



The integration of earthquake engineering resources

A thesis submitted to the

Department of Engineering Science
UNIVERSITY OF OXFORD

for the degree

Doctor of Philosophy

Ignacio Lamata Martínez
New College

Trinity term, 2014

Abstract

The integration of earthquake engineering resources

A thesis submitted for the degree of
Doctor of Philosophy

Ignacio Lamata Martínez
New College, University of Oxford
Trinity term 2014

Earthquake engineering is increasingly focusing on large international collaborations to address complex problems. Recent computing advances have greatly contributed to the way scientific collaborations are conducted, where web-based solutions are an emerging trend to manage and present results to the scientific community and the general public. However, collaborations in earthquake engineering lack a common interoperability framework, resulting in tedious and complex processes to integrate results, which cannot be efficiently used by third-party institutions.

The work described in this thesis applies novel computing techniques to enable the interoperability of earthquake engineering resources, by integrating data, distributed simulation services and laboratory facilities. This integration focuses on distributed approaches rather than centralised solutions, and has been materialised in a platform called *Celestina*, that supports the integration of hazard mitigation resources.

The prototype of *Celestina* has been implemented and validated within the context of two of the current largest earthquake engineering networks, the SERIES network in Europe and the NEES network in the USA. It has been divided into three sub-systems to address different problems: (i) *Celestina Data*, to develop best methods to define, store, integrate and share earthquake engineering experimental data. *Celestina Data* uses a novel approach based on Semantic Web technologies, and it has accomplished the first data integration between earthquake engineering institutions from the United States and Europe by means of a formalised infrastructure. (ii) *Celestina Tools*, to research applications that can be implemented on top of the data integration, in order to provide a practical benefit for the end user. (iii) *Celestina Simulations*, to create the most efficient methods to integrate distributed testing software and to support the planning, definition and execution of the experimental workflow from a high-level perspective. *Celestina Simulations* has been implemented and validated by conducting distributed simulations between the Universities of Oxford and Kassel. Such validation has demonstrated the feasibility to conduct both flexible, general-purpose and high performance simulations under the framework.

Celestina has enabled global analysis of data requirements for the whole community, the definition of global policies for data authorship, curation and preservation, more efficient use of efforts and funding, more accurate decision support systems and more efficient sharing and evaluation of data results in scientific articles.

In memory of Manuel Martínez Balaguer

Acknowledgments

I am extremely thankful to my supervisors, **Prof. Martin Williams** and **Prof. Anthony Blakeborough**. In my life I have been fortunate to work with some terrible supervisors, which made me know what a good supervisor looks like. Martin and Tony have been beyond terrific, they have not only provided me with great support and advice, they have taught me engineering concepts, and sometimes made me think twice and pay attention to small yet crucial details. They have always been very supportive on the research directions I decided to follow and, in essence, have made it very difficult for my next supervisors to come.

I would also like to thank my additional supervisor, **Dr. Markus Krötzsch**, whose meticulous and detailed feedback has contributed to improve substantially the quality of the thesis. I have been privileged to get advice from one of the experts in some of the topics of this thesis.

This thesis is all about collaborations. Because of this, I had the opportunity to work with many diverse people in two of the largest (and best) earthquake engineering networks, at Europe (SERIES) and the United States (NEES). On the SERIES side, I would like to thank **Dr. Anna Bosi**, **Dr. Pierre Pegon** and **Fabio Taucer** (JRC), **Ioannis Ioannidis**, **Prof. Stathis Bousias**, **Antony Chazapis** and **Dr. Dionysis Biskinis** (University of Patras), **Ferrán Obón Santacana** (University of Kassel), **Dr. Francesco Lunghi** and **Davide Silvestri** (EUCENTRE), **Dr. Jean-Louis Chazelas** (IFSTTAR), **Gheorghe Telimbeci** (“Gheorghe Asachi” Technical University of Iași), **Prof. Oreste Bursi** (University of Trento), **Dr. Vlatko Sesov** and **Prof. Zoran Rakicevic** (IZIIS), **Dr. Alain Le Maoult** (CEA), **Dr. Gian Piero Lignola** and **Prof. Andrea Prota** (Università degli Studi di Napoli Federico II). I would like to make special mention to Dr. Pegon and Dr. Bosi for their support in the development and testing of the SVDB - which involved thousands of emails, for their tireless help answering my questions, and for teaching me a great part of what I know of earthquake engineering. Additional thanks to Giannis Ioannidis for the SVDB testing, Ferrán Obón for the Celestina Simulations and RT-DHT testing and Dr. Lunghi for the Celestina Data testing.

On the NEES side, I would like to thank the personnel at Purdue University, **Prof. Shirley Dyke**, **Stanislav Pejša**, **Gemez Marshall**, **Brian Rohler**, **Limin Dong**, **Richard White**, **Prof. Julio Ramírez**, **Barbara Fossum**, **Dr. Tom Hacker**, **Gaby Ou**, **Cheng Song**, **Marti LaChance**, **Peggy MacNorton** and **Keith Adams**. I would particularly like to thank Barbara Fossum for hours of enriching conversation, her hospitality and her friendship; Dr. Ramírez for envisioning and bringing me the opportunity to formalise my collaboration with NEES. Prof. Dyke for her continuous support during the development of Celestina Data, including multiple inspiring meetings; Standa, for his support with the NEES data model (history lessons included) and his sincere friendship; Gemez for his assistance with NEES systems; Lily for her support developing the NEES side of the Celestina Data application; Brian and Richie for all the operational support that was needed to complete the work. They constitute a very professional team.

Part of the work presented had its origin in the UK-NEES project. I would like to thank **Dr. Mobin Ojaghi** for all the user requirements and useful ideas he brought into the DHT versions prior RT-DHT, his lessons about earthquake engineering and his true friendship. Also, thanks to Dr. Ojaghi and **Dr. Matt Dietz** for the testing of DHT.

I am grateful to **Dr. Matt Dietz** for the proof reading and formidable feedback on an early version of this thesis, and to **Dr. Roberto Rubio** for his suggestions on the appendix about RSA.

At the computing services of the Engineering Department, I would like to thank **Tony Gilham** and **Chris Flux** for providing me with support on IT issues such as departmental network management and software licences. I am especially thankful to Tony for facilitating the hardware I needed to complete the work for this thesis.

I would like to thank **Dr. Bernardo Cuenca Grau**, **Graham Klyne** and **David Wallom** for their time, giving me slight orientation in some aspects at the beginning of my research.

I would like to express my gratitude to **Dr. Javier Parra Fuente** for bringing me the luck to meet opportunity and preparation, and for his initial support with DHT. I am thankful to **Wiebke Lotte Albers** for her encouragement at the early stages of my studies, and to **Juan Antonio Blanco** for his support prior to starting my programme.

I must thank the European Commission, the NSF and the EPSRC, which provided the funding required to complete this work.

At the Engineering Department, I would like to express my gratitude, for different reasons, to **Dr. Suresh Dash**, **Dr. Giacomo Perantoni**, **Dr. Sena Serhadlıoğlu**, **Dr. Tao Zhao**, **Clive Baker**, **Alison May**, **Chen-Ket Chai**, **Dr. Ross McAdam**, **Alejandra Albuérne**, **Dr. Elizabeth Williams**, **Dr. Richard Sandford**, **Daniel Claff**, **Robin Malloy**, **Dr. Conor Fleming** and **Sukarni Wheeler**.

I would like to thank some of my friends for the good times in Oxford, in particular (with no specific order): my old friend, **Carlos Gutierrez David**; the Spanish team: **Dina**, **Jaco**, **Marga** and **Maia**; the Egyptian team: **Tony**, **Mowafy**, **Zaki**, **Arto** and **Khaled**; the amazing *Abbey* team: **Ángel**, **José Borja**, **Mendieta**, **Jara** and **Roberto**; my three girls: **Vanessa**, **Cova** and **Mireia**; my dearest couple, full of excellent conversation, **Chris Holmsten** and **Joselyn**; the great comedian and friend, **Roberto Rubio**; the Mexican fresa, **Neo**; the Spanish teachers: **Esteban**, **Leo**, **Rosa** and **Pilar**; the Italian team of New College: **Giacomo Dell**, **Sam Luigi**, **Mobin Silvio** and **Genia Priscilla**; the kind **Sonia** and **Juan Carlos**; my most friendly entrepreneur advisor, **Placi**.

I am sincerely thankful to **Monica** for her care, her continuous support and her (sometimes extraordinary) understanding while I have been pursuing my objectives, and to my sister, for her unconditional love and cheerful encouragement.

As a child, I was fascinated with any expression of visual art, but I was discouraged to follow an artistic career professionally. At fifteen, I found computers as a mean to create 3D graphs, which somehow led me to explore the internals of my father's computer in an unusual way. This did not make my dad especially happy, who never understood why I did such things and thought I was going to break the computer. But thanks to those experiments, two years later I successfully reverse engineered and modified my first program by using assembly. Once, I was told a story of a crazy man in a village who threw stones to the sky, believing he could reach the moon. Day after day, he threw thousands of stones. Nobody in the village understood why he did such things. Everyone laughed at him, thinking how insane he was keeping his impossible belief... until there was a stone throwing competition in the village. Sometimes, we follow dreams that make us do things that might look absurd, but they may become useful one day. I would like to thank my parents for their enormous support and understanding during all these years, including their assistance to direct my future to meet my dreams.

Publications

The work presented in this thesis has contributed to the following papers:

Lamata Martínez I., Ioannidis I., Fidas C., Williams M., Pegon P. (2015) *The SERIES Virtual Database: Architecture and Implementation*. Geotechnical, Geological and Earthquake Engineering 35 (Ed. Taucer F., Apostolska R.), Springer.

Bosi A., Kotinas I., Lamata Martínez I., Bousias S., Chazelas J. L., Dietz M., Hasan M. R., Madabhushi S. P. G., Prota A., Blakeborough T., Pegon P. (2015) *The SERIES Virtual Database: Exchange Data Format and local/central databases*. Geotechnical, Geological and Earthquake Engineering 35 (Ed. Taucer F., Apostolska R.), Springer.

Lamata Martínez I., Obón Santacana F., Williams M. S., Blakeborough A., Dorka U. E. (2015) *A Support Platform for Distributed Hybrid Testing*. Geotechnical, Geological and Earthquake Engineering 35 (Ed. Taucer F., Apostolska R.), Springer.

Lamata Martínez I., Obón Santacana F., Williams M. S., Blakeborough A., Dorka U. E. (2014) *Celestina-Sim: A Framework to Support Distributed Testing and Service Integration in Earthquake Engineering*. J. Comput. Civ. Eng., 10.1061/(ASCE)CP.1943-5487.0000455, 04014119.

Ojaghi M., Williams M. S., Dietz M. S., Blakeborough A., Lamata Martínez I. (2013) *Real-time distributed hybrid testing: coupling geographically distributed scientific equipment across the Internet to extend seismic testing capabilities*. Earthquake Engineering & Structural Dynamics. DOI: 10.1002/eqe.2385.

Lamata Martínez I., Ioannidis I., Pegon P., Williams M., Blakeborough A. (2013) *The Process and Future of Data Integration within the European Earthquake Engineering Laboratories*. J. Comput. Civ. Eng., 10.1061/(ASCE)CP.1943-5487.0000308

Ojaghi M., Dietz M., Lamata Martínez I., Williams M.S., Blakeborough A. (2012) *Current Capabilities and Future Roles for Internet Distributed Large Scale Real-Time Seismic Testing*. In Proc. The 15th World Conf. on Earthquake Engineering, Lisbon, Portugal, 24–28 September 2012, Paper 1328.

de la Flor G., Ojaghi M., Lamata Martínez I., Jirotko M., Williams M. S., Blakeborough A. (2010) *Reconfiguring practice: the interdependence of experimental procedure and computing infrastructure in distributed earthquake engineering*. Philosophical Transactions of the Royal Society A, 368, 4073-4088.

Ojaghi M., Lamata Martínez I., Dietz M., Williams M.S., Blakeborough A., Crewe A. (2010) *Real-time hybrid testing in geographically distributed laboratories*. 14th European Conference on Earthquake Engineering, Ohrid, Macedonia, Paper 872.

Madhabushi S.P.G., Haigh S.K., Ali M., Williams M. S., Ojaghi M., Lamata I., Blakeborough A., Taylor C.A., Dietz M. (2010) *Distributed testing of soil-structure systems using web-based applications*. 7th International Conference on Physical Modelling in Geotechnics, Zurich.

Ojaghi M., Lamata Martínez I., Dietz M., Williams M.S., Blakeborough A., Crewe A., Ali A., Haigh S., Madhabushi S.P.G. (2010) *UKNEES - Distributed hybrid testing between Bristol, Cambridge and Oxford Universities*. 9th US/10th Canadian Conference on Earthquake Engineering, Toronto, Paper 1024.

de la Flor G., Ojaghi M., Lamata I., Jirotko M., Williams M. S., Blakeborough A. (2009) *Reconfiguring practice: the interdependence of experimental procedure and computing infrastructure in distributed earthquake engineering*. UK e-Science All Hands Meeting, Oxford (Winner of Tony Hey Award for e-Science).

Contents

1	Introduction	1
1.1	Objective, scope and research context	2
1.2	Contribution	3
1.3	Celestina	4
1.4	Terminology	6
1.5	Thesis layout	6
2	Literature review	8
2.1	Earthquake engineering databases.	9
2.2	Recent advances on database systems	11
2.3	Earthquake engineering software for distributed simulations.	13
2.4	Real-time and quality of service on the Internet	15
2.5	Conclusions	18
3	Foundations of experimental data	19
3.1	Experimental data structure overview	20
3.2	The SERIES database.	22
3.2.1	The relational model	23
3.2.2	SERIES data model overview.	25
3.2.3	Validation	27
3.3	Conclusions	29
4	Data integration in the European laboratories	30
4.1	Basic requirements	31
4.2	Architecture overview	32
4.2.1	Constituents	33
4.2.2	General architecture	33
4.2.3	SVDB nodes.	36
4.2.4	SVDB central site	37
4.2.5	Security considerations.	37
4.3	Implementation	39
4.3.1	SVDB node services	40
4.3.2	Large file download management	42
4.4	Validation.	43
4.4.1	How to update	44
4.4.2	Who should initiate the update.	45
4.4.3	When to perform the update	45
4.4.4	Potential updating issues	45
4.5	Results.	46
4.5.1	Implications of large projects	47
4.5.2	Management of distributed components	48
4.5.3	Importance of user-friendly interfaces	49
4.5.4	Importance of software contracts	49

4.6	Conclusions	50
5	A data model to support world-wide data integration	51
5.1	Analysis of existing earthquake engineering data systems	52
5.2	Motivation for a new technology	56
5.3	The Web of Data	57
5.3.1	RDF, the data model	59
5.3.2	OWL, a language to define ontologies	62
5.3.3	Semantic considerations	66
5.4	The Celestina Data ontology	69
5.4.1	Ontology overview	71
5.4.2	Project module	73
5.4.3	Specimen module	76
5.4.4	Material module	76
5.4.5	Experimental activity module	79
5.4.6	Experimental input / output	84
5.4.7	Facilities and devices	85
5.4.8	Persons and organisations	88
5.4.9	Media and data module	89
5.5	Validation	91
5.6	Conclusions	93
6	Celestina Data	95
6.1	Basic requirements	95
6.2	Architecture overview	96
6.2.1	Constituents	97
6.2.2	General architecture	98
6.2.3	Celestina Data nodes	99
6.2.4	Celestina central site	102
6.2.5	Applications	103
6.2.6	Security considerations	103
6.3	Implementation	104
6.3.1	Celestina Data node implementation	104
6.3.2	Large files	109
6.3.3	Celestina central site implementation	110
6.4	Validation	112
6.4.1	The Celestina Data Accessor	114
6.4.2	SPARQL, the Semantic Web query language	117
6.4.3	Querying the triplestore	119
6.5	Results	120
6.5.1	Complexity of node implementation	122
6.5.2	Performance	123
6.5.3	Query results	125
6.5.4	Benefits of a Semantic Web data model	127
6.6	Other applications (Celestina Tools)	130
6.7	Maintenance considerations	132
6.7.1	The Celestina Data consortium	132
6.7.2	Joining Celestina Data	134
6.8	Conclusions	134

