHENOS[6]), Research Data Infrastructures (RDIs) (e.g., D4Science[7]).

Not all the e-Infrastructures can be considered at the same architectural level since some of them enable the creation or the operation of the others. For example, high-speed connectivity infrastructures are essential facilities for grid and cloud computing infrastructures [20]; grid and cloud computing infrastructures provide excellent support to address data/computation intensive paradigm hence they are often used as facilities to build SCIs, DHIs and RDIs.

In the past, the term e-Infrastructure has been used as a synonym of Virtual Research Environment (VRE), but there is a substantial difference between them. e-Infrastructures can be considered facilities to build VREs or as counterpart a VRE can be seen as an interface to a e-Infrastructure. Literature clarifies this distinction.

For example, Carusi and Reimer [36] denotes that, while the term ’e-Infrastructure’ (or ’cyberinfrastructure’ in the US) refers to the digital side of research infrastructures, VREs are an interface to that infrastructure allowing access to data and services in an environment designed for a particular research activity.

Van der Vaart [116] evidences that while infrastructure aims at providing solutions for handling research data and information, Collaboratories are virtual locations where researchers can work together as part of the e-infrastructure. As part of a wider e-infrastructure, Collaboratories need to connect up with other systems and tools, within and outside the parent institution(s). These places demand on systems with respect to options for interoperability or integration.

Substantially, a VRE is a view built on the e-Infrastructure resources enabling the collaboration among scientists through a web-based environment. Hence, an information system for VREs operated by e-Infrastructures has to represent the information describing all the e-Infrastructure resources, and it has to enable the creation of views representing the VREs.

The next section presents Hybrid Data Infrastructure (HDI) which is as a particular type of RDI enabling the creation of multiple VREs.

[3] `https://www.geant.org/`
[4] `https://www.egi.eu/`
[5] `https://www.openaire.eu/`
[6] `http://www.parthenos-project.eu/`
[7] `https://www.d4science.org/`

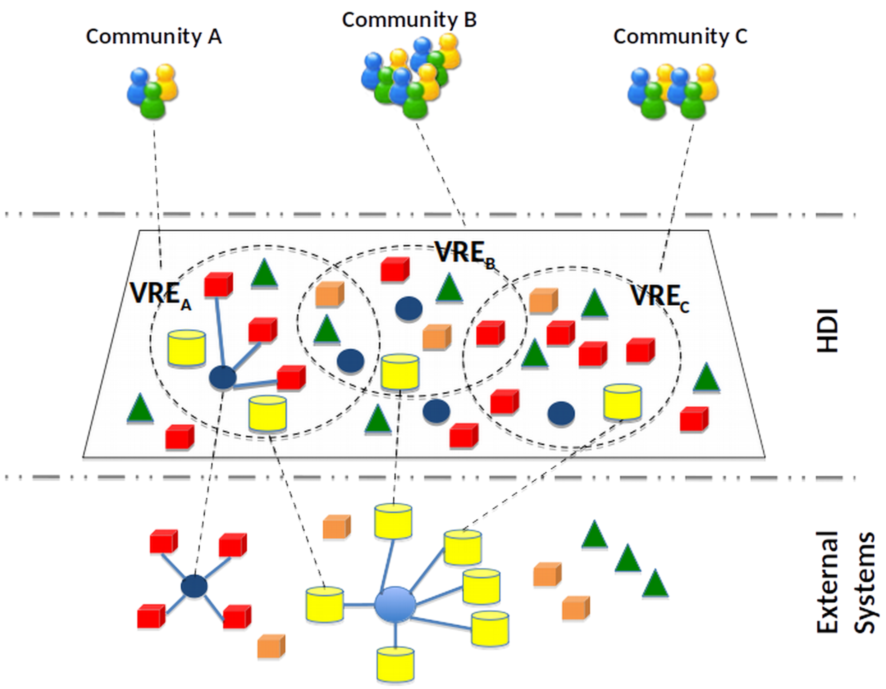


Figure 2.1: *HDI conceptual representation. The intermediate level shows the HDI resources composition (i.e., a wide set or resources). Some of resources composing the HDI are federated from to external systems (represented in the layer below). The dashed circle represents the set of resources which are available to a VRE. The VREs share some resources. Each VRE is represented as a "view" (represented as dashed circles) in then whole HDI. Each VRE corresponds to a community.*

2.4 Hybrid Data Infrastructures

Definition 2.4.1. *An Hybrid Data Infrastructure (HDI) is an e-Infrastructure which leverages external systems (e.g., data, storage services, computational resources, grid and cloud computing infrastructures) to:*

- *expose them as a common unified resource space;*
- *serve heterogeneous communities of researchers;*
- *provide access to data, services and sharing tools through a dedicated, flexible, web-based and on-demand environments called Virtual Research Environment (VRE).*

Figure 2.1 shows the conceptual representation of an HDI where the resources federated from external systems became part of the common unified space of resources. Different systems can contribute to the HDI by providing either diverse type of resources or resources of the same type. In both cases, the resources are contributed by systems operating under different administrative domains.

These resources are used by the HDI to build VREs. The VRE is a view on the common unified space of resources. This view makes available a subset of them to a specific research community. The research community can vary in the number of members (people) and organisations.

Any resource of the common unified space can belong to different VREs, as well as people belonging to different communities can join the same VREs.

The resources made accessible through a VRE can vary in numbers and typologies depending on many factors such as the number of members and the business agreement between the community and the infrastructure committee. Moreover, the resources assigned to a VRE can vary in number and types over the time such as the number of leveraged external systems.

HDI is a novel concept which does not find any reference in literature, but analysing the definition (see definition 2.4.1) it is possible to re-conduce it to a System of Systems (SoS).

Nielsen et al. [100] defined a SoS as "system composed of independent constituent systems, which act jointly towards a common goal through the synergism between them". In their survey, they evidenced the eight dimensions characterising a SoS:

Autonomy of Constituents: each constituent system behave with its own rules;

Independence: each constituent system is capable of operating autonomously if detached from the others;

Distribution: constituent systems are dispersed so that they require some form of connectivity to create a synergy;

Evolution: each constituent system evolves independently from the others;

Interoperability: ability of the SoS to incorporate different constituent systems;

Dynamic Reconfiguration: represent the adaptability property of a SoS which allow to support changes in compositions and structure of the constituent systems with no planned intervention;

Emergence of Behaviour: SoS provides higher functionality than the constituent systems as result of the synergy between them;

Interdependence: any constituent system which objective depends on the SoS have to sacrifice its degrees of independence to meet the requirements of joining the SoS.

Autonomy of constituents, independence, distribution, and evolution are properties characterising the constituent systems of a SoS, while dynamic reconfiguration, interoperability, the emergence of behaviour, and interdependence emerging by the creation of the SoS.

The primary objective of any HDI is to provide higher functionality to VRE users (Emergence of Behaviour) by composing different systems. We can affirm that this emerging of behaviour property is the reason why different organisations decide to join the federation i.e., to allow their member to access to broader and higher functionality by providing as a counterpart the resources they own.

Joining the federation could sacrifice the independence of the system as less as the whole systems have mechanisms to support differences. When an external system decides to join the federation, is because it considers the cost of joining less than the benefit received for its members.

An Information System (IS) for an HDI must be capable of representing such heterogeneity and dynamism while enabling the different perspectives offered through the VRE.

Next section presents the characteristics of an IS for an HDI.

2.5 An Information System for Hybrid Data Infrastructures

As reported in section 2.1, the purpose of the Information System (IS) is to provide access to well-organised information. In the context of Hybrid Data Infrastructure (HDI), such information is related to the set of resources describing the infrastructure. Some of them are core resources of the infrastructure, others are leveraged from external systems. Hence the IS has to deal with:

- different set of managed resources: each system composing the federation defines the resources it wishes to contribute to the HDI (elicited by Autonomy of Constituents, Independence);
- different models for describing resource characteristics: each system has its own model to define a resource (elicited by the Autonomy of Constituents, Independence). This mainly means that the same resource type may be described differently by the different organisations contributing to the HDI;
- different workflows governing registration and update of resources: each system may update any resource description at different time-frame and according to different policies (elicited by Evolution).

An information system capable of supporting such challenges does not only address the interoperability dimension required for a System of Systems (SoS) but also reduces the costs for a system that wants to join the federation since such a system needs neither to conform to a particular model for describing the resources nor modify its policies. Hence, this enhances the possibility of a system to join the HDI and to enrich the set of capabilities it offers.

As reported in section 2.1, the way the IS provides access to the information is another important aspect. In any HDI, the subset of resources exposed to a Virtual Research Environment (VRE) are usually different from the one exposed to another VRE. The subset of resources of a VRE can intersect (and usually it does it) the subset of resources of another VRE. A change occurred to one or more resources exposed to different VREs must be reflected consistently across all VREs (accuracy and timely see section 2.1).

The way resources are exploited in different VREs can vary according to the needs of the VRE. This also implies that different dependencies among the same set of resources can be established in different VREs. Hence, it is essential representing the relationship between resources as it is essential representing the resources. We can affirm that in an IS for an HDI, the common resource space of the infrastructure organise the resources for the purpose of providing views to VREs (see section 2.1). Instead, the context where these resources receive enhanced relevance and meaning (see section 2.1) (Emergence of Behaviour, see section 2.4) is the VREs thanks to the relationships created among them.

The IS must provide different perspectives to the different actors depending on the actor context. These different perspectives provide the infrastructure understanding which decreases infrastructure uncertainty (see section 2.1).

To address all these challenges, we need

- a flexible model capable of supporting:
  - an open-ended set of manageable resources and relationship among them;
  - an open-ended model for describing resources;
  - the ability to evolve with the evolving needs of the infrastructure at no cost for its clients:
    - by supporting new resource types;
    - by supporting evolution in the way a resource is described;
    - by supporting the same resource type described by different models.
- a software system capable of:
  - supporting a flexible model;
  - providing coherent and consistent global and partial views of the resources and their relationships transparently to the clients (the perspective of the client depends on the context the client is running and never by a filter, a parameter or a configuration);
  - supporting consistent updates across the provided views;
  - scaling according to the workload.

We can conclude affirming that an IS for HDI is the key service for any research infrastructure federation. It acts as a registry of the infrastructure, and it offers through either query answering or notifications [13] a global view (infrastructure level) and a partial view (VRE level) of:

- its resources (e.g., computing, storage, services, software, datasets);
- their current status (e.g., available, down, not responding);
- their relationships with other resources;
- the policies governing their exploitation.

The next chapter analyses the state of the art concerning the presented challenges.

CHAPTER 3

State of the Art

An Hybrid Data Infrastructure (HDI) federates systems usually based on a service-oriented architecture that manage either own resources or resources provisioned by cloud and grid computing infrastructures or both. Giving that HDI is a novel concept with poor references in literature, the analysis of the state-of-the-art has been conducted looking for Information Systems (ISs) designed to operate in Cloud, GRID, and federated systems.

As explained in the previous chapter, an IS enabling Virtual Research Environments (VREs) on HDI has to (i) maintain a description of all the resources of the infrastructure with a flexible model; (ii) provide the view of the resources assigned to a VRE and their relationship; (iii) support consistency across different views; and (iv) support workload changes.

This analysis of the state-of-the-art briefly presents the existing solutions and then it; (i) evidences the set of managed resources; (ii) verifies the capability of satisfying the requirements.

This chapter is organised as follows. Section 3.1 presents an overview of the information technologies used in the context of cloud computing. Section 3.2 presents Berkley Database Information Index (BDII). BDII is the most important information system in grid computing. Section 3.3 presents Universal Description, Discovery, and Integration (UDDI). Sections 3.4, 3.5 and 3.6 presents respectively Grid Operations Centre Database (GOCDB), ATLAS Grid Information System (AGIS), and Global Service Registry which are ISs designed to address the limitation of BDII. Finally, section 3.7 presents the gCube Information System V.1 which is the ancestor of the presented research proposal.

3.1 Cloud Computing Information Systems

Different solutions have been proposed from industrial and scientific communities to support cloud infrastructures. Approaches range from the one designed for a specific purpose (e.g., configuration coordination services [112], message queuing systems[1] and leader election services [110]) to more complex Information and Communication Technology (ICT) solutions which integrate one or more of these functionalities e.g., Zookeeper [2], Apache Mesos [3], Etcd [4].

In the context of Cloud Computing, service discovery technologies provide solutions supporting load balancing, fail-over recovery, and High-Availability (HA). Examples of service discovery technologies are Eureka[5]; Mesos-DNS[6] which adds service discovery capability to Apache Mesos by using the Domain Name System (DNS); Consul; Kubernetes and others.

Consul [7] provides service discovery, configuration and segmentation. Service discovery is used for load balancing in conjunction with health checking to provide HA [120]. Service segmentation is used to restrict and regulate the communication between services by using service policies implemented through identity and authorisation. Consul proved to support the description, the discovery, and the access to service descriptions, but it does not support to register and manage entity different from services, and it does not support the definition and management of multiple operational contexts.

Kubernetes [8] is a tool for automating deployment, scaling, and management of containerised applications. Kubernetes can automatically manage service scalability (up and down) by monitoring the service metrics and adding/removing redundant service instances. Service discovery is supported to enable load balancing between instances. Kubernetes is suitable for hybrid cloud infrastructure (mix of public and private cloud infrastructures) but it does not provide the support for managing the federation of diverse systems, and for supporting virtual views of the federated resource (i.e., the VRE).

All the analysed technologies are designed for cloud computing infrastructures and support service configuration, service coordination and orchestration, leader elections and distributed consensus protocol. Services discovery services are designed to provide HA and load balancing over dynamically discovered service instances. HA is achieved through services monitoring and health checking in conjunction with the capability of spanning new service instances which in some cases is used to provide the required Quality of Service (QoS).

The resources managed by these technologies are services, configurations, virtual machines and containers (i.e., a running computer program responsible for managing the lifecycle of another program).

Providing access to resource descriptions is not the focus of these technologies. Even if they provide a way to store and retrieve different information regarding the

[1]`https://aws.amazon.com/sqs/`
[2]`https://zookeeper.apache.org/`
[3]`https://mesos.apache.org/`
[4]`https://coreos.com/etcd/docs/latest/`
[5]`https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance`
[6]`https://mesosphere.github.io/mesos-dns/`
[7]`https://www.consul.io/`
[8]`https://kubernetes.io/`

various resources they support, they are designed to support load balancing, elasticity, scalability, and high-availability. The way the resources are represented (i.e., the resource model), the typology of access and the provided views are optimised to support the above specific facilities. Hence, they do not fit the context of the study. The information system proposed in this dissertation, instead, is focused on the way to provide correct, coherent and consistent access to and management of resource descriptions.

The analysis of these technologies has been essential to identify the characteristic of the resources to report in the resource description giving that cloud computing infrastructure may be part of the HDI federation.

3.2 Berkley Database Information Index (BDII)

BDII is an IS for Grid Computing infrastructures based on Lightweight Directory Access Protocol (LDAP)[9] [111].

BDII maintains information about static and dynamic grid resources which are heterogeneous and span multiple trust domains i.e., VO.

Many information providers update the information contained in BDII. An information provider[10] is implemented via a script that obtains static information from a configuration file and dynamic information about local services [62]. Periodically, the BDII queries information providers which compare the content of the database with the obtained information to create a LDAP Data Interchange Format (LDIF) [82] file of difference. BDII uses the LDIF file to populate/update the LDAP database. The resource model of BDII is the Grid Laboratory Uniform Environment (GLUE) schema [7].

BDII has been developed in the context of Enabling Grid for E-Science (EGEE) (EGEE) project which led to the creation of the homonym grid infrastructure. In EGEE, BDII is deployed at site (i.e., datacenter) level and at (infrastructure) top level. Instances at infrastructure top level are used to aggregate information from all sites participating in the EGEE infrastructure offering a global overview of all the resources available in the infrastructure. Multiple instances of BDII at infrastructure level are deployed to reduce the query load on each instance and to enable fault tolerance for clients that have implemented a fail-over mechanism [63].

In EGEE, several Glue Schema have been defined to capture the different resources available through the infrastructure (including their state and Virtual Organisation (VO)-specific views). The main represented resources are Computing Element (i.e., Glue CE), Storage Element (i.e., Glue SE), Service (i.e., Glue Service), Site (i.e., Glue Site), Cluster (i.e., Glue Cluster), Software (i.e., Glue Software) and GlueCESEBind. GlueCESEBind is a binding schema representing a mean for advertising relationships between an instance of a Computing Element with an instance of a Storage Element (or several Storage Elements).

In the context of EGEE, there is a central authority which defines the schema, and the sites joining the infrastructure must ensure to adhere to the specification. It is a federation of systems of the same "nature" (i.e., grid sites). When the central authority decides to change schema specification or the publishing policy, the sites are enforced to

[9]LDAP is a distributed directory information services

[10] A list of providers is available at `https://twiki.cern.ch/twiki/bin/view/EGEE/ISproviders` [Date Accessed May 2018]

comply with the new rules. A site is de-facto a separated grid infrastructure physically located in a remote location.

Looking at different Glue Schema specifications[11] is possible to find different specifications for the same resources and different resources between the specifications. As expected, a resource profile changes during the time. Burke et al. [32] regarding the schema in the LCG/EGEE production grid dedicated a paper to highlight the complexity of “having a clear definition of schema attributes, the construction of standard information providers and difficulties encountered in mapping an abstract schema to diverse real systems”.

Field et al. [61] highlighted as the most effort on standardisation to support interoperability of different Grid infrastructures was related to the definition of the Glue Schema (which required "several years"). In the same paper, the authors also highlighted that updates on standardisation for interoperability could have an infrastructure-wide impact especially affecting information system and the security infrastructure. They indicate the "interoperability tests" as the way to manage such updates and maintain the infrastructure operation.

Field et al. [61] evidenced that despite the effort for the standardisation, every federated infrastructure evolved differently both on the middleware as well as on the operational side: “The main reason is that Grid infrastructures use different interfaces/services and different operational procedures”. The authors also highlighted that

> While seamless resource usage is possible within a certain infrastructure, cross-infrastructure usage is still a problem and as such the original problem has not been solved but merely turned into the Grid Interoperation problem. The main reason why the original problem remains unsolved is that many different solutions exist, for a variety of reasons.

In the context of the HDIs, the situation is even more complicated. The infrastructure includes systems of diverse nature which continuously, even slowly, join and leave the infrastructure. It is not possible waiting for the required "several years". The federating infrastructure (i.e., the HDI) cannot impose any specification to adhere. Instead, each federated system (i) is autonomous (Autonomy of Constituents see 2.4); (ii) defines independently from others the resources to expose and the properties to describe such a resource (Independence see 2.4); (iii) changes independently from the other systems the schema and types of resources (Evolution see 2.4) It is in charge of the HDIs to dynamically support the system evolution (Dynamic Reconfiguration see 2.4).

BDII has some scalability issues causing the infrastructure to slow-down while the infrastructure grows [63]. Field et al. [60,63] indicated two solutions to solve or at least mitigates the scalability issues; (i) differential updates, i.e., updates involving only the

[11] Example links of different online documentation (available in September 2018) are:

- `https://documents.egi.eu/public/ShowDocument?docid=1324`
- `https://grid-deployment.web.cern.ch/grid-deployment/eis/docs/LCG-2-UserGuide/LCG-2-UserGuide.html`
- `https://www.nikhef.nl/grid/use/tutorial-june04.pdf`
- `https://edms.cern.ch/ui/file/454439/2/LCG-2-UserGuide.html`
- `https://twiki.cern.ch/twiki/bin/view/EGEE/GlueUse`
- `http://glite.web.cern.ch/glite/documentation/`

information which changes and not the whole resource representation (ii) publish-subscribe mechanism to avoid a continuous client polling to services which contribute to reducing the workload and permits a decoupling from the IS implementation.

Concluding, BDII provides service registration enabling service discovery, but the Glue Schema does not provide adequate support for evolving service description. It supports infrastructure and VO views, but it is in charge of the client to filter the resources to obtain the contextual views. Finally, the relations between resources and contexts are only virtual with no support for referential integrity.

In the next section is presented the distributed registry UDDI and the integration with BDII to enhance the latter with service description capabilities.

3.3 Universal Description, Discovery, and Integration (UDDI)

UDDI [115] is a specification for distributed registries which allows dynamic discovery of services and their description. UDDI supports registry federation. Every registry of the federation replicates the information contained in the other federated registry.

Pastore et al. [106] proposed a solution to integrate UDDI as a grid resource. Becoming a new kind of grid resource, UDDI is accessible through the grid information system BDII. The proposed approach serves to support the integration of the Service-Oriented Computing model as a "paradigm for distributed computing and e-business processing in grid environments" [106]. Such an integrated framework add to the grid middleware a facility for coordinating, discovering and reusing web services while tailoring them to the member of the VO [66].

The proposed integration shows the feasibility to add higher level service discovery and description facility to BDII by leveraging the LDAP schema for UDDI specification [99].

Even considering that BDII + UDDI could provide a broader range of resource types to federate research infrastructure, the rigid resource model of BDII poses severe issues on using such systems for HDI.

3.4 Grid Operations Centre Database (GOCDB)

GOCDB is a Configuration Management Database (CMDB) to store and manage resources of e-Infrastructure projects. It is mainly used from EGI.eu[12] and EUDAT[13] projects to define resources topology. The resource types range from projects, sites, groups, users and their roles, services-groups, services-endpoints and downtimes [96]. It allows creating relations between managed resources. Resource profiles are well-constrained and reflect a subset of the information model defined in the GLUE Specification v. 2.0.

GOCDB provides a scope-tagging[14] mechanism which allows grouping resources in many categories. It avoids duplication of information and guarantees the integrity of the topology across different infrastructures and projects. The scope-tags allow for example to identify resources used by different projects, or to identify infrastructure grouping. The system reserves some scope-tag for internal use by denying the clients

[12]`http://www.egi.eu`

[13]`http://www.eudat.eu`

[14]In GOCDB terminology, the term 'scope' can be considered a synonym of 'context'

to use them. The scope-tagging does not realise an authorisation environment. Rather, it remains under the responsibility of the client to filter by scope-tag to get the view of a specific scope.

GOCDB provides a complex role-based permission control[15] which allows a user to perform specific operation rather than others depending on the owned role. GOCDB uses relational database as persistence and it exposes REpresentational State Transfer (REST) Application Programming Interfaces (APIs) to allow other infrastructure services to retrieve topology information.

GOCDB has been designed, among other things, to provide resource relations (resources topology) and scope views. The integrity between different scopes is guaranteed while discovering and accessing a specific scope. GOCDB manages a pre-defined set of resource types, and the resource schema is rigid.

3.5 ATLAS Grid Information System (AGIS)

ATLAS is a particle physics experiment at the at Conseil Européen pour la Recherche Nucléaire (CERN). AGIS has been developed to store data extracted from the different sub-grids composing the ATLAS grid infrastructure i.e., EGEE, Nordic Data Grid Facility (NDGF), Open Science Grid (OSG). AGIS provides configuration parameters and topology to properly operate ATLAS Distributed Computing (ADC) systems and services [10–12]

Specific agents collect static information from different information systems like BDII, GOCDB, and OSG Information Management System (OIM)[16].

AGIS helps to solve; (i) information duplication issues; (ii) inconsistencies between data stored in different information systems, configuration files, and hard-coded applications; (iii) synchronisation issues. It allows to reduce code duplication and simplify application code logic [12].

AGIS facilitates, enables, and defines missing relationships between computing services 'provided by' a site and the services 'used by' an experiment. AGIS allows the experiments to define their actual organisation of the resources. AGIS exposes REST APIs to allow ADC applications and services to retrieve the infrastructure topology and define the missing relationships required by the experiments [11].

AGIS exposes the resources provided by the grid federation as a common space, and it allows to create relations among them for the ADC applications and services. The capability of creating a common unified space from different types of information systems, the possibility of creating relations among the exposed resources are essential features for the IS defined in this dissertation. Unfortunately, AGIS has been conceived to support one context view. Moreover, the central authority is the only entitled to extend the exposed model.

3.6 Global Service Registry (GSR)

Global Service Registry (GSR) is an aggregator of information from multiple information systems. GSR has been developed independently from any resource model to

[15] `https://wiki.egi.eu/wiki/GOCDB/Input_System_User_Documentation#Permissions_associated_to_roles`

[16] OIM is the information system used by OSG. It can be seen as the equivalent of GOCDB for OSG.

decouple the registered information from the underlying information systems and to support the evolution of the information model of those systems [59]. For each information source, it exists a plugin which maps and translate the source information model to the model used by the GSR instance.

GSR has been developed in the context of Worldwide LHC Computing Grid (WLCG) where it uses GLUE 2.0 model. It imports information from BDII, GOCDB, OIM to provide a comprehensive and consistent overview of the grid infrastructures. For each of these sources, it exists a plugin to read and map information from the source to the selected GLUE 2.0 model.

The designers of GSR perceived the necessity of being agnostic from any resource model. GSR has been designed to provide an overview of resources imported from different sources to VOs such as done by AGIS [59]. Unfortunately, GSR does not support multi-context overview.

3.7 gCube Information System

D4Science is a HDI managed by the gCube framework. The gCube framework is an open-source software toolkit used for building and operating HDI by promoting data sharing and reuse [34]. In particular, gCube has been designed to support the orchestration of federation of e-infrastructures specifically.

gCube core components are the IS and the accounting and authorisation frameworks. The gCube Authorisation framework is a token based authorisation system compliant with Attribute-Based Access Control (ABAC) [84]. It defines an access control paradigm using policies to grant access rights to users.

The gCube IS V.1 is a Simple Object Access Protocol (SOAP) service which uses an array of mediators for interfacing with existing "systems" and their enabling technologies including middleware for distributed computing (e.g., gLite, EMI [1]), cloud (e.g., Globus [6], OCCI [55]) and data repositories (e.g., OAI-PMH [92], SDMX [86]). Via these mediator services, the storage facilities, the processing facilities, and the data resources operated by external infrastructures can be perceived and exploited as gCube resources independently from the e-infrastructure operating it. [14].

The gCube IS V.1 exposes six type of resources [80] (i) Hosting Node; (ii) Running Instance; (iii) Service Endpoint; (iv) Software; (v) WS-Resource; (vi) Generic Resource.

A Hosting Node represents a node empowered with the gCube container facilities. The gCube container enables management, accounting, and authorisation and provides clients to simplify the interaction with the information system.

A Running Instance represents a network service accessible through the infrastructure which lifecycle is managed by the Hosting Node. The Running Instance schema contains either the service description as Web Services Description Language (WSDL) (for SOAP services) or as Web Application Description Language (WADL) (for REST services). The gCube IS V.1 provides service description capability similarly to UDDI, and it supports service discovery by exploiting the Running Instance representations.

A Service Endpoint represents the contact endpoint of a network service which uses its own authorisation rules and which lifecycle is not managed by gCube Hosting Node. A database used as a backend by another service (e.g., the Running Instance) of the infrastructure is an example of a service endpoint.

Software represents a software component that the infrastructure is capable of managing and dynamically deploying.

WS-Resources are used to persist the service state of a stateful SOAP service.

A Generic Resource can be exploited to describe and register entities that cannot be represented by any of the previously described resource type. A Generic Resource is typically used to represent configurations, policies and so on.

All the resources are encoded as eXtensible Markup Language (XML) and the resource schema is defined using XML Schema Definition (XSD) [80]. In gCube IS V.1, the clients are not allowed to create new resource types.

The gCube IS V.1 divides the network services into two main groups; (i) Running Instances containing all the services which lifecycle is managed by the gCube Hosting Node; (ii) Service Endpoint used to support dynamic discovery of network services not managed through the gCube Hosting Node. These services may use their specific authorisation policies e.g., based on user-name and password, may be restarted, stopped, and modified without any prior negotiation with the other services of the federation.

In the past, every time a new system was federated in the infrastructure, apart from the effort to write or to configure a new mediator, the infrastructure staff had to evaluate the feasibility and possibility to modify a resource profile (or adding a new resource) to support the new system. For this reason, the resource profiles have changed along the time. At a certain moment, it was clear that it was not possible to sustain such an approach. A change to a resource type required to support a new system often required to change also the implementation of the existing mediators. Field et al. evidenced similar issues in BDII [61].

Giving that the standardisation approach proposed by Field et al. was not achievable in the D4Science context, the gCube designer added to every resource a way to provide arbitrary content in order to (partially) solve this issue. Moreover, they introduced the Generic Resource type which is composed by an arbitrary body. The validation of these arbitrary parts is limited to the XML validity. This solutions provided a certain degree of flexibility and eliminated the continuous resource schema redefinition.

The gCube IS V.1 supports three different views; the infrastructure resources view; the VO views; and the VRE views. These views are hierarchical. Each VOs contains a subset of the resources registered in the infrastructure, and each VREs contains a subset of the resources registered in the VOs. Moreover, the assignment of a group of infrastructure resources to a VO can be either exclusive or shared with other VOs. Finally, to a VRE can only be assigned the resources belonging to the parent VO.

The rationale of this organisation is that usually, a group of scientists represented by the VO requires more than one VRE to analyse the different aspects of the same scientific domain. VREs are used to support the activities of either a specific study or experiment, while VOs are used to support the activities of scientists working in specific research domains. Therefore, in gCube IS V.1 the VREs are sub-group of VOs.

The IS context sharing rules are simple. Every resource registered in a context is also visible in the parent contexts. A resource can be added to a context if and only if is present in the parent context. A resource belonging to a VO is not exploitable by a VRE belonging to another VO. It sounds like a business rule, the resources assignable to a group of researchers (i.e., VRE) are only the resources owned by the organisation (i.e., the VO) the researcher belongs to.

In gCube IS V.1, the relations between resources are only virtual. A resource could have one or more properties having the ID of other resources to relate (e.g., The Running Instance has an attribute indicating the id of the Hosting Node and viceversa), but the system does not provide any referential integrity support between them. Daily tasks ensure the environment consistency [68]. These daily tasks can resolve inconsistencies by exploiting the information contained in the relation properties of the resource profiles (e.g., they can delete a Running Instance profile if the Hosting Node reported in its profile was previously removed from the VRE). Unfortunately, these tasks are not able to recognise the virtual relations created by clients either in the body of the Generic Resources or the arbitrary part of the other resources.

The gCube IS V.1 was developed ten years ago to provide support for HDIs. It has a rigid resource model alleviated by the definition of generic resources and arbitrary contents in the other resource types. It has a rigid context hierarchy and a set of predefined context sharing rules. This research proposal overcomes the limitations of this IS.

3.8 Summary

Table 3.1 summaries the analysis of the state-of-the-art by highlighting the properties required by an IS for HDI (see 2.5). The N/A values present in the table indicate that the referenced property is not applicable because it depends on another property that is not satisfied by the analysed technology. For example, the Referential Integrity property depends on the availability of relations and the Integrity Across Contexts and Global/Partial View properties depend on the capability of the technology to support more than a single context.

	BDII	**UDDI**	**GOCDB**	**AGIS**	**GSR**	**gCube IS V.1**
Open-ended Set of Resource	N	N	N	N	Y[17]	N[18]
Support different way of describing the same Resource	N	N	N	N	N	N[19]
Relations between Resources	Virtual	N	Y	Y	Y	Virtual
Referential Integrity	N	N/A	Y	Y	Y	N
Multi-Context Support	Y	N	Y	N	N	Y[20]
Integrity Across Contexts	Y	N/A	Y	N/A	N/A	Y
Global/Partial View	Y	N/A	Y	N/A	N/A	Y

Table 3.1: *State of the Art supported properties*

[17]The Resource Model when instantiated is not extensible.

[18]The Resource Model provides a general purpose resource i.e., Generic Resource.

[19]The resources can be extended using an arbitrary content.

[20]Predefined Hierarchy of Contexts and support for resource instances sharing across Contexts

Analysing the table, it is possible to highlight that even if some of the analysed proposals could satisfy one or more requirements, none of them is capable of addressing all of them.

Moreover, the analysed cloud technologies are designed to achieve different objectives than realising an ISs for Research Infrastructures (RIs). Hence they are not comparable with the presented properties. However, we analysed these technologies to highlight the types of resources they expose.

	Cloud Technologies	BDII	UDDI	GOCDB	AGIS	GSR	gCube IS V.1
Configuration	Y	Y	N	?	?	Y	Y[21]
Service	Y	Y	Y	Y	Y	Y	Y[22]
Actor	N	Y	N	Y	?	Y	N
Dataset	N	N	N	?	?	N	Y[21]
Software	Y[23]	?	N	?	?	Y	Y
Virtual Machine	Y	Y	N	?	?	Y	Y[21]
Container	Y	Y	N	?	?	Y	Y
Site	Y	Y	N	Y	Y	Y	N

Table 3.2: *State of the Art supported resources*

Table 3.2 shows a summary of the resources supported by the different analysed proposals. The available literature not always clarifies the support of a specific technology to the representation, discovery, and access of the reported resource types.

Since none of the analysed proposals is capable of supporting a flexible resource model, this research proposal starts with the definition of a new resource model (see section 4.2).

Next chapter presents the new Resource Model (i.e., Information System Model) and in particular a specialisation of it capturing all the resources and their relations needed in a HDI (see section 4.3). In chapter 7, actual use cases demonstrate how this new Resource Model has been exploited in different production environments.

[21] Via the exploitation of the open-ended Generic Resource profile.

[22] Two different resource profiles (Running Instance, Service Endpoint) are used to distinguish between managed and non-managed services.

[23] To deploy new instances.

CHAPTER 4

Resource Model

The analysis of existent solutions has highlighted the limits of the resource models to fit the context of the study. Most relevant missing requirements are:

- lack of support for relations among modelled entities. In the best case, the relations are mocked by inserting the id of the referenced entity as an additional field in the entity description. Unfortunately, this prevents correct referential integrity and the navigation of the relations in both directions;
- fixed type for entities. In our context, the entities to be modelled can vary over the time due to the addition or removal of a constituent system;
- rigid schema to describe an entity;
- rigid model to share an entity across different contexts/namespaces (when available).

The Information System (IS) Model (IS Model) has been conceived to overcome these limitations and provide the building blocks needed to develop an IS for Hybrid Data Infrastructure (HDI) [77] [78].

This chapter introduces the mathematical notation (section 4.1) to be used in formal definition of the IS model (see section 4.2). Then, as proof of concept, it describes a proposal for a complete model supporting Virtual Research Environments (VREs) management on HDI, namely gCube Model (paragraph 4.3) defined on top of IS Model.

4.1 Notation

Types, sub-types, instances and stipulations are formally described in this section according to the notation introduced by Pierce [107].

We say that a term t is an *instance* of T (or t has type T, or "t belongs to T", or "t is an element of T"), written $t : T$ [1] to mean that t "obviously" evaluates to a value of the appropriate form — where by "obviously" we mean that we can see this statically, without doing any evaluation of t. We refer to this as typing relation.

We say that S is a *subtype* of T (or "T is a supertype of S"), written $S <: T$ to mean that any term of type S can safely be used in a place where a term of type T is expected. We refer to this as subtype relation. This view of subtyping is often called the principle of safe substitution.

A more straightforward intuition is to read $S <: T$ "every value described by S is also described by T", that is, "the elements of S are a subset of the elements of T".

The bridge between the typing relation and this subtype relation is provided by adding a new typing rule, the so-called rule of subsumption:

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t <: T} \tag{4.1}$$

This rule tells us that, if $S <: T$, then every element t of S is also an element of T.

We make two main stipulations for subtyping: first is reflective,

$$S <: S \tag{4.2}$$

and second, that it should be transitive:

$$\frac{S <: U \qquad U <: T}{S <: T} \tag{4.3}$$

These rules follow directly from the intuition of safe substitution.

We also say that k is not an *instance* of T and we express this as $k :\!/T$ (or $k \notin T$); L is not a subtype of T (or T is a not supertype of L) and we express this as $L \not<: T$.

4.2 Information System Model

The IS Model (henceforth IS Model) is a graph model with ***Entities*** as nodes and ***Relations*** as edges.

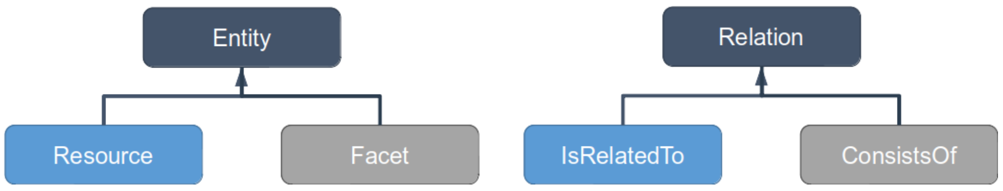
According to Angles and Gutierrez [9]

> Graph DB models are applied in areas where information about data interconnectivity or topology is more important, or as important, as the data itself. In these applications, the data and relations among the data, are usually at the same level. Introducing graphs as a modelling tool has several advantages for this type of data.

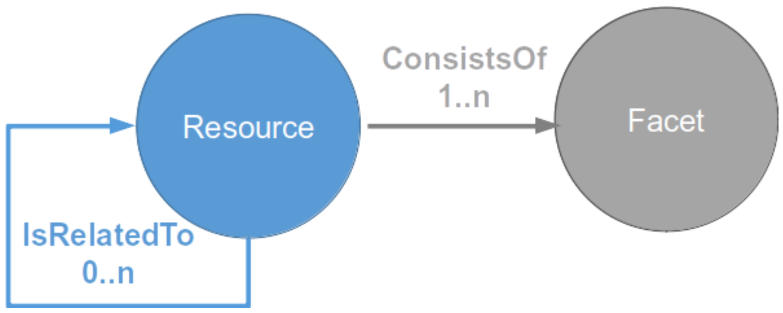
Two classes of ***Entities*** are envisaged (see. Fig. 4.1a): ***Resources*** i.e., entities representing a description of a "thing" to be managed; ***Facets*** i.e., entities contributing to "build" a description of a Resource. Each Resource is described by a set of Facets, each of them capturing a certain aspect / characterisation of the resource.