7	Foundations of distributed simulations	136
7.1	Types of earthquake engineering simulations.	137
7.2	Basic requirements for distributed hybrid simulations	138
7.3	Simulation infrastructure overview	139
7.4	Activities conducted by the simulation control program	141
7.5	The DHT program	143
7.6	Results from the DHT program	146
	7.6.1 Seek for computing expertise.	146
	7.6.2 Understand operating system internals.	147
	7.6.3 Design networks correctly.	148
	7.6.4 Operate in real-time = with time constraints	149
	7.6.5 Choose appropriate network protocols	150
	7.6.6 Tedious to plan and organise	150
7.7	Conclusions	151
8	Laboratory integration	152
8.1	Real-time and extreme performance	153
8.2	Real-time mechanisms for laboratory integration.	155
8.3	Elements involved in laboratory integration	157
	8.3.1 Wide area physical network	158
	8.3.2 Local physical network.	159
	8.3.3 Distributed simulation on the Internet.	160
	8.3.4 The simulation control program	161
8.4	RT-DHT: DHT program improvements for real-time simulation	163
	8.4.1 Discarding TCP as a network protocol	164
	8.4.2 Measuring time.	165
	8.4.3 Operating system scheduling	166
	8.4.4 Code optimisation	168
	8.4.5 Results from RT-DHT.	173
8.5	Security in laboratory integration	178
8.6	Conclusions	179
9	Celestina Simulations	181
9.1	Phases of an experiment	183
9.2	The Celestina Simulations framework	185
	9.2.1 Networking.	187
	9.2.2 Definition.	191
	9.2.3 Testing	195
	9.2.4 Services workflow overview.	199
	9.2.5 Semantics in Celestina Simulations	200
9.3	Implementation	201
	9.3.1 Implementation of Networking services.	202
	9.3.2 Implementation of Definition services.	202
	9.3.3 Implementation of Testing services	203
9.4	Validation.	206
	9.4.1 Celestina Simulations services operation.	208
	9.4.2 The HardwareConnector: FileConnector	209
	9.4.3 The HardwareConnector: NetworkTCPConnector.	209
	9.4.4 Validation of CS as a distributed hybrid simulation tool.	210

9.4.5	Collection of results	212
9.4.6	Recording data that is not transmitted	212
9.4.7	The end of the data exchange	213
9.4.8	Endianness	214
9.4.9	Performance and payload	215
9.5	Using the Celestina Simulations framework	216
9.5.1	Light adoption of Celestina Simulations in existing software	217
9.5.2	Possible uses of the framework	218
9.6	Conclusions	218
10	Conclusions	220
10.1	Impact of the research	221
10.2	Global lessons learnt	222
10.3	Future work	223
Appendix A	Public-key cryptography	225
Appendix B	Computational complexity	229
Appendix C	Logics for knowledge representation	232
Appendix D	The Internet protocols	238
	References	241

Chapter 1

Introduction

Earthquake engineering is the scientific discipline that studies the nature of earthquakes and their effect on structures, such as buildings or bridges, in order to diminish their damage and with the ultimate goals of saving human life and minimising economic loss. The required understanding of structural, ground and material behaviour is bound to the fact that human casualties of earthquakes are more often caused by the collapse of a structure rather than from the seismic motion itself. Sometimes, the study of how an earthquake affects different structures and their materials is conducted over structures that have been under the effect of a real earthquake. Some other times, the study is conducted by means of experiments, where part or a full structure, soil or devices are placed under a seismic stimulus in a laboratory and their behaviour recorded for subsequent analysis.

Experiments provide valuable data for the development of mechanisms to minimise the impact of earthquakes, but their execution can be very costly. Computing advances have enabled new methods of conducting experiments, for example by simulating the behaviour of structures in a computer. However, physical parameters and limitations on the understanding of the behaviour of the element tested sometimes impede the use of computer models, requiring the actual physical testing of the structure.

Data collected from real earthquakes and experimental activities in structures are a highly valued resource in earthquake engineering, since laboratory experiments can be extremely expensive and time consuming to replicate, and the effect of real earthquakes are impossible to record a second time.

Earthquake engineering institutions are increasingly adopting computer-based mechanisms to support experimental activities and to formalise the storage and manipulation of experimental data. However, there is a patent lack of interoperability between the institutions forming the earthquake

engineering community, which prevents efficient collaboration between them. This work aims to solve this interoperability problem by applying novel computing techniques to integrate distributed earthquake engineering resources. In the context of this thesis, *resource* means an asset, primarily referred to experimental data, laboratory facilities and distributed simulation software.

1.1 Objective, scope and research context

This thesis researches the integration of earthquake engineering resources, to facilitate collaboration between earthquake engineering institutions and enhance research outcomes. Current collaborations in earthquake engineering lack a formalised computing framework to efficiently support the activities conducted jointly. For example, there exists no common data format to exchange experimental data between institutions in Europe and the United States, which complicates the procedures to obtain and process results in a joint activity. Likewise, there is a lack of standards to facilitate the definition of the experimental workflow and to support the integration of distributed simulation software.

The scope of this research is limited to the integration of three resources: experimental data, laboratory facilities and distributed simulation software. The integration of experimental data is discussed in Chapter 3, Chapter 4, Chapter 5 and Chapter 6. Distributed simulations are introduced in Chapter 7, followed by the integration of laboratory facilities discussed in Chapter 8, and the integration of distributed simulation software in Chapter 9. This research has implemented functional prototypes and extracted results from their evaluations.

The present research has been primarily conducted in the context of three programmes: **UK-NEES**, **SERIES** and **NEES**. UK-NEES (United Kingdom - Network for Earthquake Engineering Simulation) was a collaborative project between the Universities of Oxford, Bristol and Cambridge, funded by the EPSRC (Engineering and Physical Sciences Research Council) and carried out from the end of 2006 to the beginning of 2009.

SERIES¹ (Seismic Engineering Research Infrastructures for European Synergies) is a four year, €8.4 million project involving twenty-three institutions from twelve countries in Europe. It was funded by the European Commission, and was carried out from 2009 to 2013, being the

¹ <http://www.series.upatras.gr/>

largest collaborative project conducted in the European earthquake engineering community up to the current date.

NEES² (Network for Earthquake Engineering Simulation) is currently under a four year, \$105 million project involving fifteen institutions in the United States of America. Funded by the NSF (National Science Foundation), it is the largest earthquake engineering network in the United States. NEES was created in 2004, and its current management by Purdue University runs from 2010 to 2014.

1.2 Contribution

Because the work presented in this thesis is strongly based on collaborations, it is convenient to clearly identify my contribution:

- I have formalised the relational model presented in Chapter 3. The informal data requirements have been defined by Dr. Anna Bosi and Dr. Pierre Pegon (at the Joint Research Centre, in Italy) and Dr. Matt Dietz (at the University of Bristol, United Kingdom), with very marginal suggestions by myself and some contributions from other participants of the SERIES project. I have created a relational database that have been tested by myself jointly with Dr. Anna Bosi and Dr. Pierre Pegon (JRC) and Dr. Jean-Louis Chazelas (at IFSTTAR, France). Other SERIES participants have sporadically contributed to the testing and the discovery of errors.
- Based on some SERIES requirements, I have formalised the design for the SERIES Virtual Database discussed in Chapter 4, and implemented the node services, by means of Web Services, and the services definition (WSDL). The SERIES central site has been entirely implemented by Ioannis Ioannidis (at University of Patras, Greece) and it does only partially adopt the internal design suggested and presented in this thesis. I have designed the system for the download of large files and implemented the required component on the node side, whereas the component on the central site side has been implemented by

² <https://nees.org/>

Ioannis Ioannidis. Testing of the services, security and download of large files has been done by myself jointly with Ioannis Ioannidis and Dr. Pierre Pegon.

- I have created the ontology described in Chapter 5, and conducted all the testing myself.
- I have designed and created the infrastructure of Celestina Data and implemented all the components described in Chapter 6, with exception of the modifications on the NEES Project Display (the NEES Website), that were conducted by Lilly Dong (at the University of Purdue, United States), following my decisions on how to present the data. Validation of the system was conducted by myself, and the installation of components in EUCENTRE was done by Francesco Lunghi (at EUCENTRE, Italy).
- I have created the distributed hybrid software called DHT discussed in Chapter 7. The software was tested by myself in a local environment and used and tested by Dr. Mobin Ojaghi (at the University of Oxford, United Kingdom) and Dr. Matt Dietz in a distributed environment.
- I have created the RT-DHT program discussed in Chapter 8. The program was validated with the help of Ferrán Obón (at the University of Kassel, Germany), who executed the server side during the tests.
- I have designed and implemented the Celestina Simulations framework described in Chapter 9. Validation of the system was done in partnership with Ferrán Obón, who created the C program that ran underneath Celestina Simulations.

Due to the nature of the project, extensive support was offered by personnel from SERIES and NEES.

1.3 Celestina

The prototypes used for the integration of resources have been gathered under a computer-based platform called **Celestina**, which obtains its name from a 15th century Spanish novel (*Comedia de Calisto y Melibea* by Fernando de Rojas), where a woman called *Celestina* manages to bring together a man and a woman into a loving relationship, although they initially seem incompatible.

This is used as an analogy to a platform that makes interoperation possible between distributed systems that were initially not designed to communicate or coexist together.

The platform is divided into three sub-systems:

- **Celestina Data** is concerned with the infrastructure for the integration of experimental data and the associated data services. This sub-system serves as the foundation for the management and exchange of data.
- **Celestina Simulations** is concerned with the framework to conduct distributed simulations. It defines high-level specifications and integrate simulation services provided by software to exchange simulation data on the network. This sub-system serves as the foundation for distributed testing, and utilises Celestina Data services as a common data framework to formalise simulation results.
- **Celestina Tools** is concerned with the applications that make use of Celestina services, especially those related to data. These applications make the data integration useful for the end-user by providing services to interact with Celestina data, such as data visualisation tools or decision support systems.



Figure 1.1. Celestina logo design in colour and black and white (by Ignacio Lamata Martínez)

The use of a common platform responds to a strategy to unify dispersed efforts under the same integration system. The branding depicted in Figure 1.1 has been created to use the platform as a product.

Celestina is currently a platform for earthquake engineering but it is intended to expand to other research areas. This includes the expansion to other types of earthquake engineering data (not only experimental data), other types of hazard mitigation disciplines (such as wind engineering and fire engineering), and collaborations with the publishing industry and profit organisations.

1.4 Terminology

In this thesis, the terms *experiment* and *simulation* will be used interchangeably, although the term *experiment* will normally involve the use of a physical specimen and the term *simulation* will normally involve the use of a specimen emulated by a computer model. The term *distributed simulations* will be used interchangeably with the term *distributed testing*, and without any implication of the nature of the specimen tested. The term *experimental activity* will be used when a physical experiment or a numerical simulation has to be referred without being explicit of the nature of the specimen tested.

1.5 Thesis layout

The thesis is organised as follows:

- Chapter 2 reviews past research efforts related to the work presented in this thesis, specifically about databases in earthquake engineering, software for distributed simulations, real-time on the Internet and recent advances on data management systems.
- Chapter 3 introduces the basis of earthquake engineering experimental data, and presents a data model that supports the *SERIES database*, used by the European earthquake engineering institutions.
- Chapter 4 describes the technical infrastructure adopted by the leading earthquake engineering institutions participating in SERIES, called the *SERIES Virtual Database*. This

infrastructure enables both interoperability between the European institutions and access to European data for the scientific community, and it has been validated in partnership with the Joint Research Centre and the University of Patras. This integration effort is extrapolated to a worldwide scenario in the following chapters.

- Chapter 5 discusses the creation of a data model that will serve as the foundation to address worldwide data integration, called the *Celestina Data ontology*. This data model has used the existing data models at SERIES and NEES as references.
- Chapter 6 presents *Celestina Data*, a technical infrastructure to accomplish worldwide data interoperability. The infrastructure has been validated between data sources from SERIES and NEES, and an application has been created to provide the NEES community with data from the University of Oxford and EUCENTRE.
- Chapter 7 introduces the basis of distributed simulations, and presents the example of the *DHT program*, a piece of software utilised for distributed hybrid testing between the universities of Oxford, Bristol and Cambridge. Two main shortcomings are identified in distributed simulations: the lack of appropriate distributed testing software for real-time and the lack of high-level standards that enable integration of distributed testing software. These are addressed in the following chapters.
- Chapter 8 discusses laboratory integration, as a means to integrate remote laboratory facilities as if they were in the same laboratory. This entails the necessity of real-time operation, which is addressed by means of the *RT-DHT program*, an optimised version of DHT that is aware of time constraints.
- Chapter 9 presents *Celestina Simulations*, a high-level framework to support the definition and execution of the experimental workflow and the integration of distributed simulation software. A possible implementation of the framework has been created and it has been validated in partnership with the University of Kassel.
- Chapter 10 summarises the conclusions of the present research.

Chapter 2

Literature review

The subject presented in this thesis covers a wide range of topics. This chapter reviews some of the research and advances made on the main topics related to this thesis, primarily regarding experimental data and simulations. The chapter is organised as follows:

- **Earthquake engineering databases** is discussed in Section 2.1, which presents some of the most relevant efforts to create repositories for experimental data in earthquake engineering.
- **Recent advances on database systems** is discussed in Section 2.2, which summarises new developments in data management systems, often motivated to tackle limitations of the relational model experienced by some novel applications.
- **Earthquake engineering software for distributed simulations** is discussed in Section 2.3, which presents some of the computing tools utilised to conduct distributed simulations in earthquake engineering.
- **Real-time and quality of service on the Internet** is discussed in Section 2.4, which presents the problems of real-time operation on the Internet and some of the main initiatives and methods that have been conducted to address them.

Review of the Semantic Web is presented in Chapter 5 and in Chapter 6, where the technology is used.

Finally, Section 2.5 lists some conclusions that motivated the research presented in this thesis.

2.1 Earthquake engineering databases

Developing data repositories and sharing facilities are topics of wide interest in science and engineering. Computing advances in the way data is stored and managed have motivated earthquake engineering institutions around the world to develop data models and storage solutions for the experimental data they produce.

The most noteworthy example is the database developed by NEES¹ (Network for Earthquake Engineering Simulation). NEES was created by the *National Science Foundation* in 1998 to “aggressively move forward the development of improvements and innovations in infrastructure design and construction practices to prevent or minimize damage during the event of an earthquake or tsunami”, and is composed of a headquarters and fourteen laboratories distributed around the United States.

The NEES infrastructure (Hacker *et al.*, 2011) has a centralised data repository currently called the *Project Warehouse*, to store and share experimental data, whose data model is described by Pejša *et al.* (2014) and Pejša (2012). The quality of the data is verified and improved by means of a process called *data curation*, in which the data are monitored and approved for publication (Pejša *et al.*, 2013). NEES has commitments in terms of digital data preservation, to assure that data remain accessible and usable (Pejša *et al.*, 2011), and provides open licences² and digital identifiers³ for both documents and data resulted from experimental activities.

The NEES database provided data integration to earthquake engineering institutions in the USA. The database is supported by a number of tools such as *inDEED* (Howlett *et al.*, 2009), and *PEN* (Dong *et al.*, 2014). Extended information about the NEES data model can be found in Section 5.1 of this thesis. Besides supporting the centralised data repository, NEES also provides direct access to a group of other databases (NEESDB, 2014).

Peng *et al.* (2004) attempted the creation of an ontology-based data model to support NEES metadata (the term *ontology* is defined in Section 5.3.2). Although it had limitations in the range of experimental data that could be represented, this was probably the first attempt to use an ontology in earthquake engineering. The ontology was extended up to the point that was too

¹ <https://nees.org/>

² <https://nees.org/legal/licensing>

³ <http://www.doi.org/>

complicated to use and, therefore, impractical. Bardet *et al.* (2004) directed new efforts to create a better documented, simpler, more usable data model from the previous ontology, although it was again never adopted to support NEES experimental data. The original data model has been lately discussed by Bardet *et al.* (2010), and seems to have been disassociated from the original ontology-based ideas related to the *Semantic Web* (the Semantic Web described in Section 5.3), while attempting to somehow accommodate some of its features. Nevertheless, these data models were one of the first attempts to formally document an earthquake engineering data model.

These failed attempts to create an ontology, however, provided some useful lessons. The first one is that the use of an emerging technology must have both a clear application and a benefit for the end user to succeed, demonstrated by the fact that the mere use of an ontology did not manage to show any benefit to substitute or complement the existing relational database. The second one is that extremely complicated data models can be rejected by their own users, and should be avoided. Those two lessons have been considered in the development of the system described in Chapter 5 and Chapter 6.

Another data repository was developed in UK-NEES, a collaborative grid-based research network established between the Universities of Oxford, Bristol and Cambridge (UKNEES, 2009). The UK-NEES data repository was a simple centralised database to store experimental results from the three universities. The repository was very limited and it was superseded by the databases discussed in this thesis.

Finally, the Joint Research Centre in Ispira, Italy, have used from 1999 a database to store results of experimental activities conducted at ELSA⁴ (European Laboratory for Structural Assessment), as defined by Bosi *et al.* (2009). The data model of this database has served as one of the foundations for the data model discussed in Chapter 3.

Other databases have been internally created at different institutions around the world, such as EUCENTRE in Pavia, Italy, and E-defense in Japan, but they have not been documented publicly.

Earthquake engineering databases are relevant for Chapter 3, Chapter 4, Chapter 5 and Chapter 6.

⁴ <http://elsa.jrc.ec.europa.eu/>

2.2 Recent advances on database systems

The most widespread methods to store digital data on a database are likely to be based on the relational model invented by Codd (1970), revised in Codd (2000). Relational databases have been demonstrated to be an efficient and robust mechanism to store and query data for general applications.

However, the emerging success of the Internet, established as a necessity in modern societies, has encouraged the creation of new applications for which data requirements do not perfectly fit in the relational model. These applications are characterised for managing massive sets of data that sometimes change rapidly, requiring high data availability and serving millions of users. The infrastructure required to meet these demanding requirements needs features such as *scalability* (the capacity of maintaining performance when workload increases) and *replication* (duplicating the information in a redundant computer), which conflict with the strong consistency that relational databases have (DeCandia *et al.*, 2007).

High data availability has been addressed in relational database management systems by using a technique commonly known as *sharding*, which consists of splitting a single database into several databases on different computers. Each of these computers works on an isolated data set, so ideally connecting new computers to the network will automatically increase the system capacity when its limit is nearly reached (this is known as *horizontal scalability*). However, sharding requires careful design on how to split the data set and brings high complexity to some data processes when data from several computers are required.

This has forced some organisations to explore novel database management systems and create their own for those applications in which the relational model did not satisfy their operational demands. Common features of these systems are: high data availability, unstructured data storage, inexistence of predefined schema, management of very large collections of data, simple API to access the system, replication support, normally distributed, horizontally scalable, fault-tolerant and eventually consistent. They are sometimes classified according to their main data model: key-values stores, column family stores, document-oriented databases and graph databases (Burtica *et al.*, 2012).

DeCandia *et al.* (2007) describe *Dynamo*, a key-value distributed store that has been successfully used to meet the demands of Amazon applications. Dynamo data model uses simple structures

that can be retrieved with a unique key, and high availability has been prioritised over consistency, meaning that the importance of accessing data prevails over accessing the most updated version of the data.

Chang *et al.* (2006) present *Bigtable*, a distributed storage system that maps values of data by a row key, a column key and a timestamp. Bigtable uses simple data models and enables users to define their own data layout and format underneath. Bigtable has been used to bring scalability, high performance, and high availability to various applications at Google.

BigTable has inspired the creation of HBase⁵, a column family store based on Hadoop⁶. Hadoop is a framework for the distributed processing of large data sets stored across different computers by using a model called *MapReduce*, described by Dean *et al.* (2004). Hadoop is an emerging technology in science and industry, where processing very large sets of data is required.

Lakshman *et al.* (2010) introduce *Cassandra*, another distributed column-based store that enables users to define their own data layout and format.

Robinson *et al.* (2013) discuss the benefits of linked data and the discovery of information based on weighted data relationships. They claim that many database management systems deal poorly with relationships and that a more natural and efficient way to manage connected data is by means of graphs. Specifically, they have shown how graphs overcome a relational database performance when querying a set of linked data as its depth increases. In this respect, Celko (2004) has discussed optimisations to deal with hierarchical data in the relational model, such as the *nested set* model. Neo4j⁷ and Bigdata⁸ are examples of graph databases. Sometimes they are supported by linked structures called trees, such as the Bigdata *B⁺-trees*, described by Elmasri *et al.* (2010).

Finally, a novel type of data repository are *triplestores*, which store data structured in tuples of three elements called triples, normally identified as *subject*, *predicate* and *object* (note that this can be also represented as a directed graph). They are mainly used to store RDF data on the Web of Data (discussed in Section 5.3), which gives triplestores the advantage of having an standardised data model and query language. Although the performance of triplestores have been

⁵ <http://hbase.apache.org/>

⁶ <http://hadoop.apache.org/>

⁷ <http://www.neo4j.org/>

⁸ <http://www.systap.com/>

questioned, triplestores have performed adequately in some production applications, as for example shown in tests done by Krötzsch *et al.* (2007). Additionally, some triplestores provide functionality to store quads (tuples of four elements) and some are supported by relational databases underneath. Examples of triplestores are Stardog⁹, Virtuoso¹⁰ and Oracle 11g¹¹. A study on performance and functionality of some of these triplestores has been conducted by Haslhofer *et al.* (2011).

Finally, note that the relational model is still widely used and performs extremely well in many current applications. The non-relational databases discussed in this section are just a solution that simply solve specific problems more efficiently.

The discussion of data management problems and the solutions proposed by novel database systems is relevant for the data management systems discussed in Chapter 3 and Chapter 5.

2.3 Earthquake engineering software for distributed simulations

Much of the experimental data are obtained by conducting experiments and computations. Sometimes these experimental activities take place between geographically separated laboratories in what is called distributed testing. There exists a variety of programs to support distributed testing in earthquake engineering, which mainly focus on data exchange on the Internet between remote computers. The most widely used is probably *OpenFresco*, developed at the University of California, Berkeley (Schellenberg *et al.*, 2010). OpenFresco is a middleware tool that facilitates distributed hybrid simulation of structural systems. The tool is well-suited to performing generic distributed tests, supporting a wide variety of libraries and allowing integration of both testing-support tools and experimental control and data acquisition systems. Moreover, it has a good level of abstraction and generalization, and allows users to program TCL (Tool Command Language) scripts, which favours flexibility. However, OpenFresco lacks support for laboratory integration, as described in Chapter 8, and lacks a user-friendly experiment planning support, as discussed in Chapter 9.

Kwon *et al.* (2008) developed *UI-SimCor*, which is a framework to support multi-site distributed hybrid testing. It is a modular software that can be extended, based on MATLAB and can interact

⁹ <http://stardog.com/>

¹⁰ <http://virtuoso.openlinksw.com/>

¹¹ <http://www.oracle.com/>

with some other software, including OpenFresco. However, UI-SimCor has limitations when testing rate dependent structures, and the fact that it is MATLAB-based might have some performance implications when running real-time simulations. Note that this issue is common to OpenFresco and other software.

Mosqueda *et al.* (2008) describe a system known as *FASTMOST* (Fast Multi-Site Online Simulation Test). It uses a modified bespoke protocol called NTCP (NEESgrid Teleoperations Control Protocol) to exchange control data, which handles data transmission errors (including data retransmission, which is normally not desirable in real-time simulations). The original NTCP protocol works with a centralised coordinator that interacts with every remote simulation participant. FASTMOST has been superseded by more recent software.

Another example of software is *UK-NEES DHT*, developed at the University of Oxford, which has been successfully used in real-time distributed testing (Ojaghi *et al.*, 2013). It seems to be the only program that permits use of different network protocols other than TCP. DHT is described in Chapter 7 of this thesis, and it has been superseded by *RT-DHT*, which is described in Chapter 8.

Pan *et al.* (2006) describe a system called *P2P Internet Online Hybrid Test System*, which is based on a computing network connecting independent, loosely coupled systems to a centralised coordinator system named “supernode”. The distributed testing systems do not directly connect to each other, performing all communication through the centralised supernode. The system has been catalogued as *peer-to-peer* (P2P) by its authors. However, Schollmeier (2001) has defined a P2P network as “a network where all the machines within it share some of their hardware resources and where any machine is accessible by any other machine in the network without intermediaries”, which disqualifies the system defined by Pan *et al.* (2006) as P2P, since all communication is performed through the so-called supernode.

The authors of the P2P Internet Online Hybrid Test System assert that socket communication is a fast and reliable solution to conduct distributed testing, which is in agreement with other authors’ findings (Yang et al, 2007b; Ojaghi *et al.*, 2010). The system does not seem to implement any real-time operation mechanism, and its specifications suggest that TCP is used as a network protocol.

A final example is the Platform for Networked Structural Experiments, developed at the National Center for Research on Earthquake Engineering, Taiwan (Yang et al, 2007; Yang et al,

2007b), which describes two approaches to conduct multi-site simulations with a central entity in the middle of every simulation participant. The “Database Approach” connects all testing resources called “Facility Controllers” by means of a centralised entity called the “Data Centre”. The “Application Protocol Approach” utilises a central server to communicate with all the clients in an event-driven environment, by sending messages marked in an own application-layer protocol. Both approaches depend on a central entity, and there is no direct communication allowed between the testing participants. Facility Controllers’ response is collected as a whole, which can cause slower Facility Controllers to delay the whole system operation during a multi-site test, overshadowing the performance of faster Facility Controllers. In addition, Facility Controllers do not implement any mechanism to ensure that the polling system implemented (continuous checks of the data availability) is indeed carried out without external interferences (*e.g.* the operating system execution). This is a crucial point since the real-time synchronization is executed without implementing proper mechanisms, which can result in ruined experiment results.

Distributed simulation software is relevant for Chapter 7, Chapter 8 and Chapter 9.

2.4 Real-time and quality of service on the Internet

The Internet is a massive network of networks which use heterogeneous hardware and software and are controlled by different organizations. For their common understanding, common protocols are required, normally referred to as the TCP/IP protocol suite (Stevens, 1994). This suite is mainly, but not exclusively, formed by IP (Internet Protocol), TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) - a description of these protocols is given in Appendix D. These protocols were originally designed for a highly distributed, fault-tolerant network interconnection able to cope with operation disruption (Tanenbaum *et al.*, 2010), without considering special timing requirements: the network is able to guarantee delivery but not at any specific time range.

Quality of Service on the Internet has been typically achieved by distinguishing between data needing urgent delivery and the rest of data, and is demanded by applications required to work under specific time constraints - also known as real-time, as described in Section 8.1. Real-time needs time predictability, which originally was not a design feature of the Internet protocols. Real-time on the Internet is often addressed in its different layers (see Figure D.1 in Appendix D for

a division of network layers).

In the transport layer, TCP performance for real-time application has been widely discussed since it is not a predictive protocol and its behaviour depends on current network load conditions (Huston, 2000). Different approaches for high-speed TCP have been discussed (Huston, 2006) and some performance improvements have been proposed (T/TCP; Braden, 1994) whose impracticality were later demonstrated (Duke *et al.*, 2006). In practice, UDP normally substitutes TCP in applications demanding real-time functioning. Compression of UDP/IP headers has been studied as a way to decrease communication latency (Degermark *et al.*, 1999). The Network Time Protocol (Mills *et al.*, 2010) has been developed to synchronize computer clocks on the Internet, although in practice it can cause non-monotonic clock operation as discussed in Section 8.4.2. Park *et al.* (1997) proposed the LAN-TCP algorithm acting on the transport layer to allow soft real-time operation.

TCP operation is indeed not the only problem for real-time operation on the Internet. The Ethernet, a link-layer network protocol that operates underneath TCP and IP and is widely used on local area networks connected to the Internet (see Appendix D), uses an algorithm of collision detection to send data to the network cable (Tanenbaum *et al.*, 2010), consequently causing nondeterministic communication latency. This means that if there are five computers on the same network, they all have to share the network cable, delivering data when necessary in turns - instead of simultaneously, causing waits or packet collisions that ruins any real-time operation. Collision with other devices in the network can be avoided by using *switched Ethernet*, although with this solution, data might require buffering. Since Ethernet has become an inexpensive technology widely used nowadays, many initiatives to provide time synchronization over it (which are intrinsic characteristic in other similar technologies) have been developed. One of them is Synchronized Ethernet (SyncE; Ferrant *et al.*, 2008), a standardization effort to enable synchronization and clock signal transference over Ethernet. Also, the IEEE 1588 standard (IEEE 1588-2008, 2008) attempts to provide clock synchronization with sub-microsecond accuracy in a local area network (normally, but not exclusively, over Ethernet) by using the Precision Time Protocol (PTP). Alternatively to Ethernet, other standards have been used in the aerospace and defence industry to provide fault-tolerant deterministic architectures for hard real-time control bus of distributed systems (IEEE 1394-2008, 2008; SAE, 2006).

Common methods to implement real-time over networks operated by TCP/IP/Ethernet include (i) using low-level programming on the microprocessor (sometimes with the support of additional hardware), (ii) avoiding the use of TCP/IP (whose implementation might be too poor in many network devices) and (iii) using a token passing protocol (preventing computers from sending data unless requested, so collisions are minimized). Some of these solutions might only be valid when the computers are close to each other in a controlled environment (*e.g.* with no publicly shared external devices in the network).

Real-time on the Internet is not a new topic. In 1973, the necessity of real-time Internet data communication was envisaged (Pfeifer *et al.*, 1973). General real-time on the Internet focuses on soft real-time data exchange for video and sound applications, and has motivated the development of a variety of techniques (such as *Forward Error Correction* or *interleaving*; Tanenbaum *et al.*, 2010; Rubenstein *et al.*, 1998) as well as protocols such as the Real-time Transport Protocol (RTP; Schulzrinne *et al.* 2003). RTP’s main goal is to support multi-participant video and audio applications. Video and audio applications do not have the same requirements as non real-time applications (*e.g.* e-mails or file transfer). Whereas the latter might require complete reliability, video and audio applications could cope with some data loss providing the data arrive on low-latency terms. An example of this is that some noise in a phone conversation can be acceptable, but intermittent signal cuts might be irritating for the users. RTP is described as a “transport protocol that is implemented in the application layer” (Tanenbaum *et al.*, 2010), since it provides transport-layer services over the UDP transport protocol. However, the RTP protocol does not guarantee any quality of service in the delivery of real-time data and just relies on the lower network layers (mainly IP) to provide it.

As mentioned before, Internet protocols were not designed with quality of service in mind, and therefore all traffic is treated the same way, with no priorities. Although the IP packet was designed to provide limited quality of service features (such as the *Type of Service* header field), it is not used in practice by any existing routing protocol (Tanenbaum *et al.*, 2010). IP version 6 was designed to improve IP quality of service for real-time applications (Deering, 1998), and it implements two mechanisms to support quality of service by means of the *Traffic class* and the *Flow label* header fields.

Another effort to deliver real-time data on the Internet is RSVP (Resource reSerVation Protocol; Zhang *et al.*, 1997). RSVP acts in the network layer and is based on resource reservation on routers to set a path to deliver the data flow. RSVP has also been described as a “switch state establishment protocol” (Zhang *et al.*, 1993). Further information on RSVP can be found in Farrel (2004).

Real-time on the Internet is relevant for Chapter 8, and partially for Chapter 7.

2.5 Conclusions

The review discussed in this thesis exhibited the following issues, which have primarily motivated the research presented in this thesis: (i) A **lack of data integration** in earthquake engineering, meaning that there is no formalised way to exchange experimental data between the earthquake engineering institutions world-wide. (ii) Most of the software for distributed testing is **not well suited to real-time**, and seems to have failed to explore alternatives for network communication. (iii) **Lack of integration of distributed testing software**, meaning that homologous testing software must be used during simulations, and the resulting data they generate are not defined in a standardised format. (iv) **Lack of an infrastructure that facilitates experimental planning** in a user-friendly way.

Chapter 3

Foundations of experimental data

Experimental data are the data collected prior, during and after the execution of an experiment or a computation. They often consist of a variety of data that include experimental results, the experimental set-up and the context in which the experimental activity takes place, *e.g.* the project that motivated its execution.

Experimental data are a highly valued resource in earthquake engineering, since laboratory experiments can be extremely expensive and time consuming to replicate a second time. This necessitates the storage of the data produced.

Earthquake engineering experiments are also uniquely characterised by its wide variety of disparate set-ups, which can involve an extensive range of devices, facilities, components, materials and possible configurations. This freedom of design often demands careful preparation, and leaves many crucial experimental decisions to the researcher's interpretation. Other sciences, such as biology, count with a more rigid classification of experimental set-ups. The open nature of earthquake engineering experimental set-ups is directly exhibited in the data models that store the data.

Experimental data provide a source of support for new research projects. However, issues such as the identification of which data were used to support other research works, and methods to guarantee control of the data by their owners still remain challenging.

This chapter discusses general data needs in earthquake engineering and describes the implementation of a data model to store earthquake engineering experimental data for the European institutions participating in SERIES¹ (Seismic Engineering Research Infrastructures for European Synergies). Section 3.1 presents a brief overview of data needs by discussing the common structure of earthquake engineering experimental data. The SERIES database, which is used in the leading earthquake engineering institutions of Europe, is described in Section 3.2. Finally, some conclusions

¹ <http://www.series.upatras.gr/>

can be found in Section 3.3.