Two classes of ***Relations*** are envisaged (see. Fig. 4.1a): ***IsRelatedTo*** i.e., a relation linking two Resources; ***ConsistsOf*** i.e., a relation connecting a Resource with each of its Facets. Relations have a direction with a "source" and a "target". Nevertheless, when

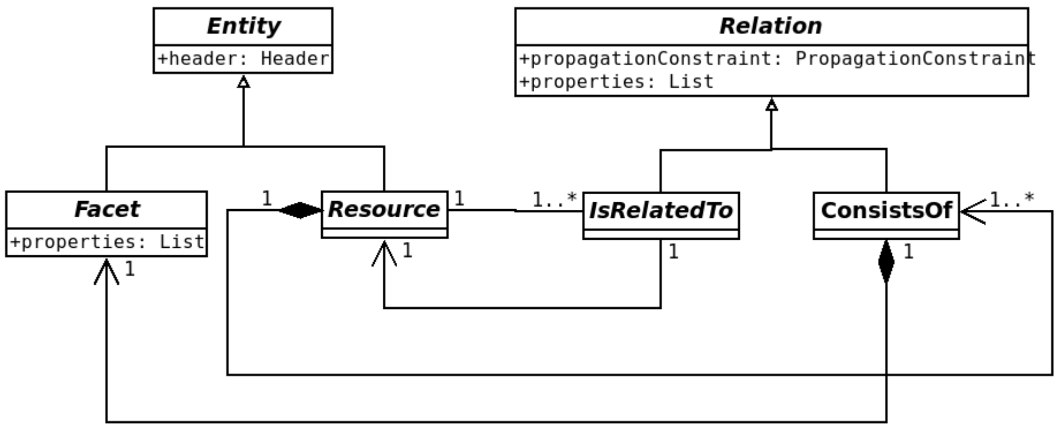
[1] is also often written as $t \in T$



(a) *Inheritance model of entities and relations*



(b) *Conceptual graph model of defined entities and relations*



(c) *UML Class Diagram*

Figure 4.1: *Concepts of the IS Model*

inspecting the graph (e.g., at query time) relations can be navigated in both directions, i.e., from source to target and from target to source.

In our model, a *Facet* describes a characteristic of a *Resource* definition. To enforce this semantic, a Relation can never have a *Facet* as source.

We derived the term Facet directly from the dictionary definition. Merriam Webster Dictionary [50] defines **facet** as "any of the definable aspects that make up a subject (as of contemplation) or an object (as of consideration)". The Free Dictionary [51] defines a facet as "One of numerous aspects, as of a subject". serv

Before describing the model, we introduce the notation convection:
We formalise the IS Model with the following notation and definitions:

Definition 1. *$Resource$ is a subtype of $Entity$:*

$$Resource <: Entity$$

Definition 2. *$Facet$ is a subtype of $Entity$:*

$$Facet <: Entity$$

Notation 1. *$Relation\langle Resource, Entity\rangle$ denoted the Relation type which has the type $Resource$ as a source and the type $Entity$ as target.*

Definition 3. *$IsRelatedTo$ is a subtype of $Relation$ having the $Resource$ type both as source and target:*

$$IsRelatedTo\langle Resource, Resource\rangle <: Relation\langle Resource, Entity\rangle$$

The semantic of relating two resources is captured by specialising this relation.

Definition 4. *$ConsistsOf$ is a subtype of $Relation$ having $Resource$ as source and $Facet$ as target.*

$$ConsistsOf\langle Resource, Facet\rangle <: Relation\langle Resource, Entity\rangle$$

The semantic of relating a Resource to a Facet (i.e., contributing to "build" a description of a Resource) is sufficient to instantiate this relation. Additional semantics can be captured by subtyping.

4.2.1 Instances

Relations are instantiated in the model and we formalise a relation instance as:

Definition 5. *Relation instance in the IS Model*

$$rel(r, e) : Relation\langle Resource, Entity\rangle$$

$$\textit{where}$$

$$r : Resource, \quad e : Entity$$

The next paragraphs explain how the IS Model relates and is used throughout contexts (see Figure 4.2). More specifically: how contexts composing an application domain are organised (section 4.2.2); how the IS Model instances can be shared across the different contexts (section 4.2.3); what are the allowed specialisation rules of the basic types of IS Model (section 4.2.4).

4.2.2 Application Domain and Contexts

A Context is the environment where an instance lives and can be used. The same context can include multiple instances. As an example, the context c_1 might include resource instances r_a, r_b and entity instances e_a, e_b, e_c. Context c_2 could include instances r_a and e_c.

It is important to point out that the same instance can belong to more than one context. This is useful to express the fact that, for instance, a service can belong to multiple VREs.

We denote $\mathbb{C}^i$ the set of contexts the instance i belongs to. More formally

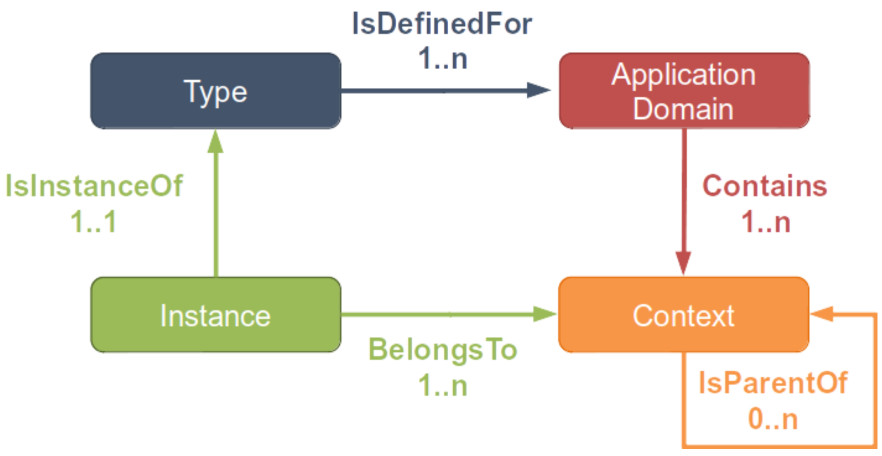


Figure 4.2: *How Types and their Instances relates to Application Domains and Contexts*

$$i \in \mathbb{C}^i$$

where

$$\mathbb{C}^i = \{c_1, \dots, c_k\}$$

In the following it will be useful to define the set of all the existing contexts:

$$\mathbb{C} = \{c_1, c_2, \dots, c_n\}$$

4.2.3 Instances and Contexts

Definition 6. *Contexts of an instance of a Relation*

To create a new instance of a relation, we need both an instance of the source resource and an instance of the target entity. This implies that it can be created only within those contexts shared by context of the source resource and the target entity, i.e., in one or more contexts of the intersection of the two sets of contexts.

$$\mathbb{C}^{rel(r,e)} \subseteq \mathbb{C}^r \cap \mathbb{C}^e \tag{4.4}$$

Definition 7. *ConsistOf Instance and Context*

$$\mathbb{C}^f \subseteq \mathbb{C}^r \quad \forall\ co(r, f) \tag{4.5}$$

where $co(r, f) : ConsistsOf\langle Resource, Facet\rangle$ is a $ConsistsOf$ relation instance between a resource $r : Resource$ and a facet $f : Facet$.

Definition 8. *Resource Instance and its Contexts*

$$\mathbb{C}^r \equiv \cup \mathbb{C}^{f_i} \quad \forall i\ co(r, f_i)$$

The set of contexts of a resource r is the union of all the contexts of all the facets f_i for every $co(r, f_i)$.

Summarising, a resource instance can consist of (i) a set of facets in a context c_1, (ii) plus, another different set of facets in a context c_2, plus, another set of facets in a third context c_3, and so forth. This aspect is significant because it allows characterising a resource instance in different ways, depending on the context we are considering.

Do note that a resource can be related to another resource in a context c_1 but not in the context c_2, where it is eventually related to a different set of resources.

This approach enables a different perspective from the common resource space representing all the application domains. Each perspective is related to a context, some of these contexts are VREs, others can be just a logical group defined for a business scope.

4.2.4 Specialisation Rules

Each Resource, Facet, ConsistsOf relation and IsRelatedTo relation can be specialised.

Definition 9. *Entity Subtyping*

1. *If T is a subtype of $Resource$, it can not be also a $Facet$ subtype.*

$$T <: Resource \iff T \not<: Facet$$

2. *Any subtype T of a $Facet$ can only be a facet subtype.*

$$T <: Facet \iff T \not<: Resource$$

3. *A type T can be subtype of more than one resource types*

$$T <: (R^1, R^2, \ldots, R^k), \quad \forall k R^k <: Resource$$

4. *A type T can be subtype of more than one facet types*

$$T <: (F^1, F^2, \ldots, F^k), \quad \forall k F^k <: Facet$$

For relation type definition, we have the following rules:

Definition 10. *Relation subtyping*

1. *$IsRelatedTo\langle Resource, Resource\rangle$ specialisation*

$$I\langle R^s, R^t\rangle <: IsRelatedTo\langle Resource, Resource\rangle$$

$$\iff$$

$$R^s <: Resource, R^t <: Resource$$

2. *A type $I\langle R^s, R^t\rangle$ can be a subtype of more than one $IsRelatedTo\langle Resource, Resource\rangle$ types*

$$I\langle R^s, R^t\rangle <: (I^1\langle R^{s_1}, R^{t_1}\rangle, \ldots, I^k\langle R^{s_k}, R^{t_k}\rangle)$$

$$\iff$$

$$\forall k I^k\langle R^{s_k}, R^{t_k}\rangle <: IsRelatedTo\langle Resource, Resource\rangle,$$

$$R^s <: (R^{s_1}, \ldots, R^{s_k}), R^t <: (R^{t_1}, \ldots, R^{t_k})$$

3. *$ConsistsOf\langle Resource, Facet\rangle$ specialisation*

$$CO\langle R^s, F^t\rangle <: ConsistsOf\langle Resource, Facet\rangle$$

$$\Longleftrightarrow$$

$$R^s <: Resource, F^t <: Facet$$

4. *A type $CO\langle R^1, F^1\rangle$ can be a subtype of more than one $ConsistsOf\langle Resource, Facet\rangle$ types*

$$CO\langle R^s, F^t\rangle <: (CO^1\langle R^{s_1}, F^{t_1}\rangle, \ldots, CO^k\langle R^{s_k}, F^{t_k}\rangle)$$

$$\Longleftrightarrow$$

$$\forall k CO^k\langle R^{s_k}, F^{t_k}\rangle <: ConsistsOf\langle Resource, Resource\rangle,$$
$$R^s <: (R^{s_1}, \ldots, R^{s_k}), R^t <: (F^{t_1}, \ldots, F^{t_k})$$

4.2.5 Entities and Relations properties

Entity and *Relation* have one or more properties. *Header* is a mandatory property, automatically generated for the sake of identification and provenance of the specific information. Apart from the *Header*, no additional properties are allowed for resource types.

Every *Facet* and *Relation* types - apart from the *Header* - can define zero or more properties for their schema.

Relation types, also have a *PropagationConstraint* property.

A *Facet* specialisation defines an additional semantic in respect to the parent type. The specialisation can also define additional properties.

Specialising a *Relation* means defining a schema concerning properties and concerning source and target *Entity* types.

Specialising a *Resource* means defining a schema regarding the *Facets* and the *ConsistsOf* relations used to describe them.

Facet and *Relation* instances can have additional properties not defined in the schema (henceforth schema-mixed mode). Similarly, *Resource* instances can have other *Facets* describing them.

Property Definition

Any Property defined in the schema is characterised by:

Name: the property name;

Type: the type of the property;

Description: the description of the property [default=null];

Mandatory: indicate if the property is mandatory or not [default=false];

ReadOnly: the property cannot change its value [default=false];

NotNull: whether the property must assume a value diverse from 'null' or not [default=false];

Max: whether the property can be limited to a maximum value [default=null];

Min: whether the property can be limited to a minimum value [default=null];

Regex: a regular expression to validate the property value [default=null].

Property Types The type of a property is one of the following:

1. a primitive type in Java or other programming languages i.e., Boolean; Integer; Short; Long; Float; Double; String; Enum; Date; Byte; Binary;
2. a composition of primitive types, called Embedded;
3. a collection of elements of the same primitive type: list, set or map;
4. a date, represented as the difference, measured in milliseconds, between the creation time and midnight, January 1, 1970, UTC (also known as "the epoch time");
5. an enumeration, represented using an integer and setting the values for Min and Max or using a string validated with the following regular expression:

```
^(FIRST-ENUM-STRING-REPRESENTATION | ... | LAST-ENUM-STRING-REPRESENTATION)$
```

Some common useful types such as Universally Unique Identifier (UUID), Digital Object Identifier (DOI), Uniform Resource Identifier (URI), Uniform Resource Name (URN), Internationalised Resource Identifier (IRI), Uniform Resource Locator (URL) can be created by using String as concrete type and a validating regular expression.
For example an UUID can be validated by using the following regular expression:

```
^([a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}){1}$
```

Embedded Type

Embedded type allows defining a complex type composed of two or more properties of primitive types or others embedded types.

Header The Header contained in entities and relations is an embedded type defined by the following properties:

uuid: [String, Mandatory=true, NotNull=true, ReadOnly=true]. This UUID is used to univocally identify the *Entity* or the *Relation*;

creator: [String, Mandatory=true, NotNull=true, ReadOnly=true]. The user that created the *Entity* or *Relation*;

modifiedBy: [String, Mandatory=true, NotNull=true]. The user that made the last update to the *Entity* or *Relation*;

creationTime: [Date, Mandatory=true, NotNull=true, ReadOnly=true]. Creation time;

lastUpdateTime: [Date, Mandatory=true, NotNull=true]. Last Update time.

Propagation Constraint At any time entities and relations can be added or removed to/from a context or deleted.

The *PropagationConstraint* property contained in each relation is a predefined embedded type which indicates the behaviour to be held on a target entity when an event related to a context occurs in the source resource [2] or directly to the relation.

This type contains the two following properties:

remove: [Enum, Regex=^(*cascadeWhenOrphan*|*cascade*|*keep*)$] : i.e., indicate the behaviour to implement for the target Entity when a remove action is performed on the source Resource.

> **cascadeWhenOrphan**: when a remove action is performed on the source Entity of the relation, or directly on the relation, then the same remove action apart on the relation is performed to the target entity if it has no other incoming relations;
>
> **cascade**: when a remove action is performed on the source Entity of the relation, or directly on the relation, then the same remove action is performed on the relation and its target entity;
>
> **keep**: when a remove action is performed on the source Entity of the relation, or directly on the relation, then the same remove action is performed on relation but never to the target entity.

add: [Enum, Regex=^(*propagate*|*unpropagate*)$] : i.e., indicate the behaviour to implement for the target Entity when a add action is performed on the source Resource.

> **propagate**: when an add action is performed on the source Entity of the relation, or directly on the relation, then the same add action is performed on the relation and its target Entity;
>
> **unpropagate**: when an add action is performed on the source Entity of the relation, or directly on the relation, then the same add action is performed on relation but never to the target entity.

The default values of propagation constraints are:

- *IsRelatedTo*: remove=keep, add=unpropagate;
- *ConsistsOf*: remove=cascadeWhenOrphan, add=propagate.

Remove actions are: (i) the operation of removing an instance from a context; (ii) the operation of deleting an instance (it has an impact on all contexts). Add action is the operation of adding an instance to a context.

4.2.6 Information System Model Summary

Resources, *Facets*, *IsRelatedTo* and *ConsistsOf* relations are the basic building blocks of the IS Model.

The model defines rules to specialise such types and to attach them properties if any. Two "special" types of property have been identified: the first for the sake of identification and provenance i.e., *Header*; the second to maintain referential integrity between entities across contexts i.e., *PropagationConstraint*.

[2] the source instance of a relation instance is always a resource instance.

The model exposes a way to define new types of properties starting from the basic types.

The model clarifies that any specialisation type is available across the whole application domain.

The model defines the constraints to be respected by any system using it. Each instance belongs to one and only one type; the source and target instance of a relation must be hierarchically compatible with the entity types defined by the extending relation type.

Instances can be shared across contexts. The propagation constraint defined on the relation instances specify the behaviour to hold on the target entity of the relation when adding or removing the source entity.

The next section introduces the gCube Model, a concrete instance of the IS Model used to represent resources and relations in the D4Science HDI.

4.3 gCube Model

gCube Model is a resource model capturing the different aspects of the resources and their relations playing significant roles in a HDI.

The modelling activity consisted in the (i) analysis the resources identified in literature (chapter 3); (ii) identification of the resources defined by different types of infrastructures and services federated in the context of D4Science HDI; (iii) overcoming of issues encountered in gCube IS V.1 as described in section 3.7; (iv) study of multiple federation scenarios such as the one required to support Building Research environments fostering Innovation, Decision making, Governance and Education to support Blue growth (BlueBRIDGE) (see 7.2) and Social Mining & Big Data Ecosystem (SoBigData) (see 7.3) European projects. As further proof of concept, the ontological model defined in Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization and Synergies (PARTHENOS) (see 7.4) (i.e., PARTHENOS Entities Model (PE Model) [29]) by the experts of Information Systems Laboratory at Foundation for Research and Technology-Hellas (FORTH) (Crete - GR) has been mapped on top of gCube Model (see section 7.4). This activity helps define coherent entities and relation hierarchy and to identify the proper nomenclature.

The modelling activity started by identifying the resource types (and the relations among them i.e., the *IsRelatedTo* relations), and then the different aspects of each resource description (i.e., the *Facets*). Similar aspects of different resources have been compared to find a common way to describe them. Every time an aspect of a *Resource* could be assimilated to the one of another *Resource* a *Facet* type has been defined.

When a *Resource* has two (or more) aspects captured with the same *Facet* but having different semantics, a *ConsistsOf* specialisation is required.

This section presents the *Embedded* types (section 4.3.1) used by the identified *Facet* types (section 4.3.2). For each *Facet*, we introduce its properties, the identified *ConsistsOf* relations, and then the identified *Resource* type. We describe each *Resource* in terms of *Facets* and the identified incoming or outcoming *IsRelatedTo* relation.

4.3.1 Embedded Types

Within the gCube Model, we defined one *Embedded* type named *ValueSchema* (section 4.3.1) .

ValueSchema

$$ValueSchema <: Embedded$$

This type aims at exposing a value which can be automatically managed by any client with no knowledge of its format.

value: [String, Mandatory=true, NotNull=true]. The value which schema is available at the URI provided in the schema property;

schema: [URI, Mandatory=true, NotNull=true] An URI containing a schema used to validate/interpret the content of the value. It is only an informative field. The validation is charge of the client.

The client can retrieve the schema at the provided URI, i.e., schema property, and use its content to understand/validate/manipulate the information contained in the value property.

The client must have the capability to understand the semantics of content retrieved at the URI endpoint. Examples of application of such type are eXtensible Markup Language (XML) values which can be validated by a Document Type Definition (DTD) or XML Schema Definition (XSD).

4.3.2 Facets and ConsistsOf Relations

Nineteen facet types have been identified and defined (see. Fig. 4.3) in order to describe the common aspects of the resources identified in gCube Model (see 4.6).

When a resource presented two (or more) different aspects of its description which could be described by the same *Facet* type but with different semantics, it has been necessary to identify a *ConsistsOf* relation to capture such a difference. In other words, if two aspects of the same resource could be defined as a single concept but they have to be discriminated, a *ConsistsOf* relation type has identified. Instead, when it has been recognised that a resource has the same aspect of the description repeated two (or more) times with no need of discriminating between them, no dedicated *ConsistsOf* relation has been identified.

AccessPointFacet

$$AccessPointFacet <: Facet$$

AccessPointFacet captures information on an "access point" of a resource, i.e., any web-based endpoint to programmatically interact with the resource via a known protocol.

The following properties characterise the *AccessPointFacet*:

entryName: [String] A distinguishing string to be used by clients to identify the access point of interest;

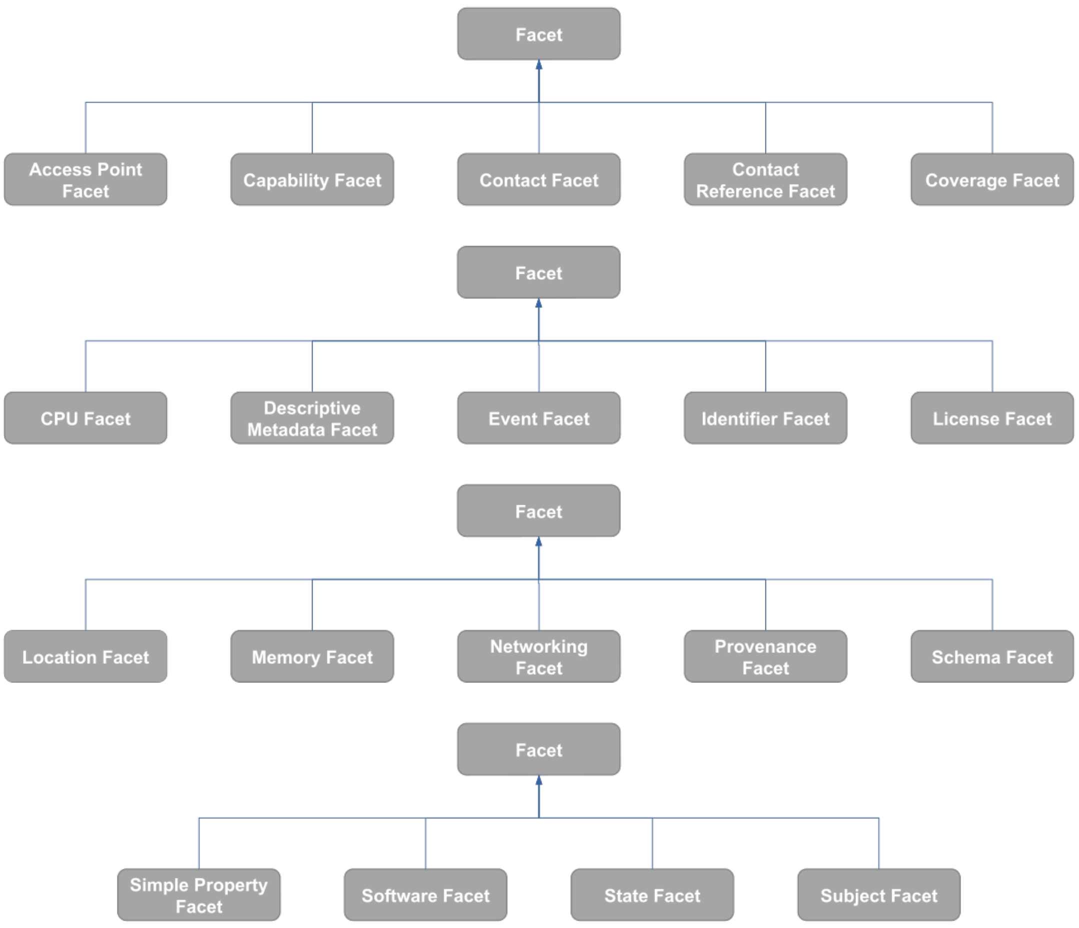


Figure 4.3: *Overview of gCube Facets*

endpoint: [URI, Mandatory=true, NotNull=true, ReadOnly=true]. The URI which characterises the specific endpoint instance;

protocol: [String] The high-level protocol used by the access point. The String could contain the version if needed. e.g., Web Map Service (WMS) and not Hyper-Text Transfer Protocol (HTTP) which is already contained in the URI;

description: [String] A human-oriented text accompanying the access point;

authorization: [ValueSchema] Contains authorisation information. e.g., a token, the couple username:password. By relying on a schema, it should be sufficient to capture also whether the content is encrypted or not. The schema is used for a proper interpretation of the authorisation value.

It is used to define the network endpoint to contact the service. The endpoint can expose a well-known high-level protocol.

AccessPointFacet has similar meaning of *PE29 Access Point* defined in PARTHENOS Entities Model (PE Model) [29] (see section 7.4.1).

CapabilityFacet

$$CapabilityFacet <: Facet$$

CapabilityFacet captures a defined facility to perform a specified task supported by a given Resource. It is characterised by the following properties:

name: [String, Mandatory=true, NotNull=true, ReadOnly=true] The distinguishing name of the capability;

description: [String] A human oriented description of the capability;

qualifier: [String] A string used to specialise the capability.

It is mainly used to provide a human-readable description of the capabilities of a resource (e.g., the support for transactions of an electronic device or some non-functional properties of a service like its replicability to support High-Availability (HA)).

ContactFacet

$$ContactFacet <: Facet$$

ContactFacet captures information on a point of contact for the resource, i.e., a person or a department serving as the coordinator or focal point of information concerning the resource. The following properties characterise the facet:

title: [String] A name describing the profession or marital status of the point of contact. e.g., Dr, Mrs, Mr.;

name: [String, Mandatory=true, NotNull=true] First Name;

middleName: [String] Middle Name;

surname: [String, Mandatory=true, NotNull=true] Surname;

eMail: [String, Mandatory=true, NotNull=true] Email address.

HasContact relation has been defined as base relation type to capture the diverse points of contact associated with a resource

$$HasContact\langle Resource, ContactFacet\rangle <: \\ ConsistsOf\langle Resource, Facet\rangle$$

This relation is abstract because if not specialised it does not add any semantic of relating the resource with the target *ContactFacet*. Instead, every specialisation refines the reason of using the *ContactFacet* allowing to discriminate between two or more *ContactFacet*s attached to the same resource.

The identified specialisations are *HasContributor*, *HasCreator*, *HasCurator*, *HasDeveloper*, *HasMaintainer*, *HasManager* and *HasOwner*.

$$HasContributor\langle Resource, ContactFacet\rangle <: \\ HasContact\langle Resource, ContactFacet\rangle$$

HasContributor indicates that the target *ContactFacet* contains the information related to a contributor to the source resource e.g., the contact points of the contributor of software or the contributor of a dataset.

$$HasCreator\langle Resource, ContactFacet\rangle <: \\ HasContact\langle Resource, ContactFacet\rangle$$

HasCreator indicates that the target *ContactFacet* contains the information related to a creator of the source resource e.g., the contact points of the creator of a dataset.

$$HasCurator\langle Resource, ContactFacet\rangle <: \\ HasContact\langle Resource, ContactFacet\rangle$$

HasCurator indicates that the target *ContactFacet* contains the information related to a curator of the source resource e.g., the contact points of the curator of a dataset.

$$HasDeveloper\langle Resource, ContactFacet\rangle <: \\ HasContact\langle Resource, ContactFacet\rangle$$

HasDeveloper indicates that the target *ContactFacet* contains the information related to a developer of the source resource e.g., the contact points ofthe developer of a software.

$$HasMaintainer\langle Resource, ContactFacet\rangle <: \\ HasContact\langle Resource, ContactFacet\rangle$$

HasMaintainer indicates that the target *ContactFacet* contains the information related to a maintainer of the source resource e.g., the contact points of the maintainer of a software or a dataset.

$$HasManager\langle Resource, ContactFacet\rangle <: \\ HasContact\langle Resource, ContactFacet\rangle$$

HasManager indicates that the target *ContactFacet* contains the information related to a manager of the source Resource e.g., the contact points of the manager of a research infrastructure or a data centre.

$$HasOwner\langle Resource, ContactFacet\rangle <: \\ HasContact\langle Resource, ContactFacet\rangle$$

HasOwner indicates that the target *ContactFacet* contains the information related to the owner of the source resource e.g., the contact points of the owner of dataset.

ContactReferenceFacet

$$ContactReferenceFacet <: Facet$$

ContactReferenceFacet captures information on the primary and authoritative contact for the resource it is associated with. It is characterised by the following properties:

website: [String] The main website;

address: [String] A physical address;

phoneNumber: [String] A phone number.

CoverageFacet

$$CoverageFacet <: Facet$$

CoverageFacet captures information on the extent of the resource, i.e., any aspect aiming at capturing an indicator of the amount/area the resource covers be it a geospatial area, a temporal area, or any other "area". It contains one mandatory property:

coverage: [ValueSchema, Mandatory=true, NotNull=true] The value indicates the "area" covered by the dataset according to the schema.

Let consider a dataset containing information regarding the salinity of a specific ocean area in a certain amount of time. What is essential in the information system is not representing the data of the salinity. Instead, to specify the temporal period and the ocean area the dataset is valid. This information is captured using the same schema i.e., the *CoverageFacet* but using different relations to distinguish between them with no need to understand the value or the schema of the coverage.

$$HasCoverage\langle Resource, CoverageFacet\rangle <: \\ ConsistsOf\langle Resource, Facet\rangle$$

For such a reason it has been defined the abstract *HasCoverage* relation and the specialisation *HasTemporalCoverage* and *HasSpatialCoverage* have been defined to refines the reason of using the *CoverageFacet*.

$$HasSpatialCoverage\langle Resource, CoverageFacet\rangle <: \\ HasCoverage\langle Resource, CoverageFacet\rangle$$

HasSpatialCoverage indicates that the target *CoverageFacet* indicates a spatial coverage information e.g., the geographic area indication for the dataset.
The *CoverageFacet* attached to the resource with *HasTemporalCoverage* indicates a spatial coverage information e.g., the temporal period indication for the dataset.

$$HasTemporalCoverage\langle Resource, CoverageFacet\rangle <: \\ HasCoverage\langle Resource, CoverageFacet\rangle$$

CPUFacet

$$CPUFacet <: Facet$$

CPUFacet captures information on the Central Processing Unit (CPU) of the resource it is associated with. A resource which needs to indicate a multi-processor/multi-core CPU will consist of more than one *CPUFacet*. Even if more than one *CPUFacet* is associated with a resource (i.e., an *HostingNode*), we did not find any reason to differentiate the CPUs. It is characterised by the following properties:

model: [String, Mandatory=true, NotNull=true] CPU Model;

vendor: [String, Mandatory=true, NotNull=true] CPU Vendor;

clockSpeed: [String, Mandatory=true, NotNull=true] Clock speed expressed with the unit, e.g., 1 GHz.

DescriptiveMetadataFacet

$$DescriptiveMetadataFacet <: Facet$$

DescriptiveMetadataFacet captures information on descriptive metadata to be associated with the resource. It is characterised by the following properties:

metadata: [ValueSchema, Mandatory=true, NotNull=true] A metadata record representing the descriptive metadata according to the schema.

This facet is mainly used to attach metadata to a Dataset.

EventFacet

$$EventFacet <: Facet$$

EventFacet captures information on a certain event/happening characterising the life cycle of the resource. It is characterised by the following properties:

date: [String, Mandatory=true, NotNull=true] The time the event took place/occurred;

event: [ValueSchema, Mandatory=true, NotNull=true] The typology of event according to the schema.

Examples of an event are the start time of a virtual machine or the activation time of an electronic service.

IdentifierFacet

$$IdentifierFacet <: Facet$$

IdentifierFacet captures information on identifiers (other than the ones automatically generated by the system) that can be attached to a resource. It is characterised by the following properties:

value: [String, Mandatory=true, NotNull=true] The identifier. e.g., `http://fr.dbpedia.org/resource/Thunnus`, `de305d54-75b4-431b-adb2-eb6b9e546014`;

type: [Enum, Mandatory=true, NotNull=true] The typology of identifier, i.e., URI, DOI, IRI, URL, URN, UUID;

isPersistent: [Boolean, Mandatory=true, NotNull=true] To indicate whether the identifier is persistent or not.

LicenseFacet

$$LicenseFacet <: Facet$$

LicenseFacet captures information on any license associated with the resource to capture the policies governing its exploitation and use. It is characterised by the following properties:

name: [String, Mandatory=true, NotNull=true] The common name of the license. e.g., European Union Public Licence (EUPL) 1.1, GNU General Public License (GPL) 2, Berkeley Software Distribution (BSD), Common Creative (CC);

textURL: [URL, Mandatory=true, NotNull=true] The URL to the actual text of the license.

Example of use is the licence of a dataset e.g., Creative Commons Attribution (CC-BY) or the licence of software such as GPL.

This facet is used to provide for human knowledge, but it is not excluded the usage by infrastructure services which enforces the respect of the licence e.g., a service which denies the usage of a dataset with Creative Commons Attribution No-Derivatives (CC-BY-ND) (CC-BY-ND) licence to produce a new dataset.

LocationFacet

$$LocationFacet <: Facet$$

LocationFacet captures information on a physical area characterising the resource it is associated with. This should not be confused with *CoverageFacet* described above (see 4.3.2). The *LocationFacet* provides information of a location (eventually using latitude and longitude), instead *CoverageFacet* provide a way to to define the spatial or the temporal extent the resource represent.

LocationFacet is characterised by the following properties:

country: [String] The English name of the country;

location: [String] The City name;

latitude: [String] Latitude;

longitude: [String] Longitude.

It is mainly used to locate a data centre or to the geographic references of a legal body playing the role of an actor in the infrastructure.

MemoryFacet

$$MemoryFacet <: Facet$$

MemoryFacet captures information on computer memory equipping the resource and its usage. It is characterised by the following properties:

size: [String, Mandatory=true, NotNull=true] Total amount of memory;

used: [String, Mandatory=true, NotNull=true] Used amount of memory;

unit: [Enum, Mandatory=true, NotNull=true] The unit of measure used to report size and used attributes. Allowed values are B (Byte), kB (kilobyte 10^3), MB (megabyte 10^6), GB (gigabyte 10^9), TB (terabyte 10^{12}), PB (petabyte 10^{15}), EB (exabyte 10^{18}), ZB (zettabyte 10^{21}), YB (yottabyte 10^{24}).

Any resource describing a computing machine must have at least two types of memories i.e., persistent and volatile. For such a reason, it has been identified the *ConsistsOf* relation called *HasMemory*.

$$HasMemory\langle Resource, MemoryFacet\rangle <: \\ ConsistsOf\langle Resource, Facet\rangle$$

It is in charge of the specialisation *HasVolatileMemory* and *HasPersistentMemory* to clarify the semantics of the memory.

$$HasVolatileMemory\langle Resource, MemoryFacet\rangle <: \\ HasMemory\langle Resource, MemoryFacet\rangle$$

HasVolatileMemory indicates that the target *MemoryFacet* is a volatile memory, i.e., a memory which requires power to maintain the stored information. Volatile memory is also known as main memory, internal memory or primary storage.

$$HasPersistentMemory\langle Resource, MemoryFacet\rangle <: \\ HasMemory\langle Resource, MemoryFacet\rangle$$

HasPersistentMemory indicates that the target MemoryFacet indicates a non-volatile memory, i.e., a memory which does not lose the data when the device is powered down. This type of memory is also known as secondary memory, external memory, auxiliary storage, or secondary storage.

Of course more than one *MemoryFacet* related with *HasPersistentMemory* can be attached to the same resource, but from an infrastructure management point of view, we did not find any reason to further specialise the *HasPersistentMemory* relation.

NetworkingFacet

$$NetworkingFacet <: Facet$$

NetworkingFacet captures information on any (computer) network interface associated with the resource. It is characterised by the following properties:

hostName: [String] Host Name;

domainName: [String] Domain Name;

IPAddress: [String, Mandatory=true, NotNull=true] Internet Protocol (IP) Address.

mask: [String] Network Mask;

broadcastAddress: [String] Broadcast Address.

It is mainly used to describe the network interface of a host. It should not be confused with the *AccessPointFacet* which instead describes the protocol and the endpoint of a web-based service.

ProvenanceFacet

$$ProvenanceFacet <: Facet$$

ProvenanceFacet captures information on provenance/lineage of the entire resource.

relationship: [Enum, Mandatory=true, NotNull=true] i.e., wasDerivedFrom, wasGeneratedBy;

document: [ValueSchema, Mandatory=true, NotNull=true] Provenance Document e.g., an XML according to the reference schema.

It is mainly used to describe provenance information of a Dataset.

SchemaFacet

$$SchemaFacet <: Facet$$

SchemaFacet captures information on any schema, i.e., a vocabulary used to validate a document associated with a resource. It is characterised by the following properties:

name: [String, Mandatory=true, NotNull=true] Schema Name;

description: [String, Mandatory=true, NotNull=true] Schema Description;

schema: [ValueSchema, Mandatory=false, NotNull=true] The 'value' property contains the defined "schema" that in turn is validated by the schema available at the URL indicated in the 'schema' property. An example could be an XSD schema instantiation as 'value' and the URL of the DTD defining the XSD as 'schema' i.e., `https://www.w3.org/2009/XMLSchema/XMLSchema.dtd`.

Examples of schema are JavaScript Object Notation (JSON) schema and XML schema. JSON schema "is a vocabulary that allows you to annotate and validate JSON documents" [103]. JSON schema is under standardisation by Internet Engineering Task Force (IETF) (see references at `http://json-schema.org/specification.html`). XSD defines the legal building blocks of an XML document. DTD defines the structure and the legal elements and attributes of an XML document.

SimplePropertyFacet

$$SimplePropertyFacet <: Facet$$

SimplePropertyFacet captures information on any property by a simple name-value pair. It is characterised by the following properties:

name: [String, Mandatory=true, NotNull=true] The name associated to the value;

value: [String, Mandatory=true, NotNull=true] The value.

It is mainly used to add key-value pairs to the resource in order to describe some resource characteristics. Before using *SimplePropertyFacet* a developer should evaluate if it is possible to identify a specific *Facet* to capture the particular aspect of the resource. The usage of *SimplePropertyFacet* should be reduced to the maximum.