3.1 Experimental data structure overview

From an abstract, high-level perspective, experimental data can be divided into three main groups of information that must be stored in an experimental activity:

- **Experimental context**, which consists of the description of the experimental activity, such as the dates when the experiment or computation was conducted, the personnel that participated as test agents, the objectives pursued with the experimental activity, whether the experimental activity was physical or numerical (as described in Section 7.1), *etc.* It also includes information about the project in which the experimental activity is conducted and funded. These data describe the organisations involved and the relations with other experimental activities, which give a higher view of the objectives that the experimental activity aims to achieve as part of the big picture.
- **Experimental set-up**, which describes the detailed configuration of the experimental activity. This can be subdivided into three parts: (i) Information about the specimen that is tested. This could be a physical structure, such as a column, or a device, such as a viscous damper. The specimen can be described in terms of sub-components, for which their composite materials are described. Detailed description of the specimen is desired to correctly understand the results of the experiment. Sometimes, the specimen can be simulated in a computer model. (ii) Information about the physical equipment used in the experiment execution. This describes the properties and specific use of devices (*e.g.* an actuator, a sensor, *etc.*) and facilities (*e.g.* a shaking table, a strong floor, a centrifuge system, *etc.*). (iii) Information about the non-physical systems used, such as software or computer models.
- **Experimental execution**, which represent all the data that are used to conduct the experimental activity. This can be divided into loading data, meaning all the data that are provided to the experimental activity for its execution, and collected data, meaning all the data that is obtained as a result of the experimental activity execution, and is needed for

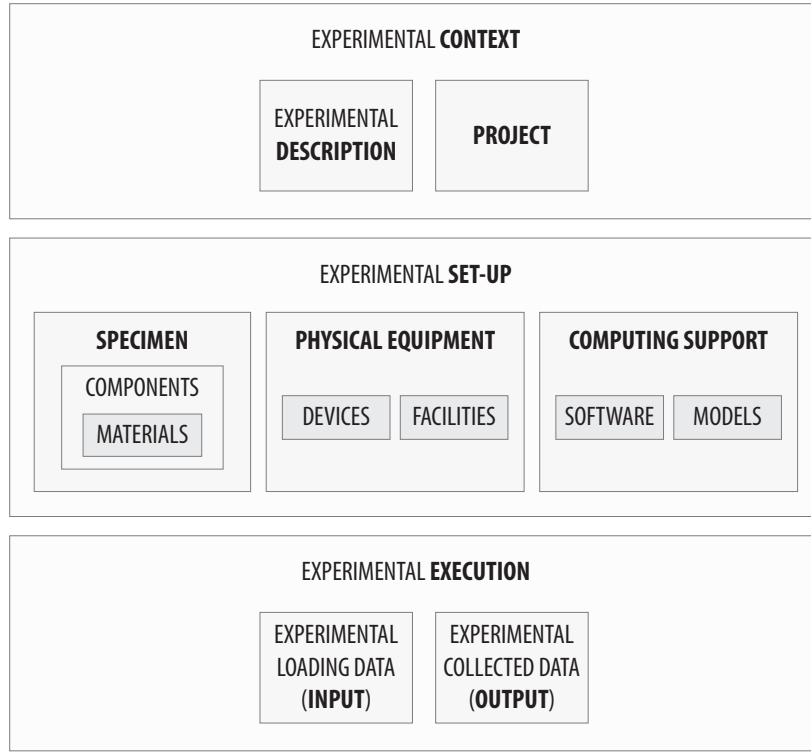


Figure 3.1. High-level view of the experimental data structure

subsequent analysis. If the experimental activity is repeated, experimental execution data will certainly change, whereas the data of the other two groups will often stay the same.

A graphical view of this three-group structure is depicted in Figure 3.1. Often, the *experimental context* and the *experimental set-up* are described as *metadata*, whereas the *experimental execution* is described as *data*. This abstract view can be implemented in practice in many different ways, attending to most efficient ways to formalise, represent and restructure the data. This will be shown in the data models described in Sections 3.2, 5.1 and 5.4 where, for example, the project context information is a hierarchical superior data structure, and some data belonging to every group is stored in a separated data structure that can be managed more efficiently by itself (*e.g.* documents).

Creating classifications of elements in earthquake engineering data has demonstrated to be challenging. It is also complicated by the fact that earthquake engineering is actively researching on novel materials, new experimental methods and other innovations. Deciding the parameter over which classify elements is not always a straightforward decision. One possible parameter is the engineering specialisation: structural (normally related to structural engineering) or ground (normally related to geotechnical engineering). This parameter might be a candidate in the classification of composite materials used, since structural and earth materials rarely share materials. Other

classifications can use different parameters. For example, experimental activities can be classified by the nature of the specimen they test: if it is tested physically, simulated by a computer model or a mixture of both is used (as described in Section 7.1). Other classification parameters are easier to select, such as the classification of devices according to the type of device: actuator, sensor, *etc.*, which can themselves be subdivided according to extra parameters. For example, sensors could be categorised according to the physical unit they measure. However, even having a clear decision of the classification parameter, the actual classification might result a challenging task.

The following section presents the practical implementation of a database for experimental data used at the leading earthquake engineering institutions in Europe. Note that the work presented does not address the creation of an abstract model, but the practical formalisation of a data model from a previously defined abstract model.

3.2 The SERIES database

In the context of SERIES, Lamata Martínez *et al.* (2010) defined data requirements in an informal way. These requirements have been implemented formally in a relational data model that has supported the creation of a database. Note that the work presented does not evaluate the correctness of the data requirements, but simply formalises them in computer data structures.

The SERIES database serves as the foundation for the data integration of European institutions discussed in Chapter 4.

The decision to formalise the data in the relational model is because relational databases are a conventional, well established method to store information, which provides shared access to large banks of formatted data and incorporates query and update capabilities (Abiteboul *et al.*, 1994). Because it is a well known technology, it is easy to find technical support and people with knowledge in relational databases. These points are crucial for the maintenance of data systems in an environment where people are in continuous rotation, such as the students of an academic environment, and there might exist limited IT support, such as in civil engineering.

The relational model is introduced in Section 3.2.1. The SERIES data model is briefly described in Section 3.2.2, whose validation is discussed in Section 3.2.3. The validation was conducted into two ways, jointly with two SERIES institutions to prove suitability of the data model and

by analysing potential optimisations on the data model.

3.2.1 The relational model

The relational model is one of the most widespread database approaches for data storage. It specifies tables (also called *relations*, sets of elements of the same nature) and relationships between them. Tables have a fixed (but alterable) set of columns (also called *attributes*) and a, normally increasing, number of rows (also called *tuples*). An example of a table *Experiments* is depicted in Figure 3.2. The table contains three experiments (three tuples) for which four attributes are stored: experiment identifier, experiment name, experiment date and a number of times the experiment was conducted. Attributes have constraints over the type of data they store, such as numerical values, string of characters, dates, binary data, *etc.*

As elements of a set, tuples must be unique within the table and their order is not relevant. Tuples have one or more attributes that uniquely identify a tuple within the table. These attributes are referred as the *primary key* of the table and they cannot be *null* (the marker that represents a missing value). When a table primary key is referenced by another table, this is called a *foreign key* in the external table. Foreign keys support to keep data consistent between the tables so, for example, a tuple cannot be deleted if it is referenced as a foreign key by a tuple of another table, and unmatched foreign keys cannot exist. An example of the relation of two tables is shown in Figure 3.3. The table *Projects* uses a foreign key to specify which experiment it conducts. Any experiment referenced from a project cannot be deleted while the reference exists, and projects cannot reference inexistent experiments.

TABLE EXPERIMENTS

id	name	date	repetitions
1	Shaking table-01	22/04/2011	3
2	Viscous damper test	06/10/2012	1
3	Experiment-007	10/10/2014	10

Row or tuple →

↑ Column or attribute "repetitions"

Figure 3.2. Example of table for experiments

To assure the quality of the design of tables, there exists a process called *normalisation*. A normalised database assures that the database design represents “a fact one way, one time and in one place” (Celko, 1999). Normalisation avoids data redundancy and prevents errors. In the process of normalisation, there exist different normalization levels (also called *normal forms*), each of them more restrictive or “pure”. The most basic normalisation level, the first normal form, requires that every attribute on a table must have only one value (not structured data such as arrays or lists). The second normal form additionally requires that every table stores one single concept, so every attribute in the table totally depends on the table primary key. The third normal form additionally requires that non-primary key attributes are independent each other. Although there exists more restrictive normal forms, a database is considered normalised when it complies with the third normal form. An example of normalisation is depicted in Figure 3.4. The tables of Figure 3.4b require only a single write operation if any of the main activity texts has to be modified, whereas the table of Figure 3.4a will require one writing operation for each of the tuples using the main activity. Also, normalisation saves space because data are stored once and referenced, and unifies the content of attributes avoiding mismatch errors (*e.g.* retrofit techniques, ret. tech., Retrofit Technique, *etc.*).

Although a relational database should be normalised and have no data redundancy (Codd, 2000), these restrictions might be avoided when performance is a priority. Modifications on a normalised database are normally faster because a single writing operation is required, but reading from a normalised database could require access to several tables, with a corresponding decrease on response time. It is a common practice to denormalise a database in order to increase reading times.

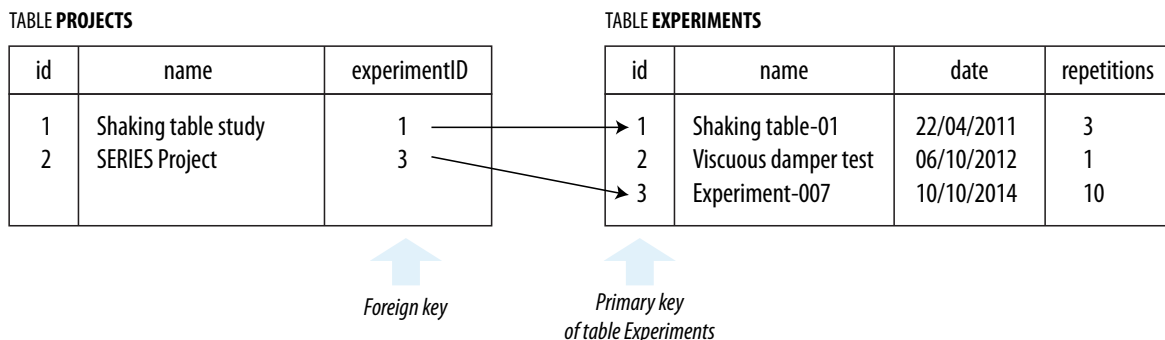


Figure 3.3. Example of foreign keys between the Projects and the Experiments tables

The relational model is supported by a query language called *SQL* (Structured Query Language; Burleson *et al.*, 2003), which allows data extraction and data updating in a relational database. SQL is based on multisets, which allow repetition of elements within a set - hence a query could return repeated rows. When data does not exist in the model, it is considered *false* and not *unknown*. This behaviour is known as the *Closed World Assumption*.

Extended information on the relational model can be found in Abiteboul *et al.* (1994) and De Haan *et al.* (2007).

3.2.2 SERIES data model overview

A normalised relational data model has been created from the data requirements defined by Lamata Martínez *et al.* (2010). The model is based on four hierarchically organised levels: project, specimen, experiment/computation and signal. A fifth independent level can be considered, containing metadata of media resources: documents, videos and images, as depicted in Figure 3.5.

Since the laboratories conduct different kind of experiments, the SERIES data model has been designed broadly enough to cope with information coming from the wide variety of experiments and simulations that are conducted in the community, including field monitoring, geotechnical centrifuges, shaking tables and reaction walls.

The SERIES database uses the term *experiment* to refer to physical experimental activities, and *computation* to refer to numerical experimental activities. The concept of *signal* represents paired

(a)

TABLE **PROJECTS**

id	name	main activity
1	Shaking table study	Model calibration
2	SERIES Project	Retrofit techniques
3	UKNEES Project	Model calibration

(b)

TABLE **PROJECTS**

id	name	main activity
1	Shaking table study	5
2	SERIES Project	8
3	UKNEES Project	5

Foreign key to
Project Activities table

TABLE **PROJECT ACTIVITIES**

id	activity name
5	Model calibration
8	Retrofit techniques

Figure 3.4. Example of normalisation (a) A denormalised table (b) The same table normalised

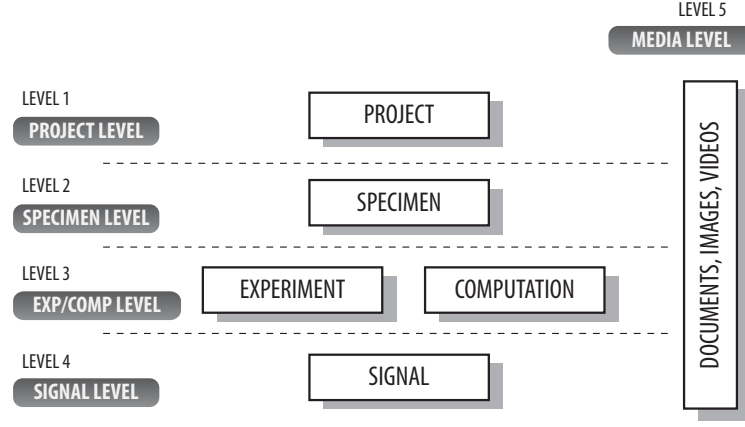


Figure 3.5. SERIES database hierarchy

vectors of physical quantities and their associated discrete times when they were recorded. Signals are used in the experimental execution group of Figure 3.1. For example, a signal could contain an accelerogram of an earthquake, with a vector of data for force values and its corresponding vector of time values when such forces happen. The term signal will be widely used in the chapters of this thesis.

The project level is a high level description of the project under whom a series of experiments are conducted. This level contains information such as the project description, people participating in the project and project objectives. The specimen level describes the structure element that is the target of the experiments. This description includes the materials the specimen is made of and their properties. The experiment/computation level contains all the details attached to an experiment, such as configurations for an experiment, loading characteristics, software used in computations, and information of sensors and devices. Finally, the signal level stores signals data. Note that the approach followed by the data requirements is that multiple experimental activities are conducted over a single specimen. Herein, specimens are hierarchically superior with respect to experiments and computations.

A table has been created for the main elements of each hierarchical level: *Projects*, *Specimens*, *ExperimentComputations*, *Signals*, *Documents*, *Images* and *Videos*. Other additional tables have been created, for a total of 66 tables in the data model. In order to simplify both the model and query operations, restrictions have been applied over the number of elements that can be related from one hierarchical level to another. Projects can be related to multiple specimens, but a specimen can only be related to one project. Likewise, specimens can be related to multiple experiments and computations, but an experiment or computation can only be related to one specimen. Signals,

documents, images and videos do not have such restriction.

Experiments and computations have been stored in the same table, since the attributes that describe them are very similar. Likewise, instead of having different tables for the documents of projects, specimens, *etc.*, documents have been modelled in a single table. To relate documents with other elements, *junction tables* have been used (junction tables support many-to-many relationship, meaning that multiple tuples of one table can be related to multiple tuples of another table). The same configuration exists for images and videos.

Devices were split into two tables, *Sensors* and *Devices* because, although sensors can be considered a type of device, they normally play a distinctive role in the experiment. A hierarchical structure was created for the devices, to allow arbitrary relations parent-child between devices.

Because the data model is intended to be used to share experimental data, three privacy levels are used to tag the information: private to the organization itself, shared with other trusted institutions and public to the worldwide community. Privacy levels are very important to maintain control of the data by an institution. Some data were considered always private, such as the tables for internal files, devices and sensors. However, this privacy is not reflected in the data model, and privacy restrictions on these tables are delegated to the component that manages the data.

Extended information of the SERIES data model can be found in Lamata Martínez *et al.* (2010).

3.2.3 Validation

The SERIES data model has been validated by implementing it on a MySQL database². The validation consisted of the installation of the database at the Joint Research Centre (in Italy) and IFSTTAR (Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux, in France), which input structural and geotechnical experimental data to verify the suitability of the data model. As a result of the validation, some structures in the data model were refined.

Following the previous validation, some potential optimisations were identified. For example, the device table allows the creation of complex hierarchies of device in parent-child relations by using a device configuration, as shown in Figure 3.6a. This can create some performance complications,

² <http://www.mysql.com/>

since relational databases do not deal particularly well with this type of structure. Also, the data model cannot implement efficient methods to prevent users from creating very complex or even incorrect hierarchies like the one shown in Figure 3.6b. Reading from this structure can be very inefficient, especially considering obtaining all the child elements of a device. One approach to obtaining the device tree is querying the database once to retrieve all first-level devices used in a device configuration and another one to obtain all device relationships in the configuration. Processing this information in a programming language will allow the creation of a list of devices and their links in memory to efficiently run a single additional query to finally obtain all device information. In many cases, this process can be quite expensive and time consuming: following the hierarchy to obtain all children nodes will require more than a single query.

Hierarchies have been categorized in four groups by Celko (2004): (i) static nodes and static edges, (ii) static nodes and dynamic edges, (iii) dynamic nodes and static edges and (iv) dynamic nodes and dynamic edges, and some optimization techniques have been proposed to access these tree-like hierarchical structures. One of these optimizations is the *nested set model*, represented in Figure 3.7a. This uses two additional numbers, one for the “left” part of the node and another for the “right” part, as depicted in Figure 3.7b. These two numbers define a range containing the numbers of all children elements. In the figure, *device C*’s children are those with the left value greater than 6 and the right value lower than 11. The nested set model works very efficiently with reading operations that obtain all children of an element. However, inserting or deleting new elements in the tree structure is quite expensive, since these operations might require to update many other elements (the “left” and “right” values of some other elements). To minimise the cost of inserting and deleting, non-sequential numbering can be used in specific cases. For instance, instead of numbering elements sequentially (1,2,3, *etc.*) non adjacent numbers can be used (10, 20, 30, *etc.*), so inserting a node might not always require updating other elements in the tree. The nested set model was not finally implemented in the data model, but it could be if reading the device hierarchy entails a performance issue.

Besides the devices hierarchy, other database aspects can be optimised. One optimization is related to reading operations to find out if some elements possess documents, since those require reading an additional junction table, as explained before. This extra reading is not always desirable in tables where the tuples are unlikely to be related to documents. In this case, an extra attribute

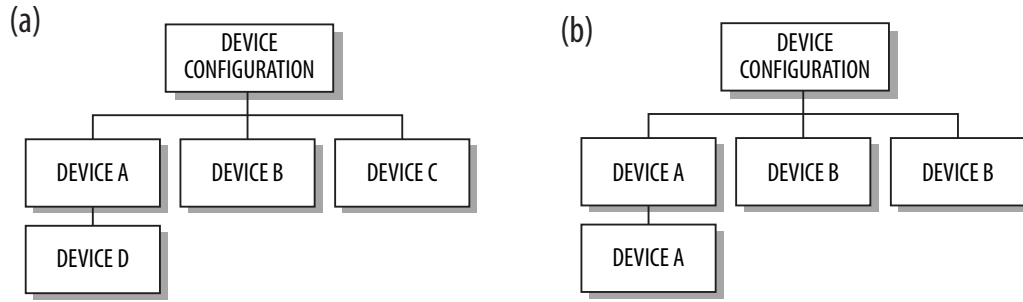


Figure 3.6. Hierarchy of devices (a) correct device tree structure (b) incorrect device tree structure

has been created to indicate whether the tuple is related to documents or not. An automatic process in the database has been established to set the attribute in the tuple when the junction table relates it with a document, and to clear the attribute if such relation does not exist.

Finally, performance optimisations based on table denormalization have not been considered since data requirements do not particularly require a low time response to justify the disadvantages of the denormalization.

3.3 Conclusions

This chapter has discussed some of the main experimental data requirements in earthquake engineering, and, as a practical implementation of a system to store experimental data, the relational database of the SERIES institutions has been briefly introduced. The database has been validated jointly with the Joint Research Centre and IFSTTAR, and some possible optimisations of the data model have been proposed. The following chapter utilises the SERIES data model as the foundation for the data integration of the European earthquake engineering institutions.

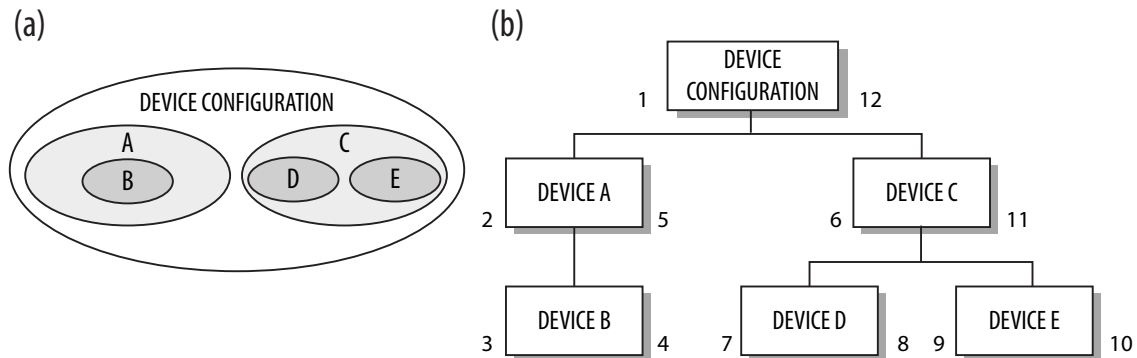


Figure 3.7. Optimisations on tree-like structures in a relational database (a) the nested set model (b) implementation of the nested set model in the device hierarchy

Chapter 4

Data integration in the European laboratories

The previous chapter has discussed the creation of a data model to store experimental data. With a common data format for the European earthquake engineering institutions, mechanisms have to be enabled to use and share the data. This involves interfaces to mediate both with humans (to interact with a repository by adding and extracting information in a user-friendly way) and computers (to exchange information in an automatic, structured way). Enabling computers to use the data opens research opportunities to share data and integrate them with other data sources, which enriches the global information dissemination, quality and completeness.

The European earthquake engineering laboratories have typically worked independently, establishing frequent collaborations but dealing with experimental results locally, with their own internal processes and data formats. Moreover, most laboratories¹ have not stored results in a computerised, uniform way and some of the laboratories employed no designated technological storage infrastructure at all. This is the case for many of the twenty-two leading earthquake engineering institutions participating in SERIES.

One of the main challenges is the fact that the SERIES institutions are very different from each other. Albeit they all belong to the same earthquake engineering community, they have different objectives, engineering infrastructures, capabilities, working languages, hardware and software platforms and they create and utilise data from many different types of experiments.

This chapter discusses the creation of a technical infrastructure called the **SERIES Virtual Database** (SVDB; Lamata Martínez *et al.*, 2013), which aims to support the data integration of the European earthquake engineering laboratories and create channels to share the data with the whole earthquake engineering community. The chapter is structured as follows: Section 4.1 identifies

¹ The concepts earthquake engineering institution and laboratory will be used interchangeably in this thesis.

a list of requirements for the system. Section 4.2 presents the architecture of the SVDB, whose implementation is discussed in Section 4.3. The validation of the system has been conducted in a real environment and is explained in Section 4.4. Some results and lessons learnt are discussed in Section 4.5 and, finally, some conclusions are presented in Section 4.6.

4.1 Basic requirements

Some requirements have been identified:

- Institutions must be in control of their own data (RQ1).
- Users downloading large files of data must agree with a legal policy. (RQ2).
- Hardware and software at institutions might be heterogeneous (RQ3).
- A single access to the data must be provided for the community (RQ4).
- Because of limited human resources, the system must be as automated as possible, requiring as little human interaction as possible (RQ5).
- Institutions might define data that are only shared within SERIES institutions (RQ6).
- Disparity in the technical development of the institutions must be minimised (RQ7).

The amount of knowledge that a laboratory produces is difficult to estimate. A busy laboratory can complete six projects a year, which might generate metadata in terms of megabytes and large files in terms of gigabytes. How often these data are updated is not predictable, but in any case they are not intensively updated. Moreover, the data do not need to be shared immediately or within specific time constraints. The number of users is not predictable, but it is a specialised community. Similar platforms have registered a number of 3,135 registered users and a total of 118,016 registered and unregistered users² (which might not be unique) as of September 2014.

The following sections discuss the system architecture, that will meet this summary of basic system requirements.

² <https://nees.org/usage>

4.2 Architecture overview

Existing architectures to store earthquake engineering data have typically used a single, centralised repository (see Chapter 2). However, this implies that data control must be delegated to a single institution, which might conflict with requirements RQ1 and RQ7.

Supported by the fact that SERIES institutions have to be in control of their own data, and in the light of the differences in institutional practice within the SERIES consortium, the decision was taken to develop a *virtual database*³, in which data are stored locally at participating laboratories but accessed via a single, centralised gateway. The aim is that the distributed nature of the database should be invisible to the end user, whose experience should be similar to accessing a single data repository.

This decentralised approach is quite different from that typically adopted in earthquake engineering, and has been encouraged mainly by its European setting. The main component of the system is not a central repository, but the individual laboratory. Instead of transmitting data to a central file store, laboratories work for their own benefit, which also contributes to make any economical investment in hardware and software worth it in the long term, not just valid for a specific project. All the information is stored and controlled locally. Sharing mechanisms are created to participate in the virtual database, but laboratories decide what and what not to share. This approach also avoids the need to motivate a dispersed scientific community to feed a centralised database, which anecdotally has proved to be a more difficult and complicated task than at first imagined.

The Series Virtual Database (SVDB) is the infrastructure that allows access to the SERIES European laboratories' data through a single user interface, and enables data integration between the different SERIES laboratories. Supporting the chosen infrastructure is a series of design decisions which established a balance between the different technical developments required by different laboratories. This design is presented in this section.

The rest of the section is organised as follows: Section 4.2.1 introduces the constituents of the SERIES Virtual Database. The architecture overview of the system is given in Section 4.2.2.

³ The system developed is described as a “virtual database”, meaning that it provides single-point access to different distributed data sources (many data sources are accessed using a common method). This term avoids ambiguity with a “distributed database”, which is a single source of data that is normally stored in a geographically distributed form, as systems described by Chang *et al.* (2006) and DeCandia *et al.* (2007).

Sections 4.2.3 and 4.2.4 describe the main constituents of the SVDB, which are the distributed data sources (represented by nodes) and the centralised access (represented by the central site). Security considerations for the system are discussed in Section 4.2.5.

4.2.1 Constituents

The main constituents of the SVDB are the following:

- **SERIES nodes**, which are the producers of the experimental data. They form a network of trusted nodes. Physically, the nodes are distributed around Europe (UK, Italy, France, Greece, Portugal, *etc*). SERIES nodes will be also called SVDB nodes, SERIES institutions, laboratories, nodes or partners in this thesis. They are discussed in Section 4.2.3.
- **SERIES central site**, which consumes the experimental data from the different nodes and provides a user-friendly, unified access to these data for the community (as required by RQ4 in Section 4.1). The SERIES central site, also called SVDB central site, is located at the University of Patras (Greece), and it will be discussed in Section 4.2.4.
- **SERIES users**, also called external users, which represent the earthquake engineering community accessing data from the SERIES Virtual Database.

The general architecture of the SVDB is discussed in the following section.

4.2.2 General architecture

The architecture of the SVDB is characterised by a *centralised access to distributed data sources*. The decision of having distributed data sources instead of a centralised data source is justified by multiple reasons: (i) The European laboratories are required to retain full control of their own data. This requires a data integration scheme based on routing access to the data, and not collecting it all in one place. (ii) A decentralised architecture encourages the enhancement of the technical computing infrastructure in each laboratory, making investment locally worthwhile even if in the end no data are provided by the local site (as required by RQ7 in Section 4.1). (iii) The political

context is significant here. Places using centralised repositories, such as the United States, might enjoy a territorial and political unity that does not exist in the European Union.

Having a centralised access to the distributed data sources is desirable since it provides a single route to the data, and enables operations, such as searching over the data as a whole instead of accessing each laboratory's data separately. Users will also need to become familiar with a single interface, instead of having to learn how to use several different ones. Despite these benefits, one disadvantage of a central repository remains: the centralised access is a single point of failure. Even if all the distributed repositories are available, they cannot be accessed if the centralised access is off-line.

The distributed data sources are represented by the SERIES nodes, managed and controlled by their respective institutions, whereas the centralised access is offered by the SERIES central site. Note that the primary component of the SVDB is the node - not the central site. External users access all the information by connecting to the central site, which must have a user-friendly interface to access the data.

The communication between nodes and the central site has been done by using the client-server model, in which one computer is the server and provides services to the rest of the computers, known as clients. The SVDB is an example of a distributed system, which has been defined as "a collection of distinct processes which are spatially separated, and which communicate in turn with one another by exchanging messages" by Lamport (1978). Within distributed systems, SOA (Service Oriented Architecture) is an architectural paradigm that focuses on organizing and utilising distributed capabilities of heterogeneous systems under the control of different owners (MacKenzie *et al.*, 2006). This reference model allows interoperability between different systems, which satisfies the European requirements perfectly.

Therefore, the SVDB is a client-server, service-oriented architecture, in which nodes provide the necessary mechanisms to access their data, behaving as servers providing services for a single client, the central site. This aspect is a fundamental difference between this and other systems governed by a centralised component, and allows at the same time promoting sharing of data as well as coping with the strong sense of ownership laboratories usually have for the test results they produce. This schema is shown in Figure 4.1, in which the external users (the wider earthquake engineering community, who seek validated, organized experimental data) use the centralised

access implemented by the central site to access all nodes' distributed information.

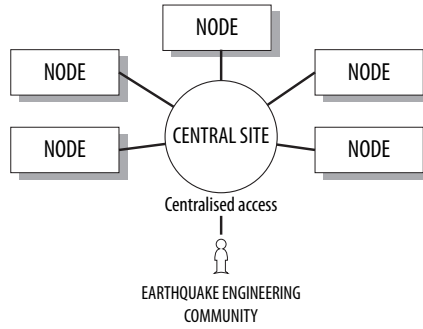


Figure 4.1. The SVDB centralised access to distributed data sources represented by nodes

In order to minimise access time to the data, the central site keeps a repository to cache data collected from the SVDB nodes. When external users access the centralised data, they do not have to wait until the data are collected from the distributed sources, and the local copy in the central site can be used. However, although the central site can provide much information directly, large files such as videos, documents and images are stored in and accessed from the nodes, as described in Section 4.3.2. This satisfies requirement RQ1.

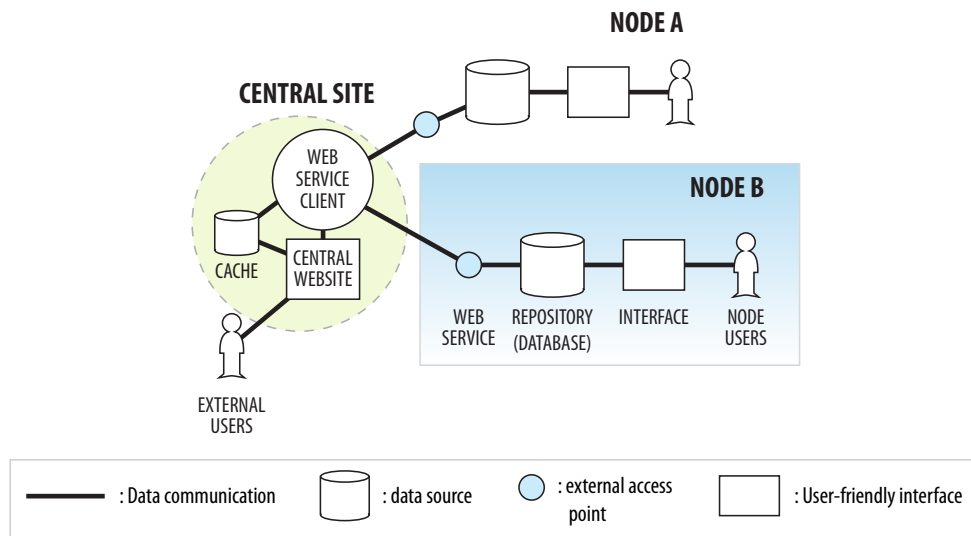


Figure 4.2. The SERIES Virtual Database architecture

Data integration is achieved by defining a common list of services that each node must provide, as discussed in Section 4.2.3. Any node that joins the virtual database must at least provide these services but is free to provide additional ones. The key component in this data integration design is the use of “contracts”. The contract formalised the list of services, by defining the

operations that can be transacted, and specifying the input and output information required by each operation. An example of operation is “getProject”, which returns a project’s data based on a provided project identifier. As long as a piece of software complies with the contract, the system will operate correctly. This creates an extremely flexible architecture, since different data sources, programming languages and hardware can be used to implement each server. The effort of creating each implementation falls on each site. If a node site wants to extend the software, or create its own, the costs are borne by the node.

The general system architecture of the SVDB is shown in Figure 4.2, and its different components will be described in the following sections.