SoftwareFacet

$$SoftwareFacet <: Facet$$

SoftwareFacet captures information on any software associated with the resource. It is characterised by the following properties:

name: [String, Mandatory=true, NotNull=true] The name of the software artifact being described, e.g., artifactId in maven, the software name for retail software such as Office (in Microsoft^{TM}Office 2013-SP2);

group: [String, Mandatory=true, NotNull=true] The name of "group" the software artifact belongs to, e.g., groupId in Maven, company name for retail software Microsoft^{TM}(in Microsoft^{TM}Office 2013-SP2);

version: [String, Mandatory=true, NotNull=true] The particular release of the software artifact, e.g., version in maven, the software version for retail software such as 2013-SP2 (in Microsoft^{TM}Office 2013-SP2);

description: [String] A human oriented description of the software artifact being described;

qualifier: [String] A qualifier for the software, e.g., packaging or scope in maven, target architecture for retail software x86 or amd64;

optional: [Boolean] Used to indicate the software optionality e.g., optional dependency in maven.

StateFacet

$$StateFacet <: Facet$$

StateFacet captures information on state to be associated with the resource. The state is captured by any controlled vocabulary which is an integral part of the facet.

state: [ValueSchema, Mandatory=true, NotNull=true] The value of the state expressed according to the schema.

Examples of usage are the state of service e.g., running or down or the state of a virtual machine e.g., activated or unreachable.

SubjectFacet

$$SubjectFacet <: Facet$$

SubjectFacet captures information on subjects associated with the resource for description, classification and discovery purposes.

subject: [ValueSchema, Mandatory=true, NotNull=true] The value of the subject according to the schema.

IsIdentifiedBy

$$IsIdentifiedBy\langle Resource, Facet\rangle <:$$
$$ConsistsOf\langle Resource, Facet\rangle$$

Each gCube resource has been defined to have at least a *Facet* linked with an *IsIdentifiedBy* relation. *IsIdentifiedBy* indicates that the target facet represents a sort of identification for the source resource. For instance, a software can consist of one or more *SoftwareFacet*s but the one related with *IsIdentifiedBy* represents the identify of the software.

4.3.3 Resources and IsRelatedTo Relations

Seven main resource typologies have been identified and defined (see. Fig. 4.4), namely *Dataset*, *Actor*, *Schema*, *Configuration Template*, *Site*, *Service* and *Software*. In some cases these typologies have been further specialised to capture specific entities, e.g., *ConcreteDataset* is a sub-type of *Dataset*, *EService* is a subtype of *Service*.

A *Resource* Type identified as "Abstract" cannot be instantiated. It is expected that one of its specialisations is instantiated. It is not required that an Abstract class establishes an *IsIdentifiedBy* relation with a *Facet*.

Sixteen *IsRelatedTo* relation typologies have been identified and defined (see. Fig. 4.5). In some cases, these typologies have been further specialised to capture specific semantic.

In the reminder of this section the defined *Resource* and *IsRelatedTo* types are described.

Fig 4.6 presents the resources, their specialisations and the relations among them. It can be used as reference to visualise the resources and relations described below. *IsRelatedTo* hierarchy is not indicated in the picture.

Dataset

$$Dataset <: Resource$$

A *DataSet* is a set of digital objects representing data and treated collectively as a unit. It is the key resource of a HDI, even more, it is the reason the HDI exists.

A *Dataset* can be correlated to another *Dataset* by using *IsCorrelatedTo* relation (see fig. 4.6).

Dataset has similar meaning of *PE18 Dataset* defined in PARTHENOS Entities Model (PE Model) [29] (see section 7.4.1).

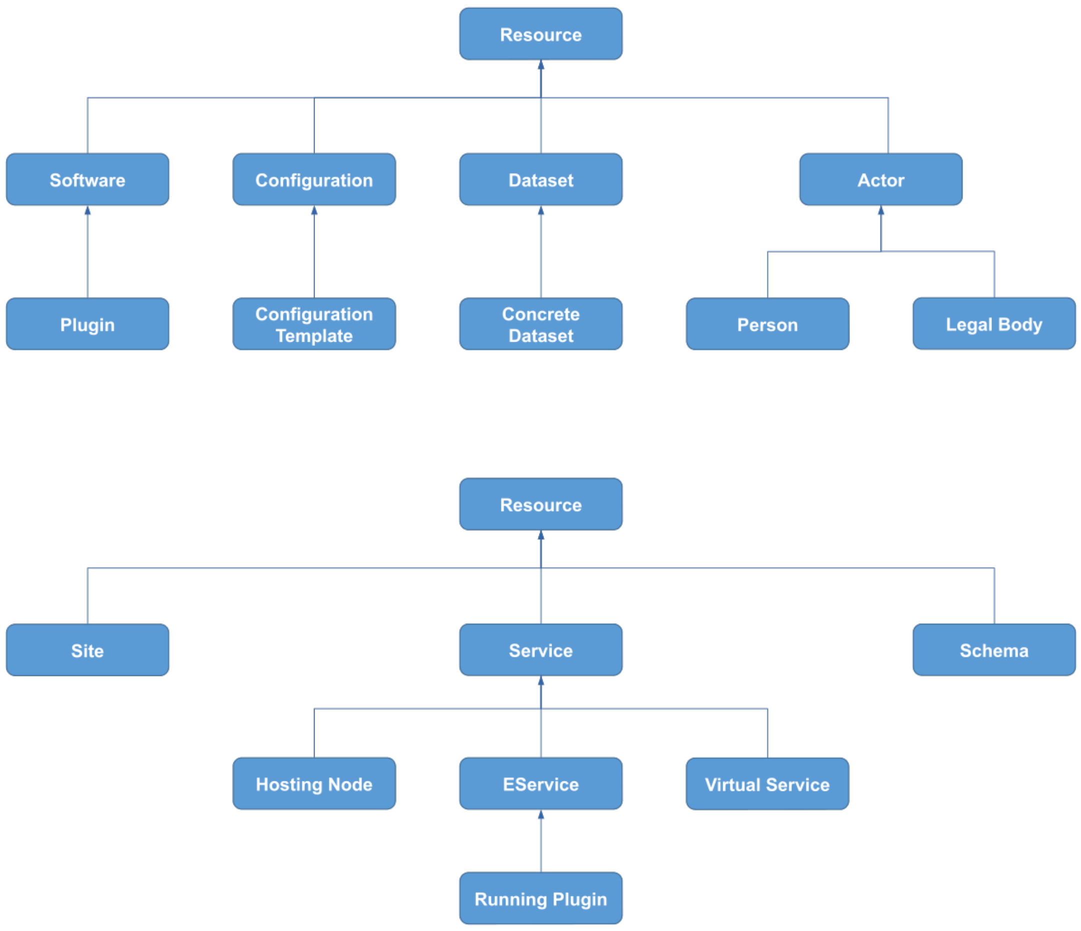


Figure 4.4: *Overview of gCube Resources*

$$ConcreteDataset <: Dataset$$

ConcreteDataset is any incarnation/manifestation of a dataset or part of it. The relation *IsPartOf* is used when a *ConcreteDataset* is part of a *Dataset* (see fig. 4.6).

A *DataSet* and its specialisations *ConcreteDataset* are characterised by the following facets:

- *IdentifierFacet* (associated with *IsIdentifiedBy*) : this facet captures information on Identifiers (other than the ones automatically generated by the system) that can be attached to the dataset, e.g., for discovery purpose;
- *ContactFacet*: this facet captures information of a point of contact for the dataset, i.e., a person or a department serving as the coordinator or focal point of information concerning the dataset. There are several potential points of contact that can be associated to the dataset, and the role of the association is captured by using a specific *ConsistsOf* relation e.g., to represent the owner, the responsible, the creator, the curator, the maintainers and any contributors of the dataset;
- *AccessPointFacet*: this facet captures information of an "access point" of a dataset,

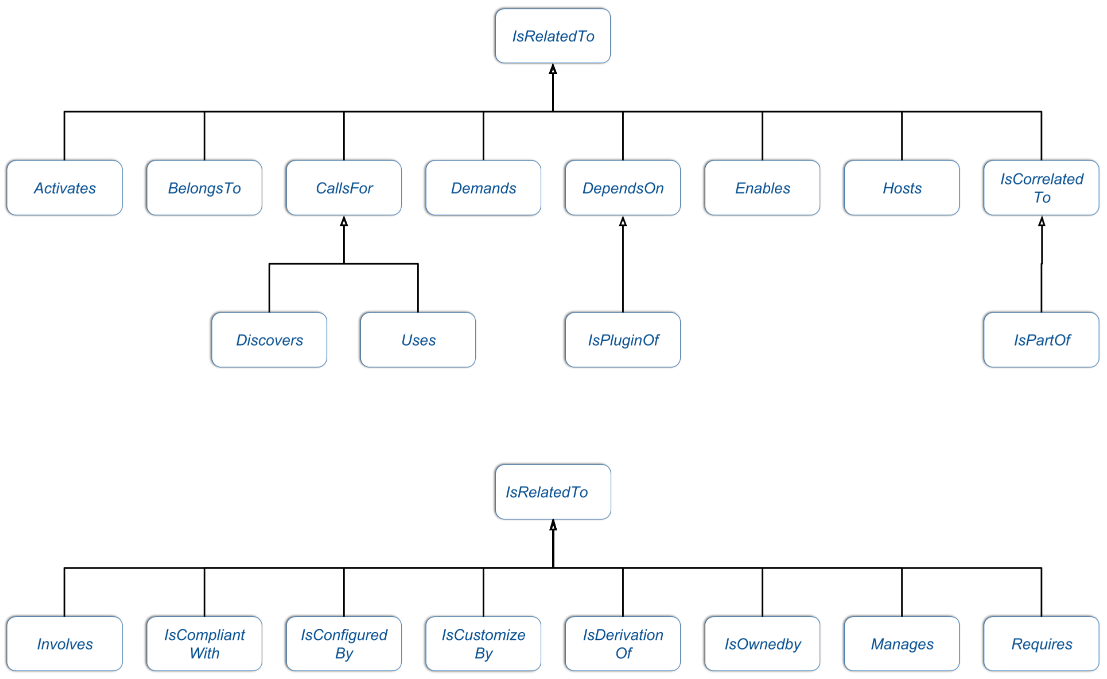


Figure 4.5: *Overview of gCube IsRelatedTo relations*

i.e., any web-based endpoint to programmatically interact with the resource via a known protocol. It represents the access point to access to the dataset. The embargo state can be modelled through access policy defined in the *ConsistsOf* relation;

- *LicenseFacet*: this facet captures information on any license associated with the dataset to capture the policies governing its exploitation and use. The duration of the license (if any) can be captured through expiry date property defined in the *ConsistsOf* relation;

- *ProvenanceFacet* : this facet captures information on provenance/lineage of associated with the dataset;

- *CoverageFacet* this facet captures information on the extent of the dataset, i.e., any aspect aiming at capturing an indicator of the amount/area the resource covers be it a geospatial area, a temporal area, or any other "area". Any temporal coverage information characterising the content of the dataset, e.g., the time-span covered by the dataset can be described with this facet associated with the dataset with a specific *ConsistsOf* relation. Any geospatial coverage information characterising the content of the dataset, e.g., the area covered by the dataset can be described with this facet associated with the dataset with a specific *ConsistsOf* relation;

- *DescriptiveMetadataFacet* : this facet captures information on descriptive metadata to be associated with the dataset, e.g., for discovery purposes;

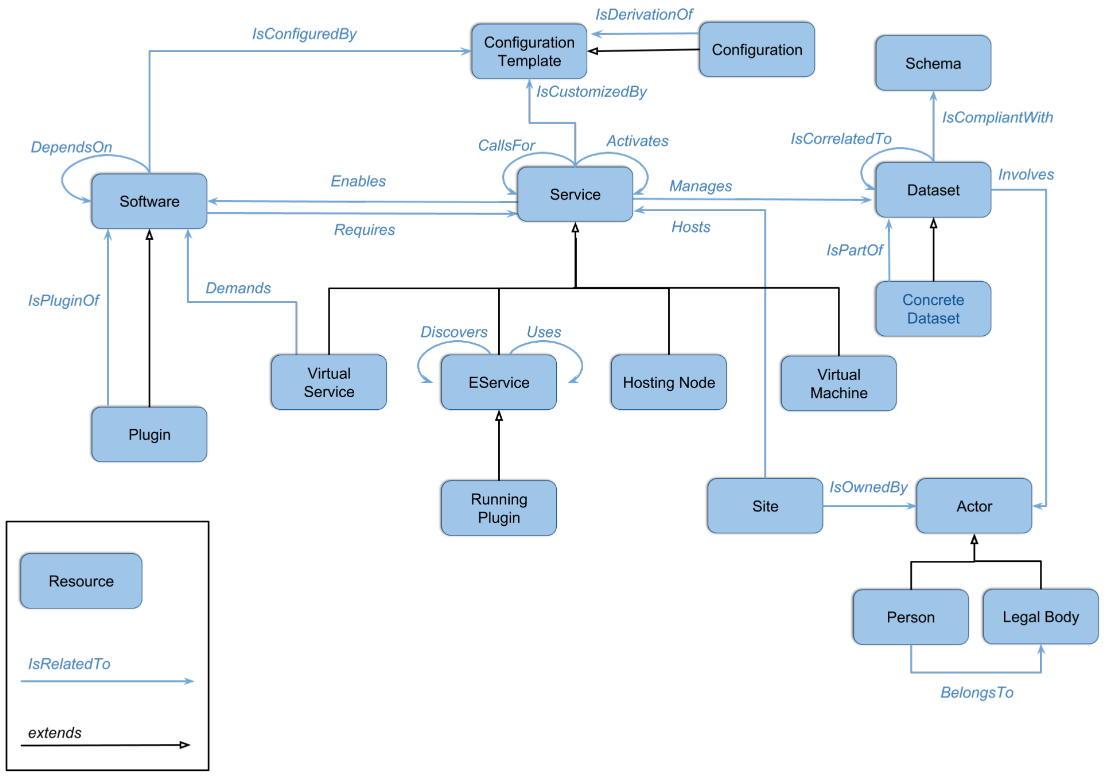


Figure 4.6: *Overview of gCube Model IsRelatedTo relations and the Resources they relates. The resource inheritance is also indicated.*

- *SubjectFacet* : this facet captures information on subjects associated with the dataset for descriptive and discovery purposes;
- *SimplePropertyFacet* : this facet captures information on any additional property by a simple name-value pair.

Actor

$$Actor <: Resource$$

Actor (Abstract) is any entity (human or machine) playing an active role in the infrastructure. *Actor* has two specialisation, *LegalBody* which represent any legal entity and *Person* which is any human playing the role of *Actor*. An *Actor* can belong to a *LegalBody* and this is expressed using the defined *BelongsTo* relation (see fig. 4.6).

$$LegalBody <: Actor, Person <: Actor$$

Actor and its specialisation are characterised by the following facets:

- *ContactFacet* (associated with *IsIdentifiedBy*) : an *Actor* has at least a *ContactFacet* which permit to identify the *Actor* per se. An *Actor* can have other *ContactFacets* which provide secondary contact information;
- *ContactReferenceFacet* : this facet captures information on the primary and authoritative contact for the resource it is associated with;

- *LocationFacet* : mainly used for *LegalBody* to provide a geographic references.

Actor specialisation are used from *Dataset* to indicate the involved *Actor* using *Involves* relation (see fig. 4.6).
Actor has similar meaning of *E39 Actor* defined in CDOC-CRM [85]. Similarly *LegalBody* and *Person* have similar meaning of *E40 Legal Body* and *E21 Person* respectively.

Schema

$$Schema <: Resource$$

Schema is any reference schema used to specify compliant values. Examples include controlled vocabularies, ontologies, and others. This resource is mainly used by *Dataset* to evidence that is compliant with a *Schema* by using *IsCompliantWith* relation (see fig. 4.6).
Schema is characterised by the following facets:

- *SchemaFacet* (associated with *IsIdentifiedBy*) : this facet captures information on any schema associated with a *Resource*. There are diverse type of schema that can be associated to the schema each one is capture by a dedicated *SchemaFacet*;
- *ContactFacet* : this facet captures information on a point of contact for the schema;
- *DescriptiveMetadataFacet* : this facet captures information on descriptive metadata to be associated with the schema, e.g., for discovery purposes;
- *SubjectFacet* : this facet captures information on subjects associated with the schema for descriptive and discovery purposes.

Configuration Template

$$ConfigurationTemplate <: Resource$$

Configuration Template represents a template for a configuration. It describes how a configuration has to be realised.

$$Configuration <: ConfigurationTemplate$$

Configuration is a specialisation of *ConfigurationTemplate* and is an instance of a configuration template characterising the behaviour and shape of the resource it is attached to. The *Configuration* can be related to the template it derives using *IsDerivationOf* (see fig. 4.6).
ConfigurationTemplate and *Configuration* are characterised by the following facets:

- *IdentifierFacet* (associated with *IsIdentifiedBy*) : the set of identifiers associated with the configuration template instance;
- *SimplePropertyFacet* : this facet captures information on any property by a simple name-value pair.

Software

$$Software <: Resource$$

Software is an entity worth being represented for management purposes. The relation *DependsOn* indicates dependencies between two *Software* captured for management purposes (see fig. 4.6).

Plugin is a piece of *Software* extending the capabilities of another *Software* (main) and requiring the main Software to be executed. The relation between the main *Software* and the *Plugin* is expressed by *IsPluginOf* relation (see fig. 4.6). *IsPluginOf* is an extension of *DependsOn*.

Any *Software* and its specialisation *Plugin* are characterised by the following facets:

- *SoftwareFacet* (associated with *IsIdentifiedBy*) : software coordinates which identify the software per se;
- *SoftwareFacet* : apart the one connected by the *IsIdentifiedBy* relation the others identify the software in other way (e.g., Maven coordinates);
- *AccessPointFacet* : identify endpoint useful for software download, documentation, source code etc e.g., links to maven artifact on public maven repositories, javadoc, wiki, SVN;
- *LicenseFacet* : the software license characterising its possible exploitation and use e.g., EUPL, GNU Lesser General Public License (LGPL), GPL, Apache2;
- *StateFacet* : this facet captures information on state to be associated with the resource. State is captured by any controlled vocabulary which is an integral part of the facet e.g., Deprecated, Active, Obsolete;
- *CapabilityFacet* : any facility supported/offered by the software.

The relation *IsConfiguredBy* indicates that the source *Software* requires a configuration when it is instantiated (see fig. 4.6).

Software has similar meaning of *D14 Software* defined in CRMdig [52].

Service

$$Service <: Resource$$

Service (Abstract) represents any typology of service worth registering in the infrastructure. As evidenced by the fig 4.6 a service has relations with quite all other resources. If on one side, an HDI is created to manage dataset, on the other side the HDI born to manage such datasets digitally. We could affirm that dataset and services are the two pillar resources around which revolves the entire infrastructure. It is important to highlight that service has a general meaning and must not be intended as a network-callable service which is represented instead by one of its specialisation called *EService*.

Any *Service* is characterised by the following facets:

- *DescriptiveMetadataFacet* : any descriptive information associated with the service, e.g., for discovery purposes;

- *SubjectFacet* : any subject / tag associated with the service for descriptive, cataloguing and discovery purposes;
- *CapabilityFacet* : this facet captures a defined facility for performing a specified task supported/offered by a given Service.

Giving that *Service* is abstract no *IsIdentifiedBy* association with a facet is provided which in turns is responsibility of the specialisation.

Looking at fig. 4.6 we can identify that any specialisations of a *Service* can manage a *Dataset* which is captured using *Manages* relation. *IsCustomizedBy* relation instead evidences that Any *Service* can be customised by a *ConfigurationTemplate*.

Giving that *Service* is abstract, to present in a more understandable way *IsRelatedTo* relations having a service as source or as target, it is more convenient to introduce identified *Service* specialisation i.e., *EService*, *HostingNode* and *VirtualMachine*, *VirtualService* and then argue about the relations.

Service could be a specialisation of the entity *PE1 Service* defined in PARTHENOS Entities Model (PE Model) [29] (see section 7.4.1) and the entity *D1 Digital Object* defined in CRMdig [52].

EService

$$EService <: Service$$

EService is any running service that is registered in the infrastructure and made available by an access point.

In addition to the facets describing any *Service*, *EService* is characterised by the following facets:

- *SoftwareFacet* (associated with *IsIdentifiedBy*) : this facet captures information on any software associated with the resource. The one associated associated with *IsIdentifiedBy* represent the main software enabling the *EService* capabilities (this facet is the one identifying the *EService*);
- *SoftwareFacet* : software available in the *EService* environment that characterises the specific *EService* instance;
- *AccessPointFacet* : identify the endpoints of the *EService*;
- *EventFacet* : this facet captures information on a certain event / happening characterising the current status and the life cycle of the *EService* events (e.g., Activation Time, Deployment Time);
- *StateFacet* : this facet captures information on the current operational state of the *EService* it is associated with (e.g., started, ready, down, failed);
- *LicenseFacet* : this facet captures information on any license associated with the *EService* to capture the policies governing its exploitation and use.

EService has similar meaning of *PE8 EService* defined in PARTHENOS Entities Model (PE Model) [29] (see section 7.4.1).

RunningPlugin

$$RunningPlugin <: EService$$

RunningPlugin allows differentiating which is the primary service and which is an additional capability of a such a service. Keeping the two concepts separated enables to share a service across VREs with a subset of its capabilities.

Virtual Machine

$$VirtualMachine <: Service$$

VirtualMachine (Virtual Machine (VM)) is the typical resource represented in a cloud infrastructure. It is an emulation of a physical computer which appears to the running operative system as real hardware. A VM provides operative system functionality and potentially allows to install any software designed for the running operative system.

A *VirtualMachine* is characterised by the following facets:

- *NetworkingFacet* (associated with *IsIdentifiedBy*) : this facet captures information on the network interface associated with the VM;
- *CPUFacet* : this facet captures information on the Central Processing Unit of the resource it is associated;
- *MemoryFacet* : this facet captures information on computer memory equipping the resource and its usage such as the persistent memory (i.e., the Disk Space Capacity of the hosting node) by using *HasPersistentMemory* relation or the volatile memory (i.e., the RAM Capacity of the hosting node) by using *HasVolatileMemory* relation;
- *EventFacet* : this facet captures information on a certain event / happening characterising the life cycle of the hosting node, e.g., the activation time;
- *StateFacet* : this facet captures information on the operational status of the VM (e.g., started, ready, certified, down, failed);
- *SimplePropertyFacet* : this facet captures information by a simple <key, value> pair property worth associating with the VM, e.g., environment variables;
- *SoftwareFacet* : this facet captures information on any software associated with the Hosting Node. Useful to report the hosted software such as the operating system.

VirtualMachine could be though as a specialisation of the entity *PE6 Software Hosting Service* defined in PARTHENOS Entities Model (PE Model) [29] (see section 7.4.1).

Hosting Node

$$HostingNode <: Service$$

The *HostingNode* represent a container capable of managing the lifecycle of an electronic service, i.e., being capable to host and operate an *EService*. Examples are docker, tomcat. A container is a service which is conceived to enable any services respecting the container rules to be operative. The container does not typically provide any functionality rather than allowing the hosted service to operate.

The *HostingNode* characterisation in terms of facets reflects the one presented for *VirtualMachine*. In particular, facets representing memory, CPU and networking interface are used to describe the *HostingNode* when the *VirtualMachine* enabling the *HostingNode* is not represented.

Federated systems can provide virtual machines as resource or containers as resources. In some cases, the description of a container includes (virtual) hardware information.

It is important to highlight that *HostingNode* is not a subclass of *VirtualMachine*.

HostingNode could be though as a specialisation of the entity *PE6 Software Hosting Service* defined in PARTHENOS Entities Model (PE Model) [29] (see section 7.4.1).

Virtual Service

$$VirtualService <: Service$$

VirtualService is an abstract service (non-physically existing service) worth being represented as an existing *Service* for management purposes. Examples of usage include cases where classes or set of services have to be managed like an existing unit. This resource is essential from infrastructure management point of view because it allows easily share a pool of services across VREs as a single unit.

VirtualService mainly consist of a service definition which uses relations to *ConfigurationTemplate*, *EService*, *Software* (using *Demands* relation) to properly support the sharing across VREs. The correct sharing is feasible thanks to the propagation constraint of the model. The IS provides only the support for resource sharing as a bundle. Instead, the actions required to deploy a *Software* are a responsibility of the service invoking the sharing operation.

This resource emerged thank to the experience maturated with gCube IS V.1 where this resource was represented as a Generic Resource (see 3.7) containing the list of the resource's id forming the bundle which often lead to inconsistency.

Service Relations It is now easier to present the *Service* incoming and out-coming relations as depicted in fig. 4.6 by describing them and providing examples existing in real scenarios.

Any service representing running code of a specific software has the relation *Enables* targeted to the corresponding *Software*. *Enables* is used for example by *EService* to indicates the running software; from *HostingNode* to indicate the running software container; within *RunningPlugin* and the software represented as *Plugin*.

The relation *Requires* from a *Software* to any kind of *Service* is used to indicate that the source *Software* when activated requires to interact with a specific *Service* instance i.e., the target of the relation. This relation usually reflects an administrative constraint than a technological limitation. When there are no constraint on the instance the relation is instead *DependsOn* which is between two *Software*.

Not necessarily all the *Software* running in the infrastructure are represented as resources. Only the *Software* required for infrastructure administration or the ones to be managed with specific policies are represented. Indeed, *Requires* relation is normally used as policy constraint.

Let imagine an open source software which uses a MySQL database as backend. In many cases, what characterises the software instance is the data present in the backend. The infrastructure manager could stipulate an agreement with a provider having a particular set of data which is not for public domain. The software once deployed, giving the business agreement, is entitled to use the instance of the provider and not a generic MySQL database instance. The infrastructure manager imposes the use of such a particular instance because of the data it contains.

A *Service* instance can require another *Service* instance to properly operate and this is indicated with *CallsFor*. Motivations similar to the ones for *Requires* relation conducted to the definition for this relation.

Activates has a *Service* instance both as source and target as well as *CallsFor* but with a complete different semantic. *Activates* is used to indicates that the source *Service* of the relation enable the operation of the target *Service*. Examples are between a *VirtualMachine* and an *HostingNode* to capture the VM is operating the *HostingNode*. Another example is between a *VirtualMachine* and an *EService* e.g., between a VM and a database instance. This relation can be created also between an *HostingNode* and one *EService* e.g., to represent a container e.g., Tomcat and the web-service(s) is operating. *Activates* is also used between an *EService* and a *RunningPlugin* which enrich the functionality offered by the source service.

Any *EService* or its specialisations can be related with another *EService* with *Discovers* relation. *Discovers* relation inform that the source *EService* discovers the target through the information system. *Uses* relation instead inform regarding the network invocation of the target *EService* by the source. *Discovers* and *Uses* relations specialise the semantic of *CallsFor*.

Site

$$Site <: Resource$$

Site is a resource representing the location (physical or virtual) hosting the resources associated. The following facets characterise it:

- *IdentifierFacet* (associated with *IsIdentifiedBy*) : the identifier associated with the site instance;
- *ContactFacet* : there are diverse points of contact that can be associated to the site and the role of the association is captured by using a specific *ConsistsOf* relation e.g., to represent the manager and the maintainers of the site;
- *LocationFacet* : this facet captures information on a physical area characterising the resource it is associated with e.g., the Global Positioning System (GPS) coordinates of the site or the geographical address of the site. This should not be confused with *CoverageFacet*;
- *NetworkingFacet* : this facet captures information on any (computer) network interface/access point associated with the resource. A site has one or more IP sub-networks to address the machines in the site.

Hosts relation is used from a *Site* a *Service* instance. The target of the *Hosts* relation depends on the service offered by the *Site*. When the resources provided by a site are

VMs, *Hosts* relation is created from a *Site* to a *VirtualMachine*. When, instead a *Site* provides web-services, *Hosts* relation is created with *EService*. If a site provides container facilities *Hosts* relation is created with *HostingNode*. By defining *Hosts* relation with target *Service*, the model is capable of representing the diverse type of federated systems and service.

Site allows to identify all the services that will be affected by downtime due to a scheduled maintenance, as well as the impact on the infrastructure that an accidental loss of connectivity could cause. This resource allows to study and define the replication scenarios or to provide an adequate redundancy level to a VRE.

Any *Site* is owned by an *Actor* and this is captured by the *IsOwnedBy* relation. The referenced *Actor* can be used as a contact point during an emergency, to agree on the scheduling of a site downtime and to request additional resources during the downtime of another site.

4.4 Model Analysis

From the Context of Study (see section 2.5) it emerged the requirement for a flexible model capable of supporting:

1. an open-ended set of manageable resources and relationship among them;
2. an open-ended model for describing resources;
3. the ability to adapt to the evolving needs of the infrastructure at no cost for its clients:
    (a) by supporting new resource types;
    (b) by supporting evolution in the way a resource is described;
    (c) by supporting the same resource type described by using different models.

The possibility to specialise entities and relations provides the open-ended set of manageable resources and relationships (requirement 1).

The possibility of describing a resource with facets provides the open-ended model for describing a resource (requirement 2).

Specialisations enable to define and support new resource types (requirement 3a). When the model of a resource provided by a specific system changes, it is sufficient to change the facet instances attached to the resource. The IS Model delivers a flexible and straightforward way to support the evolution (requirements 3, 3a, 3b). New facets can be created to describe a characteristic of a resource which original model (the model provided by the federated system) change. Any facet instance can be described with additional attributes compared to the schema (mixed-schema mode).

Resource instances of the same type can consist of different facets. Hence the resource description can support different models (requirement 3c).

Chapter 6 presented a complete software architecture capable of managing the IS Model (section 4.2) and enabling dynamic creation of any model build on top of IS Model. The next chapter presents a RESTful Transaction Model capable of providing Atomicity, Isolation, Durability and Consistency (ACID) transaction to the IS.

CHAPTER 5

RESTful Transaction Model

During the last decade, REpresentational State Transfer (REST) has emerged as a best practice to design web services. REST is an excellent architectural style to support scalability of service while keeping the complexity of design, implementation, and deployment at very affordable costs. For such a reason, REST has guided the design of the Information System (IS) (see chapter 6).

REST is an architectural style defined in 2000 by Roy Thomas Fielding [65]. REST defines 6 principles and 4 constraints but it does not provides any concrete guideline or architecture. An example of concrete architecture for REST is Resource Oriented Architecture (ROA) which is based on HyperText Transfer Protocol (HTTP) 1.1 [64].

In the rest of this chapter are presented the principles of REST (cf. Sec. 5.1) and then the concrete implementation provided by ROA (cf. Sec. 5.2). Section 5.3 presents our research proposal for a resilient RESTful transaction model.

5.1 REpresentational State Transfer (REST)

Fielding [65] indicates six principles which characterise the REST architectural style. Those principles were derived by observing the design rationale behind the web architecture and by identifying the set of constraints applied to elements within that architecture:

Client-Server Paradigm: refers to the separation of concern between client and server. Both the server and the client can evolve independently, provided that the exposed interface be left unaltered;

Statelessness: indicates that the server does not store any information regarding previous interaction with the client. Instead, the client sends to the server all the information required to understand and elaborate the request correctly. This principle improves “visibility, reliability and scalability while decreasing network performances” [65];

Cacheability: indicates to clients (and intermediates such as a proxy) that they have to cache the responses based on server indication (explicit or implicit). Cacheability allows clients to reuse data and reduce the needs of some interactions with servers. Cacheability, among other things, helps to balance/mitigate the potential inefficiency in network performance introduced with the stateless constraint;

Uniform Interface: allows decoupling the architecture from the implementation. This principle defines four additional constraints to satisfy:

Identification Of Resources: resources must be individually identifiable. The crucial emerging concept is the resource. “A resource is any information that can be named. It is any concept that might be the target of an author’s hypertext reference [65]”. The identifier does not change if its representation change;

Manipulation Of Resources Through Representations: the resource representation capture the state of the resource transferred between components;

Self-descriptive Messages: every message exchanged between client and server must contain all information regarding how to elaborate the message itself “in order to support intermediate processing of interactions [65]” (e.g., the media type of the resource representation, cache control information);

Hypermedia As The Engine Of Application State (HATEOAS): the use of hypermedia drives clients interaction with servers.

Layered System: suggests adding an arbitrary number of intermediary components between the client and the server. This property allows decoupling the service logic from higher level facilities. Layers can be used to improve the system scalability; to provide encapsulation facility for legacy and non-rest systems (by exposing them through the uniform-interface); to achieve transparent cacheability across different clients; to add higher-level facilities such as security. As a counterpart, layering adds overhead and network latency;

Code on Demand (optional): (depending on the application and context) server can extend its functionality by providing clients code to execute. It allows distributing the computational load. It is only an optional constraint because in some cases some intermediary can/must limit the transfer of code (e.g., for security reason).

It is possible to evidence why this architectural style is called REST by keeping in mind the "key" concept of "resource" [1]. A client is interested in the representation of a resource. Once the client receives the representation, the client is posed in a specific state, as soon as the client requests another resource its state change.

[1] In REST, the term "resource" is used to identify an entity managed by the web service. In this section the term resource is used indicating the REST meaning.

5.2 Resource Oriented Architecture (ROA)

REST is only an architectural style and does not provide any implementation. In 2007, Richardson and Ruby [109] proposed a concrete architecture to implements REST web services by using HTTP 1.1. The authors refer to such an architecture with the name ROA. In other terms, ROA is a set of guidelines to design REST web services using standard technologies such as Uniform Resource Identifier (URI), HTTP and eXtensible Markup Language (XML).

ROA uses standards HTTP methods applied to URI to realise Create, Read, Update, Delete (CRUD) operations [94]. Most used HTTP methods in ROA are `POST` , `GET` , `PUT` and `DELETE` .

According to HTTP specification [64, 101], `POST` is used to create a new resource without providing the URI of creating resource. The representation of the resource is sent, as part of the HTTP body, via `POST` to the collection that will contain the resource. The server determines its appropriate location, and it provides the resulting URI to the client. Also, `PUT` can be used to create a new resource if the client provides the URI where the resource will become available.

`GET` is used to get the information about a resource. `PUT` is used to update an existing resource. This operation instructs the server to apply a new representation as a replacement of the previous one. `DELETE` is used to delete an existing resource.

`GET` , `PUT` and `DELETE` must be idempotent, i.e., the same operation repeated multiple times has the same side effect than using it one time. Request For Comments (RFC) 7231 clarify that “repeating the request will have the same intended effect, even if the original request succeeded, though the response might differ” [64].

`GET` must have no side effect, and this is also known as safe operation. “This does not prevent an implementation from including behaviour that is potentially harmful, that is not entirely read-only, or that causes side effects while invoking a safe method” [64].

Table 5.1 shows the mapping between HTTP methods and CRUD operations. Moreover, it shows the property of safety and idempotency the methods must satisfy.

CRUD Operation	**HTTP Method**	**Safe**	**Idempotent**
Create	`POST`	No	No
Read	`GET`	Yes	Yes
Update [2]	`PUT`	No	Yes
Delete	`DELETE`	No	Yes [3]

Table 5.1: *Mapping between CRUD operation and HTTP methods enriched by safety and idempotency property they must satisfy.*

ROA gives particular emphasis on *"make it a resource"* paradigm and proposes descriptive and predictable URI as technology to satisfy the resource identification constraint. Hence, any resource in ROA has a URI.

[2] Also Create to the URI where the resource will be available

[3] Allamaraju [2] argues that `DELETE` idempotency should be accomplished client-side. The server should inform the client if a delete succeeded because the resource was really deleted or it was not found i.e., 404 Not Found error is suggested instead of 204 No Content. The latter situation should be treated as idempotent by the client.

To satisfy the uniform interface constraint, ROA indicates the way to construct URI for resources and how to use HTTP methods to them.

Operation	HTTP Method	URI [4]	Non-REST URI [5]
Listing	`GET`	`/users/{USERNAME}/bookmarks`	`GET /posts/list`
Create	`POST`	`/users/{USERNAME}/bookmarks`	`GET /posts/add`
Create	`PUT`	`/users/{USERNAME}/bookmarks/{URI-MD5}`	`GET /posts/add`
Read	`GET`	`/users/{USERNAME}/bookmarks/{URI-MD5}`	`GET /posts/get`
Update	`PUT`	`/users/{USERNAME}/bookmarks/{URI-MD5}`	`GET /posts/update`
Delete	`DELETE`	`/users/{USERNAME}/bookmarks/{URI-MD5}`	`GET /posts/delete`

Table 5.2: *ROA compliant URI compared to non REST services.*

Table 5.2 shows an example on how to create a compliant bookmark service as proposed by the authors. The first column evidences the intended operation, and the second indicates the HTTP method used to contact the URI presented in the third column (ROA URI). The last column shows an example of an old-fashion Non-ROA Application Programming Interfaces (APIs).

HTTP 1.1 defines also the HTTP method `HEAD` , `CONNECT` , `OPTIONS` and `TRACE` .

RFC 7231 [64] defines these methods as follows:

> `HEAD` method is identical to `GET` except that the server MUST NOT send a message body in the response. This method can be used for obtaining metadata about the selected representation without transferring the representation data and is often used for testing hypertext links for validity, accessibility, and recent modification;
>
> `CONNECT` method requests that the recipient establish a tunnel to the destination origin server identified by the request-target and, if successful, thereafter restrict its behaviour to blind forwarding of packets, in both directions, until the tunnel is closed. `CONNECT` is intended only for use in requests to a proxy;
>
> `OPTIONS` method requests information about the communication options available for the target resource, at either the origin server or an intervening intermediary. This method allows a client to determine the options or the requirements associated with a resource, or the capabilities of a server, without implying a resource action. A `OPTIONS` request with an asterisk ("*") as the request-target applies to the server in general rather than to a specific resource;
>
> `TRACE` method requests a remote, application-level loop-back of the request message.

Richardson and Ruby evidenced that `HEAD` and `OPTIONS` methods are also part of the uniform interface design and suggest their usage [109, p. 97].

Richardson and Ruby suggest the use of the HTTP methods defined for Web-based Distributed Authoring and Versioning (WebDAV) [53] as a plus to respect the uniform interface (e.g., `MOVE` and `COPY`), but they also highlight to avoid the HTTP methods

[4]The path variable parameters are shown within { } and using capital letters. This convection will be used in the rest of the dissertation.

[5]The information regarding which username and URL save as bookmark or which URL retrieve are provided via `GET` URL parameters not shown in the table.

that lead to deviate from *"make it as resource"* paradigm. For example, they suggest to create lock collections and manipulate them like all the others collections (by using `POST` , `GET` , `PUT` , `DELETE`) in place of using `LOCK` and `UNLOCK` methods to resource URI.

The next section presents the defined RESTful Transaction Model. It is important to note that RESTful is an abused term. It is easy to find services declared RESTful because they comply with HTTP methods semantics but they violate the uniform interface pattern. It is also easy to find services syntactically compliant with Richardson and Ruby guidelines (URI and HTTP methods) but violating one of the principles of REST as defined by Fielding (e.g., services violating the staleness). In the next section instead, we will use the term RESTful to refer to web services compliant with the set of ROA rules and with REST principles.

5.3 Resilient Two-Phase Locking Transaction Model

An IS enables the management and assignment of resources to one or more Virtual Research Environments (VREs). This assignment is compliant with the policies defined by the Hybrid Data Infrastructure (HDI) and aims to satisfy the needs of the community exploiting the VRE with a fit for purpose approach: no more than the necessary resources have to be allocated for its intended use. This approach maximises the number of VREs that can be operated by the HDI and requires that resources can be dynamically allocated and disposed according to the observed usage of those resources. Moreover, different and non-collaborative clients can concurrently modify the resources and the way they are accessed and exploited in the VREs. Concurrent modifications of the resources and their dynamic allocation and disposal could bring the IS in an inconsistent state that cannot be tolerated in a production-quality environment. To avoid those issues, it is required a transaction model capable of providing Atomicity, Isolation, Durability and Consistency (ACID) properties.

Mihindukulasooriya et al. [97, 98] in their surveys analysed the research literature identifying the main transaction models (see section 5.4). Unfortunately, these models fail to conform to one or more REST principles, and the proposed approaches are not suitable to be used in a production environment where scaling, and resiliency (i.e., the ability to return to its original state after a component failure, see section 5.5.2) are mandatory requirements.

For such a reason it has been necessary to investigate and propose a new RESTful Transaction Model.

The Transaction Model is designed as a layered system and is capable of supporting existing (i) services not designed to provide transactions; (ii) and existing clients not designed to perform transactions.

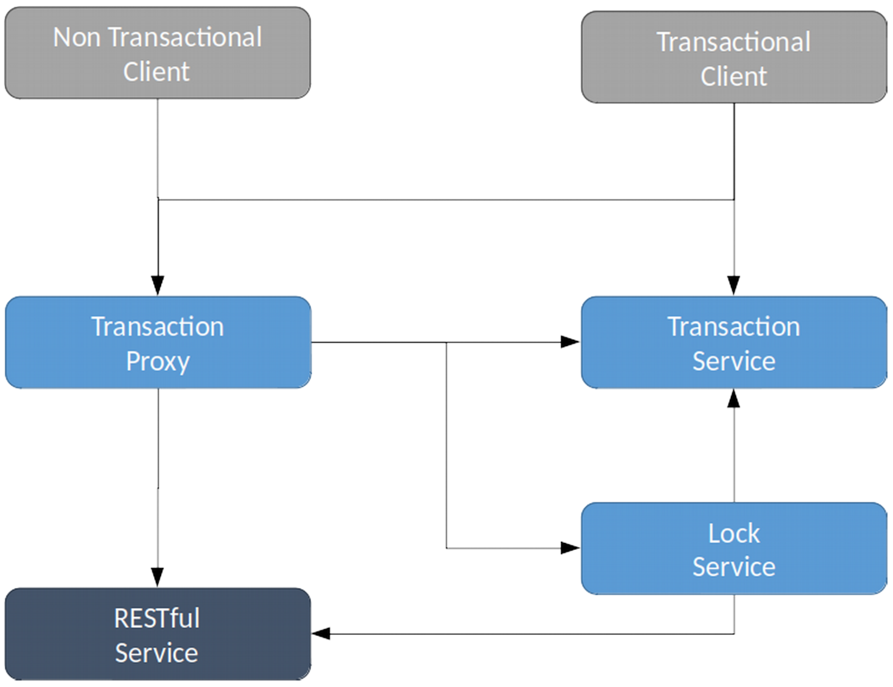


Figure 5.1: *Transaction Model Architecture. The arrow indicates that the source services in entitled to perform REST request to the target service.*

Figure 5.1 presents the six components that may be involved in a transaction governed by the Transaction Model.

- RESTful Service: the RESTful web service (i.e., ROA based web service) is transaction agnostic. From now on, we will refer to this component as service or effective service interchangeably;
- Lock Service: it provides lock capabilities for resources exposed by the effective service;
- Transaction Service: it provides transactions as resource facilities. It also exposes transaction logging facilities as resources. Logging is used to achieve compensations in case of rollbacks;
- Transaction Proxy: it intercepts all requests made by clients to the effective service and forwards them only after performing the required either checks or actions. We will refer to this component as proxy or transaction proxy interchangeably.

Figure 5.1 presents two types of clients:

- Non-Transactional Client: client either unaware of the transaction model or not interested in performing transactions;
- Transactional Client: transaction aware client.

From now on, we will use clients to generically indicates any Transactional or Non-Transactional clients. Moreover, we will use "additional services" to generically refer to Lock Service, Transaction Proxy and Transaction Service.

	Share	**eXclusive**
Share	true	false
eXclusive	false	false

Table 5.3: *Lock compatibility table.*

The networking policies play an important role in this architecture. The following rules are defined: (i) the effective service is only accessible from Transaction Proxy and Transaction Service services; (ii) the Lock Service is only accessible from Transaction Proxy and Transaction Service; (iii) the Transaction Proxy intercepts every request coming from clients.

Additional services support both XML and JavaScript Object Notation (JSON) as the content format to represent the resources involved in the transaction. By supporting both types, we do not enforce any Transactional client to use two different formats, but we make the content format adopted by the effective service also available for managing a transaction.

The Transactional client indicates the required format resources by indicating it (i.e., `application/xml` or `application/json`) in `Accept` HTTP header [64, section 5.3.2] or in `Content-Type` HTTP header [64, section 3.1.1.5]. Each additional service reply indicating the format in `Content-Type` header field according to the received request. In this dissertation, we present all the examples using the JSON format.

The next three sections are dedicated to present each of the additional service and their behaviour to provide transaction facilities to Transactional and Non-Transactional clients.

5.3.1 Lock Service

The Lock Service provides locking capabilities. It exposes two type of locks: eXclusive (X) and Shared (S). A shared lock allows multiple clients to read the referenced resource but does not allow any client to modify such a resource. Multiple shared locks can exist for the same resource. The exclusive lock allows only one client to modify (and read) the referenced resource.

The algorithm to grant a lock is pretty straightforward: the Lock Service checks only the locks for the correspondent resource (considering the URI) and applies the rules presented in table 5.3. This simple procedure allows to efficiently implements the algorithm, and the execution does not add overhead in addition to the one expected by the additional call to the Lock Service.

Operation	**HTTP Method**	**URL**
Create	`POST`	`/locks/`
Read	`GET`	`/locks/{LOCK_ID}`
Update	`PUT`	`/locks/{LOCK_ID}`
Delete	`DELETE`	`/locks/{LOCK_ID}`

Table 5.4: *Lock Service RESTful APIs*

Lock Service accepts `GET` , `POST` , `PUT` and `DELETE` methods to manage locks (see table 5.4). `POST` creates a new lock for a resource. `PUT` can only modify a lock type. `DELETE` removes a lock. `GET` allows reading the lock resource representation. This service does not allow the use of `PUT` to create a new lock.

The Lock Service receives a request to create a new lock with `POST` . If the Lock Service grants the lock, it creates the resource representing such a lock. Listing 5.1 presents an example of the representation of lock resource. The URI of the created lock resource is returned in `Location` HTTP header. The HTTP `201Created` status code is returned to indicate the requested succeeded.

Listing 5.1: *Example of lock resource*

```
{
    "type": "X",
    "resource-uri": "http://example.org/resources/A",
    "transaction-uri": "http://transaction.example.org/
       transactions/T1"
}
```

The Lock Service receives a request to modify a lock with `PUT` . The Lock Service allows only to upgrade the lock type from 'S' to 'X' and never vice-versa. The Lock Service verifies if the lock upgrade can be conceived and updates the resource. The Lock Service can use either `200OK` [64, section 6.3.1] or `204NoContent` [64, section 6.3.5] status code to indicate that the request succeeded but never `202Accepted` [64, section 6.3.3] which is used for asynchronous operations.

When the Lock Service receives a request to delete a lock with `DELETE` , it removes the associated lock resource. The Transaction Service is the only service authorised to invoke the removal of a lock and it does it either on commit or if the rollback procedure (see section 5.3.6) is terminated.

5.3.2 Transaction Service

The Transaction Service accepts `POST` , `PUT` and `DELETE` methods to manage a transaction (see table 5.5). `POST` for transaction creation, `DELETE` to rollback a transaction, `PUT` to commit the transaction. This service does not allow the use of `PUT` to create a new transaction.

Operation	HTTP Method	URL
Create Transaction	POST	/transactions
Commit a Transaction	PUT	/transactions/{TRANSACTION_ID}
Rollback a Transaction	DELETE	/transactions/{TRANSACTION_ID}
Create Initial Resource	PUT	/transactions/{TRANSACTION_ID}/{RESOURCE_RELATIVE_PATH}
Create Log Resource	POST	/transactions/{TRANSACTION_ID}/{RESOURCE_RELATIVE_PATH}/operations/

Table 5.5: *Transaction Service RESTful APIs*

The Transaction Service receives a request to create a new transaction with `POST` . The URI of the created transaction resource is returned in `Location` HTTP header [64, section 7.1.2]. The HTTP `201Created` status code is returned to indicate the request succeeded [64, section 6.3.2].

The content of the response (see listing 5.2) contains (i) the 'timestamp' of the transaction i.e., unix timestamp expressed in milliseconds; (ii) the 'timeout' i.e., the amount of time (expressed in milliseconds) the transaction will be automatically rollbacked if not committed or rollbacked from the client; (iii) the protocol version i.e., 1.0.

In case of failure (i.e., The transaction cannot be created) either a `4xx` or a `5xx` error [64, section 6.5, section 6.6] is returned to client depending on the failure occurred.

Listing 5.2: *Example of representation of a transaction resource*

```
{
    "timestamp": 1520329149000,
    "timeout": 2400,
    "protocol-version": "1.0"
}
```

The Transaction Service receives a request to commit a transaction with `PUT` . The transaction service allows only to add the property 'commit' with value true to the representation of the transaction resource (see listing 5.3). Once the property *commit* is set to *true* the transaction is closed and the Transaction Service denies any further operation on the transaction resource and subordinates by returning `403Forbidden` [64, section 6.5.3].

Listing 5.3: *Example of representation of the transaction resource sent to commit the transaction*

```
{
    "timestamp": 1520329149000,
    "timeout": 2400,
    "protocol-version": "1.0",
    "commit" : true
}
```

The Transaction Service receives a request to rollback a transaction with `DELETE` . If the target transaction resource exists and it has not been already committed, the Transaction Service starts a compensation procedure (see 5.3.6). The Transaction Service starts the compensation procedure also if the transaction expires due to the timeout being exceeded.

Initial and Logging Resources

For each 'active' transaction, the Transaction Service allows the Transaction Proxy to create subordinated resources (i.e., "resources that exist in relation to some other "parent" resource" [109]). In this case, the subordinated resources of the transaction contain the representation of a resource of the effective service in a certain moment. The Transaction Proxy saves the representation of any resource involved in the transaction before any attempt of modification to the resource sent to the effective service. We will refer to this resource as the initial resources. Initial resources are used by the compensation procedure (see 5.3.6) in case of rollback.

Listing 5.4 shows an example of the representation of the initial resource. The representation contains: (i) the resource URI i.e., 'resource-uri'; (ii) the URI of the lock conceived by the Lock Service to access the resource available to the resource URI i.e., 'lock-uri'; (iii) the content type of the resource representation i.e., 'content-type'; (iv) the initial representation of the resource on the effective service i.e., 'content'.

Listing 5.4: *Example of initial resource*

```
{
    "lock-uri": "http://lock.example.org/locks/lock-a",
    "resource-uri": "http://example.org/resources/A",
    "content-type": "application/json"
    "content": {
        // Representation of resource
    }
}
```

The Transaction Proxy creates initial resources with `PUT` to the URI calculated by composing the transaction URI with the relative URI of the resource on the effective service (e.g., `http://transactions.example.org/transactions/T1` + `/resources/A` results in `http://transactions.example.org/transactions/T1/resources/A`).

For each initial resource, the Transaction Service exposes a collection, named "operations", to enable logging of the operations made by the client on the effective resource (via Transaction Proxy). The Transaction Service allows logs creation with `POST` to the collection "operations" e.g., `POST http://transactions.example.org/transactions/T1/resources/A/operations`.

Listing 5.5 shows an example of a log resource which contains the HTTP request made by the client.

The URI of the created log resource will be created post-pending the timestamp to the log collection URI (e.g., `http://transactions.example.org/transactions/T1/res/A/operations/1520329149363`)

Listing 5.5: *Example of log resource*

```
{
  "method" : "PUT",
  "headers" : {
    "Accept" : "application/json",
    "Content-Type" : "application/json"
  },
  "content-body" : "....."
}
```

5.3.3 Transaction Proxy

The Transaction Proxy intercepts all requests directed to the effective service. Listing 5.7 shows how the Transaction Proxy manages intercepted requests.

As first action, the Transaction Proxy extracts the content of the request type (see row 2) both to use it for any interaction with additional services and to generate the responses.

Then, the Transaction Proxy checks the HTTP Method. If the HTTP method is `OPTIONS` (row 4), it sends the list of supported Transaction Services to the client (row 5). An example of `OPTIONS` response is shown in listing 5.6

Listing 5.6: *Example of proxy response to OPTIONS request*

```
{
  "transaction-managers" : [{
    "uri" : "http://transaction.example.org/transactions"
  }]
}
```

`OPTIONS` allows any Transactional client to know which Transaction Service(s) can be used to perform the transaction on the Effective Service with no prior knowledge (see section 5.5).

If the HTTP method is not one of `HEAD` , `GET` , `PUT` or `DELETE` , the Transaction Proxy replies (see rows 8 and 20) to the client with an error `405MethodNotAllowed` [64, section 6.5.5].

When the HTTP method is supported, the Transaction Proxy checks if the received request contains the HTTP header `X-Transaction-URI` (rows 9,10). The request arriving with this header from a Transactional Client is managed as a transaction action (row 12) as described in the next paragraph. Instead, the requests arriving from a Non-Transactional Client are managed as a "mini transaction" (row 15).

Listing 5.7: *Pseudo Code for Proxy function receiving HTTP request from clients*

```
1  function onReceive(httpRequest):
2    global var contentType = getContentType(httpRequest)
3
4    if httpMethod == OPTIONS then
5      return sendSupportedTransactionServicesList()
6    endif
7
8    if httpMethod in [ HEAD, GET, PUT, DELETE ] then
9      var transactionURI = httpRequest.getHeader("X-Transaction-URI")
10     if transactionURI != null then
11       // Transactional Client
12       return manageTransactionAction(httpRequest, transactionURI)
13     else
14       // Non-Transactional Client
15       return createMiniTransaction(httpRequest)
16     endif
17   endif
18
19   // POST and other methods are not allowed
20   return sendErrorResponseToClient(405, "Method Not Allowed")
```

Managing Transaction Requests

Listing 5.8 shows how the Transaction Proxy manages transaction actions. First of all, the Transaction Proxy checks if the request contains any references to previous obtained locks (i.e., lockURI, rows 2, 3).

If the Transactional Client does not provide any lock reference then the Transaction Proxy creates the lock for the target resource URI (i.e., resourceURI) (row 4). It is invoked the *createLock()* function to create the lock with a `POST` request to the Lock Service. The Lock Service creates the lock representation as indicated in listing 5.1 by using the provided argument. The HTTP Method (i.e., httpMethod) argument is used to identify correct lock type i.e., a shared lock is used for `HEAD` and `GET` while exclusive lock is used for `PUT` and `DELETE` . When the lock is granted, the Transaction Proxy reads the resource on the effective service with `GET` (row 5) and creates the initial resource on Transaction Service (row 7) using the function *createInitialResource()*.

If the Transactional Client provides the lock URI, then the Transaction proxy verifies and eventually upgrades (from S to X depending on the requested HTTP Method) the lock using the Lock Service (row 9).

The Transaction Proxy logs the action with a `POST` to the collection "operations" subordinated to the initial resource before forwarding the request to the effective service (rows 25). The `POST` content contains the received HTTP request information (see the example 5.5);

Finally, the Transaction Proxy forwards the request to the effective service and collects the received response (row 27). The Transaction proxy adds the lock URI (row 29) to the received response using `X-Lock-URI` HTTP header and returns it to the requesting client (row 33).

If a client sends a request either to delete or to create a resource, the Transaction Proxy also locks the parent URI (which represents the resource collection) with an exclusive lock (rows 19,20). This lock is required to implement the isolation property properly (see section 5.5).

A Transaction Client owning the parent lock URI (i.e., `X-Parent-Lock-URI` HTTP header) sends always it in any request. The Transaction Proxy verifies this header as it does for `X-Lock-URI` HTTP header (see row 14).

If the Transaction Proxy receives a `PUT` request, it checks it to discriminate among a create or an update request (rows 15-17). A create request is identified by checking the initial content of the resource.

The Transaction Proxy adds the parent lock URI to the response header if any (rows 30-32).

Listing 5.8: *Pseudo Code for Proxy function which handles any transaction action*

```
1  function manageTransactionAction(httpRequest, transactionURI):
2    var lockURI = httpRequest.getHeader("X-Lock-URI")
3    if lockURI == null then
4      lockURI = createLock(httpMethod, transactionURI, resourceURI)
5      var response = getResource(resourceURI)
6      var content = response.content
7      var initialResourceURI = createInitialResource(transactionURI,
          resourceURI, lockURI, content)
8    else
9      verifyAndUpgradeLock(httpMethod, lockURI, transactionURI, resourceURI)
10     var initialResourceURI = getInitialResourceURI(transactionURI,
          resourceURI)
11     var content = getContentFromInitialResource(initialResourceURI)
12   endif
13
14   if httpRequest.getHeader("X-Parent-Lock-URI") == null then
15     if httpMethod == PUT AND content == null then
16       var create = true;
17     endif
18     if create OR httpMethod == DELETE then
19       var parentURI = getParentURI(resourceURI)
20       var parentLockURI = createLock(httpMethod, transactionURI, parentURI)
21       // initial resource of collection is not needed
22     endif
23   endif
```

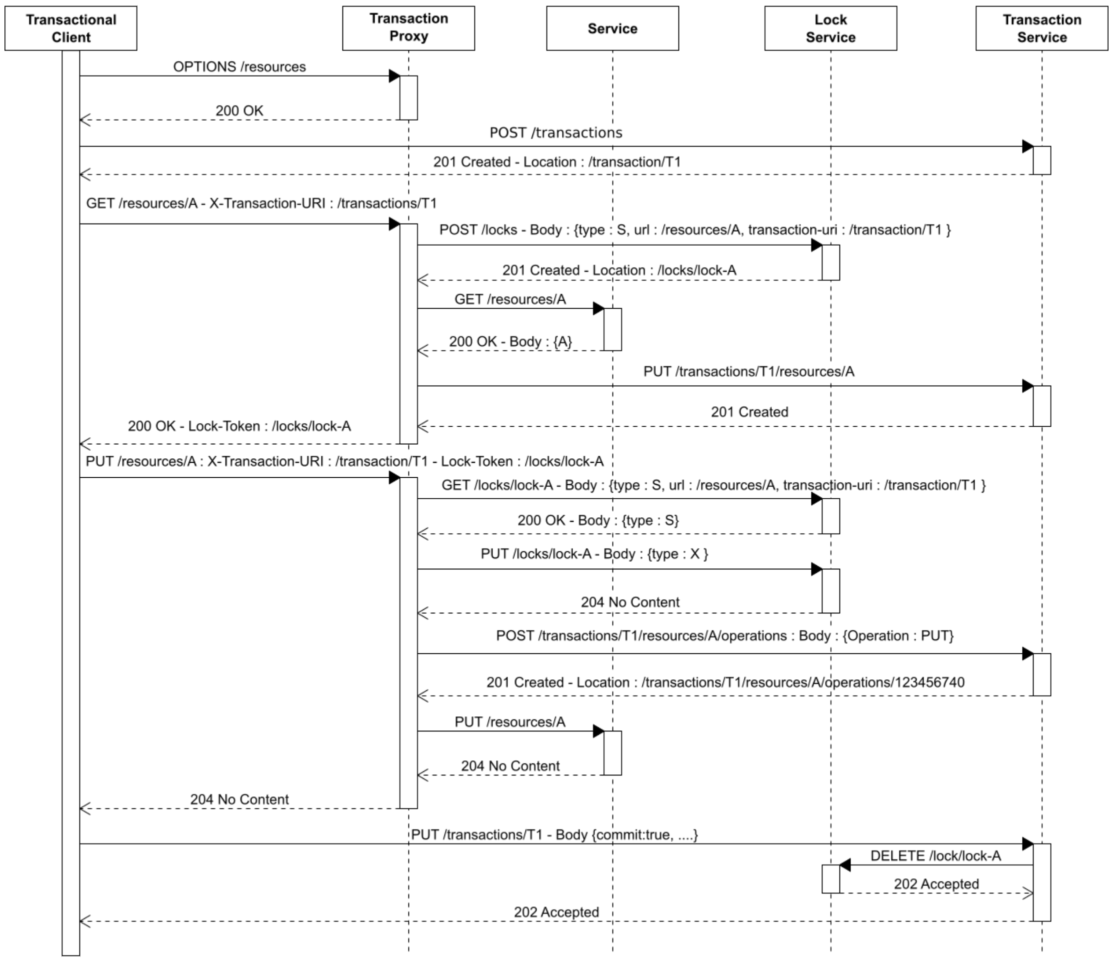

Transactional Client
Transaction Proxy
Service
Lock Service
Transaction Service
OPTIONS /resources
200 OK
POST /transactions
201 Created - Location : /transaction/T1
GET /resources/A - X-Transaction-URI : /transactions/T1
POST /locks - Body : {type : S, url : /resources/A, transaction-uri : /transaction/T1 }
201 Created - Location : /locks/lock-A
GET /resources/A
200 OK - Body : {A}
PUT /transactions/T1/resources/A
201 Created
200 OK - Lock-Token : /locks/lock-A
PUT /resources/A : X-Transaction-URI : /transaction/T1 - Lock-Token : /locks/lock-A
GET /locks/lock-A - Body : {type : S, url : /resources/A, transaction-uri : /transaction/T1 }
200 OK - Body : {type : S}
PUT /locks/lock-A - Body : {type : X }
204 No Content
POST /transactions/T1/resources/A/operations : Body : {Operation : PUT}
201 Created - Location : /transactions/T1/resources/A/operations/123456740
PUT /resources/A
204 No Content
204 No Content
PUT /transactions/T1 - Body {commit:true, ...}
DELETE /lock/lock-A
202 Accepted
202 Accepted

Figure 5.2: *Example of a sequence diagram showing the interaction between different components. In this example, a client creates a transaction, reads a resource, and then updates such a resource. Finally the client commits the transaction.*

```

    logRequest(initialResourceURI, httpRequest)

    HttpResponse actionResponse = forwardAction(httpRequest)

    actionResponse.setHeader("X–Lock–URI", lockURI)
    if parentLockURI != null then
      actionResponse.setHeader("X–Parent–Lock–URI", parentLockURI)
    endif
    return actionResponse
```

Figure 5.2 shows the sequence diagram of the operations for a transaction involving the read and the update of a resource made by a Transactional Client.

Proxy Transaction for Non-Transactional Client

Listing 5.9: *Pseudo Code for Proxy function which handles non transactional client requests*

```
function createMiniTransaction(httpRequest):
    String transactionURI = createTransaction()
```

```
3   var actionResponse = manageTransactionAction(httpRequest, transactionURI)
4   commitTransation(transactionURI)
5   actionResponse.removeHeader("X-Lock-URI")
6   actionResponse.removeHeader("X-Parent-Lock-URI")
7   actionResponse.removeHeader("X-Transaction-URI")
8   return actionResponse
```

When the Transaction Proxy receives a request not containing any transaction reference, the client is considered Non-Transactional. The Transaction Proxy creates a transaction (row 2), then it manages the requested operation as any other transaction action (row 3) (see paragraph 5.3.3). The Transaction Proxy commits the transaction (row 4) on behalf of the client. The Transaction Proxy cleans the obtained response from the additional header before returning the response to the Non-Transactional Client (row 8) (rows 5-7)

Every action made from a Non-Transactional Client is de-facto a mini-transaction involving a single operation on the Effective Service.

5.3.4 Non-Transactional Client

A Non-Transactional Client is either a client which is not aware of the transaction model (e.g., any legacy client) or a client which is not interested in creating a transaction. This client behaves as it would have been if there were no transactions. It performs REST requests to the Effective Service and uses the responses (intercepted and managed by the Transaction Proxy) to continue its workflow.

5.3.5 Transactional Client

A Transactional Client discovers the Transaction Service by requesting `OPTIONS` to the resource collection URI. The request is intercepted and managed by the Transaction Proxy which provides the list of supported Transaction Services. The client creates a transaction by sending a `POST` to the Transaction Service and then it requests all the transaction operations via the Transaction Proxy.

A Transactional Client always sends the owned transaction URI in the header of the HTTP request (using `X-Transaction-URI` header).

The Transactional Client collects every lock URI it receives (via `X-Lock-URI` header in the response) and associates them to the proper resource (using the resource URI).

Anytime the Transactional Client requests an action, it indicates the owned locks using the HTTP headers. If a Transactional Client does not include the lock it already owns, it receives the errors code `423Locked` [53, section 11.3].

If a Transactional Client receives a parent lock URI, it associates the lock to the resource collection. The Transactional Client sends the parent lock URI it owns for any succeeding resources creation and resources deletion.

The Transactional Client can terminate the transaction by sending either a commit (by using `PUT`) or rollback (by using `DELETE`) request.

5.3.6 Compensation Procedure

If a client sends a request to rollback a transaction or the transaction timeout expires, the transaction service starts the compensation procedure.

The procedure consists in analysing all the subordinated resources of the transaction (initial resources and logs). The resources accessed only with safe operations (`GET` or `HEAD`) do not require any action. Updated resources require (`PUT`) compensation to the initial values. Deleted resources have to be re-created with a `PUT` to the original resource URI and using the initial values. The resources created within the transaction must be deleted (by using `DELETE`).

Table 5.6 shows the compensation operations used to rollback.

CRUD Operation	HTTP Method	Compensation HTTP Method	Compensation Body Content
Create	`PUT`	`DELETE`	No
Exists	`HEAD`	Unneeded	N/A
Read	`GET`	Unneeded	N/A
Update	`PUT`	`PUT`	Initial Representation
Delete	`DELETE`	`PUT`	Initial Representation

Table 5.6: *Compensation Operations*

The next section analyses the RESTful Transaction State of the Art.

5.4 RESTful Transaction Model State of the Art

Mihindukulasooriya et al. [97, 98] have surveyed the state of the art of the transaction models introducing a set of scenarios (see section 5.5.1) and analysing the support of each solution for the presented scenarios. No one of the solution analysed by the authors is capable of addressing all the scenarios. They also highlighted the adherence of the transaction models to the REST principles (see table 5.8) and to the HTTP and other relevant properties (see table 5.9).

Table 5.7 enumerates the state of the art proposals which are going to briefly described.

In this dissertation, it is proposed a model capable of addressing all scenarios presented by Mihindukulasooriya et al. [97, 98] and capable of addressing most of the challenges they evidenced.

This section summarises the solutions analysed in the cited survey and compares each of them with the proposal reported in this dissertation.

The overloaded `POST` pattern has been the first approach to RESTful transactions [97,98]. The overloaded `POST` pattern is characterised by putting several HTTP operations in the payload of a single `POST` operation. This approach fits well for batched and short-lived transactions. Unfortunately, the approach does not respect the HATEOAS constraint and is not suitable for distributed transactions.

In 2007, Richardson and Ruby proposed a transaction as resource approach [109]. In their proposal, the client opens a transaction by creating a resource to a transaction service. The client performs every subsequent operation by creating a resource with `PUT` as a subordinate of the transaction resource. This solution supports only the resources creation.

Khare [89] proposed an enhancement of the REST architectural style for distributed and decentralised systems which includes five different extensions. One of these extensions, REST with Delegation (REST+D), dealing with ACID transactions, proposes a mutex lock proxy component which provides mutually exclusive access to the origin

	Type	Year	Transaction Model	Authors
1	Technical	~2000	Batched Transaction with Overloaded `POST`	
2	Technical	2007	Transaction as Resource	Richardson, L., Ruby, S [109]
3	Scientific	2009	Optimistic technique for transaction using REST	da Silva Maciel, L.A.H., Hirata, C.M. [42]
4	Scientific	2009	A consistent and recoverable RESTful transaction model (RETRO)	Marinos, A., Razavi, A., Moschoyiannis, S., Krause, P. [93, 108]
5	Scientific	2010	Timestamp-based two phase commit protocol for RESTful services (TS2PC4RS)	da Silva Maciel, L.A.H., Hirata, C.M. [43, 44]
6	Scientific	2011	Try-Cancel/Confirm pattern (TCC)	Pardon, G., Pautasso, C. [104, 105]
7	Scientific	2012	Atomic REST batched transactions	Kochman, S., Wojciechowski, P.T., Kmieciak, M. [91]
8	Scientific	2015	REST+T	Dey, A., Fedeke, A., Röhm, U. [48]
9	Scientific	2018	Resilient Two-Phase Locking	This Research Proposal

Table 5.7: *RESTful Transaction Model State of the Art*

server and ensures total serialisation of all updates to a resource. Their proposal, such as REST which tries to extend, is an architectural style and it “does not provide any details regarding how to execute the scenario” [98].

da Silva Maciel et al. [42] proposed an optimistic technique. In their model, the REST service must support resource versioning. The proposal uses compensation techniques to rollback a transaction, but it also extends it by clarifying the requirements of performing the compensation actions within locks.

Marinos et al. proposed RETRO [93, 108] which uses the concepts of transaction as resource, locks and temporary resources to achieve isolation. Their solution uses the hyperlink to meet HATEOAS constraint. The temporary resources approach introduces link transparency issues [98]. Our solution shares some ideas with RETRO i.e., transaction as resource, locks and use of hyperlinks but instead of using the concept of temporary resources, it uses an approach based on initial resource representation and logging as resource to support rollbacks via the appropriate compensation technique. RETRO uses non-standard HTTP methods and header. On the contrary, our solution uses standard HTTP methods only.

da Silva Maciel et al. [43,44] proposed a timestamp based two-phase commit RESTful transactions (TS2PC4RS) designed for reservation-based services. In their last version [45], they also introduced the concept of logs “as a fault-tolerant mechanism capable of recovery connection and server failures”. Our research proposal does not require the effective service to adhere to a specific pattern. We also use the logs to support compensation and to be fault-tolerant, but we expose them as resource to fully comply with ROA.

Pardon and Pautasso [104, 105] proposed try-cancel/confirm (TCC) approach. Their solution shares some ideas with TS2PC4RS, but TCC is only applicable if the reservation *fits directly into the business model* [105] (i.e., a ticket reservation). Our solution does not require services to adhere to any particular business model.

Kochman et al. [91] proposed a solution based on batched transactions which use mediators and proxies. Their solution allows transactional and non-transactional clients to coexists. Our solution uses a proxy to support the same clients, but we use a two-phase locking protocol instead of the batched transactions.

Dey et al. proposed REST with Transaction (REST+T) [48]. This solution is not stateless and uses non-standard HTTP methods.

5.5 Analysis of the Resilient Two-Phase Locking Transaction Model

Table 5.8 shows a summary of the adherence of the proposal enumerated in table 5.7 to the transaction properties and REST principles .

According to ROA, the resource collection URI returns the list of the resources (list of URIs). Every URI can be considered a resource per se. A client read the resource collection by using the HTTP `GET` method. The only way to modify the resource collection is by either deleting a resource or creating a new one.

The lock service grants a lock only by analysing if exists another lock for the same resource (by using its URI).

To delete a resource, the client performs a `DELETE` to the resource URI. The Resilient Two-Phase Locking Transaction Model requires two exclusive locks, one for the resource and one for the resource collection giving that it is going to modify two representations (the resource and the collection). The lock for the resource prevents that any other client can interact with such a resource. The lock for the collection prevents that any other client can interact with the resource collection.

To create a resource, the client has to use the HTTP `PUT` request. With `PUT` , the client performs a request the URI where the resource will be available. The resource creation with `PUT` requires two exclusive locks, one for the resource and one for the resource collection for the same considerations of the delete operation.

It is not so straightforward to create a resource with `POST` . The URI of the resource created with a `POST` is not known, and the Transaction Proxy cannot request a lock for the created resource URIs. The lock for the resource collection is not enough because a concurrent client could create a resource with `PUT` directly to the resource URI (which could result in an update which violates the Isolation property). The Transaction Proxy could transform a `POST` request into a `PUT` request only if it is capable of defining the appropriate URI for the resource. This is usually possible if the Transaction Proxy knows the semantic of the Effective Service. For this reason, our transaction model does support `POST` only when it is tailored to a known Effective Service.

The presented solution works with RESTful services which do not provide nested resource collection (e.g., the bookmark ROA example presented in table 5.2). To delete

[6]Possible Lost Update Problem
[7]Clients are made aware of pending uncommitted actions
[8]Temporary Resources having Uniform Resource Locator (URL) have been criticized
[9]Could be added

Property	**Description**	**Transaction Model**								
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	Transaction Properties									
Atomicity	All or Nothing	Y	Y	Y	Y	Y	Y	Y	Y	**Y**
Isolation	Hidden events between concurrent running transaction	Y	Y[6]	N	Y	N	N	Y	Y[7]	**Y**
	REST Principles									
Statelessness	Server does not stores any information regarding previous interaction with the client (e.g. no session exists)	Y	Y[8]	N	Y[8]	N	Y	Y	N	**Y**
Uniform Interface	Management of transaction and management of resources involved in the transaction are made using the same paradigm	Y	N	Y	Y	Y	Y	Y	Y	**Y**
Identification Of Resources	All involved resources must be identifiable (transactions, locks, etc)	N	Y	N	Y	N	Y	N	Y	**Y**
Manipulation of Resources Through Representations	Create and Update must contains a representation. Read must return a representation not just hyperlink or successful messages	Y	Y	Y	Y	N	N	Y	Y	**Y**
Self-descriptive Messages	The messages flowing between server and client must not have implicit information	Y	Y	Y	Y	N	N	Y	Y	**Y**
HATEOAS	Hypermedia As The Engine Of Application State	N	N	N	Y	N	N	N[9]	N	**Y**
Layered System	The transaction is provided by a dedicated layer or everything is demanded to the service implementation	Y	Y	N	Y	N	N	Y	N	**Y**

Table 5.8: *RESTful Transaction Model Properties*

a resource having subordinates, no one should be capable of interact with the subordinates. To support this scenario, the Lock Service should be enhanced to use a different algorithm that supports subordinated resources.

Marinos et al. [93, 108] have formally demonstrated that their locking mechanism is well formed and sound. However, it supports only read and update operations with a two-phase locking protocol by releasing acquired locks only at commit or rollback time. Our proposal does not only provide a similar approach to their proposal. Rather, we also support the creation (via HTTP `PUT` request only) and the deletion of resources. These operations also generate an update of the resource collection (which is a resource per se), and therefore our proposal provides a solution to safely interact with the resource collection besides the created/deleted resource.

Property	Description	Transaction Model								
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
HTTP related properties										
Semantic not violated	Safety and Idempotency of HTTP methods must be respected	Y	Y	Y	Y	N	Y	Y	?	**Y**
Common verb supported	HTTP methods and header field are standard	Y	Y	N	N	Y	Y	Y	N	**Y**[10]
HTTP Methods adherence	HTTP verbs are used as defined in HTTP 1.1 standards	Y	Y	N	N	Y	Y	Y	N	**Y**
Low overhead	Additional Round Trips	Y	Y	Y	N	N	Y	Y	Y	**Y**[11]
ROA URI adherence	Compliance with ROA proposed URI pattern	Y	Y	Y	Y	Y	Y	Y	Y	**Y**
Miscellaneous properties										
Supported operations (CRUD)	Operation supported within the transaction	?	?	Y	RU	CR (U)	CD	Y	Y	**Y**[12]
Optionality	Client and Server transaction unaware can co-exist with the one supporting them (important for interoperability and wide adoption of the model)	Y	?	?	Y	?	?	Y	N	**Y**
Discoverable	All the metadata needed to execute the transactions can be discovered in RESTful way without out-of-band knowledge (i.e. following links)	?	?	?	Y	?	?	Y	N	**Y**
Distributed Transactions	REST transactions involving more than one services	N	N	Y	?	Y	Y	?	?	**Y**
Service Paradigm Required	The REST service has to adhere to a certain paradigm	N	N	Y	?	Y	Y	N	Y	**N**
Heterogeneity of Service Types	Different service type (e.g. reservation based and pure resource ones) can be part of the same transaction (only available for distributed transaction)	N/A	N/A	Y(?)	?	?	N	?	?	**Y**

Table 5.9: *RESTful Transaction Model Challenges*

Table 5.9 shows a summary of the challenges for the RESTful transactions models presented in Table 5.7.