4.2.3 SVDB nodes

The node is the principal component of the SVDB architecture, being the actual source of information. Laboratories are free to choose the details of the node’s internal data structure most appropriate for them, as long as the node complies with the Web Services contract (Booth *et al.*, 2004). This section discusses the specific node design depicted in Figure 4.2, that divides the node into a series of independent component modules: (i) Web Services software, which provides the services for computers to access the data of the node. (ii) Repository, which formally stores experimental data in a database. The repository has been implemented as a relational database in the format discussed in Chapter 3. (iii) Interface, which allows laboratory users to interact with the repository in a user-friendly way. It is suggested the interface is developed as a Web interface, that permits to avoid client software installation and just use a Web browser to provide and visualise data. (iv) Node users, who generate the experimental data.

In this node structure, node users use the interface to feed the local repository with data, which are automatically published through Web Services. This architecture has been designed so from the moment the node user provides information to the local interface, no other human interaction is necessary to make this information available for the external users. This both avoids human mistakes and saves efforts, and satisfies requirement RQ5.

4.2.4 SVDB central site

The central site is the entry point for the earthquake engineering community to access the data. Its role is important because it centralises access to the information, but it does not have to be very complex internally.

The central site serves data by means of the *Central Website* component depicted in Figure 4.2. This Website has been named the Data Access Portal (DAP) in SERIES. The central site accesses the different nodes by means of the *Web Service Client* component of the figure, and it keeps a copy of the nodes' metadata in a local repository, to minimise access time for the external users. Herein, most of the data requests can be served locally without the need of creating continuous network traffic to the nodes. This impels the central site to define policies about the updating process of the cached data. Although most of the data is served from a cached copy in the central site, large files (such as documents, images or videos) are indexed at the central site and served directly from the respective nodes to avoid unnecessary network congestion. This also avoid the need of great storage capability at the central site and the difficult management of legal issues, such as data ownership transfer, but forces nodes to provide additional components to serve large files to external users, as defined in Section 4.3.2.

Therefore, the central site operates more like an index rather than a centralised repository: storing metadata of large files such as signals, videos, documents or images but providing references to their main content. These references link to the node that stores and serves the large file.

Performance (meaning short response times) must prevail over efficient management of data space in the central site. For this reason, central site components should be very light, and data structures in a relational database should be *denormalised* (a definition of *normalisation* is provided in Section 3.2.1), which is a common technique to improve performance of data access in a relational database.

4.2.5 Security considerations

In a distributed system using public networks such as the SVDB, there are many important security issues that need to be considered (Shirey, 2000): (i) *Authentication*: The process of identifying a user. For example, a node must ensure that a request comes from the central site, and not from an

intruder. (ii) *Confidentiality*: Ensuring that information is accessible only to those authorized to have access. This is usually achieved by using cryptography. (iii) *Integrity*: Ensuring that information has not been modified while it travels on the network, or that modification can be detected. (iv) *Accountability*: The ability of a system to identify and register user activity. (v) *Authorization*: The process of determining if a user is allowed to access a resource (and how he/she can access it) or perform an operation. For example, external users can only access information marked as public. (vi) *Availability*: The capacity of a resource or service to be accessible or operative. For example, the central site is accessible by external users or the nodes can be accessed by the central site.

All of these security measures can be implemented by using a wide range of mechanisms (*e.g.* the digital signature used in Section 4.3.2 is a mechanism that deals with integrity and authentication).

The communication between the different laboratories is conducted on public and semi-public Internet networks, so the transmitted data should be protected from unauthorised access (*i.e.* confidentiality must be ensured). Also, the information should be protected from any unconsented modification (*i.e.* integrity of the data must be guaranteed). To implement confidentiality and integrity, direct authentication with public key certificates over the *HTTPS*⁴ protocol has been used. Herein, the communication between the central site and the nodes is encrypted. Public-key certificates are based on public-key cryptography, which is explained in Appendix A.

Every system in the SVDB (nodes and central site) has a public key and a private key. Private keys are kept in secret, but the public keys are exchanged between the central site and each of the nodes. To validate the requests made by the central site, every time the central site accesses a node services, it is first authenticated by the node with support of the central site public key stored at the node and the central site private key stored at the central site. If the authentication is correct, the communication is secured by encryption, with the support of the public and private keys of the node and the central site.

In the current SVDB, no user can access the node information without using the single access gate on the central site. Thus, the node should be able to distinguish between clients, identifying if its client is allowed to access the information (authorization). The only client of a node in the SVDB is the central site, and the node should authorise any access to its local information. To implement authorization, public key certificates have again been used. Since the “private” key is

4 Hypertext Transfer Protocol Secure, a protocol for secure Web communication.

secret to one entity, only this entity is able to create a message that can be decrypted with its “public” key.

Both node and central site should be accessible and operative (availability). However, whereas a node could stop working for some time without affecting the SVDB operation, any service disruption in the central site will lead to a whole system disruption. It is especially important to protect the central site from malicious external activity, considering it is the main interaction gate where the user inputs data. Both the central site and the node must restrict the resources used in any request.

Communication with the central site is not the only thing that must be protected. Security measures were also taken on the nodes. The use of a DMZ infrastructure (Demilitarized Zone) has been proposed for every node, protecting public services from the rest of the internal network with a *firewall*⁵. It is important to realise that some attacks could come from inside the University or organization (*e.g.* from the node users). A common mistake is to develop security measures based on the assumption that every potential attack must come from the outside world.

Finally, it should be noted that security does not only involve technical issues. Node users must be trained to understand the security structures and practices used, since many security mechanisms are useless if users do not understand how to use them. Furthermore, understanding security practices implies understanding that security is not a *product* but a *process*, as described by Schneier (2012).

4.3 Implementation

This section discusses the development of some of the components described in the previous section. Specifically, the nodes services are discussed in Section 4.3.1, whereas the node component to support the download of large file is discussed in Section 4.3.2.

The central site implementation is not discussed in this section, since it has been entirely implemented by the Human Computer Interface group of the Department of Electrical and Computer Engineering, at the University of Patras (Greece). Implementation details of the central site has been described by Ioannidis *et al.* (2012), and the user manual for the central Website

⁵ A piece of software or hardware that monitors communications and accept them or reject them based on a security policy.

(also called the DAP) can be found in Ioannidis *et al.* (2012b).

4.3.1 SVDB node services

Node services are the software component that serves data requests from external systems, such as the central site. They have been implemented with a service-oriented architecture by means of *SOAP Web Services*, as described by Miltra *et al.* (2007). Web Services exists to allow machine to machine communication between heterogeneous systems on the Web. Services are defined by means of WSDL (Chinnici *et al.*, 2007), an XML-based language to describe operations that can be called, messages exchanged for each operation and data types of the attributes of the exchanged messages. The WSDL component is what actually provides the integration of data, because every node might comply with it, independently from its internal structure.

The services provided by a SVDB node have been categorised into four groups:

- **Testing Services**, to check the correct operation of the node, such as correct functioning of the Web Services and the local database. These services are especially useful to trace possible errors externally, when a node joins in the platform.
- **Data Services**, to deliver data from the node database. These services have been defined according to the following parameters: (i) approximation of the size of the data sent per service, (ii) number of data services that have to be called in an average operation (*e.g.* the number of services required to obtain a full project), (iii) relation between data structures (*e.g.* it might not be useful to obtain material data without its properties), and (iv) database independence for a loosely coupled system (*e.g.* even if the data model described in Chapter 3 defines identifiers for structural components, other databases might not; the services should work independently from specific database implementations, as required by RQ3). Data services are listed below.
- **Updating Services**, related to potential updates of data by external systems. For example, a service that informs of projects whose data are not expected to change in the long term, and therefore might not be necessary to update often.

- **Control Services**, to control the system operation (*e.g.* informing a node that a process to collect its data has been started) and to obtain system information (*e.g.* version of the different components).

Data services conform the core of the services, whereas the rest of the groups support management aspects. The services to obtain data are: `getProjectIDs`, `getProjectData`, `getSpecimenData`, `getStructuralComponentData`, `getExperimentData`, `getComputationData`, `getExperimentLoadingData`, `getComputationLoadingData`, `getProjectDocuments`, `getSpecimenDocuments`, `getExperimentDocuments`, `getComputationDocuments`, `getSpecimenImages`, `getExperimentImages`, `getComputationImages`, `getExperimentVideos`, and `getComputationVideos`. The name of the service describes the type of data that is offered by the service. One aspect that makes the services loosely coupled with the underlying database containing the data is, for example, the services *getExperimentData* and *getComputationData*, which serve data that come from the same table in the SERIES database (see Chapter 3) but they are defined as different services for data storage independence. This loose coupling is also reflected in the node components. Herein, the communication between Web Services and the node database has been implemented by a *Java interface*⁶. If a node requires to replace its current database for another one, all that has to be done is the creation of a Java component that implements the Java interface and accesses the new database, without any modification on the Web Services component. This satisfies requirement RQ3, allowing different database systems at different nodes.

Data services provide mechanisms to obtain node's metadata. For instance, the service *getProjectDocuments* will deliver information such as the document title, document size, authors, format, document summary, *etc.* for the documents of a given project, but it will not deliver the document itself (for the reasons explained in Section 4.2). Thus, large files are downloaded by using an additional component that will be discussed in Section 4.3.2.

As a final note, nodes in the SVDB normally serve a single client, the central site, but services are designed to allow new clients in future (*e.g.* laboratories may wish to establish collaboration with other laboratories without going necessarily through the central site).

⁶ A Java interface is an abstract definition of methods that should be coded in all classes implementing the interface. This way, it is guaranteed that any class implementing the interface will provide methods to deal with specific functionality.

4.3.2 Large file download management

In the SVDB, nodes store and provide access to their large files, understood as documents, images, videos and signal data. This implies that nodes have to provide a mechanism for external users to download the large files. The process is the following: external users access the SERIES DAP (the central site Website) and browse to find some data of interest. Amongst these data, the DAP shows links to download large files. When the user accesses one of these links, the DAP shows legal terms and, after acceptance, the user is redirected to the node that stores the file to automatically download the file. This communication with the node is transparent to the external user, who seems only to deal with the central site. With such a mechanism, large files, which are the effective valuable result of the node's work, are neither stored at the central site nor transferred through it during the download process.

This process offers two potential problems: (i) Nodes have to allow external access to users, which might imply security risks (note also that, contrarily to the central site, external users are not authenticated). (ii) Users can create copies of the download links, and surpass the acceptance of legal terms to download and use the data, which is a requirement (RQ2, described in Section 4.1). To minimise these risks, a download mechanism has been created, consisting of the use of a special link that cannot be modified arbitrarily without being noticed. These links are created by the central site and validated by the different nodes.

The link structure is presented in Figure 4.3. The *Node File Manager* represents the address (*IRI*) of the component that manages download requests. This is followed by some parameters. The *Object ID* parameter represents an identifier that, together with the *Object Type*, identifies unequivocally the object to be downloaded. The *Object Domain* parameter specifies the node for which the link was created. This prevents malicious users from reusing a link that was intended for one node on a different node. The *Request Rights* parameter identifies if the link was created for a user belonging to a SERIES institution or for a public user. This enables privacy of data, as required by requirement RQ6. The *Request Timestamp* records the time at which the download request was created, which helps to determine whether the link is expired or not. This enables links to expire after some time, forcing users to access the DAP again to obtain a new valid link. The central site only records the request time and nodes can check whether the request has expired or not, according to their own criteria. Finally, to assure the integrity of these parameters and that



Figure 4.3. Download link to request large files from a SVDB node

the download request is legitimate, the central site creates a *digital signature* in the *Signature* parameter, allowing the node to detect any unauthorised modification on the link. Digital signatures work by means of public-key cryptography (see Appendix A for extended information).

This download mechanism validates that the download of the file was requested through the central site, which requires users to accept legal download terms, and that the link comes from a trusted system.

4.4 Validation

The validation of the SERIES Virtual Database infrastructure has been conducted in two different scenarios:

- By conducting multiple tests between two nodes at the University of Oxford and the Joint Research Centre (in Ispra, Italy) and the central site at the University of Patras (Greece). These tests were conducted during 2010 and 2011, and verified the feasibility of the SVDB architecture defined in this chapter.
- By implementing SVDB nodes at eighteen of the SERIES institutions, which stored some experimental data that have ultimately being served by the central site. This validation has been conducted from May to August 2013.

Tests between the universities of Oxford and Patras and JRC validated the correctness and efficiency of the data services. These had to be adjusted to the final form presented in Section 4.3.1 to avoid network bottlenecks and permit repository independence. Thus, multiple services to collect data (*e.g. getStructuralComponent, getMaterials, getMaterialNominalProperties* and *getMaterialActualProperties*) were grouped in more compact services, (*e.g. getSpecimen*, which include all the aforementioned data).

Security measurement tests were conducted to validate, amongst others, that data were sent encrypted, that requests of systems that were not trusted were rejected, that alteration of download

links were noticed and rejected, that expired download links were rejected, that illegitimate download links were rejected and that external users could download large files by using legitimate links. The tests were eventually successful but, however, complete security cannot be proved.

The most relevant series of tests were conducted to evaluate the *updating process* from the central site. The data served by nodes through the data services change over time. If an external system accesses the node at the moment the data are demanded, the data are always up-to-date. However, if the external system creates a local copy of the data (such as the caching conducted by the central site to improve performance) a process to be regularly updated is needed. Issues related to this process are how to update, discussed in Section 4.4.1, who should start the process, discussed in Section 4.4.2, and when it should be done, discussed in Section 4.4.3. Finally, Section 4.4.4 discusses potential problems on the adopted updating solution.

4.4.1 How to update

Different approaches were tested to evaluate how the updating process should be conducted. The first approach to be investigated was to provide update information from the node. This means that the node should detect and provide details of any data modification. The modifications were time dependent and this is especially problematic when more than one client is involved. Some tests were performed using this first approach: for independence of the database used, the database was provided with automatic procedures to record when a change was made on a new database table. New services were created at the node to collect the update information. Testing of this approach showed that it was extremely inflexible and very complex to extend and maintain, especially considering the data model.

The second approach was to use the normal services to obtain all the information directly from each node. When using this method, the information is already up-to-date. A test was done at the Joint Research Centre, providing the approximate quantity of data that they produce in one year (from around six projects). The size of the metadata with no large files was in the order of kilobytes, and the whole updating process was completed in an acceptable amount of time (60 s. in this case). Even if the whole updating process took many times longer than this, overnight this is not a substantial load and therefore quite acceptable. This shows that apparently unsophisticated

solutions can be the most appropriate option, under specific circumstances.

To improve this solution, a new attribute was set at the Project level, allowing a project to be “closed” when no project modifications are expected in the long term. In this way, the central site will only request update information from open projects.

4.4.2 Who should initiate the update

According to the system design, where the data lie in the nodes, the external systems (the central site) should be the initiators of the update process. This way the intelligence of triggering the update process is collected into a single place, releasing nodes from the task and responsibility of updating, including dealing with any update errors. Thus, the updating process has a single update policy, rather than one policy for each node, and it is controlled by a single component. Moreover, if a node has multiple clients, it is not necessary to keep an update schema for all of them. However, some mechanisms have been enabled to allow update requests when a node does consider it necessary.

4.4.3 When to perform the update

The update process should be performed when the workload is low for both the central site and the node. The central site could have activity during the whole day, but the nodes are more likely to have workload only during the usual daytime hours. The update process is therefore initiated on each node’s local time, when experimental updating is unlikely - overnight. The geographical distribution of laboratories across different time zones eases the load on the central site.

4.4.4 Potential updating issues

The services that collect information from the node are basically stateless operations. This characteristic provides the system with simplicity, scalability and flexibility. Nevertheless, the stateless nature of the services defined does not guarantee the information consistency of the set of operations that must be called to collect all the information of a project. This is called a *race*

condition, which is defined by Göetz *et al.* (2006) as “the possibility of incorrect results in the presence of unlucky timing”.

To explain the race condition, let us consider the following example illustrated in Figure 4.4. In a typical updating process, data stored at the project level in the database is requested by an external system (*e.g.* the central site). The data are sent by the node and, while these data are being processed by the external system, some project level data are changed in the node by a node user before a later call from the central site to the specimen level information is made. Because of the unfortunate timing, the node user could have changed project level data that is not in accordance with the specimen level data sent to the external system subsequently.