[10]Three non standard HTTP header field

[11]Clients perform three additional requests. The first uses `OPTIONS` to discover the Transaction Service. The client may cache the result. Hence, the client performs the `OPTIONS` request only before the first transaction. The remaining two requests are used to create and commit (or rollback the transaction). It is the minimum number of additional operations of every transaction mechanism.

[12]Create is supported via `PUT` (not via `POST`).

Our solution provides ACID transactions and all the three envisioned additional services, i.e., the Transaction Proxy, the Transaction Service, and the Lock Service, are stateless . The Transaction Service saves an application state [109, page 90] to support compensation and resiliency (i.e., the ability to return to its original state after a component failure, see paragraph 5.5.2). Conversely from Marinos et al. [93, 108], such an application state does not create temporary resources, and this avoids the link transparency issue they introduced [97, section 3.3].

Our solution is layered, it respects the uniform interface constraint, and the fours additional constraints the uniform interface introduces (manipulation of resources through representations; self-descriptive messages; Hypermedia As The Engine Of Application State (HATEOAS); and identification of resources).

The `OPTIONS` method allows achieving discoverability. This means that “all the metadata needed to execute the transactions can be discovered in a RESTful manner without out-of-band knowledge (i.e., following links)” [97]. By exploiting this feature, our model can exploit the out-of-band knowledge to support a distributed transaction. By querying all the involved proxies, a client discovers if a common transaction service exists. If it exists, then all the lock services used by the proxies can interact by implementing a protocol for deadlock detection.

The proposed approach respects the semantics of HTTP 1.1 methods. It only uses three non-standards HTTP headers: `X-Transaction-URI` , `X-Lock-URI` and `X-Parent-Lock-URI` . `Lock-Token` header defined for WebDAV [53, section 10.5] could be a possible standard alternative to `X-Lock-URI` .

Moreover, clients not supporting our model can coexist with the ones supporting it (optionality). Services are always not aware of the extended behaviour introduced by our transaction model, and they do not need to conform to any particular pattern. They have only to be RESTful and do not expose subordinated resources.

Khare and Taylor [89] [90] define the differences between decentralised and distributed systems. Decentralisation permits independent agencies to make their own decisions. In distributed scenarios, instead, agents share control of a single decision.

Our model supports RESTful distributed transactions, i.e., transactions of different services, under the following additional constraints. First of all, all the services should trust a common Transaction Service. The Lock services should share a common distributed deadlock detection algorithm. Moreover, distributing the transactions will imply to build the initial resource URIs encoding the effective services base URI. This will allow to distinguish different services and expose their collections to the same relative URI.

5.5.1 Scenarios

Mihindukulasooriya et al. proposed nine scenarios [98, section 5.2] enabling to evaluate the coverage of any RESTful transaction model.

In Scenario I, two resources belonging to a single application are updated. Table 5.10 shows how our protocol supports such a scenario. Our model requires seven client calls to execute the scenario instead of the four ideals calls (3-6). Three additional calls represent 75% of overhead in the presented scenario, but this number is constant because the number does not change with the number of operations made in the transaction.

HTTP Method	URL	Response
OPTIONS	`http://example.org/resources`	200 OK
PUT	`http://transaction.example.org/transactions/`	201 Created Location: `http://transaction.example.org/transactions/T1`
GET	`http://example.org/resources/A`	200 OK
GET	`http://example.org/resources/B`	200 OK
PUT	`http://example.org/resources/A`	204 No Content
PUT	`http://example.org/resources/B`	204 No Content
PUT	`http://transaction.example.org/transactions/T1`	204 No Content

Table 5.10: *Scenario I*

Table 5.11 presents Scenario II using `PUT` since our proposal does not support the resource creation with `POST` .

HTTP Method	URL	Response
OPTIONS	`http://example.org/resources`	200 OK
PUT	`http://transaction.example.org/transactions/`	201 Created Location: `http://transaction.example.org/transactions/T1`
GET	`http://example.org/resources`	200 OK
PUT	`http://example.org/resources/C`	201 Created
GET	`http://example.org/resources/A`	200 OK
PUT	`http://example.org/resources/B`	204 No Content
PUT	`http://transaction.example.org/transactions/T1`	204 No Content

Table 5.11: *Scenario II*

Scenario III is similar to Scenario I, but the update operation must be asynchronous because the call is expected to take a long time to execute. In scenario III, the Response to the client `PUT` operation is `202Accepted` in place of `204NoContent` . Our proposal supports this scenario, but it does not have a good fit for long-running transactions because it uses pessimistic locks which block resources.

Scenario IV is also similar to the scenario I, but the two resources belong to two different applications, see table 5.12. We support such a scenario for distributed services but not for decentralised ones, as presented in the previous section. Steps one and two are executed to discover the supported Transaction Service. Clearly, the client can proceed only if it finds a common Transaction Service.

HTTP Method	URL	Response
OPTIONS	`http://example.org/resources`	200 OK
OPTIONS	`http://remote.example.org/res`	200 OK
PUT	`http://transaction.example.org/transactions/`	201 Created Location: `http://transaction.example.org/transactions/T1`
GET	`http://example.org/resources/A`	200 OK
GET	`http://remote.example.org/res/B`	200 OK
PUT	`http://example.org/resources/A`	204 No Content
PUT	`http://remote.example.org/res/B`	204 No Content
PUT	`http://transaction.example.org/transactions/T1`	204 No Content

Table 5.12: *Scenario IV*

Scenario V is a rollback scenario where a server rejects an update (i.e., the second update) by responding with `409Conflict` [64, section 6.5.8] because another client updated the same resource before. This scenario cannot happen in our transaction model because the actions of the first client lock the resource and the second client cannot obtain an Exclusive lock for the same resource. In such a case, the second client obtains a `403Forbidden` while it tries to read the resource. It is in charge of the client either to retry after a delay (scenario V.a see table 5.13) or rollback by deleting the transaction resource (scenario V.b see table 5.14).

HTTP Method	URL	Response
OPTIONS	`http://example.org/resources`	200 OK
PUT	`http://transaction.example.org/transactions/`	201 Created Location: `http://transaction.example.org/transactions/T1`
GET	`http://example.org/resources/A`	200 OK
PUT	`http://example.org/resources/A`	204 No Content
GET	`http://example.org/resources/B`	403 Forbidden
GET	`http://example.org/resources/B`	200 OK
PUT	`http://example.org/resources/B`	204 No Content
PUT	`http://transaction.example.org/transactions/T1`	204 No Content

Table 5.13: *Scenario V.a with delayed retry*

HTTP Method	URL	Response
OPTIONS	`http://example.org/resources`	200 OK
PUT	`http://transaction.example.org/transactions/`	201 Created Location: `http://transaction.example.org/transactions/T1`
GET	`http://example.org/resources/A`	200 OK
PUT	`http://example.org/resources/A`	204 No Content
GET	`http://example.org/resources/B`	403 Forbidden
DELETE	`http://transaction.example.org/transactions/T1`	202 Accepted

Table 5.14: *Scenario V.b with rollback*

Scenario VI (see table 5.15) shows a voluntary rollback of the client.

HTTP Method	URL	Response
OPTIONS	`http://example.org/resources`	200 OK
PUT	`http://transaction.example.org/transactions/`	201 Created Location: `http://transaction.example.org/transactions/T1`
GET	`http://example.org/resources/A`	200 OK
GET	`http://example.org/resources/B`	200 OK
PUT	`http://example.org/resources/A`	204 No Content
DELETE	`http://transaction.example.org/transactions/T1`	202 Accepted

Table 5.15: *Scenario VI*

In Scenario VII, the client fails in the middle of a transaction, for instance after the second PUT in scenario I. In such a situation, the transaction is not committed. The Transaction Service rollbacks the transaction if the timeout expires.

In Scenario VIII, the server fails in the middle of a transaction, for instance with a '500 Internal Server Error' [64, section 6.6.1]. This situation is similar to scenario V, the client can decide either to retry with a delay (scenario V.a) or rollback the transaction (scenario V.b).

Scenario IX is about communication losses and message losses. Either the request or the response message could get lost due to unreliable network communication. This scenario opens different cases to analyse.

The first case occurs when the client does not receive a reply from the Transaction Proxy while updating a resource. If the Transaction Proxy fails before the lock is requested, then the client is still able to get the lock and proceed by retrying the operation. If instead the Transaction Proxy fails after the lock is obtained (or the reply message is lost due to network issues), then the lock has been set on the resource, but the client does not receive the lock URI. If the client retries the operation, it obtains a `403Forbidden` because the Transaction Proxy is not able to obtain the lock and the client can only rollback the transaction. In any case, the Transaction Service rollbacks the transaction if the timeout expires. Alternatively, the Lock Service could use a `301MovedPermanently` [64, section 6.4.2] if it recognises that a Transaction Proxy is trying to create the same lock for the same resource and transaction.

The second case occurs when the client does not receive the response. The client can retry the operation because the `PUT` operation is idempotent.

5.5.2 Resiliency

All the three envisioned additional services, i.e., the Transaction Proxy, the Transaction Service, and the Lock Service are stateless and can be replicated and load balanced to improve scalability and availability.

The Transaction Service annotates the progress of the rollback by annotating each step on the resources representing the transaction. In particular, the Transaction Service

- adds a property to the transaction resource representation to indicate it is starting the rollback;
- for each initial resource representation:
  - adds a property to indicate it is going to compensate the corresponding resource;
  - modifies the previous property to indicate that the corresponding resource has been compensated;
  - add a property to indicate it is going to release the associated lock;
  - modifies the previous property to indicate it has released the associated lock;
- modifies the properties on the transaction resource representation to indicate that the rollback is terminated successfully.

This procedure behaves like a journal which enables any Transaction Service to take in charge the rollback procedure of another instance of the service by restarting it from the last completed step.

Scenarios VII, VIII and IX show the management of failures either on the Proxy or Lock services.

Clearly, the Lock Service must prevent starvation and deadlock. It is possible to replicate and distribute the Lock Service by using one the distributed deadlock detection algorithms surveyed by Ahmed K. Elmagarmid [56].

Next chapter presents the IS architecture which is compliant with REST principles and which contains an extension of the transaction model presented in this section. The extension shows the capacity of adapting our RESTful Transaction Model to support actual RESTful services which differ from the ideal scenario. The layered design of this proposal provides this adaptability.

CHAPTER 6

Transactional REST Information System

The Information System (IS) is the key service for any research infrastructure federation. It acts as a registry of the infrastructure, and it offers through either query answering or notifications [13] a global view (infrastructure level) and a partial view (Virtual Research Environment (VRE) level) of:

- its resources (e.g., computing, storage, services, software, datasets);
- their current status (e.g., available, down, not responding);
- their relationships with other resources;
- the policies governing their exploitation.

This chapter presents the design of the IS starting from the identification of the functional and non-functional requirements (section 6.1). Section 6.2 presents the IS architecture. Section 6.3 presents the core component of the IS (i.e., the Resource Registry) and the exposed functionalities. Section 6.4 presents an assessment of the database families to argue about the choice of Graph Database. Section 6.5 defines an extended version of the transaction model presented in chapter 5. Section 6.6 shows how the information system has been interfaced with a subscription notification system and motivates the benefits of including such extension. Finally, section 6.7 introduces the required deployment architecture to satisfy the non-functional requirements.

6.1 Requirements

The analysis of the context of study (see chapter 2) led to model the infrastructure as a System of Systems (SoS). This choice allowed the identification of a number of requirements (see section 2.5). In particular, the IS must be capable of:

- supporting a flexible model;
- providing coherent and consistent global and partial views of the resources and their relationship to the clients (the perspective of the client depends on the context the client is running and never by a filter, a parameter or a configuration);
- supporting consistent updates across the provided views;
- scaling according to workload changes.

In software engineering, requirements are typically divided into Functional and Non-Functional requirements. Functional requirements define a function of a system or its components. Non-Functional requirements specify criteria that can be used to evaluate the operation of a system, rather than specific behaviour.

From the functional requirements point of view, the IS must be capable of:

1. supporting the IS Model (section 4.2). The system must support the constraints and capabilities defined by the IS Model. In particular:
   (a) contexts management. It requires Application Programming Interfaces (APIs) to manage hierarchical contexts as defined by the IS Model;
   (b) types management. The system must enable IS model instantiation (e.g., gCube Model). This requires :
      i. the definition of a Data Definition Language (DDL);
      ii. the design of dedicated APIs.
   (c) instances management. Create, Read, Update, Delete (CRUD) of any entity and relation types defined in the system. To support these requirements the system must:
      i. provide a Data Manipulation Language (DML);
      ii. support univocal identification of any entities and relations;
      iii. support instances validation against the registered schema.
   (d) referential integrity. The system has to guarantee that any incoming or outcoming relation to/from a resource is removed if that resource is removed;
   (e) propagation constraints. The system has to enforce the defined *remove* propagation constraints to the target entity if a client deletes either the source entity or the relation between that source and target entities.
2. supporting multi-tenancy[1] exploitation of resources in contexts, hereafter shortly referred as multi-context by offering:
   (a) APIs to share instances across contexts;
   (b) consistent views across contexts: entity and relation representations must be observable at the same time from any context the instances belong to;
   (c) context views as well as global view at any level of the context hierarchy;

[1]Historically is considered a non-functional requirement, but in this context it provides a behaviour having its own features.

(d) propagation constraints enforcement to observe both *add* and *remove* propagation constraints to the target entity if a client adds or remove either the source entity or the relation between that source and target entities;

3. supporting dynamic queries. The system must support queries built dynamically from clients rather than provided as an explicit access pattern from the system;
4. supporting Atomicity, Isolation, Durability and Consistency (ACID) transaction. The system must support transactions to enable the client to perform more than one action as a single unit;
5. supporting subscription notification. The system must provide the possibility to subscribe for a certain event and notify the subscriber when the event occurs. This feature avoids continuous polling to the IS to capture modifications of the resources description.

The identified non-functional requirements are:

1. High-Availability (HA): it is a characteristic of a system, which aims to ensure an agreed level of operational performance for a higher than the normal period;
2. Eventual Consistency: it is a consistency model used in distributed computing to achieve high availability. It informally guarantees that, if a given data item does not receive new updates, eventually all accesses to that item will return the last updated value [22]. Consistency, Availability, Partitionability (CAP) theorem [27] states that it is impossible for a distributed computer system to provide more than two of the following three guarantees:
   - Consistency (C): every read receives the most recent write or an error;
   - Availability (A): every request receives a response, without a guarantee that it contains the most recent version of the information;
   - Partitionability (P): the system continues to operate despite arbitrary partitioning due to network failures.

   Given the CAP theorem and the fact that Availability and Partitionability are mandatory requirements for Hybrid Data Infrastructures (HDIs), we indicated the Eventual Consistency requirement instead of the more strong Consistency.
3. Horizontal Scalability (HS): it is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth [118] by adding more nodes to (or remove nodes from) a system.

It is important to stress that there is an interdependence between these two types of requirement and the boundaries are not always explicit [54]. The provided implementation of a functional requirement could impact on the achievement of a non-functional requirement. Moreover, the practical achievement is a mix of architectural software design and deployment architecture. The design may enable more than one deployment architectures and this is important to accommodate the needs of the specific scenarios where the system have to be exploited.

The design of the IS uses REpresentational State Transfer (REST) because it provides a consolidated architectural style to implement functional requirements while guaranteeing HA and HS of the system through replication at the same time. Moreover, REST allows maintaining the cost of designing and implementing clients and servers very affordable.

6.2 Information System Architecture

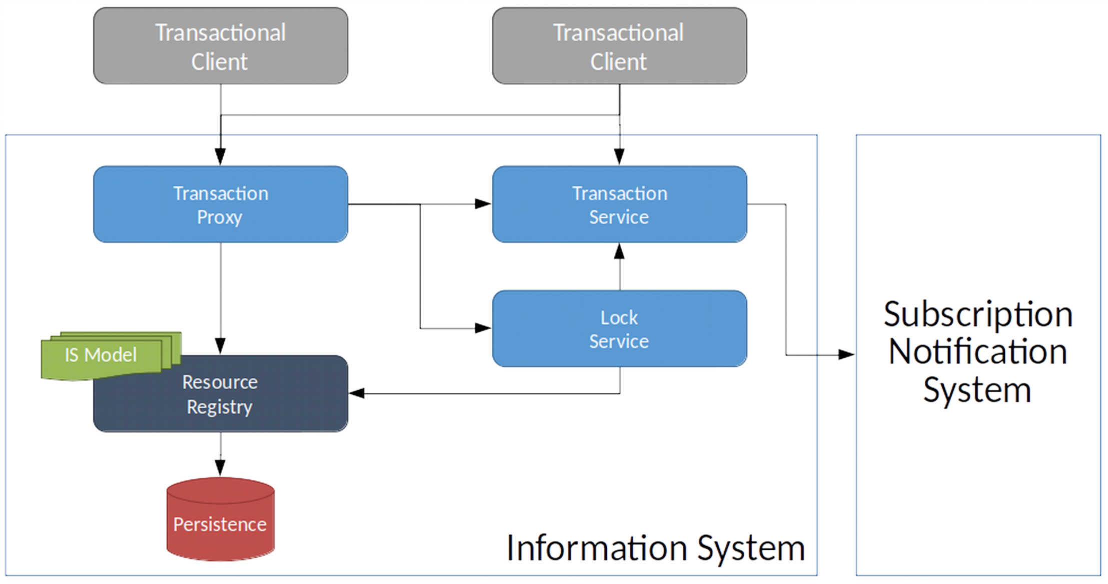


Figure 6.1: *IS Architecture*

The IS is composed by six components, as depicted in fig. 6.1:

Resource Registry: the core component of the IS;

IS Model: the model used by the Resource Registry (see chapter 4.2);

Graph Database: a graph database used as persistence by the Resource Registry;

Transaction Proxy: a proxy intercepting all the requests directed to the resource registry to provide a REST transactional behaviour;

Lock Service: provides lock as resource capabilities enabling the Transaction Proxy to safely operate on any resource;

Transaction Service: provides transaction management and notifies about the committed operations to Subscription Notification System which implements the publish-subscribe pattern.

The next section (section 6.3) presents the Resource Registry.

6.3 Resource Registry

Resource Registry is a web service which represents the core component of the IS. It is designed to comply with Resource Oriented Architecture (ROA) by grouping the required management APIs logically and making them *"as a resource"*. The Resource Registry exposes five port-type. In this dissertation the term port type is used to indicate the first level Uniform Resource Locator (URL) path starting from the service base path, i.e., if the service base path is `/` then the URLs `/a` and `/b` are two different port types. Each port type exposes RESTful APIs to satisfy one or more of the functional requirements identified in section 6.1.

Contexts Management: manages hierarchical contexts as defined by the IS Model (see chapter 4.2). A VRE is a typical context managed by the Resource Registry (see functional requirement 1a in section 6.1);

Types Management: manages the definition of entities and relations types and their schema. This choice allows for easy extension and support modification to the resource model. This is the key factor for the sustainability of the service and infrastructure that have to last for several years (see functional requirements 1b in section 6.1);

Instances Management: manages entities and relations instances (see functional requirement 1c in section 6.1);

Sharing Management: manages instances sharing across different contexts (see functional requirement 2a in section 6.1);

Query and Access: supports the discovery of instances through access patterns and queries (see functional requirement 3 in section 6.1).

The rest of this section presents these five port type by: (i) describing the exposed REST APIs; (ii) identifying the effects a requests could have to the exposed Uniform Resource Identifiers (URIs) of other port type; (iii) providing examples.

6.3.1 Contexts Management

Context Management is responsible for managing hierarchical contexts as defined by the IS Model (see chapter 4.2). It exposes the common functionalities of any collection as recommended by ROA such as:

Listing: allows to enumerate the contexts;

Create: allows to create a new context as a child of another context (if any). The context has a name;

Exists: allows to check if a Context exists;

Read: allows to read a Context;

Update: allows to rename a context or to move a context as a child of another Context;

Operation	HTTP Method	URL
Listing	`GET`	`/contexts`
Create	`PUT`	`/contexts/{CONTEXT_UUID}`
Exists	`HEAD`	`/contexts/{CONTEXT_UUID}`
Read	`GET`	`/contexts/{CONTEXT_UUID}`
Update	`PUT`	`/contexts/{CONTEXT_UUID}`
Delete	`DELETE`	`/contexts/{CONTEXT_UUID}`

Table 6.1: *Context collection RESTful APIs*

Delete: allows to delete a Context.

Table 6.1 shows the exposed APIs for contexts collection.
Any request to this port type has success if the following guarantees are satisfied:

- the hierarchy of contexts is a tree with an arbitrary number of levels;
- two contexts with the same name can only exist if they have different parents;
- any update to a context does not have any side effect on the instances belonging to the context;
- it is not possible to delete a context if it contains instances. It is a responsibility of the clients to remove the instances from the context (or delete them) before trying to delete the context.

Listing 6.1: *Example of context representation having a parent and a child*

```
{
    "name": "SoBigData",
    "uri" : "/contexts/b8a909e8-3a99-4177-83ef-
       a84410025d49",
    "header": {
        "@class": "Header",
        "uuid": "b8a909e8-3a99-4177-83ef-a84410025d49",
        "creator": "luca.frosini",
        "creationTime": "2018-06-19 12:41:22.305 +0200",
        "modifiedBy": "luca.frosini",
        "lastUpdateTime": "2018-06-19 12:41:22.305 +0200"
    },
    "parent": "/contexts/aad7371b-0bd6-4c7f-bd76-
       f1ceca1e573c",
    "fullName" : "/d4science/SoBigData",
    "children": [ "/contexts/c380bda0-71cf-49fb-a3a3
       -771479e9ce59"]
}
```

A Context representation is similar to the one shown in listing 6.1. It contains the name of the Context (i.e., SoBigData), the URI of the Context, the parent Context URI, the full name of the context (i.e., the composition of the names of the contexts starting from the root context, each name is preceded by a '/') and the list of children contexts as list of URIs. During updates, the service considers only 'name' and 'parent' fields (it ignores all the other fields). The listing APIs provides a JavaScript Object Notation (JSON) array containing a list of URIs.

The representation contains information related to the representation of parent and children contexts. Hence, update and delete operations have an impact on the representation of parent and children contexts. The representation of a context has implications on the Transaction Model (i.e., required locks) as described in section 6.5.

6.3.2 Types Management

Types Management is responsible for managing the instantiation of the IS Model by allowing the definition of entities, relations and embedded types and their schema.

Giving the REST principle *Manipulation Of Resources Through Representations* the defined DDL (see functional requirement 1(b)i in section 6.1) is a specification of the representation of a type.

The following attributes describe a type representation (* indicates a mandatory attribute):

Name*: the type name [String];

Description: the description of the type [String, default=null];

Abstract: indicate if the type is an abstract or concrete. It is not possible to instantiate an abstract type [Boolean, default=false];

SuperClasses*: the list of parents types. The only entitled types having this attribute to null are the system type *Embedded*, *Entity* and *Relation* [List<String>];

Properties: any types, except *Resource* and its specialisation, includes a properties array [List]. Any property is described by the following attributes as defined by the model (see section 4.2.5):

- **Name***: the property name [String];
- **Type***: the type of the property (see paragraph 4.2.5);
- **Description**: the description of the property [String, default=null];
- **Mandatory**: indicate if the property is mandatory or not [String, default=false];
- **ReadOnly**: the property cannot change its value [Boolean, default=false];
- **NotNull**: whether the property must assume a value diverse from 'null' or not [Boolean, default=false];
- **Max**: whether the property can be limited to a maximum value [Integer, default=null];
- **Min**: whether the property can be limited to a minimum value [Integer, default=null];
- **Regex**: a regular expression to validate the property value [String, default=null].

Any *Relation* has two additional mandatory attributes:

Source: indicate the required source type to instantiate such relation;

Target: indicate the required target type to instantiate such relation.

Any *Resource* instead of *properties* expose two arrays:

Facets: this array defines which *Facet* types describe the resource and which *ConsistsOf* relation type must be used to connect the facet instance;

Resources: this array defines which other *Resource* types could be related to the defined type and which *IsRelatedTo* relation type could be used to connect the instances of them. The array contains only the outbound relations.

Each element of the 'facets' and 'resources' arrays contained in the *Resource* definition is composed of six attributes (* indicates a mandatory attribute):

Source*: it is always the name of the defined type [String];

Relation*: the relation type name to be used to connect the source type to the target type [String];

Target*: the target type name of the relation [String];

Description: the description of the reason why the source and the target should be related [String, default=null];

Max: the upper bound number of relations between the source and target types (null means unbounded) [Integer, default=null];

Min: the lower bound number of relations between the source and target types (null is the same as zero which means that the relation is optional). Optional relations are specified as to provide suggestion to whom is interested in instantiating the resource type [Integer, default=null].

Listing 6.2: *Entity type representation*

```
{
    "name": "Entity",
    "description": "Entity Base Type",
    "superClasses": null,
    "properties": [
        {
            "name": "header",
            "type": "Header",
            "description": "Automatically generated for
               the sake of identification and provenance
               of the specific information",
            "mandatory": true,
            "readonly": false,
            "notnull": true,
            "max": null,
            "min": null,
            "regexp": null
        }
    ],
    "abstract": true
}
```

Listing 6.2 shows the JSON representation of the base type for any entity. The type is abstract because it cannot be instantiated. It does not have any super-types and contains the header property which is inherited by all sub-types. Appendix A presents the IS Model types representation (internally managed by the resource registry) and few examples of gCube Model types.

This port type exposes the type management functionalities trough a collection as recommended by ROA. The exposed APIs are (i) listing; (ii) create; (iii) exists; (iv) read; and (v) delete.

Table 6.2 shows the exposed RESTful APIs for type operation.

Operation	HTTP Method	URL
Listing	`GET`	`/types`
Create	`PUT`	`/types/{TYPE_NAME}`
Exists	`HEAD`	`/types/{TYPE_NAME}`
Read	`GET`	`/types/{TYPE_NAME}`
Delete	`DELETE`	`/types/{TYPE_NAME}`

Table 6.2: *Types REST APIs*

Types Management does not provide the capability to update the specification of a type. No one could know which is the impact on changing the schema of a type because potentially any client could create them. The IS Model provides by design the support to evolution via inheritance and schema-mixed mode.

The Types Management implements the following policies:

- it ignores all the properties a client tries to define for a resource;
- it deletes a type only if the type has no instances and no sub-types;
- it supports multiple inheritance;
- it enforces inheritance rules (see section 4.2.4). A type can have only one ancestor between *Resource*, *Facet*, *IsRelatedTo*, *ConsistsOf* or *Embedded*. The specialisations of any relation must have source and target entity types hierarchically compatibles with the parent relations.

This section presented the APIs design an the defined DDL to satisfy the functional requirement 1b (see section 6.1). The defined DDL allows to create a new type and it enables the Instances Management port type (see section 6.3.3) to validate the instances against the registered schema (see functional requirement 1(c)iii in section 6.1). The next section presents the Instances Management port type.

6.3.3 Instances Management

The Instances Management port type is responsible for the management of entities and relation instances. It offers the following APIs:

- Create: it allows to create a new entity or relation instance in a certain context;
- Exists: it allows to check if an instance exists in a certain context;
- Read: it allows to get the representation of the requested instance in a certain context;
- Update: it allows to update an instance in a certain context;
- Delete: it allows to delete an instance.

The Instances Management implements the following policies:

- it manages the *Header* automatically;
- it allows to identify an instance via the Universally Unique Identifier (UUID) specified in the *Header*;
- it allows the creation of an instance only if the declared type is already present in the system (previously registered via the Type Management port type);
- it validates the instance against the schema of the defined type;
- it imposes the default values of propagation constraints [2] when the client does not specify their values;
- it guarantees propagation constraints.

Operation	**HTTP Method**	**URL**

Create	`PUT`	`/instances/{RESOURCE_TYPE_NAME}/{UUID}`
		`/instances/{IS_RELATED_TO_TYPE_NAME}/{UUID}`

Exists	`HEAD`	`/instances/{RESOURCE_TYPE_NAME}/{UUID}`
		`/instances/{IS_RELATED_TO_TYPE_NAME}/{UUID}`

Read	`GET`	`/instances/{RESOURCE_TYPE_NAME}/{UUID}`
		`/instances/{IS_RELATED_TO_TYPE_NAME}/{UUID}`

Update	`PUT`	`/instances/{RESOURCE_TYPE_NAME}/{UUID}`
		`/instances/{IS_RELATED_TO_TYPE_NAME}/{UUID}`

Delete	`DELETE`	`/instances/{RESOURCE_TYPE_NAME}/{UUID}`
		`/instances/{IS_RELATED_TO_TYPE_NAME}/{UUID}`

Table 6.3: *Instances REST APIs*

Table 6.3 shows the exposed APIs for instances operations. The table shows only the URI for *Resource* and *IsRelatedTo* instances to improve the table readability. Instead, for each operation the port type, it exposes also the following URIs:

```
/instances/{RESOURCE_TYPE_NAME}/{UUID}/
  {CONSISTS_OF_TYPE_NAME}/{UUID}

/instances/{RESOURCE_TYPE_NAME}/{UUID}/
  {CONSISTS_OF_TYPE_NAME}/{UUID}/{FACET_TYPE_NAME}/{UUID}
```

[2]The IS Model defines the following default values: *IsRelatedTo* (remove=keep, add=unpropagate); *ConsistsOf* (remove=cascadeWhenOrphan, add=propagate).

The resource instances are available to URI constructed using the resource type name and the UUID as path parameters i.e., `/instances/{RESOURCE_TYPE_NAME}/{UUID}`. The resource instance is a subordinate of the resource type collection. Creating a resource type via Types Management (see section 6.3.2) results in the creation of the resource type collection.

ConsistsOf relations are de-facto subordinates of the resource instance. Hence, they are exposed consistently `/instances/{RESOURCE_TYPE_NAME}/{UUID}/{CONSISTS_OF_TYPE_NAME}/{UUID}`. Any created *ConsistsOf* relation type is exposed as a collection of any resource instance compatible with the specified source type of the relation type.

The same rationale is used to define the URI of the facet instances `/instances/{RESOURCE_TYPE_NAME}/{UUID}/{CONSISTS_OF_TYPE_NAME}/{UUID}/{FACET_TYPE_NAME}/{UUID}`.

IsRelatedTo relations are exposed at the same level of resources `/instances/{IS_RELATED_TO_TYPE_NAME}/{UUID}`

ConsistsOf relations and *Facet* instances are implementation of a design pattern which provides the logical separation of the resource concept from the different aspects of the resource description. This separation enables:

1. differential updates of resources to enhance the scalability of the service (see Berkley Database Information Index (BDII) scalability issues in section 3.2);
2. different representation of the resource aspect(s) in different contexts.

The instances are created and modified through representation.

Listing 6.3: SoftwareFacet *instance representation*

```
{
  "@class": "SoftwareFacet",
  "header": {
    "uuid":"69f0b376-38d2-4a85-bc63-37f9fa323f82",
    "creator":"luca.frosini",
    "lastUpdater":"luca.frosini",
    "creationTime":"2018-09-29 11:16:24",
    "lastUpdateTime":"2018-09-29 11:16:24
  },
  "name": "resource-registry",
  "group": "InformationSystem",
  "version": "2.0.0",
  "description": null,
  "qualifier": null,
  "optional": false
}
```

Listing 6.3 shows an example of JSON representation of a *SoftwareFacet* instance. The instances are encoded using JSON. The *@class* attribute declares the instance type.

Analysing the REST APIs, we notice that the context is not explicit in the URI. The context is provided via an HyperText Transfer Protocol (HTTP) header. This allows to expose the instances to the same URI in every context they are shared. This URI approach enables concurrency in REST transactions (see section 6.5).

The gCube framework uses an authorisation token in the HTTP header to identify the user and the context of each request. The authorisation framework equips the container running the web services. It intercepts any requests, resolves the token and forwards the request to the service if authorised, along with the user and the operating context. The Resource Registry uses the context to identify the belonging instances and the user to manage the *Header* properly; (i) at creation time to initialise *creator* and *modifiedBy* properties; (ii) at updated time to updated *modifiedBy* property.

The presented APIs satisfies the instances management functional requirement (see functional requirement 1c in section 6.1). The instances representation provides the required DML (see functional requirement 1(c)i). The exposed URIs allow to satisfy the univocal identification of any entities and relations (see functional requirement 1(c)ii).

The implementation of the exposed APIs must (i) satisfy the validation requirement (see functional requirement 1(c)iii in section 6.1); (ii) provide the required referential integrity guarantees (see functional requirement 1d in section 6.1); (iii) guarantee the observation of propagation constraints (see functional requirement 1e in section 6.1); (iv) guarantee consistent views across contexts (see functional requirement 2b in section 6.1).

The Sharing Management presented hereafter shows how the service supports the sharing of the instances created with Types Management port type across contexts.

6.3.4 Sharing Management

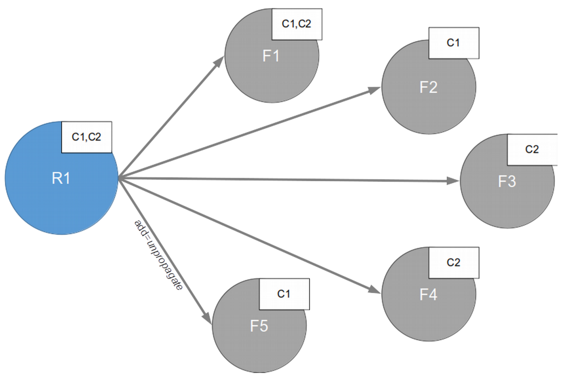
The Sharing Management port type allows a client to:

- add an instance to a context;
- remove an instance from a context (the Resource Registry deletes the instance when it is available only in the removing context).

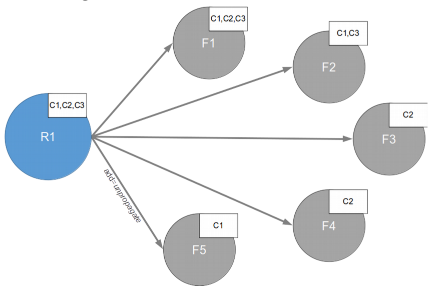
When the resource registry receives a request to add a resource instance to a context (or remove from a context), it applies the propagation constraints defined in the outcoming relations. This behaviour can have a "cascade impact" because also the target resource will be added (removed) and so forth.

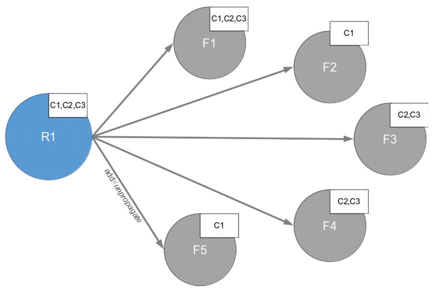
Any request to add an instance to context has a source context which means that the resource registry analysed only the propagation constraint of the relations belonging to the source contexts. Hence, sharing an instance from a context C1 to a context C3 can have a different result than sharing the same instance from the context C2 to the context C3.

Fig. 6.2a shows a resource R1 described by the facets F1, F2, F3, F4, F5 (across all the contexts it belongs to). In the context C1 the resource R1 is described by the facets F1, F2 and F5. In the context C2, instead, the resource R1 is described by the facets F1, F3, F4. The *ConsistsOf* relations between the resource R1 and the facet F5 has the *PropagationConstraint* field with add=*unpropagate* and remove=*cascade*. Instead, the others *ConsistsOf* relations has the default values for *PropagationConstraint* (i.e., add=*propagate*, remove=*cascade*).



(a) *Initial State of resource R1 which belongs to C1 and C2 contexts as well as F1. F2 and F5 belongs to C1. F3 and F4 belongs to C2 contexts.*





(b) *Result of adding R1 to target context C3 from source context C1*

(c) *Result of adding R1 to target context C3 from source context C2*

Figure 6.2: *Main concepts of the IS Model*

A request to add R1 to the context C3 starting from C1 will result in the resource R1 described by the facets F1 and F2 (not F5 because the *add* property is set to *unpropagate*) (see fig. 6.2b). Instead, the request starting by the context C2 will result in the resource R1 described by the facets F1, F3 and F4 (see fig. 6.2c).

Resource Registry validate the resources against the schema when an actions involves *ConsistsOf* relation and *Facet* describing the resource. Resource Registry does no add *ConsistsOf* relations (and correspondent facets) when the add propagation constraint is *unpropagate*. Hence, the resource could have an invalid profile in the target context.

Looking at the example presented in fig. 6.2c, trying to remove the facets F1 from C3 context, the operation fails if the schema of R1 defines such facet as mandatory.

Operation	HTTP Method	URL
Add To Context	`POST`	`/sharing/contexts/{CONTEXT_UUID}/{TYPE_NAME}/{INSTANCE_UUID}`
Remove From Context	`DELETE`	`/sharing/contexts/{CONTEXT_UUID}/{TYPE_NAME}/{INSTANCE_UUID}`

Table 6.4: *Sharing Management REST APIs*

Table 6.4 shows the exposed APIs for sharing operations. Both operations contain the target context UUID in the path. Moreover, the URL contains type name of the instance as well as the instance UUID.

The source context is not explicit in the REST APIs. Instead, it depends on the authorisation token in the HTTP header of the request as described in the previous section.

This port type allows to satisfy the requirement of supporting instances sharing across contexts (see functional requirement 2a in section 6.1).

The implementation of the exposed APIs must (i) satisfy the validation requirement (see functional requirements 1(c)iii); (ii) satisfy the consistent views across contexts (see functional requirement 2b); (iii) observe propagation constraint (see functional requirement 2d)

The next section presents the Query and Access port type.

6.3.5 Query and Access

Query and Access port type allows performing queries on instances in a specific context. It exposes some of the safe methods already available through dedicated port type in addition to query APIs.

Group	Operation	HTTP Method	URL
Contexts	Listing	`GET`	`/access/contexts`
	Existence	`HEAD`	`/access/contexts/{CONTEXT_UUID}`
	Read	`GET`	`/access/contexts/{CONTEXT_UUID}`
Types	Listing	`GET`	`/access/types`
	Existence	`HEAD`	`/access/types/{TYPE_NAME}`
	Read	GET	`/access/types/{TYPE_NAME}` `[?polymorphic=false]`
Instances	Existence	`HEAD`	`/access/instances/{TYPE_NAME}/{UUID}`
	Read	`GET`	`/access/instances/{TYPE_NAME}/{UUID}`
Query	Query all instances of a type	`GET`	`/access/query/{TYPE_NAME}[?polymorphic=true]`
	Get filtered entities	GET	`/access/query/{ENTITY_TYPE_NAME}` `/{RELATION_TYPE_NAME}/{REFERENCE_ENTITY_TYPE_NAME}` `[?polymorphic=true&direction=(in|out|both)` `[&reference={REFERENCE_ENTITY_UUID}` `&name1=value1&name2=value2&...]`
Raw Query	Gremlin Query to Graph	`GET`	`/query?q={query}`

Table 6.5: *Query and Access REST APIs*

Table 6.5 shows the exposed APIs grouped by base URL. The APIs to get contexts information allow to: (i) lists all existent contexts; (ii) checks if a context with a certain UUID exists; (iii) read the representation of the context identified by the UUID.

The APIs to retrieve types information allow to: (i) list all the registered types; (ii) check if a certain type exists; (iii) read the schema for the specified `TYPE_NAME`. Using the parameter `polymorphic=true`, apart the schema for the specified `TYPE_NAME`, it returns also the schema of all the sub-types.

The APIs to get instances information allow to: (i) check if a certain instance exists; (ii) read the representation of a certain instance. Conversely to the instances port type all the instances are exposed at the first level.

Finally, this port type exposes three query APIs. The first returns the list of all instances of a certain `TYPE_NAME` provided as a path parameter. This API allows a client to indicate if is interested all the instances of the specified type (all the instances of sub-types) or it requires only the instances of the indicated type `polymorphic=false` (default `true`).

The second API returns the list of instances of a specific entity type (indicated by the first path parameter i.e., `ENTITY_TYPE_NAME`) and filter them according to the following criteria:

- The second path parameter indicates which relation type (i.e., `RELATION_TYPE_NAME`) must be related to the obtained instances;
- The direction of the relation can be specified by using `direction` query variable which by default is `both`. The reference of the direction is the `ENTITY_TYPE_NAME`. Allowed values for `direction` are `(in|out|both)`;
- The third path parameter (i.e., `REFERENCE_ENTITY_TYPE_NAME`) defines the types of the entity in the opposite side of the relation;
- The `reference` query parameter enforces a specific instance of referenced type by specifying the UUID (i.e., `REFERENCE_ENTITY_UUID`);
- When the client does not indicate `reference` query parameter, it is possible filtering between the reference entities by specifying an arbitrary number of name-value couples. The API evaluates in AND the set of couple specified as query parameters (i.e., `&name1=value1&name2=value2`);
- The `polymorphic` query parameter (`default=true`) indicates if the client is requesting only instances of the indicated type (*polymorphic=false*) or also the instances of any extension of the indicated type (*polymorphic=true*).

The service will return `400BadRequest` in case the indicated types are not compliant with the model (e.g., specifying an IsRelatedTo relation between a facet type and a resource type).

This API is very versatile. Examples of use are:

- `/access/query/EService/IsIdentifiedBy/SoftwareFacet?polymorphic=false&direction=out` : this invocation allows to retrieve all the *EService* instances having a *SoftwareFacet* related with *IsIdentifiedBy*. The only allowed direction is `out` because *IsIdentifiedBy* is a specialisation of ConsistsOf which by definition 'exists' `(out)` from a resource and 'enters' `(in)` into a facet;
- `/access/query/EService/IsIdentifiedBy/SoftwareFacet?polymorphic=true&direction=out&reference=7bc997c3-d005-40ff-b9ed-c4b6a35851f1` : this invocation allows to retrieve the *EService* instance identified by a *SoftwareFacet* with UUID `7bc997c3-d005-40ff-b9ed-c4b6a35851f1`. The URL has `polymorphic=true` query parameter (it could be omitted because it is the default) this meas that the result could be for example a *RunningPlugin* (which is a specialisation of *EService*) identified by the *SoftwareFacet* with such UUID. The `direction=out` query parameter is the only valid value for the request to avoid to get a `400BadRequest` response;

- `/access/query/Resource/IsIdentifiedBy/ContactFacet?polymorphic=true&direction=out` : this invocation allows to retrieve all the *Resource* instances (any type of *Resource* instance giving the query parameter `polymorphic=true`) which is identified by a *ContactFacet*;
- `/access/query/Resource/ConsistsOf/ContactFacet?polymorphic=true&direction=out` : this invocation allows to retrieve all the *Resource* instances (any type of *Resource* instance giving the query parameter `polymorphic=true`) having a *ContactFacet* related by any type of *ConsistsOf* relation (giving the query parameter `polymorphic=true`);
- `/access/query/Service/Hosts/Site?polymorphic=true&direction=in` : this invocation allows to retrieve all the *Service* instances having an incoming *Hosts* relation (a specialisation of *IsRelatedTo*) from *Site* resource instances;
- `/access/query/Service/Hosts/Site?polymorphic=true&direction=in&reference=16032d09-3823-444e-a1ff-a67de4f350a8` : this invocation allows to retrieve all the *Service* instances hosted by (having an incoming *Hosts* relation from) the *Site* with UUID `16032d09-3823-444e-a1ff-a67de4f350a`;
- `/access/query/EService/ConsistsOf/SoftwareFacet?polymorphic=true&direction=out&group=accounting&name=accounting-service` : this invocation allows to retrieve the *EService* instances identified by a *SoftwareFacet* with `group=accounting` and `name=accounting-service` (i.e., the running Accounting Service instances).

The third API allows performing a query on the underline graph. This API is accessible only to infrastructure managers and administrators. The base path of this API differs from the others to facilitate the definition of networking and authorisation policies to restrict the access.

The next section describes the database families analysed to select the persistence of the data required by the Resource Registry.

6.4 Information System Persistence

The selection of persistence for the IS started by evaluating the most popular database families[3].

The analysed database families are:

Relational database: A relational database uses the relational model proposed by Edgar Frank Codd [38]. The Structured Query Language (SQL) is used to manage and query the database.

[3] Analysed database families are the families present in the top 100 list of database technologies defined by `db-engine.com` [47] with the exclusion of "Search Engine" family

NoSQL: Corbellini et al. evidenced [41] that “the term was coined by Carlo Strozzi in 1998 to refer to the open-source database called NoSQL not having an SQL interface”. Nowadays, the term NoSQL is used for “database providing storage and retrieval not based on relational model [41]”:

Key-value Store: A Key-value (KV) store uses the associative array (also known as hash-table, map or dictionary) as data model. The data is represented as a schema-less collection of key-value pairs allowing to store arbitrary data under a key [17,41].

Column Store: A column store saves data in tuples consisting of three elements: a key, an arbitrary value and a timestamp. Each tuple represents a column, and the attribute of the values can be considered the rows. Not all columns have the same rows [41].

Document Store: The key concept of a document store is the notion of a "document". Each document is composed of a series of fields. They normally use a standard format to represent the document such as eXtensible Markup Language (XML), JSON, Binary JSON (BSON) [17,41].

RDF Store: The Resource Description Framework (RDF) model represents information as triples in the form of subject-predicate-object. Clients use semantic queries to retrieve the triples.

Graph database: This database family is designed for data whose relations are represented as a graph consisting of elements interconnected with a finite number of relations between them.

	Resource Model	**Dynamic Query**
Relational Database	Yes[4]	Yes (Std)
Key-Value Stores	No	No
Column Stores	No	No
Document Stores	No	Yes[5]
RDF Stores	Yes	Yes (Std)
Graph Database	Yes	Yes (Std[6]

Table 6.6: *DBMS Families Comparison*

Table 6.6 summarises two aspects of the analysis: the capability of persisting the resource model, and the support of a mechanism to dynamically define a query. The table also evidences the availability of standard query languages.

Key-value stores, columns stores and document stores do not comply with the resource model mainly due to missing relations support. In some cases, it is possible to simulate the relations by inserting the key/id of the reference inside the value (e.g., in document store, the document contains the id of another document as additional property). The creation of simulated relation is a well-known practice. MongoDB (a document store technology) documentation encourages to the usage of simulated reference

[4]The schema mixed mode could be simulated with a lot of inefficiency

[5]Document Stores support dynamic query in certain implementations only (e.g., MongoDB, CouchBase)

[6]Apache TinkerPop^{TM} framework is not a standard but it provides widely implemented common APIs to query the underline graph

and divides them into two groups "manual references" and DBRefs (manual references defined according to a convention) which enables some support by the engine [7]. Unfortunately, it is not provided referential integrity for the simulated references. Key-values stores and columns stores do not provide a way to build the dynamic query.

Document stores provide map-reduce query pattern typically. Some implementations provide a sort of SQL-like query language (e.g., N1QL in CouchBase[8]) but the performances and the capabilities are not comparable with the ones obtained from a relational database.

Key-value stores, columns stores and document stores are not suitable families giving that they do not provide dynamic queries and relations support.

Relational databases provide the support to perform a dynamic query in a standard way. The relational database does not have native support for the polymorphism which should be simulated. The built-in multi-tenant support provided by the available technologies (multi-database) does not provide the required consistency across different contexts. To provide the required multi-tenant approach could be necessary using a tagging mechanism similar to the one proposed by Grid Operations Centre Database (GOCDB) (see section 3.4).

Graph databases have a de-facto standard framework called Apache TinkerPop^{TM}. Apache TinkerPop^{TM} is a graph-computing framework for both graph databases On-Line Transaction Processing (OLTP) and graph analytic systems On-Line Analytical Processing (OLAP). When a data system is TinkerPop-enabled, its users can model their domain as a graph and analyse the graph using the Gremlin graph traversal language. A graph is composed of vertexes and edges. Graph databases support properties both on vertexes an edges. This model allows persisting the entities as vertexes and the relations as edges. Thanks to TinkerPop^{TM} the graph database implementation can be changed between the compliant technologies with no cost, or very little, for the service.

RDF has a complete set of inference rules which fit the IS Model. SPARQL is a standard query language to perform semantic queries on RDF stores. A graph database has a more general structure than a RDF store, and the IS Model is de-facto a graph model.

The excellent fit for the model and the availability of standards query mechanism of RDF stores and graph databases suggested investigating the available implementation to find the best candidate.

It is outside of the scope of this dissertation present all the analysed technologies. Angles and Gutierrez [8, 9] surveyed Graph Databases while Kaoudi and Manolescu [88] surveyed RDF Stores.

Between the analysed technologies, we selected OrientDB because it provides excellent support for IS requirements. Among other things, OrientDB is released as open source software under the Apache 2 License. Apache 2 licence is compatible with the European Union Public Licence (EUPL) licence used by the gCube Framework. Next paragraph describes how the Resource Registry uses the capabilities provided by OrientDB.

[7]`https://docs.mongodb.com/manual/reference/database-references/`

[8]`https://developer.couchbase.com/documentation/server/5.1/getting-started/try-a-query.html`

OrientDB

OrientDB is a graph database [9] supporting Apache TinkerPop™.

The Resource registry persists resource and facet as vertexes and ConsistsOf and IsRelatedTo relations as edges. OrientDB provides a partitioned graphs capability which provides partitioned views across the whole graph based on Role-Based Access Control (RBAC) [58]. This feature uses authorisation to grant or deny the access to vertex and edge instances. The authenticated user navigate only the part of the graph is authorised. Any TinkerPop™ client has the feeling of interacting with the whole graph, which instead is filtered by authorisation. Partitioned graph fit perfectly with the required context views.

Resource Registry creates a graph partition for each context. It also creates some partitions to store management information (the instances of these partitions are separated from any other instances of other partitions). Resource Registry creates for each graph partition a writer role and a reader role and the associated users. Resource registry uses the writer user to modify the content of the graph partition (Create, Update, Delete) and the reader for safe operations (Read).

The Resource Registry perform sharing actions using an administrative user of the database which grant or deny the reader and writer role correspondent to the target context of the request.

OrientDB supports vertexes, edges, and embedded type definition with inheritance support, but it does not support relations validations and resource profile definitions and validation. A graph partition separated from the others is used to store the type definitions used for validation purposes. Resource Registry creates the IS Model types automatically at database creation time.

Resource Registry uses a dedicated graph partition to persist context hierarchy and context information (e.g., the name).

OrientDB supports different deployment scenarios (e.g., single instance, multi-master, master-slave) which allows deciding which two of the CAP guarantees have to be satisfied. OrientDB supports ACID transactions. The Resource Registry performs an ACID transaction for every single REST request.

The next section presents an extension of the RESTful Transaction Model presented in chapter 5.

6.5 Extended RESTful Transaction Model

The RESTful Transaction Model proposed in section 5.3 (henceforth general transaction model) does not comply with RESTful services exposing subordinated resources. A RESTful service exposing multiple collections can be considered as two services if the manipulation of a resource of a collection does not impact any other resource of another collection. In this case, the general transaction model can support services exposing multiple collections. The Resource Registry exposes subordinated resources and the different port types which have a reciprocal impact. For example, when a client creates a new resource type, a new collection became available in instances port type. When a client creates an instance, such instance becomes available in sharing man-

[9]OrientDB can be also used as Document Store or as Key-Value Store

agement port type. Hence, the general transaction model is not compatible with the presented Resource Registry.

In this section is described a REST transaction model designed for the Resource Registry purposes (henceforth extended transaction model) which modify the behaviour of Transaction Proxy and Lock Service and extends the capability of the Transaction Service.

The extended transaction model has been designed to co-exist with the general transaction model and to maintain the distributed transaction support. This design enables any client to perform a transaction involving operations on the resource registry as well as operations on any other RESTful service of the infrastructure. To address this, any client must perceive all the Transaction Proxy in the same way; Lock Service has to comply with the distributed algorithm for deadlock detection; Transaction Service must expose the same APIs and support the same capabilities.

The transaction model architecture allows deploying a Transaction Proxy instance for each Effective Service. Hence, it is possible to provide a specific version of Transaction proxy capable of supporting the specific requirements of the Effective Service. Transaction Proxy is responsible for selecting the Lock Service. Hence, it is possible providing a dedicated instance of Lock Service.

This section presents first an overview of the locks required for each REST API exposed by the Resource Registry (see paragraph 6.5.1). Paragraph 6.5.2 presents the lock algorithm used by the Lock Service of the extended model (henceforth extended Lock Service or Resource Registry Lock Service). Paragraph 6.5.4 presents the modified version of Transaction Proxy (henceforth extended Transaction Proxy or Resource Registry Transaction Proxy). Finally, paragraph 6.5.3 presents the extend Transaction Service.

6.5.1 Resource Registry APIs Analysis

Operation	HTTP Method	URL	Lock Type	Lock URI
Listing	`GET`	`/contexts`	S	`/contexts`
Create	`PUT`	`/contexts/{CONTEXT_UUID}`	X	`/`
Exists	`HEAD`	`/contexts/{CONTEXT_UUID}`	S	`/contexts/{CONTEXT_UUID}`
Read	`GET`	`/contexts/{CONTEXT_UUID}`	S	`/contexts`
Update	`PUT`	`/contexts/{CONTEXT_UUID}`	X	`/contexts`
Delete	`DELETE`	`/contexts/{CONTEXT_UUID}`	X	`/`

Table 6.7: *Context Management Port Type REST APIs with the required lock type and URI*

Table 6.7 shows the lock type and URI to interact with Context Management port type. Creating or deleting a context impacts on instances, sharing and query port type. If a client deletes a context, no one should create or share an instance in such a context [10]. Furthermore, no client should query the deleted context. The creation of a context has similar considerations because to achieve isolation the context is not available to other clients until the creator commit the transaction. For this reason, the Transaction Proxy must request to lock the whole service base path (e.g., `/`) to properly isolate delete and create operations [11].

[10]the resource registry accepts to delete a context only and only if it has no instances, so the other operations are not possible on instances port type

[11]Any client could perform parallel operations in types management. This optimisation is not presented in this dissertation.

Operation	HTTP Method	URI	Lock Type	Lock URI
Listing	`GET`	`/types`	S	`/types`
Create	`PUT`	`/types/{TYPE_NAME}`	X	`/`
Exists	`HEAD`	`/types/{TYPE_NAME}`	S	`/types/{TYPE_NAME}`
Read	`GET`	`/types/{TYPE_NAME}`	S	`/types/{TYPE_NAME}`
Delete	`DELETE`	`/types/{TYPE_NAME}`	X	`/`

Table 6.8: *Types Management Port Type REST APIs with the required lock type and URI*

Table 6.8 shows the lock type and the URI to lock to interact with Type Management port type. Similarly to context management port type, creating or deleting a type impacts on instances, sharing and query port type. When a client deletes a type, no one should create or share across contexts instances for such type [12]. Furthermore, no client should query the instances of the deleted type. The query polymorphism support enhances the possibility of including the deleted type instances because the Transaction Proxy cannot predict the involved types. For this reason, the Transaction Proxy must request to lock the whole service base path (e.g., `/`) to isolate delete and create operations properly [13].

Operation	HTTP Method	URI = Lock URI	Lock Type
Create	`PUT`	`/instances/resources/{RESOURCE_TYPE_NAME}/{UUID}`	X
Exists	`HEAD`	`/instances/resources/{RESOURCE_TYPE_NAME}/{UUID}`	S
Read	`GET`	`/instances/resources/{RESOURCE_TYPE_NAME}/{UUID}`	S
Update	`PUT`	`/instances/resources/{RESOURCE_TYPE_NAME}/{UUID}`	X
Delete	`DELETE`	`/instances/resources/{RESOURCE_TYPE_NAME}/{UUID}`	X

Table 6.9: *Instances Management Port Type REST APIs for Resource and subordinated with the required lock type and URI*

Operation	HTTP Method	URI	Lock Type	Lock URI
Create	`PUT`	`/instances/relations/{IS_RELATED_TO_TYPE_NAME}/{UUID}`	X	`/instances`
Exists	`HEAD`	`/instances/relations/{IS_RELATED_TO_TYPE_NAME}/{UUID}`	S	`/instances`
Read	`GET`	`/instances/relations/{IS_RELATED_TO_TYPE_NAME}/{UUID}`	S	`/instances`
Update	`PUT`	`/instances/relations/{IS_RELATED_TO_TYPE_NAME}/{UUID}`	X	`/instances`
Delete	`DELETE`	`/instances/relations/{IS_RELATED_TO_TYPE_NAME}/{UUID}`	X	`/instances`

Table 6.10: *Instances Management Port Type REST APIs for IsRelatedTo instances with the required lock type and URI*

Tables 6.9 and 6.10 show the required lock type and the URI to interact with Instances Management port type.

Any operation of a resource or the resource subordinated (i.e., *ConsistsOf* relations and *Facet*) requires a lock only for the requested URI.

Table 6.9 shows only the URI for *Resource* instances to improve the table readability. Instead, for each operation the port type, it exposes also the following URIs:

```
/instances/{RESOURCE_TYPE_NAME}/{UUID}/
  {CONSISTS_OF_TYPE_NAME}/{UUID}
```

[12]The resource registry accepts to delete a type only and only if it has no instances and sub-types, so the others operations are not possible on instances port type.

[13]Any client could perform parallel operations in context management. This optimisation is not presented in this dissertation.

```
/instances/{RESOURCE_TYPE_NAME}/{UUID}/
  {CONSISTS_OF_TYPE_NAME}/{UUID}/{FACET_TYPE_NAME}/{UUID}
```

Instead, operation on *IsRelatedTo* relations has a potential impact on the two resources the relation is linking. For such a reason, any operations on *IsRelatedTo* relations requires a lock on the whole instances management port type.

Operation	HTTP Method	URI	Lock Type	Lock URI
Add to Context	`POST`	`/sharing/contexts/{CONTEXT_UUID}/{TYPE_NAME}/{INSTANCE_UUID}`	X	/
Remove from Context	`DELETE`	`/sharing/contexts/{CONTEXT_UUID}/{TYPE_NAME}/{INSTANCE_UUID}`	X	/
APIs exposed only to Transaction Manager				
"Check" Add To Context	`GET`	`/sharing/contexts/{CONTEXT_UUID}/{TYPE_NAME}/{INSTANCE_UUID}?operation=POST`	S	/
"Check" Remove From Context	`GET`	`/sharing/contexts/{CONTEXT_UUID}/{TYPE_NAME}/{INSTANCE_UUID}?operation=DELETE`	S	/
Add to Context "No Follow"	PUT (with content)	`/sharing/contexts/{CONTEXT_UUID}/{TYPE_NAME}/{INSTANCE_UUID}`	X	/
Remove from Context "No Follow"	`POST` (with content)	`/sharing/contexts/{CONTEXT_UUID}/{TYPE_NAME}/{INSTANCE_UUID}`	X	/

Table 6.11: *Sharing Management Port Type REST APIs with the required lock type and URI*

Table 6.11 shows the lock type and the URI to lock to interact with Sharing Management port type. Sharing port type exposes to any clients the operations add to context and remove from context. Due to propagation constraints, this APIs can impact on a non-predictable number of instances which requires to lock the whole service. Add to Context is not the Reverse operation of Remove From Context and Vice-versa[14].

For this reason the Resource Registry exposes four additional APIs as indicated in table 6.11. Check operations return the list of the affected entities and relations for the operation indicated as query parameter with no side effects. "No Follow" operations apply the action to the target resource and all the instances indicated in the content body of the request (in this case the Resource Registry does not consider the propagation constraints).

Operation	HTTP Method	URL	Lock Type	Lock URI
Access APIs	`GET`	`access/*`	S	/
Query	`GET`	`/query`	S	/

Table 6.12: *Query and Access Port Type REST APIs with the required lock type and URI.*

To interact with query and access port type is required a Shared lock to / URI (see table Table 6.12). Any query requires a shared lock on the whole service because any

[14]Adding a resource A to a context C1 can affect the resource B if the IsRelatedTo relation has 'propagate' as propagation constraint of add action (it also affects IsRelatedTo relation). The same relation can have the 'keep' as propagation constraint value of the remove action. Hence, removing A from the context C1 does not affect the resource B and the IsRelatedTo relation connecting them

non-safe operations have a potential impact on the result.

6.5.2 Lock Service

The architecture of the Transaction Model presented in chapter 5.3 defines that is in charge of the Transaction Proxy to interact with the Lock Service. Hence, the architecture allows the definition of a dedicated Lock Service. The extended Lock Service grants locks according to the following rules:

- shared locks are compatible with another shared lock of the same URI but also with shared locks of parents and children URIs;
- exclusive locks are not compatibles with any existent locks (shared or exclusive) of parents and children URIs.

Lock Service allows Transaction Proxy to create multiple locks for the same URI but only if they are related with the same transaction.

6.5.3 Transaction Service

An extended version of the Transaction Service has been extended to support the Resource Registry REST APIs. The design of this extension is compatible with the model presented in chapter 5.3. The extended Transaction Service supports a broader number of services (non only the Resource Registry).

This model extends the representation of log resource with information describing how to compensate an action which cannot be compensated using general considerations. The extended log resource contains a field indicating the HTTP method, HTTP headers, URI, and content body of the compensation request. Transaction Service is capable of compensating the non-predictable operation of the sharing port type using this extension (see listing 6.4). It is in charge of the Transaction Proxy providing the required information when it creates the log resource.

Listing 6.4: *Example of log resource having compensation field*

```
{
  "method" : "DELETE",
  "headers" : {
    "Accept" : "application/json",
    "Content-Type" : "application/json"
  },
  "content-body" : null,
  "compensation" : {
    "method" : "PUT",
    "uri" : "/sharing/contexts/30b0e0b9-595d-47f8-88e7-3
       a7dc2ecb902/EService/d8c57e0f-a796-4e23-a83a-
       e4d298e32f4d",
    "content-body" : {
      "affected-urls":[
        "/sharing/contexts/30b0e0b9-595d-47f8-88e7-3
           a7dc2ecb902/EService/d8c57e0f-a796-4e23-a83a-
           e4d298e32f4d",
        "/sharing/contexts/30b0e0b9-595d-47f8-88e7-3
           a7dc2ecb902/EService/aede479f-726d-4bc7-842b
           -37cf88992d69",
        "..."
      ]
    },
    "headers" : {
      "Accept" : "application/json",
      "Content-Type" : "application/json"
    }
  }
}
```

Listing 6.4 shows an example of a log resource using the compensation field for remove from context operation. The extended Transaction Service uses the information contained in the the compensation field to perform the compensation instead of applying the standard compensation mechanism.

6.5.4 Transaction Proxy

The extended Transaction Proxy uses slightly different algorithm respect to the algorithm used by the general Transaction Proxy.

Listing 6.5 shows the extended version of *manageTransactionAction()*.

The code differs from the version of the general Transaction Proxy (see listing 5.8) because :

- it does not use `X-Parent-Lock-URI` HTTP header;
- it uses a different algorithm to calculate the required locks;
- it adds the *compensation* field in log resources.

The required locks are calculated in the functions *createLock()* (row 6) and *verifyAndUpgradeLock()* (row 11). The algorithm contained in such functions associate the requested URI to the required lock URI using the mapping presented in paragraph 6.5.1 (see tables 6.7, 6.8, 6.9, 6.10, 6.11, 6.12).

Listing 6.5: *Pseudo Code for Resource Registry Proxy function which handles any transaction action*

```
1  function manageTransactionAction(httpRequest, transactionURI):
2
3    var lockURI = httpRequest.getHeader("X-Lock-URI")
4
5    if lockURI == null then
6      lockURI = createLock(httpRequest, transactionURI)
7      var response = getResource(httpRequest.requestURI)
8      var content = response.content
9      var initialResourceURI = createInitialResource(transactionURI,
           httpRequest.requestURI, lockURI, content)
10   else
11     verifyAndUpgradeLock(httpRequest, lockURI, transactionURI)
12     var initialResourceURI = getInitialResourceURI(transactionURI,
           resourceURI)
13   endif
```

```

  if httpMethod == PUT AND content == null then
      var create = true;

  if httpMethod in [ HEAD, GET ] then
    // No need to log a safe operation which has not to be compensated
  else
    var compensation = getCompensation(httpRequest , create , content)
    logRequest(initialResourceURI , httpMethod , resourceURI , compensation)
  endif

  HttpResponse actionResponse = forwardAction(httpRequest)

  actionResponse.setHeader("X-Lock-URI", lockURI)

  return actionResponse
```

Listing 6.6 shows how the extended Transaction Proxy constructs the *compensation* field to insert in the log resource.

Listing 6.6: *Pseudo Code for Resource Registry Proxy function which calculate compensation field*

```
1  function getCompensation(httpRequest , create):
2
3    var compensation = new Object()
4    compensation.headers = httpRequest.headers
5
6    compensation.uri = httpRequest.requestURI
7
8    if httpRequest.requestURI.startWith("/sharing") then
9
10     compensation.content-body = getResource(httpRequest.requestURI +
          "?operation=" + httpRequest.httpMethod);
11
12     if httpRequest.httpMethod = POST then
13       compensation.method = POST
14     else if httpRequest.httpMethod = DELETE then
15       compensation.method = PUT
16     endif
17
18   else
19
20     if create then
21       compensation.method = DELETE
22     else
23       // DELETE PUT and PATCH have as compensation method PUT
24       compensation.method = PUT
25     endif
26
27     if content-body==null then
28       compensation.content-body = getResource(httpRequest.requestURI)
29     else
30       compensation.content-body = httpRequest.content
31     endif
32
33   endif
34
35   return compensation
```

In any case the compensation URI is the same of the request URI (see row 6).

If the extended Transaction Proxy receive a request for sharing management port type (`/sharing` see rows 8), then it:

- retrieves the list of affected entities and relations using the "check" operation and it sets such list as *content-body* of the compensation field (see row 10) ;
- associates the HTTP method for the compensation action depending on the HTTP method of the request (see rows 12-16). To compensate an "Add To Context" (requested by client with `POST`) it is required a `POST` (see row 12). To compensate a "Remove From Context" (requested by client with `PUT`) it is required a `DELETE` (see row 16).

For any other port type, the compensation field is calculated using the the mapping presented in table 5.6 (rows 20-25).

6.6 Subscription Notification System

The Subscription Notification System implements the publish-subscribe mechanism. The Subscription Notification System allows any services to advertise changes and any client to subscribe to the occurred changes. This mechanism avoids a continuous client polling to services which contribute to reducing their workload. Hence, it enhances the scalability of the service and permits a decoupling from the service implementation [60].

The Subscription Notification System is compliant with the standard Java Messaging System (JMS) topic. In JMS a topic provides the following guarantees:

- any subscribed clients receive a copy of the message in the topic;
- the clients receive the messages only during the subscription periods;
- the clients can filter the messages using SQL-like query.

The extended Transaction Service notifies the Subscription Notification System about the changes occurred in a transaction when the transaction is committed (after releasing the locks) to comply with the isolation property.

The extended Transaction Service uses four different topics:

- Context topic for all the changes occurred in the context port type: Create, Rename, Move, and Delete;
- Schema Topic for all the changes occurred on schema port type: Create, Update, and Delete;
- Instance Topic for all the changes occurred in instances port type: Create, Update, and Delete;
- Sharing Topic for all the sharing requests. It notifies about the affected entities and relations using the information contained in the *content-body* of the *compensation* field of the log resource.

The publish-subscribe mechanism and the design of the model, which enables differential updates, improve the scalability of the Resource Registry.

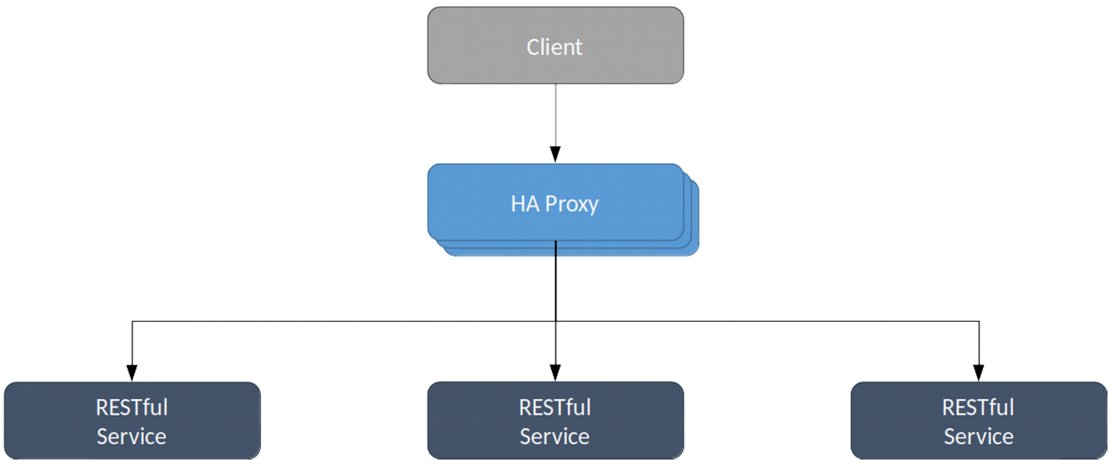


Figure 6.3: *REST Service Deployment Architecture*

6.7 Deployment Architecture

To satisfy the non-functional requirements, a proper software design and an appropriate deployment architecture are required. The design of the IS uses the REST architectural style.

Fig. 6.3 shows the deployment architecture of each REST service (i.e., Resource Registry, Transaction Proxy, Transaction Service and Lock Service). The service instances are load balanced and monitored via multiple instances of HA Proxy.

OrientDB is a cluster composed of at least three instances configured as multi-master (see fig. 6.4). This configuration provides HA and Eventual Consistency to the Persistence.

The RESTful design of the IS services and the described deployment scenarios allow to satisfy the HA, Eventually Consistency, and HS non-functional requirements (see non-functional requirement 6.1 in section 6.1).

6.8 Information System Summary

The resulting IS architecture exploits the Resilient Two-Phase Locking Transaction Model to provide a transactional system. It is composed by the following different services:

- Resource Registry: a RESTful web service supporting the IS Model to model resources in hierarchical contexts. It has been designed to support any concrete model such as the gCube Model (see section 6.3);
- Three RESTful web services (i.e., Transaction Proxy, Transaction Service, Lock Service) enabling clients to perform ACID transactions with Resource Registry;
- Graph Database: an OrientDB cluster used as persistence by the Resource Registry (see section 6.4).

The IS behaves as publisher in the publish-subscribe pattern (see section 6.1). The Transaction Proxy notifies the changes to the Subscription Notification System in com-

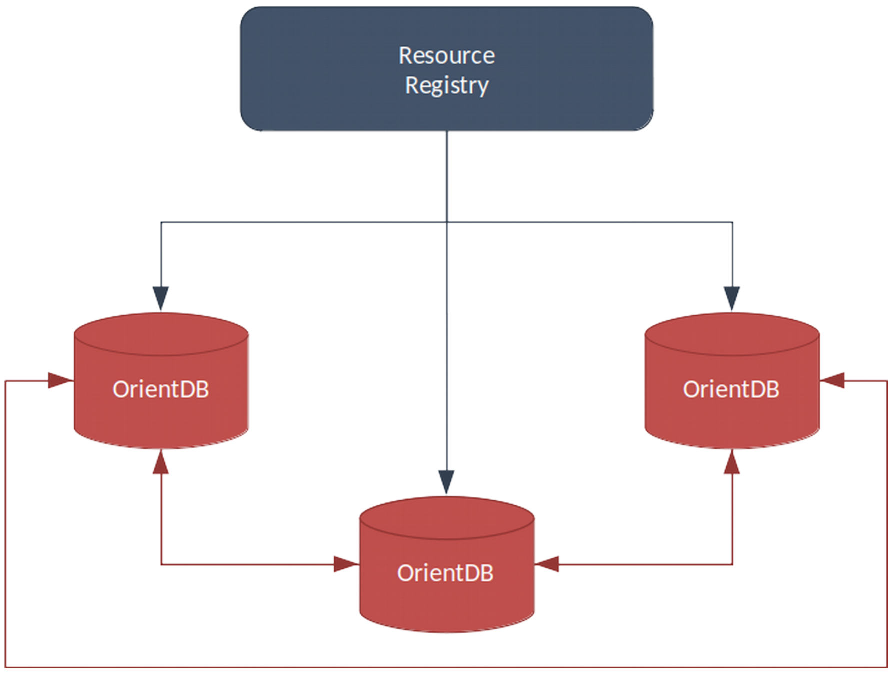


Figure 6.4: *OrientDB Deployment Architecture*

pliance with the Isolation property (see section 6.5); Clients subscribe to the Subscription Notification System to receive notifications about changes of resources representation and their relations. In doing so, the publish-subscribe pattern allows enhancing the scalability of the system by reducing the workload on the IS.

The design of IS and the deployment architecture (see section 6.7) allow to achieve the HA and the HS identified as non-functional requirements (see section 6.1).

The next chapter describes the European projects used to design and validate this research proposal.

CHAPTER 7

Use Cases in Production

The Information System presented in this dissertation has been concretely implemented in a software framework. [67, 69–75]. The framework is part of the gCube Software toolkit[1], an open source software released under European Union Public Licence (EUPL) licence[2].

D4Science is an Hybrid Data Infrastructure (HDI) connecting more than 5500 scientists across more than 50 countries, integrating about 50 heterogeneous data providers, executing more than 55,000 models & algorithms/month and providing access to over a billion quality records in repositories worldwide, with 99,8% service availability. Currently, D4Science hosts over 100 Virtual Research Environments (VREs) to serve the biological, ecological, environmental, social mining, culture heritage, and statistical communities worldwide. D4Science exploits the Information System (IS) software to create the common unified space of resources and to handle VREs.

This chapter describes how the Building Research environments fostering Innovation, Decision making, Governance and Education to support Blue growth (BlueBRIDGE), Social Mining & Big Data Ecosystem (SoBigData), and Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization and Synergies (PARTHENOS) European projects have been exploited to design and validate the presented proposal. Each of them creates a federation of systems and requires the creation of a set of VREs enabled by the D4Science[3] HDI.

This chapter is organised as follows. Section 7.1 presents an overview of the statistic usage of the infrastructure highlighting the advantages of the gCube Model. Section 7.2 presents the BlueBRIDGE European project, describes the created infrastructure and presents how the the 7.2 community exploited the gCube Model. Section 7.3 presents

[1]`https://www.gcube-system.org/`

[2]`https://joinup.ec.europa.eu/collection/eupl/introduction-eupl-licence`

[3]`https://www.d4science.org/`

the infrastructure created within SoBigData. Section 7.4 presents the Digital Human Infrastructure (DHI) created in the context of PARTHENOS.