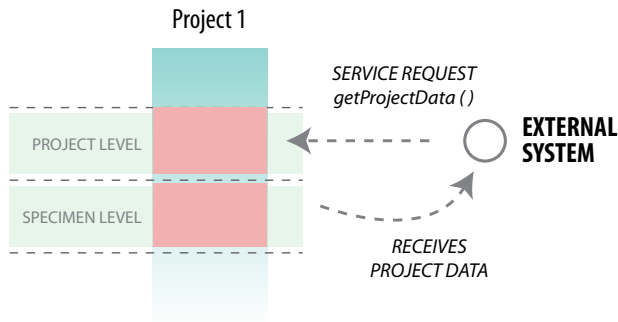
Solving this race condition implies a reduction of the number of requests that can be served at a given time by the node. This is not a problem in the current SVDB, which has the central site as the only single external system, but it can entail a limitation in future. Herein, considering the extremely unlikely probability of the race condition to happen (which must entail a data modification that has a significant impact), it was decided that, by design, the data requested is up-to-date according to a specific level and the time of the request. This means that every service provides the most up-to-date data up to the requested time, independently from any other previous request.

For the cases where this race condition must be avoided, the solution is to create a copy of the original database, at a single time, when the repository is locked to write operations. Later modifications to the database are not immediately propagated to the copy. Node services must then only consult the copy of the database, which is unchanging and therefore remains consistent. This solution causes three complications: (i) the original database must be locked to writing while the copy is made, (ii) data served by the node services might not be totally up to date, and (iii) before updating the copy, the system must assure that there is no external system in the middle of any data collection. These problems are common to most database schemas that allow querying different layers of related data independently.

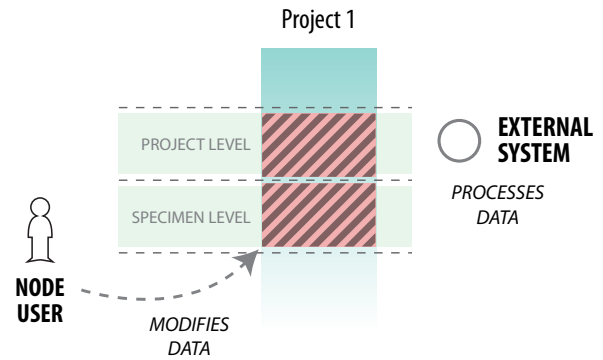
4.5 Results

Some of the most important results from the SVDB research are presented in this section, which shows findings from both the social and the technical side.

(a) The external system requests project data



(b) In the meantime, a node user modifies the data



(c) The external system requests specimen data and obtains the modified data

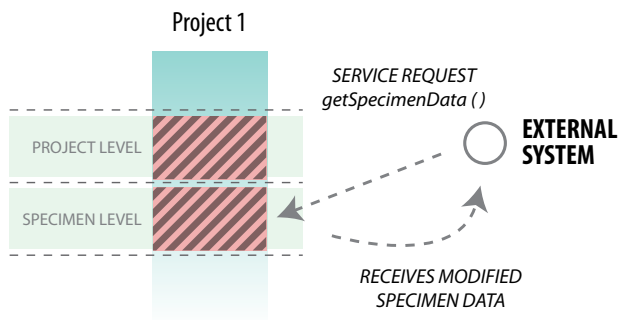


Figure 4.4. Potential race condition in the updating process

4.5.1 Implications of large projects

During the development of the SVDB, it was shown how difficult it is to come to timely agreement between twenty-two institutions. The complex process of finding common solutions for institutions with different motivations and objectives presented communication and negotiation challenges as great as the technical ones. Three methods were followed to achieve the project milestones:

- **Task forces.** To promote decision making and fast developments, the work was initially conducted by small teams of two or three very active participants (normally between three and six people). This team, or task force, worked on a complete draft, that is distributed to the rest of the project participants for testing, comments and approval. This use of very active small teams was crucial to the success of the project. The draft had a hard deadline for revision, since procrastination is very common when many participants are involved. If a participant does not produce any comment during the revision period, agreement was automatically assumed to avoid delays. The general SVDB architecture was designed at

Oxford as a draft, completed with a collaboration of some other institutions, developed in a task force, and tested by including additional participants gradually.

- **Agile methods** (AGILE, 2001), as a primary development methodology. Agile processes are feedback-oriented, and give priority to operative software over exhaustive documentation. They promote collaboration of teams over contract negotiation (Highsmith, 2002). Agile concepts have improved development speed and enabled a feasible development in a very variable environment where requirements change often and converging ideas are difficult to achieve. Agile methods were found ideal for the size of the project and they fitted very well into the task force functioning. Although a formal output documentation was finally produced, more informal documents were used during the development of the task. This saved much time producing, revising and approving formal documentation. Note that Agile methods cannot substitute project planning, but can help to speed the development of some of the tasks.
- **Proofs of concept.** Technical solutions were firstly tested by means of proofs of concept. Proofs of concept helped to set up the development environment, estimate development time, check communications, measure performance and demonstrate feasibility. They allow the system's viability to be checked to see if the system has been correctly designed before investing more resources, especially time and money.

4.5.2 Management of distributed components

The SVDB, as a distributed system, consists of geographically separated components that are controlled autonomously by their respective owners. Activities conducted in this distributed environment needed frequent collaborations to be conducted. It was experienced that those activities often required more than double the time that the activity would normally take to be completed. Because of this distributed environment, errors that happened during testing were extremely difficult and tedious to trace.

Regardless of how robust a system is, it was also experienced that components in a distributed system will eventually fail, and the system should be designed with appropriate mechanisms to

address this situation. Occasionally, the SVDB experiences unavailability of some nodes, and automatic mechanisms to monitor and report this issue have not been implemented.

4.5.3 Importance of user-friendly interfaces

Node users are the producers of data, which are the reason of existence of the SERIES Virtual Database infrastructure. The interaction between users (humans) and data repositories (computers) requires a user-friendly interface that supports the process of data storage and data presentation, ideally releasing users from tedious tasks with the support of automation tools.

During the validation that required the storage of data by eighteen of the SERIES institutions, it was experienced that supportive, user-friendly interfaces play a key role to encourage users to store the data. The original design of the SVDB underestimated the importance of good support for user-friendly interfaces, and accomplishing the task was more complicated than expected.

On the side of data presentation, the DAP (the central site Website) shows experimental data in a similar way as they are stored in the data model presented in Chapter 3. User interfaces should present data to users in the most convenient manner for humans, not attending to the manner the data are better stored on a data repository. In particular, data shown by the DAP should look similar to the informal data requirements mentioned in Section 3.2. Also, the DAP is perceived as slow, which is probably due to the high internal complexity of the central site, as presented by Ioannidis *et al.* (2012). Database tables in the central site are not denormalised and they keep unnecessary constraints for a system that only acts as a cache, especially considering that such constraints are already forced at the different nodes' databases.

4.5.4 Importance of software contracts

Software contracts, meaning a piece of software that formalises definitions that must be agreed in the interaction of two components by means of the messages that a component can or must process, played a key role for the creation of flexible, modular, loosely coupled components. Such is the case of the *WSDL*, that implemented the SVDB integration, and the *Java interface* to communicate the Web Services and the database, both discussed in Section 4.3.1.

Thanks to software contracts, every component of the SVDB could be replaced, maintaining a seamless system operation. For instance, a node can replace its database for a different database management system and still use the Web Service component to serve central site requests.

4.6 Conclusions

This chapter has presented the SERIES Virtual Database, a novel infrastructure in earthquake engineering that enables data integration within the European earthquake engineering institutions. This virtual database has enabled a formal way to exchange data between institutions as well as present them to the whole earthquake engineering community.

This virtual database is laboratory-oriented, based on a centralised access to many distributed data sources instead of the more traditional approach of a centralised repository. This design works thanks to contracts, which are key elements in the system. Contracts define operations and their expected input and output to enable integration, modularity and flexibility. The different laboratories publish their information by using a service oriented architecture through Web Services.

Two main elements define the system, the nodes that provide the data and the central site that provides the centralised access. The central site acts as a cache of each node's metadata. To obtain updated information from the nodes, the central site implements a corrective, non-selective update of open projects. The security of the communication between nodes and central site is done by using public key certificates.

Good organisation is essential to successfully conduct a project where twenty-two organisations are involved. Tasks were started by very active small groups called task forces, working to rapidly develop drafts that were consequently revised by the rest of the participants. Agile methods enabled quick group interactions in a heterogeneous environment where decision making was complicated. Finally, technical solutions were firstly tested by means of proofs of concept to demonstrate the viability of the solution and to help to estimate the effort required. Proofs of concept gave an idea of a working system in a reduced amount of time.

While European data integration represents a significant step forward, the next movement should go towards a worldwide data integration. This integration has been addressed in Chapter 5 and Chapter 6.

Chapter 5

A data model to support world-wide data integration

The *SERIES Virtual Database* discussed in the previous chapter has established a formalised infrastructure to exchange data between the European earthquake engineering institutions, which enables treatment of information in a uniform manner and brings a consistent, integrated source for European experimental data. Despite being an achievement of great relevance for the earthquake engineering community, its scope is limited to European research.

The extrapolation of this interoperability effort to a world-wide scale has never been addressed in earthquake engineering, and entails greater technical and social challenges. Additionally to the difficulties found in the initial SERIES European situation, an infrastructure to enable world-wide data integration has to consider well-established, existing data systems; disparate data processing practices, work methodologies and procedures; self-evident cultural differences, and the absence of a common funding body that enforces unified management and common directions. This latter point is one of the main differences between global and European data integrations.

To address the lack of data interoperability in earthquake engineering, and with the objective of extending this effort to other hazard mitigation disciplines, the prototype of an infrastructure called **Celestina Data** has been designed. This research has been presented in two different parts in this thesis:

- **The data model** (discussed in this chapter), which explores a method to formalise the experimental knowledge gathered by research institutions, by using two of the main earthquake engineering data models as reference (the one in SERIES and the one in NEES). Its ultimate objective is pursuing the integration of experimental data in earthquake engineering.

- **The distributed infrastructure** (discussed in Chapter 6), which deals with the systems and policies to implement the data model in the earthquake engineering institutions and to make practical use of the data integration.

Section 5.1 discusses the existing data models of SERIES and NEES, which provide data requirements for the creation of a global data model. Due to the differences in the existing data models and the distributed nature of the data sources, the use of Semantic Web technologies to support world-wide data interoperability has been explored. Its motivation is introduced in Section 5.2, and part of the technology, which is the foundation for the Web of Data, is briefly described in Section 5.3. The new data model, named the **Celestina Data ontology**, is described in Section 5.4. This model formalises earthquake engineering experimental data to enable machines to *understand* earthquake engineering knowledge and provides a Web-based infrastructure for data exchange. This ontology is the main outcome and contribution of the chapter. Section 5.5 discusses a brief validation over the new data model, that will be complemented in the following chapter. Finally, Section 5.6 presents the conclusions.

5.1 Analysis of existing earthquake engineering data systems

The data model discussed in this chapter has been inspired by the data requirements derived from existing implementations of earthquake engineering data models. The two main publicly available data models that have been studied are the SERIES data model, described in Chapter 3, and the NEES data model, that will be briefly described in this section. No relevant public data models have been found in other earthquake engineering institutions around the world, which also encourages the creation of the system described in this chapter, in order to make private data publicly available and shareable.

Currently, NEES provides access to the largest public data source of earthquake engineering experimental data by means of its *Project Warehouse*¹ (Hacker *et al.*, 2011). As of July 2014, the NEES Project Warehouse contains approximately 194 projects and 1,426 experiments. Although data requirements are a direct reflection from the Project Warehouse public interface, in order

¹ <https://nees.org/warehouse/>

to properly understand the data model, an analysis has been conducted directly over the NEES database, currently hosted at Purdue University in Indiana, United States.

The aim of the analysis was to understand NEES data, including: (i) data needs of the NEES community, (ii) purpose of each piece of data in the database, and (iii) technical data structures used as well as procedures and mechanisms to store data.

Unlike the *SERIES Virtual Database*, the NEES Project Warehouse is based on a centralised relational database. This database contains 263 tables, and the data model is organised in the hierarchy depicted in Figure 5.1b. Some similarities can be found when it is compared to the SERIES data model hierarchy of Figure 5.1a, but the content of the different hierarchical levels is quite dissimilar.

In order to conduct the analysis, the following steps have been taken: (i) Revision of the schema of the tables. This allowed understanding the semantics of the data. (ii) Inspection of table data. This reinforced the understanding of the data model and helped finding irrelevant tables and data. (iii) Conduct multiple tests interacting with the NEES Project Warehouse Web interface as a NEES user. By using the interface, it could be traced how data are actually stored and which data structures are in fact used. (iv) Meetings with Purdue team members, to resolve concerns and discuss findings.

Following the hierarchy of Figure 5.1b, the NEES project level contains general information of projects that conduct experiments and, in general terms, the NEES and SERIES project levels serve the same purpose.

The experiment/simulation level contains the overall information on experiments and simulations (referred to as *physical* and *numerical* activities, respectively, as described in Section 7.1), but

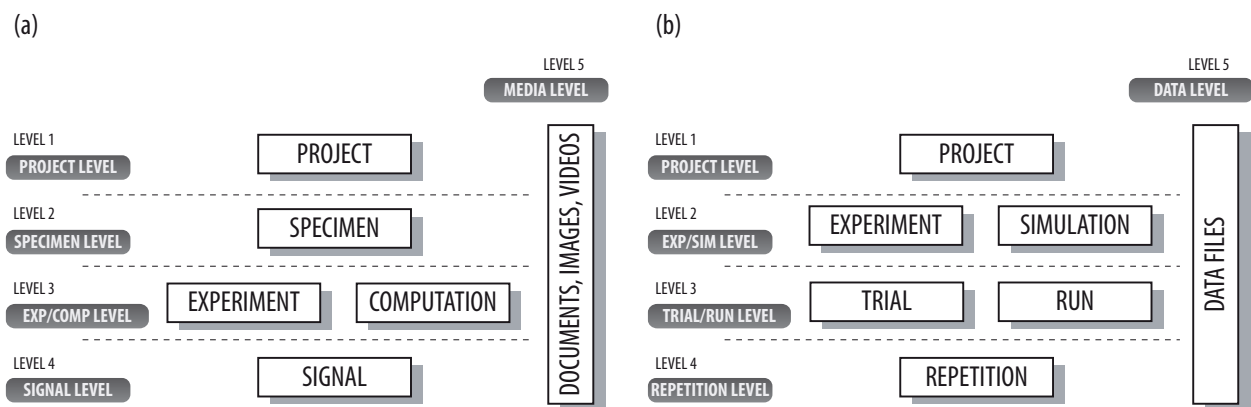


Figure 5.1. (a) SERIES and (b) NEES data model hierarchies

no specific information of their actual execution, which is provided at the trial level. One main difference from the SERIES data model is that the experiment/simulation level in NEES describes the specimen tested instead of having a different hierarchical layer as the SERIES data model does. The NEES data model considers that projects conduct a series of experiments, and each of them uses a particular specimen to obtain results. On the other hand, the SERIES data model considers that projects use a (potentially large) specimen over which experiments are conducted to obtain results. Both approaches are correct, and deciding between one or the other partly requires a subtle perspective on the execution of the experiments.

While the experiment/simulation level stores the experimental objectives and the general setup, the trial/run level stores the specific execution of a physical experiment (trial) or a numerical simulation (run). Trial and run information can be found embedded in both the experiment/computation and signal levels of the SERIES data model.

The repetition level contains the collected data, and there exists one repetition for each time the same trial is conducted under the same conditions.

Finally, the data level in the database is a mirror of the data directories in the file system, containing the directory structure and the links to all large files of the database. Data files are stored in the format used by the user, and there exist no transformation of data to a uniform “signal” as in SERIES, although, normally, CSV (Comma Separated Value) files are used for signal data. Any sort of media data, such as documents, images and videos, are also stored as data files in the data level.

The NEES database stores a large amount of metadata to compute statistical reports (number of files downloaded, user behaviour, *etc.*), which is internal and has no interest for this work.

Data in NEES have three privacy levels that are applied at the project and experiment levels:

- **Members** (private). The control of the access to the information is based on a custom list of research teams. Only people in specified research teams can access the information, regardless of their affiliation.
- **Users** (protected). Public access for registered users of the NEES Project Warehouse.
- **Public**. Everyone can access the data.

Unlike data in SERIES, where each node is responsible for the content of their data, data in NEES are “curated”, meaning that data go through a quality assurance process to guarantee the information is complete, relevant and uniform according to NEES standards, as described by Pejša (2012).

It should be noted that the data model to be created is not intended to be an integrated data model for SERIES and NEES, but to fulfil data requirements of the whole earthquake engineering community while being extendable to other hazard mitigation areas and scientific disciplines. Moreover, the new data model intends to improve the quality and versatility of existing experimental data and to fill data requirements that existing data models seem to lack.