7.1 Infrastructure Overview

D4Science organises and manages shared resources in hierarchical contexts: each resource must belong at least to one context; any context may have its own specific access and management policies under the assumption that those tailored policies are compliant with the ones provided in the higher level of the hierarchy. The level of the hierarchy are not prescribed by the technology; D4Science organised its contexts in three levels: (i) the root level provides an overview of the whole infrastructure containing all resources; (ii) an intermediate level (i.e., Virtual Organisation (VO) level) provides the overviews of the federations created by a scientific community, i.e. a network of interacting scientists sharing common scientific methods and long-term objectives; (iii) the VRE level specialises the VO to model the set of resources exploitable by a scientific sub-community focused on a particular scientific field for a determined time frame.

The D4Science infrastructure exploited the gCube IS V.1 till October 2017. In particular the IS Collector service, designed to manage the persistence of the resources with an eXtensible Markup Language (XML) database (i.e., eXist-db[4]), was used by adopting the following deployment architecture: (i) an instance of the IS Collector service was deployed to manage the root level; (ii) a dedicated instance of the IS Collector service was deployed to manage each VO level.

This deployment architecture was selected to reduce the server load by partitioning the requests across several server instances. However, any operation of update of a resource required to generate as many operations as the number of VOs exploiting that shared resource.

Table 7.1 shows the monthly number of operations managed by the IS Collector (IS V.1 see section 3.7) and the Resource Registry (IS V.2 see section 6.3) during 2017. From May to October, D4Science exploited both the ISs in accordance with the adoption plan spread over six months. This was needed given the high-number of services, clients, applications, and third-parties tools exploiting the IS Collector service to discover and exploit D4Science resources. During this period, some clients continued their exploitation of the IS V.1 while others were instructed to use the IS V.2. In some cases, the IS V.1 was used as a sort of fallback if the requested information were not available in IS V.2.

To compare the two systems, we consider April for the IS Collector and November for the Resource Registry. The reasons for comparing the two systems by using the operations performed in these two specific months are the following:

- April was the last month of the infrastructure using only the IS V.1 and November was the first month of the infrastructure exploiting only the IS V.2;
- the overall infrastructure exploitation in these two months was similar;
- the number of days was the same (30);
- the number of overall resources registered in the infrastructure was comparable;

[4]http://exist-db.org

Month	IS Collector (IS V.1) Requests	Resource Registry (IS V.2) Requests
January	20.288.338	-
February	17.783.726	-
March	26.728.493	-
April	22.511.962	-
May	20.479.330	133.745
June	27.774.748	217.630
July	25.905.594	227.776
August	7.220.950	3.135.908
September	9.434.581	5.687.902
October	4.748.933	11.505.191
November	-	14.280.738
December	-	9.158.144

Table 7.1: *Number of requests received from IS Collector (IS V.1) and the Resource Registry (IS V.2) during 2017.*

- the transaction model was not in place for IS V.2 [5];
- the subscription-notification mechanism was not in place;
- the number of contexts was the same.

In both months, the infrastructure was operating 152 contexts as follow: 1 root; 7 VOs; and 144 VREs.

Table 7.2 shows the size of the resource instances managed by the gCube IS V.1. Table 7.3 shows how the five resource types of IS V.1 were mapped to the resource types of the gCube Model (see section 4.3). The average number of instances is reported as well in both cases . The table focuses only on the resource types used to compare the behaviour of the two systems and it does not consider other resource types that were not used in this comparison (e.g., *Actor*, *VirtualMachine*).

Resource Type [6]	Avg Number	Avg Size (byte)	Geometric Mean	Max Size	Min Size	Standard Deviation
Hosting Node	191	5154	4847	42691	3326	3394
Running Instance	788	1637	1433	44498	813	1943
Service Endpoint	328	2342	1639	65629	577	4382
Generic Resource	2250	7830	1402	402061	374	36172
Software	2493	1015	1004	3175	819	177

Table 7.2: *Resources in gCube IS V.1 extracted from database dumps of April 2017.*

It is worth to highlight that the *Generic Resources* of IS V.1 was used to map very different concepts (see section 3.7) which were not supported as primary resource type. The *Generic Resource* was used in the previous IS to describe:

[5]The transaction model adds extra calls to the Resource Registry to support the rollback of the operations.

[7]The *EServices* corresponding to Running Instances can be differentiated from the *EServices* corresponding to Service Endpoints looking for an incoming *Activates* relation (see section 4.3.3) from *HostingNode*

IS V.1 Resources	April Avg Instances	gCube Model Resources	November Avg Instances
Hosting Node	191	*HostingNode*	203
Running Instance	788	*EService* [7]	811
Service Endpoint	328	*EService*	343
Generic Resource	2250	*RunningPlugin* (386) *Configuration* (372) *ConfigurationTemplate* (113) *VirtualService* (152) *Dataset* (1301)	2324
Software	2493	*Software* (1179) *Plugin* (422)	1601

Table 7.3: *Comparison between resource types and instances in April 2017 of gCube IS V.1 and resource types and instances of gCube Model (see section 4.3) in November 2017.*

- plugins running on both *Running Instances* and *Service Endpoints*: the gCube Model instead provides a specific *RunningPlugin* resource type (see section 4.3.3);
- configuration and templates: the gCube Model provides *ConfigurationTemplate* and *Configuration* resource types (see section 4.3.3);
- information about bundle of resources (e.g., datasets, running instances, service endpoints) required in a specific context to implement a macro-functionality: the gCube Model provides the *VirtualService* resource type (see section 4.3.3);
- dataset: the gCube Model provides the *Dataset* resource type (see section 4.3.3);

The heterogeneity of the information represented in *Generic Resource* emerges also by looking to the standard deviation of such resource in table 7.2.

The number of *Software* managed by IS decreased because several software were not used anymore and this became evident in IS V.2 where *Software* not referenced by any service were safely unregistered.

The number of *HostingNodes*, *Running Instances* (modelled as *Running Instances*), and *Service Endpoint* (modelled as *Running Instances*) managed by the two IS is comparable.

The *Hosting Nodes* and *Running Instances* resources of the IS V.1 and the corresponding *HostingNodes* and *EServices* IS V.2 are created and updated by gCube-empowered containers (namely gCube Smartgears see section 3.7). The *Hosting Nodes* size variation among the instances is due to the running environment variables reported in the resource description, variables that depend on the server where the *Hosting Nodes* is deployed. The *Running Instances* size variation among the instances is due to either the Web Services Description Language (WSDL) or the Web Application Description Language (WADL) reported in the resource description.

Smartgears updates the status of the created instances every ten minutes to offer always the most updated information about the managed resources to the clients of the infrastructure. This means that, each month Smartgears performs 4320 (30 days * 24 hours * 6 updates per hour) updates for each managed hosting node and service instance.

The number of updates the IS (either IS Collector or Resource Registry) receives increases linearly with the number of instances of either *Hosting Nodes* and *Running Instances* or the corresponding *HostingNodes* and *EServices*. In IS V.1 the updates are propagated to the two top-level contexts (i.e., root level and VO level) the resource belongs to.

IS Version	Resource Type	Instances	Updates	Total Updates	Avg Size (Byte)	Network Traffic
1	*Hosting Node*	191	2.475.360	12.558.240	5154	~11,9 GB
	Running Instance	778	10.082.880		1637	~15,4 GB
2	*HostingNode*	203	876.960 (35,43%)	4.380.480 (34,88%)	313 (6%)	~262 MB (2,15%)
	EService	811	3.503.520 (34,75%)		309 (18,88%)	~1 GB (6,56%)

Table 7.4: *Smartgears predictable updates analysis.*

Table 7.4 shows the number of updates performed by Smartgears. Giving that the average number of top-level contexts (root and VOs) is 3 (meaning that a resource is shared across two VOs in average), the number of monthly updates for IS V.1 has been multiplied by factor 3. Those resources can be updated also by *Running Instances* and *EServices* but the number of those updates is not significant compared to the number of updated performed by Smartgears. For this reason, we limit the analysis of the updates to the ones performed by Smartgears and we refer to them as Predictable Updates.

The average size of an update of *HostingNode* and *EService* is calculated taking into account the size of the updated facets. In particular, the most frequent updated facet is the *StateFacet* which size is i.e., 313 byte for *HostingNode* and 309 byte for *EService*.

	IS V.1	IS V.2	Differences
Operations	22.511.962	14.280.738 (63,44%)	8.231.224 (36,56%)
Predictable Updates	12.558.240	4.380.480 (34,88%)	8.177.760 (65,12%)
Remaining Operations	9.953.722	9.900.258 (99,46%)	53.464 (0,54%)

Table 7.5: *Comparison .... The percentage indicated between parenthesis are referred to the number in the first column which indicate he number of operation in the IS V.1.*

The table shows that in terms of updates the IS V.2 receives 1/3 of the requests received by IS V.1. Moreover, the network traffic decreases significantly as expected because only one faced is modified and transferred over the network instead of the entire resource description.

As highlighted in table 7.5, the remaining number of operations (i.e., creation and delete) managed by IS V.1 and IS V.2 is comparable and the differences are mainly related to the different behaviour of the clients of the infrastructure observed in the two analysed months.

It is important also to notice that the deployment architecture adopted by IS V.1 was selected to deal with its rigid model which does not allow to tailor the description of a resource to the needs of a specific context. In order to support different representations for the same resource in different contexts (particularly useful for the Generic Resources and Service Endpoints) was required to deploy a dedicated instance of IS V.1 serving just one VO.

The adoption of the new IS V.2 in a real operational infrastructure highlighted the following advantages:

- highly reduced network traffic thanks to the selective update of the facets instead of the entire resource;
- reduced number of operations thanks to the adoption of a simplified deployment architecture;
- simplified maintenance of the network of resources thanks to the intrinsic nature of the new model and its relations.

In the following sections the specific advantages introduced for each community are presented.

7.2 BlueBRIDGE

BlueBRIDGE (Building Research environments fostering Innovation, Decision making, Governance and Education to support Blue growth) is a European project supporting capacity building in interdisciplinary research communities actively involved in increasing scientific knowledge about resource overexploitation, degraded environment and ecosystem with the aim of providing a more solid ground for informed advice to competent authorities and to enlarge the spectrum of growth opportunities as addressed by the Blue Growth Societal Challenge. [8].

BlueBRIDGE provides VREs in the following areas:

- Ecosystem approach to Fisheries - services for stock assessment and the generation of unique identifiers for global stocks;
- Aquaculture - services supporting the analysis of socio-economic performance in aquaculture;
- Maritime Spatial Planning - spatial planning services to identify aquaculture and fisheries infrastructures from satellite imagery and tools to visualise, analyse and report on a range of ecologically important seafloor features within marine protected areas;
- Education - tools to set up and deliver cost-effective training courses.

Each BlueBRIDGE VRE provides their users with a web-based working environment offering the entire spectrum of applications and facilities (including services, data and computational facilities) needed to accomplish a given task. It offers specialised

[8]http://ec.europa.eu/programmes/horizon2020/en/news/blue-growth-research-innovation

functionalities for the management, processing, and visualisation of scientific data and textual content.

BlueBRIDGE federates ("bridges", using the project terminology) a rich array of information systems, infrastructures and technologies.The most relevant are: the Catalogue of Life (CoL), the Food and Agriculture Organization (FAO) Geospatial Data Catalogue (FAO GeoNetwork), the Global Biodiversity Information Facility (GBIF), the Integrated Taxonomic Information System (ITIS), the Interim Register of Marine and Nonmarine Genera (IRMNG), the Marine Species Distribution Infrastructure (AquaMaps), the Ocean Biogeographic Information System (OBIS), the World Register of Marine Species (WoRMS). BlueBRIDGE has been important to validate and refine the gCube Model, helping to identify and describe the resources of these systems.

The BlueBRIDGE gateway, available at `https://bluebridge.d4science.org/`, is the access point to the VREs created in the context of the project.

7.2.1 BlueBRIDGE gCube Model Usage

BlueBRIDGE does not provide any specialisation neither of IS Model nor of gCube Model. In particular, analysing the BlueBRIDGE VO instances we observed the following usage of the model:

- 13 of 15[9] (86,67%) *Resource* specialisations;
- 15 of 19 (~78,9%) *Facet* specialisations;
- 14 of 16 (87,5%) *IsRelatedTo* specialisations;
- 12 of 14 (~85,7%) *ConsistsOf* specialisations.

In particular the following *Entities* and *Relations* were not exploited:

- *Resource* specialisations: *LegalBody*, *Schema*;
- *Facet* specialisations: *CapabilityFacet*, *DescriptiveMetadataFacet*, *SchemaFacet*, *SubjectFacet*;
- *IsRelatedTo* specialisations: *BelongsTo*, *IsCompliantWith*;
- *ConsistsOf* specialisations: *HasMaintainer*, *HasManager*.

In this project, the usage of the new model has especially been useful to simplify the creation of VREs and to reduce the effort requested to the VRE and Operation Managers. This has been achieved by creating a set of *VirtualService* to model macro-functionality, e.g., Data Analysis, Data Mining, Marine Species Discovery, frequently deployed and activated in several VREs. Each defined *VirtualService* is related to the set of resources required to provide the functionality e.g., *EService* using *CallsFor*, *Dataset* using *Manages*, *Configuration* and *ConfigurationTemplate* using *IsCustomizedBy*. The macro category of *Dataset* is related to a set of other *Dataset* and *ConcreteDataset* using *IsCorrelatedTo*.

Each *IsRelatedTo* relation has 'propagate' as propagation constraint. This allows to add a set of resources to any newly created context by performing one single and

[9] *Actor* and *Service* are abstract resource types, hence they can't be instantiated

simple action thus significantly reducing both the number of operations activated by the Operation Manager and the time and the issues previously observed with IS V.1. The usage of *Dataset* as bunch was not initially planned, still it provided the opportunity to reflect on defining the resource type *VirtualDataset* [10].

Before the introduction of the gCube Model based IS, the creation of a VRE required much effort and knowledge also of the VRE Designer. The VRE Designer is the person entitle to design the VRE selecting the required services and datasets. With the IS V.1, the VRE Designer had to find and select every single *Dataset* and every service composing a macro-functionality. This was not sufficient since the VRE Manager had to complete this initial list with the set of *Generic Resources* containing tailored configurations for the specific context. To partially overcome this issue, in the IS V.1 some of the macro-functionality were defined using *Generic Resources* which mainly contained the id of the services and datasets. However, giving the high dynamic behaviour of the infrastructure, the usage of the *Generic Resources* often resulted into broken references since *Running Instances* could have been either removed or recreated with a different id on a different *Hosting Node*. The Operation Manager had To fix all the inconsistencies at creation and activation time of any new VRE.

BlueBRIDGE has also served a set of VREs for scientific training support. The training were mainly operated by International Council for the Exploration of the Sea (ICES). The course trainer often provided the specification of services and datasets just two or three days before the training starts. With the old IS, the VRE Manager had to allocate more than one full day to ensure that the VRE was fully operational while his effort has been reduced to less than one hour with the adoption of the new IS.

Clearly, the introduction of the new IS required the definition of the macro-functionality and macro-dataset but this effort was spent just one time saving time at each creation or modification of one of the more than sixty VREs operated by the BlueBRIDGE community.

7.3 SoBigData

SoBigData (Social Mining & Big Data Ecosystem) is a European project creating a Research Infrastructure (RI) for Big Data and Social Mining. The RI provides VREs enabling data scientists to access and manage large datasets via the exploitation of hundreds of distributed and interoperable analytics services, accessible through common interfaces [81]. SoBigData promotes repeatable and open science.

The SoBigData consortium consists of 12 partners, mostly from member countries of the European Union, i.e., Italy, United Kingdom, Germany, Estonia, Finland, The Netherlands and Switzerland.

SoBigData is organised in six thematic clusters embracing different research fields i.e., i) Text and Social Media Mining (TSMM); ii) Social Network Analysis (SNA); iii) Human Mobility Analytics (HMA); iv) Web Analytics (WA); v) Visual Analytics (VA);vi) Social Data (SD). SoBigData federates seven national infrastructures and data centres:

- SoBigData.it, a knowledge infrastructure of the European laboratory on big data

[10] The choices made were not part of this research, instead they were performed by entitled people which in some cases could not be shared

analytics and social mining, funded by Consiglio Nazionale delle Ricerche (CNR) (the Italian National Research Council) and the University of Pisa, offering data, services and expertise on TSMM, SNA, HMA, WA [11];

- GATE Cloud, a cloud-based infrastructure of the University of Sheffield for language processing and text mining offering data, services and expertise on TSMM [12];
- IVAS, a server-based infrastructure of the Fraunhofer competence centre for Information Visualization and Visual Analytics offering data, services and expertise on VA;
- Alexandria, the infrastructure of the Leibniz University in Hannover for Web Science offering data, services and expertise on WA;
- AALTO, the sociophysics and data mining laboratories of Aalto University in Helsinki offering data, services and competencies on SNA;
- E-Gov.data, the centre at University of Tartu curating the Estonian e-government and e-health data, offering access and services to 10-years dataset on SD;
- Living Archive for Open Data, a search engine of the Department of Sociology at ETH Zurich for open data on SD.

The SoBigData gateway, available at `https://sobigdata.d4science.org/`, provides users with access to different thematic research environments, called exploratories:

- City of Citizens: tells stories about cities and people living in it. Data scientists describe those territories using data, statistics and models. This exploratory allows citizens and local administrator to understand better cities and how to improve them;
- Well-being & Economy: uses data of purchases in supermarkets and investigates the changes in people's behaviour after the economic crisis. This study allows to work out an early indicator of disease. It also allows to investigate the measurement of the real cost of living by studying the price variation;
- Societal Debates: investigates on public debates through the analysis of discussions on social media and newspaper articles to understand which are the most discussed topics. Themes can be identified, following the discussions around them and tracking them through time and space;
- Migration Studies: analyses the phenomenon of international migration with Big Data tools. It looks at migration flows and stocks, migrant integration, cultural diversity, and the return of migrants;
- Sports Data Science: tells stories about sports analytics. Sports data scientists describe performances using data, statistics and models. This exploratory allows coaches, fans and practitioners to understand better and boost sports performance.

[11] `http://www.sobigdata.it/`
[12] `https://gatecloud.net/`

SoBigData gateway, also, provides access to different services via VREs such as:

- M-Atlas, a mobility querying and data mining system centred onto the concept of spatio-temporal data;
- e-Learning Area, training materials developed within the SoBigData project;
- SMAPH, entity linking on web queries and concise text, meaning it, disambiguate query terms, linking them to their unambiguous meaning represented as an entity in a Knowledge-base;
- SoBigData Lab, one-stop shop, where scientists can find, try and use SoBigData methods as integrated into the infrastructure by scientists across multiple disciplines of Social Mining;
- TagMe, an annotation tool able to identify on-the-fly meaningful short-phrases (called "spots") in unstructured text and link them to a pertinent Wikipedia.

7.3.1 SoBigData gCube Model Extension

In SoBigData, the gCube Model has been extended to support specific use cases[13].

Acknowledged specialisations are:

- *Story* <: *VirtualService*;
- *CityService* <: *VirtualService*;
- *Idea* <: *VirtualService*;
- *FreeTextFacet* <: *Facet*;

A *Story* references the set of services and datasets required to perform the story.

For example, in City of Citizens VRE, the *Story* has been used to model the 'Investigating City Mobility'. This specialisation allowed to assign fine-grained authorisation to the set of resources modelled with these specific resource type.

The 'Investigating City Mobility' *Story* is related to a set of *CityService*. Each *CityService* is related to services (e.g., *HostingNode*, *EService*, *VirtualService*) and datasets (e.g., *ConcreteDataset*,*Dataset*) which represents the whole service.

Using the IS V.1 the only way to represent such information was to define a Generic Resource containing the list of ids of services (Running Instances, Service Endpoints, Generic Resources) and datasets (other Generic Resources).

Both *Story* and *CityService* are described (among other facets normally used for services see section 4.3.3) by:

- *IdentifierFacet* (*IsIdentifiedBy*): contains the name;
- *FreeTextFacet* : contains the human description of the service.

SoBigData promotes research ideas by allowing to gateway users to propose experiments to the consortium. The idea is submitted indicating the involved people or institution, the required services or datasets and an abstract describing the experiment. The *Idea* is persisted in the IS which is mainly described by:

[13]The definition of the specialisations was not part of this study. The specialisations have been defined to solve specific issues by the entitled responsible.

- *IdentifierFacet* (*IsIdentifiedBy*): contains the proposed name for the experiment;
- *FreeTextFacet* : contains the abstract of the idea;
- *StateFacet* : contains the state of the idea (e.g., submitted, under evaluation, rejected, accepted).

7.4 PARTHENOS

DHIs are e-Infrastructures supporting researchers in the field of Humanities with a digital environment where they can find and use Information and Communication Technology (ICT) tools and research data for conducting their research activities.

A growing number of DHIs have been realised, most of them targeting a specific sector of the Humanities. Thanks to their discipline-specific feature, those DHIs offer specialised services and tools to researchers, which are now demanding support for interdisciplinary research, common solutions for data management, and access to resources traditionally relevant to different sectors (e.g., text-mining algorithms traditionally used by linguistics can also be useful to historians and social scientists).

The European Commission launched the PARTHENOS project (Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization and Synergies) to bridge existing DHIs by forming an integrated environment to allow researchers of different sectors of the humanities to collaborate and share data, services, and tools.

The DHIs federated by PARTHENOS are: ARIADNE[14] (archaeology); CENDARI[15] (history); CLARIN[16] (linguistic studies); CulturaItalia (archaeology, arts and humanities); DARIAH[17] (arts and humanities); EHRI[18] (studies on the holocaust); TGIR Huma-Num [19] (humanities and social sciences). [3]

ARIADNE [4, 95] brings together and integrates existing archaeological research data infrastructures so that researchers can use the various distributed datasets and new and powerful technologies as an integral component of the archaeological research methodology.

CENDARI [25] is a research infrastructure project aimed at integrating digital archives for medieval and modern European history. The CENDARI Repository holds information on over 1,200 collection holding institutions and over 300,000 records (metadata and some full text) relevant for the study of the medieval period and First World War.

CLARIN [117] is a research infrastructure initiative that aims at providing a single domain of Language Resource (LR) and Language Technology (LT) to researchers from the Social Sciences and Humanities (SSH) disciplines. The META-SHARE [79] registry federation is designed as a network of distributed repositories of LRs, including language data and basic language processing tools (e.g., morphological analysers, Part Of Speech (POS) taggers, speech recogniser). The Language Resource and Evaluation Map (LRE Map) initiative issued out of the FLaReNet project [114], whose mission was to develop a common vision of the area of LRs and to foster a European strategy for consolidating the sector and enhancing competitiveness at EU level and worldwide.

[14]ARIADNE: `http://www.ariadne-infrastructure.eu/`
[15]Collaborative European Digital Archive Infrastructure (CENDARI): `http://www.cendari.eu`
[16]Common Language Resources and Technologies infrastructures (CLARIN): `https://www.clarin.eu`
[17]Digital Research Infrastructure for Arts and Humanities (DARIAH) `http://www.dariah.eu/`
[18]European Holocaust Research Infrastructure (EHRI): `https://www.ehri-project.eu/`
[19]Très Grande Infrastructure de Recherch (TGIR) Huma-Num `https://www.huma-num.fr`

CulturaItalia [20] is the Portal of Italian Culture, managed by the Central Institute for the Union Catalogue of Italian Libraries (ICCU) of Ministry of Cultural Heritage, Activities and Tourism (MiBACT). CulturaItalia is an aggregator playing an important role for the development of European research infrastructures on Cultural Heritage such as ARIADNE, DARIAH and Europeana [21], making available cooperative networks and agreements and coordinating technical activities.

DARIAH [21] is a pan-European infrastructure for arts and humanities scholars working with computational methods. It supports digital research as well as the teaching of digital research methods. DARIAH currently connects several hundreds of scholars and dozens of research facilities in 17 European countries.

EHRI [30] provides online access to information about dispersed sources relating to the Holocaust through its Online Portal. It offers tools and methods that enable researchers and archivists to work with such sources collaboratively.

TGIR Huma-Num is a research infrastructure aimed at facilitating the turning of digital research in the humanities and social sciences. The TGIR Huma-Num offers services dedicated to the production and reuse of scientific data. TGIR Huma-Num supports research teams throughout their digital projects to allow the sharing, reuse and preservation of data thanks to a chain of devices focused on interoperability.

Within PARTHENOS, a complete technical framework has been produced for the federation of the presented DHIs, enabling transparent access to resources managed by different DHIs and enabling the creation and operation of VREs. The information system presented in this dissertation is one of the key components of this technical framework. The details about the usage of the information system presented in this dissertation are available in [19, 76].

The technical framework supports the realisation of the federation by offering tools for:

- the creation of a homogeneous information space where all resources (data, services, and tools) of the different DHIs are described according to a common data model;
- the discovery of available resources;
- the use of available resources (for download or processing);
- the dynamical creation of VREs where users can find resources relevant for a research topic, run services, and share the computational results.

The PARTHENOS gateway, available at `https://parthenos.d4science.org/`, is the access point to the VREs created in the context of the project.

7.4.1 PARTHENOS Entities Model

The PARTHENOS Entities Model (PE Model) [29] is the semantic data model (defined by researchers from the Information Systems Laboratory at Foundation for Research and Technology-Hellas (FORTH) (Crete - GR)).

[20] `http://dati.culturaitalia.it`
[21] https://www.europeana.eu

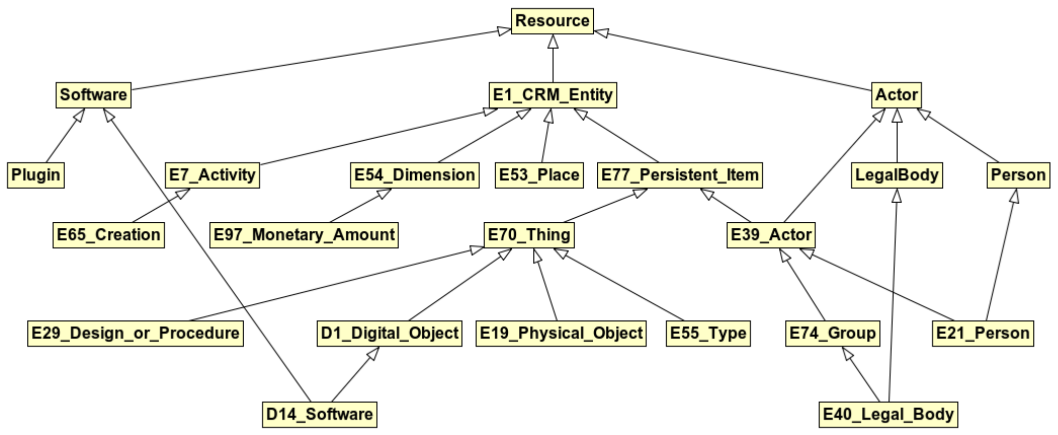


Figure 7.1: *UML class diagram of CDOC-CRM entities extending* Resource *and gCube Model resource types.*

PE Model captures and represents the knowledge generation process: which actors and which services are involved in knowledge creation, resource curation, and management chains.

To support the PARTHENOS project, the following entities have been added to the IS (some of the types belong to CDOC-CRM [85] and CRMdig [52] ontologies.)

- 56 *Resource* types, (see examples in pictures 7.1 and 7.2);
- 6 Facet types;
- 56 isRelatedTo types;
- 6 consistsOf types.

The mapping of the PE Model onto the gCube Model [3] has a dual validity: *(i)* it is a proof of the flexibility and openness of the latter and in turn a further validation of the IS Model and, *(ii)* it allowed to validate the semantics of the defined resources and relations types in the PE Model.

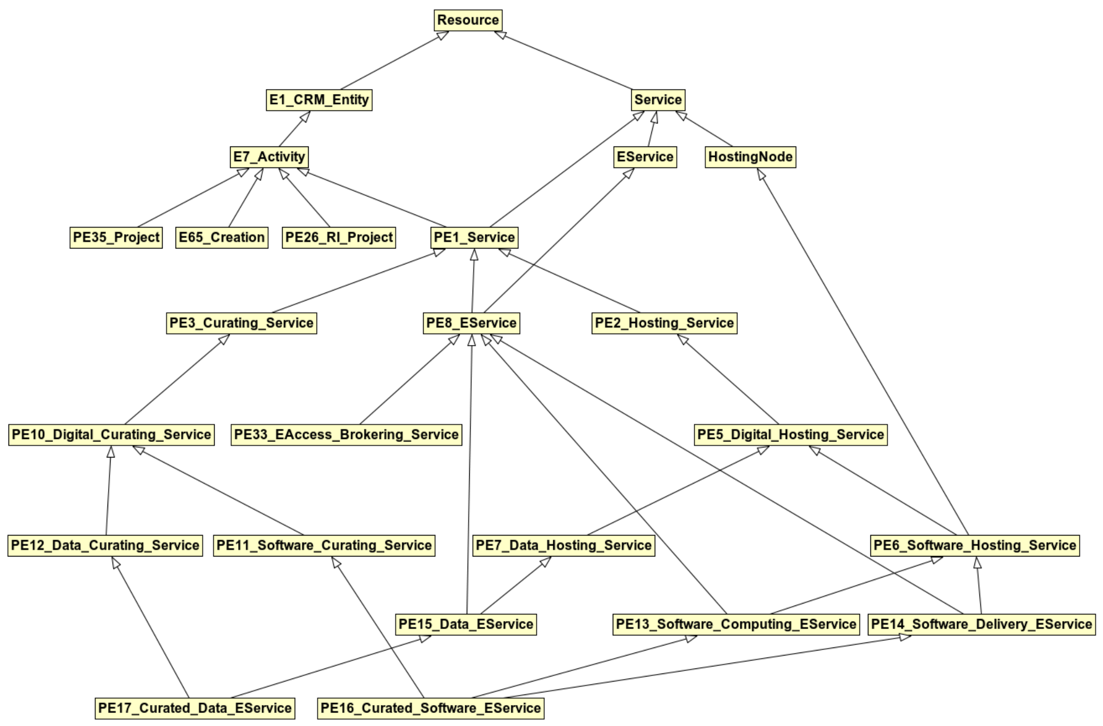

Resource
E1_CRM_Entity
Service
E7_Activity
EService
HostingNode
PE35_Project
E65_Creation
PE26_RI_Project
PE1_Service
PE3_Curating_Service
PE8_EService
PE2_Hosting_Service
PE10_Digital_Curating_Service
PE33_EAccess_Brokering_Service
PE5_Digital_Hosting_Service
PE12_Data_Curating_Service
PE11_Software_Curating_Service
PE7_Data_Hosting_Service
PE6_Software_Hosting_Service
PE15_Data_EService
PE13_Software_Computing_EService
PE14_Software_Delivery_EService
PE17_Curated_Data_EService
PE16_Curated_Software_EService

Figure 7.2: *UML class diagram of PARTHENOS entities extending* E7_Activity *and gCube Model resource types.*

CHAPTER 8

Concluding Remarks

8.1 Requirements, Resources and their Organisation

Hybrid Data Infrastructures (HDIs) manages combination of resources offering capabilities and capacities for research data management via Virtual Research Environments (VREs). These environments federate on-premises resources, i.e., data, services, tools, eventually integrating them with other resources managed by Research Infrastructures (RIs), to enable their exploitation in a "common unified space". We described this type of system (or more properly, ecosystem) as System of Systems (SoS) in section 2.4.

Resources are assigned by the HDI to different VREs. VREs provide a comprehensive, web-based, user-friendly and easy-to-use environment to researchers who can work collaboratively to meet a specific goal.

To keep track of the federated resources and their assignment to VREs, the HDI must offer publish, discovery and access features through a framework named Information System (IS). However, the different typologies of resources, the heterogeneous description of the resources of the same type, and the dynamic evolution of the participant systems that evolve independently from the federation, pose a set of challenges due to the intrinsic nature of any open-ended federation. These challenges and solutions have been analysed with the formalisation of a new resource model and the design of an IS hosting the description of the resources, supporting their heterogeneity and evolution, and providing different views of those resources organised hierarchically.

8.2 Proposed Models

A major outcome of this thesis is the formalisation of a new resource model: IS Model (see section 4.2), consisting of few constructs which can be instantiated to fulfil the requirements of any concrete scenario.

The IS model defines:

- two entity types:
  - *Resource* i.e., entities representing a description of a "thing" to be managed;
  - *Facet* i.e., entities contributing to "build" a description of a Resource. Each Resource is described by a set of Facets, each of them capturing a certain aspect/characterisation of the resource.
- two relation types. Relations have a direction with a "source" and a "target". Nevertheless, when inspecting the graph, relations can be navigated in both directions, i.e., from source to target and from target to source.
  - *IsRelatedTo* i.e., a relation linking two *Resource*;
  - *ConsistsOf* i.e., a relation connecting a *Resource* with a *Facet*.
- context definition rules;
- type inheritance rules;
- type instantiation rules;
- instances sharing across contexts rules;
- referential integrity rules.

The thesis also presents the gCube Model, a concrete instance of the IS Model (see section 4.3). With seventeen *Resource* types, twenty *IsRelatedTo* types, nineteen *Facet* types and fifteen *ConsistsOf* types, the gCube Model has proved to properly capture the resources of two different federated RIs created in the context of BlueBRIDGE and SoBigData European projects (see sections 7.2 and 7.3).

Further, in the context of PARTHENOS European project, the PARTHENOS Entities Model has been mapped on top of the gCube Model (see section 7.4). This supplemental level of abstraction demonstrates, even more, the full extent of the proposed resource model and its adaptability to highly specialised research contexts.

8.3 Design of Supporting Software

Section 6.1 introduced the set of functional and non-functional requirements identified and analysed in order to design the architecture of the IS for HDIs.

REpresentational State Transfer (REST) is a powerful service architectural style to support scalability of the service while keeping the complexity of the design, of the implementation, and of the deployment at very affordable costs. REST provides a set of principles, but it does not provide any concrete implementation (see section 5.1). The IS has been designed to comply with Resource Oriented Architecture (ROA) which defines a concrete architecture to satisfy the REST principles using HyperText Transfer Protocol (HTTP) (see section 5.2).

Since the IS enables the sharing of resources among VREs, different and non-collaborative clients can concurrently modify the resources and the way they are accessed and exploited in the HDI. Concurrent modifications could bring the IS in an inconsistent state that cannot be tolerated in a production-quality environment. To avoid those issues, it is required a transaction model capable of providing Atomicity, Isolation, Durability and Consistency (ACID) properties.

The analysis of state-of-the-art revealed that the RESTful transaction models either violate one or more REST principles or do not implement all the ACID properties (see section 5.4). Being these fundamental aspects of the IS, we adopted the Resilient Two-Phase Locking Transaction Model (see section 5.3). This proposed RESTful transaction model is designed to: (i) provide resiliency i.e., the ability to return to its original state after a component failure; (ii) support distributed transactions i.e., transaction involving different RESTful services; (iii) comply with already existent non transactional clients (i.e., optionality); (iv) add the transaction capability to already existent RESTful services (i.e., optionality) via a layered approach.

The Resilient Two-Phase Locking Transaction Model defines three RESTful services:

- Transaction Service provides transaction as resource capabilities. It enables to log the requests occurred in the transactions. In the case of rollbacks, the Transaction Service uses the logs to compensate the occurred requests;
- Lock Service provides lock as resources capabilities;
- Transaction Proxy intercepts all requests made from clients. It requests the appropriate lock to the Lock Service; then it logs the request on the Transaction Service. Finally, it forwards the request to the effective service returning the result to the requesting client.

The resulting IS architecture exploits the Resilient Two-Phase Locking Transaction Model to offer a full-fledged transactional system. It is composed by the following different services:

- Resource Registry: a RESTful web service supporting the IS Model to model resources in hierarchical contexts. It has been designed to support any concrete model such as the gCube Model (see section 6.3);
- Three RESTful web services (i.e., Transaction Proxy, Transaction Service, Lock Service) enabling clients to perform ACID transactions with Resource Registry;
- Graph Database: an OrientDB cluster used as persistence layer by the Resource Registry (see section 6.4).

The design of IS along with the presented deployment architecture (see section 6.7) allows to achieve the non-functional requirements of High-Availability (HA) and the Horizontal Scalability (HS) (see section 6.1).

8.4 Relevant Achievements

- A collaborative effort was put in place among several research groups, institutions and projects to make this work possible.
- The models presented in this dissertation have been successfully applied and validated in real scenarios.

- The IS software handling the IS model has been implemented [67, 69–75] and deployed in the D4Science[1] HDI.
- Overall, the work presented in this dissertation concretely supports the European projects BlueBRIDGE, SoBigData and Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization and Synergies (PARTHENOS). These projects have played the dual role of being major providers of requirements and validators for this work.

8.5 Lesson Learned and Future Work

Significant effort has been spent to provide fully RESTful transaction facilities both from a general perspective and for specific cases, maintaining the overall model compatibility. We consider that an important achievement. The downside is the impact on the response time of the system adopting the transaction model (see section 6.5) because of the additional round-trip and, in some cases, the required lock. As future work, we will investigate a transaction model where the Resource Registry: creates/requests the locks to Lock Service; and creates the shadow resources to the Transaction Service. The main goal is to remove the shared-lock on the entire graph while a client performs a query. The challenges will be to design the transaction model in a way to be compliant with the general transaction model; and to keep the service stateless.

Discussing with the communities participating to the European projects, it emerged the request to reuse/share a facet instance from/within multiple resources to avoid duplication of information. A typical request was to reuse/share the instance of the *ContactFacet* between instances of *Dataset* and the *Actor* instance. According to the model rationale, we explained that the right solution is to relate the *Actor*. Anyway, this request calls for a possible extension of the model by supporting the possibility to "reuse" a Facet. Facet instances "reuse" across Resource instances could be integrated with the following constraints:

- it must be guaranteed automatically by the system (defining a sort of canonical identification of a facet);
- it must be based on a configuration policy;
- it must occur only when a change in a facet instance must affect a change in all the resources described by such a facet.

This capability impacts on the transaction model because it is not predictable the required locks to modify a resource (i.e., two transaction could try to modify/read the same facet starting from different resource). Hence, this capability can be added only after the different transaction model introduced above is in place.

We are aware that the subscription-notification mechanism reduces the load on the infrastructure but we are still not able to evaluate the dimension of the impact for the following reasons:

- only few clients have been migrated to these features and the infrastructure;

[1]https://www.d4science.org/

- the subscription-notification system is used also by other services and actually, the infrastructure does not have fine-grained analytics information to differentiate between notification generated by the Resource Registry and the notifications generated by other services.

We plan to deploy a dedicated subscription-notification system to solve the last issue. Evaluating the impact of subscription-notification mechanism will be hence part of our future work.

The *Header* in all the facets and relation instances increases the size of a Resource up to 200% (with respect to the same Resource with facets and relations instances without header). Considering facets composed by only one property the size increases up to 500%. Analysing the gCube software the metadata information provided by the *Header* (except for the Universally Unique Identifier (UUID)) are actually used only by 2% of the clients. We are evaluating the possibility to move the UUID outside of the *Header* and provide the header information only when the client explicitly requests it (e.g., by using a query parameter `header=true`).

In the context of PARTHENOS, several entities and relation types have been created and used only on the related Virtual Organisation (VO). The number of types could surprise a user of another context requesting the list of types managed by the system. As future work, we will investigate the possibility to restrict the visibility of types to a context and its sub-tree. Adding this capability could impact on the exposed Types Management Application Programming Interfaces (APIs) (see section 6.3.2). Finally, further work will be done to investigate the possibility to create types with the same name but a different schema, across contexts.

Apart from the above-mentioned improvements to this study, this work opens new research questions:

New Research question 1: It is possible to automatically create/suggest new *Virtual Services* and *Datasets* (as collection of other dataset) analysing the description of the created VREs and the succeeding modification?

New Research question 2: Can a higher-level-service analyse the behaviour in registering and accessing the resources via the IS and then use artificial intelligence to answer to the previous question?

New Research question 3: How much this higher-level-service could help to reduce human intervention?

New Research question 4: Can this higher-level-service helps to enhance the sustainability of the infrastructure?

We believe that positive answers can be provided for each question.

Bibliography

[1] Cristina Aiftimiei, Alberto Aimar, Andrea Ceccanti, Marco Cecchi, Alberto Di Meglio, Patrick Fuhrmam, Emidio Giorgio, Balazs Konya, Laurence Field, Jon Kerr Nilsen, Morris Riedel, and John White. Towards next generations of software for distributed infrastructures: The european middleware initiative. In *2012 IEEE 8th International Conference on E-Science*, pages 1–10, Oct 2012.

[2] Subbu Allamaraju. *RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity*. O'Reilly, first edition, nov 2010.

[3] Nicola Aloia, Leonardo Candela, Franca Debole, Luca Frosini, Matteo Lorenzini, and Pasquale Pagano. Report on the design of the joint resource registry. Technical report, PARTHENOS Deliverable 5.2, April 2017.

[4] Nicola Aloia, Franca Debole, Achille Felicetti, Ilenia Galluccio, and Maria Theodoridou. Mapping the ariadne catalog data model to cidoc crm: Bridging resource discovery and item-level access. *SCIRES-IT - SCIentific RESearch and Information Technology*, 7(1):1–8, August 2017.

[5] Steven Alter. Information systems: A. In *Management Perspective, 2nd ed., Menlo Park, CA: The Benjamin/Cummings*. Publishing Company, 1996.

[6] Rachana Ananthakrishnan, Kyle Chard, Ian Foster, and Steven Tuecke. Globus platform-as-a-service for collaborative science applications. *Concurrency and Computation: Practice and Experience*, 27(2):290–305, mar 2015.

[7] Sergio Andreozzi, Stephen Burke, Felix Ehm, Laurence Field, Gerson Galang, Balazs Konya, Maarten Litmaath, Paul Millar, and JP Navarro. Glue specification v. 2.0. In *Open Grid Forum Recommendation Documents*. Open Grid Forum, 2009.

[8] Renzo Angles. A comparison of current graph database models. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 171–177, apr 2012.

[9] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, feb 2008.

[10] Alexey Anisenkov, Alessandro Di Girolamo, Maria Alandes, and Edward Karavakis. Agis: Evolution of distributed computing information system for atlas. *Journal of Physics: Conference Series*, 664(6):062001, 2015.

[11] Alexey Anisenkov, Alessandro Di Girolamo, Alexei Klimentov, Danila Oleynik, and Artem Sh Petrosyan. Agis: The atlas grid information system. *Journal of Physics: Conference Series*, 513(3):032001, 2014.

[12] Alexey Anisenkov, Alexei Klimentov, Roman Kuskov, and Torre Wenaus. Atlas grid information system. *Journal of Physics: Conference Series*, 331(7):072002, 2011.

[13] Massimiliano Assante, Leonardo Candela, Donatella Castelli, Roberto Cirillo, Gianpaolo Coro, Luca Frosini, Lucio Lelii, Francesco Mangiacrapa, Valentina Marioli, Pasquale Pagano, Giancarlo Panichi, Costantino Perciante, and Fabio Sinibaldi. The gcube system: Delivering virtual research environments as-a-service. *Future Generation Computer Systems*, 2018.

[14] Massimiliano Assante, Leonardo Candela, Donatella Castelli, Gianpaolo Coro, Lucio Lelii, and Pasquale Pagano. Virtual research environments as-a-service by gcube. *PeerJ Preprints*, 4:e2511v1, 2016.

[15] Massimiliano Assante, Leonardo Candela, Donatella Castelli, Francesco Mangiacrapa, and Pasquale Pagano. A social networking research environment for scientific data sharing: The d4science offering. *Grey Journal (TGJ)*, 10(2), 2014.

[16] Massimiliano Assante, Leonardo Candela, Donatella Castelli, and Pasquale Pagano. The d4science research-oriented social networking facilities. *ERCIM News*, 96:44–45, 2014.

[17] Paolo Atzeni, Francesca Bugiotti, Luca Cabibbo, and Riccardo Torlone. Data modeling in the nosql world. *Computer Standards & Interfaces*, 2016.

[18] David E. Avison and AT Wood-Harper. Multiview - an exploration in information systems development. *Australian Computer Journal*, 18(4):174–179, 1986.

[19] Alessia Bardi and Luca Frosini. Building a federation of digital humanities infrastructures. *ERCIM News*, 2017(111), 2017.

[20] Bartosz Belter, Juan Rodriguez Martinez, José Ignacio Aznar, Jordi Ferrer Riera, Luis M. Contreras, Monika Antoniak-Lewandowska, Matteo Biancani, Jens Buysse, Chris Develder, Yuri Demchenko, Pasquale Donadio, Dimitra Simeonidou, Reza Nejabati, Shuping Peng, Łukasz Drzewiecki, Eduard Escalona, Joan Antoni Garcia Espin, Steluta Gheorghiu, Mattijs Ghijsen, Jakub Gutkowski, Giada Landi, Gino Carrozzo, Damian Parniewicz, Philip Robinson, and Sebastien Soudan. The geysers optical testbed: A platform for the integration, validation and demonstration of cloud-based infrastructure services. *Computer Networks*, 61:197–216, 2014. Special issue on Future Internet Testbeds – Part I.

[21] Mirjam Blümm and Stefan Schmunk. Digital research infrastructures: Dariah. In *3D Research Challenges in Cultural Heritage II*, pages 62–73. Springer, 2016.

[22] André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP ’00, pages 195–203, New York, NY, USA, 2000. ACM.

[23] Ann Borda, Jason Careless, Maia Dimitrova, Michael Fraser, Jeremy Frey, Paul Hubbard, Stéphane Goldstein, Caroline Pung, Michele Shoebridge, and Norman Wiseman. Report of the working group on virtual research communities for the ost e-infrastructure steering group. Project report, Office of Science and Technology, mar 2006. VRC final report.

[24] Nathan Bos, Ann Zimmerman, Judith Olson, Jude Yew, Jason Yerkie, Erik Dahl, and Gary Olson. From shared databases to communities of practice: A taxonomy of collaboratories. *Journal of Computer-Mediated Communication*, 12(2):652–672, feb 2007.

[25] Nadia Boukhelifa, Mike Bryant, Nataša Bulatović, Ivan Čukić, Jean-Daniel Fekete, Milica Knežević, Jörg Lehmann, David Stuart, and Carsten Thiel. The cendari infrastructure. *J. Comput. Cult. Herit.*, 11(2):8:1–8:20, April 2018.

[26] Geoffrey Boulton, Philip Campbell, Brian Collins, Peter Elias, Wendy Hall, Graeme Laurie, Onora O’Neill, Michael Rawlins, J Thornton, Patrick Vallance, et al. Science as an open enterprise. *The Royal Society*, 2012.

[27] Eric A. Brewer. Towards robust distributed systems (abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC ’00, pages 7–, New York, NY, USA, 2000. ACM.

[28] Encyclopedia Britannica. information system | definition, examples, & facts | britannica.com. `https://www.britannica.com/topic/information-system`. Accessed May 2018.

[29] George Bruseker, Martin Doerr, and Maria Theodoridou. Report on the common semantic framework. Technical report, PARTHENOS Deliverable 5.1, apr 2017.

[30] Mike Bryant, Linda Reijnhoudt, Reto Speck, Thibault Clerice, and Tobias Blanke. The ehri project-virtual collections revisited. In *International Conference on Social Informatics*, pages 294–303. Springer, 2014.

[31] Richard A. Buckingham, Rudy A. Hirschheim, Frank F. Land, and Colin J. Tully. Information systems curriculum: A basis for course design. In Richard A. Buckingham, Rudy A. Hirschheim, Frank F. Land, and Colin J. Tully, editors, *Information Systems Education: Recommendations and Implementation*, pages 14–133. Cambridge University Press, New York, NY, USA, 1986.

[32] Stephen Burke, Sergio Andreozzi, and Laurence Field. Experiences with the glue information schema in the lcg/egee production grid. In *Journal of Physics: Conference Series*, volume 119, page 062019. IOP Publishing, 2008.

[33] Leonardo Candela, Donatella Castelli, and Pasquale Pagano. Managing big data through hybrid data infrastructures. *ERCIM News*, 2012(89), 2012.

[34] Leonardo Candela and Pasquale Pagano. Cross-disciplinary data sharing and reuse via gcube. *ERCIM News*, 2015(100), 2015.

[35] Leonardo Candela, Pasquale Pagano, Donatella Castelli, and Andrea Manzi. Realising virtual research environments by hybrid data infrastructures: the d4science experience. In *Proceedings of the symposium" International Symposium on Grids and Clouds (ISGC) 2014"(ISGC2014). 23-28 March, 2014. Academia Sinica, Taipei, Taiwan*, 2014.

[36] Annamaria Carusi and Torsten Reimer. Virtual research environment collaborative landscape study. Technical report, Joint Information Systems Committee, 2010.

[37] João Alvaro Carvalho. Information system? which one do you mean? In Eckhard D. Falkenberg, Kalle Lyytinen, and Alexander A. Verrijn-Stuart, editors, *Information System Concepts: An Integrated Discipline Emerging: IFIP TC8/WG8.1 International Conference on Information System Concepts: An Integrated Discipline Emerging (ISCO-4)September 20–22, 1999, University of Leiden, The Netherlands*, pages 259–277. Springer US, Boston, MA, 2000.

[38] Edgar Frank Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, jun 1970.

[39] European Community. Research infrastructures. `https://ec.europa.eu/research/infrastructures/index.cfm?pg=about`. Accessed May 2018.

[40] Wikipedia Community. Information system - wikipedia. `https://en.wikipedia.org/wiki/Information_system`. Accessed May 2018.

[41] Alejandro Corbellini, Cristian Mateos, Alejandro Zunino, Daniela Godoy, and Silvia Schiaffino. Persisting big-data: The nosql landscape. *Information Systems*, 63:1–23, 2017.

[42] Luiz Alexandre Hiane da Silva Maciel and Celso Massaki Hirata. An optimistic technique for transactions control using rest architectural style. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 664–669, New York, NY, USA, 2009. ACM.

[43] Luiz Alexandre Hiane da Silva Maciel and Celso Massaki Hirata. A timestamp-based two phase commit protocol for web services using rest architectural style. *J. Web Eng.*, 9(3):266–282, sep 2010.

[44] Luiz Alexandre Hiane da Silva Maciel and Celso Massaki Hirata. Extending timestamp-based two phase commit protocol for restful services to meet business rules. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 778–785, New York, NY, USA, 2011. ACM.

[45] Luiz Alexandre Hiane da Silva Maciel and Celso Massaki Hirata. Fault-tolerant timestamp-based two-phase commit protocol for restful services. *Software: Practice and Experience*, 43(12):1459–1488, 2013.

[46] Gordon Bitter Davis. *Management Information Systems: Conceptual Foundations, Structure and Development*. McGraw-Hill, Inc., New York, NY, USA, 1974.

[47] DB-Engines. Db-engines ranking - popularity ranking of database management systems. `https://db-engines.com/en/ranking/`, 2016. Accessed April 2016.

[48] A. Dey, A. Fekete, and U. Röhm. Rest+t: Scalable transactions over http. In *2015 IEEE International Conference on Cloud Engineering*, pages 36–41, March 2015.

[49] Business Dictionary. What is information system? definition and meaning - businessdictionary.com. `http://www.businessdictionary.com/definition/information-system.html`. Accessed May 2018.

[50] Merriam-Webster Dictionary. Facet | definition of facet by merriam-webster. `https://www.merriam-webster.com/dictionary/facet`. Accessed March 2019.

[51] The Free Dictionary. Facet - definition of facet by the free dictionary. `https://www.thefreedictionary.com/facet`. Accessed March 2019.

[52] Martin Doerr and Maria Theodoridou. Crmdig an extension of cidoc-crm to support provenance metadata. Technical report, Heraklion: ICS-FORTH, 2014.

[53] Lisa M. Dusseault. HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). RFC 4918, jun 2007.

[54] Jonas Eckhardt, Andreas Vogelsang, and Daniel Mèndez Fernández. Are "non-functional" requirements really non-functional? an investigation of non-functional requirements in practice. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 832–842, May 2016.

[55] Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson. Toward an open cloud standard. *IEEE Internet Computing*, 16(4):15–25, 2012.

[56] Ahmed K. Elmagarmid. A survey of distributed deadlock detection algorithms. *SIGMOD Rec.*, 15(3):37–45, sep 1986.

[57] Eckhard Falkenberg, Wolfgang Hesse, Paul Lindgreen, Björn E Nilsson, JL Han Oei, Colette Rolland, Ronald K Stamper, Frans JM Van Assche, AA Verrijn-Stuart, and Klaus Voss. Frisco-a framework of information system concepts-the frisco report. Technical report, ISBN 3-901-88201-4, 1998.

[58] David Ferraiolo, , and Richard Kuhn. Role-based access controls. In *15th National Computer Security Conference (NCSC)*, pages 554–563, 1992.

[59] Laurence Field, Maria Alandes, and Alessandro Di Girolamo. Towards a global service registry for the world-wide lhc computing grid. *Journal of Physics: Conference Series*, 513(3):032032, 2014.

[60] Laurence Field, Paul Harvey, and Tim Dyce. Designing the next generation grid information system. *Journal of Physics: Conference Series*, 331(6):062008, 2011.

[61] Laurence Field, Erwin Laure, and Markus W. Schulz. Grid deployment experiences: Grid interoperation. *J. Grid Comput.*, 7(3):287–296, 2009.

[62] Laurence Field and Markus W Schulz. Grid deployment experiences: The path to a production quality ldap based grid information system. 2005.

[63] Laurence Field and Markus W Schulz. An investigation into the mutability of information in production grid information systems. In *Journal of Physics: Conference Series*, volume 219, page 062046. IOP Publishing, 2010.

[64] Roy T. Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, jun 2014.

[65] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. AAI9980887.

[66] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[67] Luca Frosini. gcube information system model, Apr 2018.

[68] Luca Frosini. gcube information system sweeper, Apr 2018.

[69] Luca Frosini. gcube model, Apr 2018.

[70] Luca Frosini. gcube resource registry, Apr 2018.

[71] Luca Frosini. gcube resource registry api, Apr 2018.

[72] Luca Frosini. gcube resource registry client, Apr 2018.

[73] Luca Frosini. gcube resource registry context client, Apr 2018.

[74] Luca Frosini. gcube resource registry orientdb hooks, Apr 2018.

[75] Luca Frosini. gcube resource registry publisher, Apr 2018.

[76] Luca Frosini, Alessia Bardi, Paolo Manghi, and Pasquale Pagano. An aggregation framework for digital humanities infrastructures: The parthenos experience. *SCIRES-IT - SCIentific RESearch and Information Technology*, 8(1):33–50, jul 2018.

[77] Luca Frosini and Pasquale Pagano. A facet-based open and extensible resource model for research data infrastructures. *Grey Journal*, 14(2):69–76, 2018.

[78] Luca Frosini and Pasquale Pagano. A facet-based open and extensible resource model for research data infrastructures. In *19th International Conference on Grey Literature – Public Awareness and Access to Grey Literature*, volume 2017-October, pages 113–120. TextRelease, 2018.

[79] Maria Gavrilidou, Penny Labropoulou, Elina Desipri, Stelios Piperidis, Harris Papageorgiou, Monica Monachini, Francesca Frontini, Thierry Declerck, Gil Francopoulo, Victoria Arranz, et al. The meta-share metadata schema for the description of language resources. In *8th International Conference on Language Resources and Evaluation (LREC 2012)*, pages 1090–1097. European Language Resources Association (ELRA), 2012.

[80] gCube Team. Reference model - gcube wiki. `https://wiki.gcube-system.org/gcube/Reference_Model`. Accessed May 2018.

[81] Fosca Giannotti, Roberto Trasarti, Kalina Bontcheva, and Valerio Grossi. Sobigdata: Social mining & big data ecosystem. In *Companion of the The Web Conference 2018 on The Web Conference 2018*, pages 437–438. International World Wide Web Conferences Steering Committee, 2018.

[82] Gordon Good. The LDAP Data Interchange Format (LDIF) - Technical Specification. RFC 2849, jun 2000.

[83] Rudy Hirschheim, Heinz K Klein, and Kalle Lyytinen. *Information systems development and data modeling: conceptual and philosophical foundations*. Cambridge University Press, 1995.

[84] Vincent C Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (abac) definition and considerations. Technical Report 162, NIST, jan 2014.

[85] Technical interoperability Technical Committee. Iso 21127:2014 a reference ontology for the interchange of cultural heritage information, oct 2014.

[86] data elements ISO/TC 154 Processes, industry documents in commerce, and administration. Iso 17369:2013 statistical data and metadata exchange (sdmx), jan 2013.

[87] JISC. Virtual research environment programme. `http://www.jisc.ac.uk/whatwedo/programmes/vre.aspx`, 2014. Now archived and available at `http://webarchive.nationalarchives.gov.uk/20140702163345/http://www.jisc.ac.uk/whatwedo/programmes/vre.aspx` Accessed May 2018.

[88] Zoi Kaoudi and Ioana Manolescu. Rdf in the clouds: A survey. *The VLDB Journal*, 24(1):67–91, February 2015.

[89] Rohit Khare. *Extending the Representational State Transfer (Rest) Architectural Style for Decentralized Systems*. PhD thesis, University of California, Irvine, 2003. AAI3109801.

[90] Rohit Khare and Richard N. Taylor. Extending the representational state transfer (rest) architectural style for decentralized systems. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 428–437, Washington, DC, USA, 2004. IEEE Computer Society.

[91] Sebastian Kochman, Paweł T. Wojciechowski, and Miłosz Kmieciak. Batched transactions for restful web services. In *Proceedings of the 11th International Conference on Current Trends in Web Engineering*, ICWE'11, pages 86–98, Berlin, Heidelberg, 2012. Springer-Verlag.

[92] Carl Lagoze and Herbert Van de Sompel. The open archives initiative: Building a low-barrier interoperability framework. In *Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 54–62. ACM, 2001.

[93] Alexandros Marinos, Amir Razavi, Sotiris Moschoyiannis, and Paul Krause. Retro: A consistent and recoverable restful transaction model. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 181–188, Washington, DC, USA, 2009. IEEE Computer Society.

[94] James Martin. *Managing the Database Environment*. Prentice-Hall, 1983.

[95] Carlo Meghini, Roberto Scopigno, Julian Richards, Holly Wright, Guntram Geser, Sebastian Cuy, Johan Fihn, Bruno Fanini, Hella Hollander, Franco Niccolucci, Achille Felicetti, Paola Ronzino, Federico Nurra, Christos Papatheodorou, Dimitris Gavrilis, Maria Theodoridou, Martin Doerr, Douglas Tudhope, Ceri Binding, and Andreas Vlachidis. Ariadne: A research infrastructure for archaeology. *J. Comput. Cult. Herit.*, 10(3):18:1–18:27, August 2017.

[96] David Meredith. Gocdb e-infrastructure information system. `https://wiki.egi.eu/w/images/d/d3/GOCDB5_Grid_Topology_Information_System.pdf`, 2016. Accessed May 2018.

[97] Nandana Mihindukulasooriya, Miguel Esteban-Gutiérrez, and Raúl García-Castro. Seven challenges for restful transaction models. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 949–952, New York, NY, USA, 2014. ACM.

[98] Nandana Mihindukulasooriya, Raúl García-Castro, Miguel Esteban-Gutiérrez, and Asunción Gómez-Pérez. A survey of restful transaction models: One model does not fit all. *J. Web Eng.*, 15(1-2):130–169, mar 2016.

[99] Vijay Nanjundaswamy, Kent Boogert, and Bruce Bergeson. Lightweight Directory Access Protocol (LDAP) Schema for Universal Description, Discovery, and Integration version 3 (UDDIv3). RFC 4403, feb 2006.

[100] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Jan Peleska. Systems of systems engineering: Basic concepts, model-based techniques, and research directions. *ACM Computing Surveys (CSUR)*, 48(2):18:1–18:41, sep 2015.

[101] Henrik Frystyk Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, jun 1999.

[102] Joseph D Novak and D Bob Gowin. *Learning how to learn*. Cambridge University Press, 1984.

[103] JSON Schema Organisation. Json schema. `http://json-schema.org/`. Accessed June 2018.

[104] Guy Pardon and Cesare Pautasso. Towards distributed atomic transactions over restful services. In *REST: From Research to Practice*, pages 507–524. Springer New York, New York, NY, 2011.

[105] Guy Pardon and Cesare Pautasso. Atomic distributed transactions: A restful design. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 943–948, New York, NY, USA, 2014. ACM.

[106] Serena Pastore. The service discovery methods issue: A web services uddi specification framework integrated in a grid environment. *Journal of Network and Computer Applications*, 31(2):93 – 107, 2008. Trends in peer to peer and service oriented computing.

[107] Benjamin C. Pierce. *Types and programming languages*. MIT press, 2002.

[108] Amir Razavi, Alexandros Marinos, Sotiris Moschoyiannis, and Paul Krause. Restful transactions supported by the isolation theorems. In *Proceedings of the 9th International Conference on Web Engineering*, ICWE '9, pages 394–409, Berlin, Heidelberg, 2009. Springer-Verlag.

[109] Leonard Richardson and Sam Ruby. *Restful Web Services*. O'Reilly, first edition, 2007.

[110] Nicolas Schiper and Sam Toueg. A robust and lightweight stable leader election service for dynamic systems. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 207–216. IEEE, 2008.

[111] Jim Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol. RFC 4511, jun 2006.

[112] Alex Sherman, Philip A. Lisiecki, Andy Berkheimer, and Joel Wein. Acms: The akamai configuration management system. In *in the 2nd Symposium on Networked Systems Design & Implementation (USENIX)*, 2005.

[113] Dave Snowdon, Elizabeth F. Churchill, and Alan J. Munro. Collaborative virtual environments: Digital spaces and places for cscw: An introduction. In *Collaborative Virtual Environments: Digital Places and Spaces for Interaction*, pages 3–17. Springer London, London, 2001.

[114] Claudia Soria, Núria Bel, Khalid Choukri, Joseph Mariani, Monica Monachini, JEJM Odijk, Stelios Piperidis, Valeria Quochi, and Nicoletta Calzolari. The flarenet strategic language resource agenda. In *8th International Conference on Language Resources and Evaluation (LREC 2012)*, pages 1379–1386. European Language Resources Association (ELRA), 2012.

[115] Specification TC. Uddi version 3 specification, 2005.

[116] Lilian van der Vaart. Collaboratories: Connecting researchers : how to facilitate choice, design and uptake of online research collaborations. Technical report, SURFfoundation, apr 2010.

[117] Tamás Váradi, Peter Wittenburg, Steven Krauwer, Martin Wynne, and Kimmo Koskenniemi. Clarin: Common language resources and technology infrastructure. In *6th International Conference on Language Resources and Evaluation (LREC 2008)*, 2008.

[118] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, jan 2009.

[119] XSEDE. Science gateways. `https://www.xsede.org/ecosystem/science-gateways`. Accessed May 2018.

[120] Hsi-En Yu and Weicheng Huang. Building a virtual hpc cluster with auto scaling by the docker. *CoRR*, abs/1509.08231, 2015.

APPENDIX *A*

Types Definitions Examples

Listing A.1: *Entity type representation*

```
{
    "name": "Entity",
    "description": "Entity Base Type",
    "superClasses": null,
    "properties": [
        {
            "name": "header",
            "type": "Header",
            "description": "Automatically generated for the sake of
                identification and provenance of the specific information",
            "mandatory": true,
            "readonly": false,
            "notnull": true,
            "max": null,
            "min": null,
            "regexp": null
        }
    ],
    "abstract": true
}
```

Listing A.1 shows the representation of the base type for any entity. The type is abstract cause it cannot be instantiated. It does not have any super-types and contains the header property which is inherited by all sub-types. *Entity* is the base type for *Resource* and *Facet* types.

Listing A.2: Resource *type representation*

```
{
    "name": "Resource",
    "description": "Entity representing a description of a ’thing’ to be
        managed",
    "superClasses": [
        "Entity"
    ],
    "facets": [
        {
            "source" : "Resource",
            "relation" : "ConsistsOf",
            "target" : "Facet",
            "description" : "A relation linking a Facet contributing to '
                build' a description of such a Resource",
            "max" : null,
            "min" : 1
        }
    ],
    "resources" : [
        {
            "source" : "Resource",
            "relation" : "IsRelatedTo",
            "target" : "Resource",
            "description" : "A relation linking any two Resources",
            "max" : null,
            "min" : null
        }
    ],
    "abstract": true
}
```

Listing A.2 shows the representation of the base type for any resource. *Resource* type is abstract cause it cannot be instantiated. It has *Entity* as supertype from which inherits the header attribute. The definition shows that this type when instantiated (by instantiating any concrete subtype) requires at least a *Facet* related by a *ConsistsOf* relation. The definition suggests that a resource can be related to any other resource.

This type cannot be created or deleted by any client because it is one of the predefined types.

Examples of resource types definition are shown by listings A.3, A.4 and A.5. A *Service* (see A.3) is an abstract type and it does not impose any *IsIdentifiedBy* relation.

Listing A.3: *Example of resource type representation i.e., Service*

```
{
    "name": "Service",
    "description": "Represents any typology of Service worth registering in
        the infrastructure",
    "superClasses": [
        "Resource"
    ],
    "facets": [
        {
            "source" : "Service",
            "relation" : "ConsistsOf",
            "target" : "DescriptiveMetadataFacet",
            "description" : "Any descriptive information associated with the
                 service, e.g. for discovery purposes.",
            "max" : null,
            "min" : 0
        },
        {
            "source" : "Service",
            "relation" : "ConsistsOf",
            "target" : "SubjectFacet",
            "description" : "Any subject/tag associated with the service for
                 descriptive, cataloguing and discovery purposes.",
            "max" : null,
            "min" : 0
        },
        {
            "source" : "Service",
            "relation" : "HasCapability",
            "target" : "CapabilityFacet",
            "description" : "Any defined facility for performing a specified
                 task supported/offered by a given Service",
            "max" : null,
            "min" : 0
        }
    ],
    "resources" : [
        {
            "source" : "Service",
            "relation" : "CallsFor",
            "target" : "Service",
            "description" : "A reference to an invoked Services.",
            "max" : null,
            "min" : 0
        },
        {
            "source" : "Service",
            "relation" : "Activates",
            "target" : "Service",
            "description" : "A reference to an activated Service.",
            "max" : null,
            "min" : 0
        },
        {
            "source" : "Service",
            "relation" : "IsCustomizedBy",
            "target" : "ConfigurationTemplate",
            "description" : "A reference to any configuration characterising
                 the Service behaviour.",
            "max" : null,
            "min" : 0
        },
        {
            "source" : "Service",
            "relation" : "Manages",
            "target" : "Dataset",
            "description" : "A reference to any Dataset resource managed by
                the Service instance.",
            "max" : null,
            "min" : 0
        }
    ],
    "abstract": true
}
```

EService (see A.4) is a concrete type extending Service. It imposes to have a *SoftwareFacet* related by the IsIdentifiedBy relation. again this constraint already satisfies the constraints imposed by the ancestor (both *Service* and *Resource*). Additionally EService require at least an *AccessPointFacet* , an *EventFacet* . Moreover requires having one and only one *StateFacet* , i.e., the *EService* can be only in one state. The schema suggests it can have other *SoftwareFacet*, apart from the one used for sake of identification. A *LicenceFacet* is also suggested. All related resources are a suggestion for the developer.

Listing A.4: *Example of resource type representation i.e., EService*

```
{
    "name": "EService",
    "description": "Collect Electronic Service (aka Running Service)
        information through the list of its facets",
    "superClasses": [
        "Service"
    ],
    "facets": [
        {
            "source" : "EService",
            "relation" : "IsIdentifiedBy",
            "target" : "SoftwareFacet",
            "description" : "The main software enabling the EService
                capabilities.",
            "max" : null,
            "min" : 1
        },
        {
            "source" : "EService",
            "relation" : "ConsistsOf",
            "target" : "SoftwareFacet",
            "description" : "Software available in the EService environment
                that characterises the specific EService.",
            "max" : null,
            "min" : 0
        },
        {
            "source" : "EService",
            "relation" : "ConsistsOf",
            "target" : "AccessPointFacet",
            "description" : "Identify an endpoints of the EService.",
            "max" : null,
            "min" : 1
        },
        {
            "source" : "EService",
            "relation" : "ConsistsOf",
            "target" : "EventFacet",
            "description" : "Captures information on a certain event/
                happening characterising the current status and the life
                cycle of the EService events.",
            "max" : null,
            "min" : 1
        },
        {
            "source" : "EService",
            "relation" : "ConsistsOf",
            "target" : "StateFacet",
            "description" : "captures information on the current operational
                 state of the EService.",
            "max" : 1,
            "min" : 1
        },
        {
            "source" : "EService",
            "relation" : "ConsistsOf",
            "target" : "LicenseFacet",
            "description" : "Captures information on any license associated
                with the EService to capture the policies governing its
                exploitation and use.",
            "max" : null,
            "min" : 0
        }
    ],
    "resources" : [
        {
            "source" : "EService",
            "relation" : "Discovers",
            "target" : "EService",
            "description" : "The source EService discovers the target
                through the information system.",
            "max" : null,
            "min" : 0
        },
        {
            "source" : "EService",
            "relation" : "Uses",
            "target" : "EService",
            "description" : "The source EService invokes the target EService
                .",
            "max" : null,
            "min" : 0
        }
    ],
    "abstract": false
}
```

RunningPlugin (see A.5) is a concrete type extending *EService*. The mandatory *SoftwareFacet* related by the IsIdentifiedBy relation is the same constraint defined for *EService*. The resource should have at least a relation *IsActivatedBy* with an *EService*, i.e., the *EService* activating the plugin and possibly a relation with the *Software* resource representing the *RunningPlugin*.

Listing A.5: *Example of resource type representation i.e.,* RunningPlugin

```
{
    "name": "RunningPlugin",
    "description": "Collect Electronic Service (aka Running Service)
        information through the list of its facets",
    "superClasses": [
        "Service"
    ],
    "facets": [
        {
            "source" : "RunningPlugin",
            "relation" : "IsIdentifiedBy",
            "target" : "SoftwareFacet",
            "description" : "The software enabling the Running Plugin",
            "max" : null,
            "min" : 1
        }
    ],
    "resources" : [],
    "abstract": false
}
```

Listing A.6 shows the representation of the base class for any *Facet* type. The type is abstract because it cannot be instantiated. It has *Entity* as supertype from which inherits the header attribute.

Listing A.6: Facet *type representation*

```
{
    "name": "Facet",
    "description": "Entity contributing to 'build' a description of a
        Resource",
    "superClasses": [
        "Entity"
    ],
    "properties": [],
    "abstract": true
}
```

Listing A.7 shows an example of *Facet* specialisation. This facet type has three mandatory properties i.e., name, group and version and three other optional properties i.e., description, qualifier, optional. All the attributes are Strings except the one named 'optional' which is a Boolean.

Listing A.7: *Example of facet type representation i.e., SoftwareFacet*

```
{
    "name": "SoftwareFacet",
    "description": "This facet captures information on any software
        associated with the resource.",
    "superClasses": [
        "Facet"
    ],
    "properties": [
        {
            "name": "name",
            "type": String,
            "description": "The name of the software artifact being
                described.",
            "mandatory": true,
            "readonly": false,
            "notnull": false,
            "max": null,
            "min": null,
            "regexp": null
        },
        {
            "name": "group",
            "type": String,
            "description": "The name of \"group\" the software artifact
                belongs to.",
            "mandatory": true,
            "readonly": false,
            "notnull": false,
            "max": null,
            "min": null,
            "regexp": null
        },
        {
            "name": "version",
            "type": String,
            "description": "The particular release of the software artifact.
                ",
            "mandatory": true,
            "readonly": false,
            "notnull": false,
            "max": null,
            "min": null,
            "regexp": null
        },
        {
            "name": "description",
            "type": String,
            "description": "A human oriented description of the software
               artifact being described.",
            "mandatory": false,
            "readonly": false,
            "notnull": false,
            "max": null,
            "min": null,
            "regexp": null
        },
        {
            "name": "qualifier",
            "type": String,
            "description": "A qualifier for the software.",
            "mandatory": false,
            "readonly": false,
            "notnull": false,
            "max": null,
            "min": null,
            "regexp": null
        },
        {
            "name": "optional",
            "type": Boolean,
            "description": "Used to indicate the software optionality.",
            "mandatory": false,
            "readonly": false,
            "notnull": false,
            "max": null,
            "min": null,
            "regexp": null
        }
    ],
    "abstract": false
}
```

Listing A.8 shows the representation of the base class for any Relation. The type is abstract, so it cannot be instantiated. It does not have any super-types and contains the *Header* and *PropagationConstraint* properties which are inherited by all sub-types. Any relations contains source and target properties which indicate the source and target entity types for the relation. This type is the base class for *IsRelatedTo* and *ConsistsOf* relations.

Listing A.8: *Relation type representation*

```
{
    "name": "Relation",
    "description": "Relation Base Type",
    "superClasses": null,
    "properties": [
        {
            "name": "header",
            "type": "Header",
            "description": "Automatically generated for the sake of
               identification and provenance of the specific information",
            "mandatory": true,
            "readonly": false,
            "notnull": true,
            "max": null,
            "min": null,
            "regexp": null
        },
        {
            "name": "propagationConstraint",
            "type": "PropagationConstraint",
            "description": "Indicates the behaviour to be held on a target
                entity when an event related to a context occurs in the
                source resource.",
            "mandatory": true,
            "readonly": false,
            "notnull": true,
            "max": null,
            "min": null,
            "regexp": null
        }
    ],
    "source" : "Resource",
    "target" : "Entity",
    "abstract": true
}
```

Listing A.9 shows the representation of the base class for any *IsRelatedTo* relation type. It is defined as a relation between two *Resources* and is abstract.

Listing A.9: IsRelatedTo *type representation*

```
{
    "name": "IsRelatedTo",
    "description": "A relation linking any two Resources",
    "superClasses": [
        "Relation"
    ],
    "properties": [],
    "source" : "Resource",
    "target" : "Resource",
    "abstract": true
}
```

Listing A.10 shows the representation of Hosts relation type. It is a concrete type defined as a relation between a *Site* and a *Service*.

Listing A.10: *Example of* IsRelatedTo *type representation i.e.,* Hosts

```
{
    "name": "Hosts",
    "description": "Indicated the Service provided by the Site",
    "superClasses": [
        "IsRelatedTo"
    ],
    "properties": [],
    "source" : "Site",
    "target" : "Service",
    "abstract": false
}
```

Listing A.11 shows the representation of the base class for any *ConsistsOf* relation type. It is a concrete type defined as a relation between a *Resource* and a *Facet*.

Listing A.11: ConsistsOf *type representation*

```
{
    "name": "ConsistsOf",
    "description": "A relation linking any a Resource to a Facet
        contributing to 'build' a description of such a Resource",
    "superClasses": [
        "Relation"
    ],
    "properties": [],
    "source" : "Resource",
    "target" : "Facet",
    "abstract": false
}
```

Listing A.12 shows the representation of *HasOwner* relation type. It is defined as a relation between a *Resource* and a *ContactFacet*. It has as super-type *HasContact* (not shown in examples).

Listing A.12: *Example of a* ConsistsOf *type representation i.e., HasOwner*

```
{
    "name": "HasOwner",
    "description": "The target ContactFacet indicates the information
        related to the owner of the source Resource",
    "superClasses": [
        "HasContact"
    ],
    "properties": [],
    "source" : "Resource",
    "target" : "ContactFacet",
    "abstract": false
}
```