There are several challenges in the attempt of converging both data models into a unified data view that can also accommodate world-wide earthquake engineering data: (i) The data are conceptually similar in NEES and SERIES data systems because the community has similar data requirements, but both data models follow different approaches and store some dissimilar information according to their research interests. (ii) There exists no formal definition of concepts of the domain in any of the models, which makes it difficult to define accurately the meaning of some concepts. (iii) The fact that the NEES database has been inherited and developed by different groups during ten years is patent, and the lack of documentation made the analysis of the data model rather complex and time consuming. NEES information has evolved to accommodate new data requirements over time but the underlying data structures have been kept, which makes information difficult to converge into a new model. For instance, information of facilities and organisations is stored in the same table when it is semantically different, and similar information about sensors is stored in two different tables depending on the procedure it was obtained with. (iv) Information in both data models is stored with different depth. For example, calibration information of sensors is more structured in the NEES data model than in SERIES, but calibration of other types of devices is more complete in the SERIES data model. Also, SERIES translates experimental input and output data into a common, uniform structure whereas they are stored in a less structured manner in NEES.

The information obtained from the analysis of the NEES database provided the understanding of the NEES data model presented in this section and used in the creation of the data model described in this chapter. An internal report has been created to document the database schema

with an entity-relationship diagram and to propose database optimisations, but such conclusions are not presented in this thesis, being irrelevant for the discussion. With a clear vision of NEES and SERIES data needs, the following section discusses technology decisions for the new data model.

5.2 Motivation for a new technology

Although the relational model discussed in Chapter 3 provides a valid infrastructure for earthquake engineering data storage (and it is indeed the technology supporting both the SERIES and the NEES data models), it lacks some features that make other data management methodologies worth investigating. This is especially justified in the situation of a world-wide data integration, which consists of highly decentralised, highly heterogeneous autonomous systems without a common management.

For instance, the relational model lacks inherent processes to exchange information in a standard way, which has to be solved with additional technologies. Despite the multiple standards around the relational model, in practice there exist variations in the way data is accessed by different relational database management systems, and integration of data in different distributed sources is not facilitated by the model itself. Data in a relational database is assumed to be complete (meaning that what is not asserted in the database does not exist), and changes over the established data schema sometimes entail a complex process and can be difficult to make. This is not beneficial for many scientific applications whose data models are difficult to express, feature sufficient complexity so that it is difficult for all parties to agree on, can substantially vary or evolve with time or are unknown at the data modelling phase.

Another reason to investigate new technologies is the necessity to formalise data structures with well-defined semantics. Consider the example of the *SERIES Virtual Database*, where data are made available by nodes in a structured way, by means of Web Services, and the information provided is structured in an XML message (Chinnici *et al.*, 2007). Questions such as *what a project is*, *how projects and people are related* or *how concrete is different from steel* might be addressed by practical software implementations but are not based on any logical formalism. From a computer perspective, only a hierarchy of typed data is exchanged. If the same information is exchanged in a slightly different hierarchy, the applications need to be adapted consequently, because

XML presents structured information, not an abstract data model. Although Web Services and XML are perfectly valid solutions for data exchange, other technologies might be better prepared for an infrastructure to manage data globally.

Data representing the source of experimental knowledge of the earthquake engineering institutions must be supported by formal rules that define what data can be represented, how different data are related each other and how data must be understood without ambiguity. These issues have been studied in the field of **knowledge representation and reasoning** of artificial intelligence, and logical formalisms have been used to address them (Lifschitz *et al.*, 2008).

The **Semantic Web** framework, part of the Web of Data initiative of the W3C (DATAWEB, 2014), is one of the most prominent efforts to apply knowledge representation and reasoning techniques. The Web of Data aims to enable integration and processing of data on the Web, so its technologies are natively designed to work in a distributed environment and to address the integration of data. These technologies will be explored and discussed in the following section as an alternative method to address a world-wide earthquake engineering data integration.

5.3 The Web of Data

The original Web was based on structured documents containing unidirectional links to other documents. Over the years, improvements were made, such as the establishment of a clear distinction between content and presentation in Web documents (CSS, 2014; Duckett, 2011). However, from a computer perspective, Web documents were no more than text, hierarchically structured in different headers, sub-headers, lists, paragraphs, *etc.* Web content lacked mechanisms to support data processing beyond the rules of the hierarchical structure of the document (Berners-Lee, 1999).

The Web of Data is an initiative to address this lack, by enabling computers to process data on the Web (see Hitzler *et al.*, 2009 for an overview). It is also known under the name of *the Semantic Web*, since it aims to provide “semantics” to Web content. The Semantic Web relies on several technologies to mark up content and define rules to process it, some of them based on knowledge representation and reasoning techniques. Some of these technologies are:

- **IRI** (*Internationalized Resource Identifier*; Duerst *et al.*, 2005), which provides a way to identify resources in the Web. IRIs are the basic method in the Semantic Web to identify

all element types uniquely, and, contrarily to relational databases where identifiers have only a local meaning, IRIs provide universal identifiers. IRI is a generalisation of URI (*Universal Resource Identifier*; Berners-Lee *et al.*, 2005) that extends URI encoding with additional non-English characters.

- **RDF** (*Resource Description Framework*; Schreiber *et al.*, 2014), which provides a graph-based data model for the Semantic Web. It is based on triples of elements and it is briefly described in Section 5.3.1.
- **OWL** (*Web Ontology Language*; Hitzler *et al.*, 2012), which is a language based on formal logics to define classes of things and their relations. These classes are part of a domain of interest, so they are relevant elements for a specific application. One of the prime motivations to use OWL is that it enables *logical reasoning*, which means that new information can be acquired from some known information (see Appendix C for more details). The conclusions that result from reasoning are called *entailments*. OWL is discussed in Section 5.3.2.
- **SPARQL** (*SPARQL Protocol and RDF Query Language*; SPARQL, 2013), which defines a protocol and a language to query data represented in RDF or OWL. It is presented in Section 6.4.2 in the following chapter.

There exist other Semantic Web technologies that will not be discussed in this thesis.

The Semantic Web is a well established initiative, with strong industry support involving many organizations, and with an intensive, active research programme (Horrocks, 2011), which has created numerous support tools: (i) *Data stores* (often called *triplestores*) to store and manage RDF data (described in Section 5.3.1). Examples of such stores are given in Section 2.2. (ii) *Reasoners*, to compute logical entailments, as mentioned previously in OWL. Examples of reasoners are Hermit², Pellet³, FaCT++⁴ and those natively implemented in triple stores such as Stardog, Virtuoso, Oracle 11g, *etc.* (iii) *Ontology engineering tools*, to create and maintain OWL ontologies (described in Section 5.3.2), such as Protégé⁵ and TopBraid⁶.

² <http://hermit-reasoner.com/>

³ <http://clarkparsia.com/pellet/>

⁴ <http://owl.man.ac.uk/factplusplus/>

⁵ <http://protege.stanford.edu/>

⁶ <http://www.topquadrant.com/tools/ide-topbraid-composer-maestro-edition/>

Additionally, there exists an active community support via *Q&A platforms* (questions and answers), such as SemanticWeb.com⁷ and the more general StackOverflow⁸.

5.3.1 RDF, the data model

The Resource Description Framework (RDF; Schreiber *et al.*, 2014) is a formal data model for describing Web resources, which are anything that can be symbolised as data in the Web, including representations of real physical things and abstract concepts. RDF represents data in directed graphs, such as the one of Figure 5.2. This graph can be decomposed into multiple smaller graphs, as the one depicted in Figure 5.3, consisting of a *subject*, a *predicate* and an *object*. In the graph, *subject* and *object* are nodes, connected by a directed arrow named with the *predicate*. The *predicate* is a property of the *subject*, and the *object* is the value of that property. These small graphs can be encoded as tuples of three elements called **triples**. The example of Figure 5.2 can be encoded into eight triples.

Subjects, predicates and objects of triples are identified by means of IRIs. A possible representation of the triple of Figure 5.3 with associated IRIs is as follows:

```
<http://ox.ac.uk/students/ignacio_lamata_martinez> <http://example.org/  
studiesAt> <http://ox.ac.uk/university> .
```

where the first IRI represents the subject “Ignacio Lamata Martinez”, the second one represents the predicate “studies at” and the third one represents the object “University of Oxford”. Note that in the graphs of Figure 5.2 and Figure 5.3, IRIs have been replaced by more legible labels for clarity.

In some cases, nodes might not have an IRI assigned, working as mere connectors to other objects. These nodes are called *blank nodes* and their identifiers have only a local meaning within the graph.

In a triple, objects can be literals of a certain datatype instead of being identified by an IRI. Datatypes are identified by IRIs themselves, and literals with no explicit datatype are considered strings. As depicted in Figure 5.2, specific datatypes are identified by means of the symbol `^^` followed by the IRI of the datatype (in this case *xsd:positiveInteger*, as an abbreviation of the

⁷ <http://answers.semanticweb.com/>

⁸ <http://stackoverflow.com/>

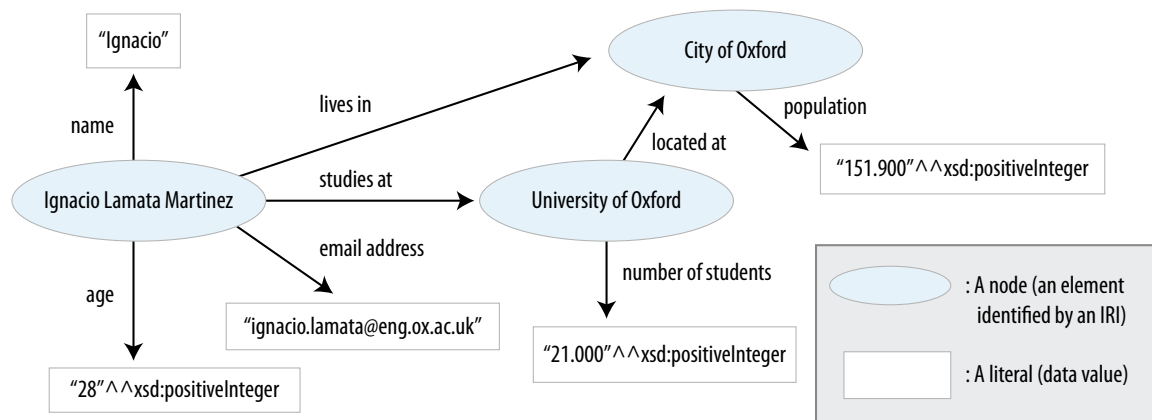


Figure 5.2. An RDF graph

IRI <http://www.w3.org/2001/XMLSchema#positiveInteger>). An example is:

```
<http://www.ox.ac.uk/universityOfOxford> <http://example.org/numberOfStudents>
"21000"^^xsd:positiveInteger .
```

RDF graphs are sets of triples, and they are called *named graphs* if the graph itself has an IRI associated. An *RDF Dataset* is a collection of RDF graphs, coded in *subject-predicate-object* triples, where exactly one of them is an unnamed graph, also called the *default graph* of the dataset (Cyganiak *et al.*, 2014).

RDF triples can be encoded into different syntaxes for transmission and computer processing. These include RDF in *XML* (RDF/XML), *Turtle* and *N-triples* amongst others (Schreiber *et al.*, 2014).

There exist database management systems able to store and manage RDF data, which are also known as *triplestores*. Examples of triplestores can be found in Section 2.2.

The semantics of RDF use the concepts of **class** and **property**. A *class* is a set of resources with similar characteristics, whose members are called *instances* of the class, whereas a *property* defines binary relations between instances (syntactically, the set of IRIs that can be used as predicates in a triple). However, note that RDF itself does not impose constraints in the way IRIs



Figure 5.3. RDF triple with a subject, a predicate and an object

are used within a triple.

Subjects, predicates and objects of RDF triples can use arbitrary vocabularies of IRIs, but there exist some IRIs with a special meaning to provide basic semantics, as defined in *RDF Schema* described by Brickley *et al.* (2014). A summary of these IRIs is listed below:

- **rdf:type**⁹ is a property that specifies the membership of the subject in a specific class. For example, the triple: `oxf:ignacio_lamata_martinez rdf:type oxf:Student` states that the subject is a member of the class “Student” represented by the IRI in the object.
- **rdfs:Class** is a class that identifies a resource as a class. For example, the triple: `oxf:Student rdf:type rdfs:Class` states that the IRI that represents “Student” is a class. In practice, instances of a class can be defined in RDF without having to assert this explicitly, so the following could be asserted `<any_IRI> rdf:type oxf:Student` without having to previously define *oxf:Student* as *rdfs:Class*.
- **rdf:Property** is a class that identifies a resource as a property. For example, the triple: `oxf:name rdf:type rdf:Property` states that the IRI that represents “name” is a property, used in a predicate.
- **rdfs:subClassOf** is a property to specify a subsumption relation between two classes. The example triple `oxf:Student rdfs:subClassOf oxf:Person` captures the concept that all members of the class “Student” are also members of the class “Person” (*Student* $\subseteq$ *Person*).
- **rdfs:subPropertyOf** is a property to specify a subsumption relation between two properties. The example triple `oxf:supervisesProject rdfs:subPropertyOf oxf:worksInProject` captures the concept that all resources linked by one property are also linked by another property, so someone that supervises a project also works in that project.
- **rdfs:domain** is a property to specify that all *subject* resources linked by a property in a triple belong to a specific class. For example, the triple `oxf:worksInProject rdfs:domain oxf:Person` states that any resource that occurs as the subject of the property *oxf:worksInProject* belongs to the class “Person”. It is common to understand this

⁹ Since IRIs are long and sometimes difficult to read, they are often abbreviated. “rdf:” is a shortcut for the IRI <http://www.w3.org/1999/02/22-rdf-syntax-ns#>, whereas “rdfs:” is a shortcut for the IRI <http://www.w3.org/2000/01/rdf-schema#>. Similarly, “oxf:” will be used in the examples as a shortcut for <http://ox.ac.uk/>.

erroneously as a restriction instead of an assignment: this triple `oxf:OxfordUniversity oxf:worksInProject oxf:MyProject` is not considered an error, but instead classifies “Oxford University” as a member of the class “Person”.

- **rdfs:range** is a property to specify that any resource that occurs as the object of the property in a triple belong to a specific class. It works similarly to *rdfs:domain*.

The term “semantics” used before is concerned with the meaning of concepts and their inter-relations. This thesis will mention the term as *formal semantics*, as mathematics-based methods to interpret and constrain what can be stated (a series of well-defined rules that permit to entail or extract conclusions from a given set of statements).

More complex semantics for RDF comes with the support of the OWL language, which is explained in the following section.

5.3.2 OWL, a language to define ontologies

The Web Ontology Language (OWL; Hitzler *et al.*, 2012) is a knowledge representation language to define ontologies, whose semantics are closely related to *description logics* (Horrocks *et al.*, 2010; see Appendix C for a brief introduction). Studer *et al.* (1998) define an ontology as a “formal, explicit specification of a shared conceptualization”, concept further explained by Guarino *et al.* (2009). Berners-Lee *et al.* (2001) have described an ontology as “a document or file that formally defines the relations among terms”. An ontology consists of a set of statements (called *axioms*) that define concepts and the specific relationships between them, capturing a shared understanding of a domain of interest for a particular application (Horrocks, 2013). This set of axioms forms a *knowledge base* in description logics terminology.

Normally, ontologies are supported by a logic, which establishes formal rules for the interpretation of concepts and enables *reasoning algorithms* to infer logical consequences and detect errors. This inference allows the creation of logically entailed axioms without requiring them to be explicitly asserted, which leads to less maintenance effort and better quality control (see reasoning details in Appendix C). Logic rules permit uniform and deterministic results to be obtained from reasoning algorithms, so different algorithms should never produce different results for the same task and set of axioms. Tools implementing reasoning algorithms are called *reasoners*.

OWL can be encoded into different syntaxes, such as *Functional-Style Syntax*, *OWL/XML*, *RDF/XML*, *Manchester Syntax* and *Turtle* notation. The examples in this thesis will use the Functional-Style Syntax as described by Motik *et al.* (2012), and IRIs will be abbreviated for clarity¹⁰.

The main elements of OWL are **classes**, **properties** and **individuals**¹¹, whose definition is the same as in the RDF Schema discussed in the previous section: classes are sets of resources with similar characteristics, individuals are members of classes (and individuals that belong to a class are called *instances* or *objects* of that class) and properties define binary relations.

For example, the axiom `Declaration(Class(:Concrete))` in Functional-Style Syntax defines the class *Concrete*¹², whereas the axiom `SubClassOf(:PrestressedConcrete :Concrete)` establishes a subsumption relation between *PrestressedConcrete* and *Concrete* (equivalent to $PrestressedConcrete \sqsubseteq Concrete$ in description logic syntax, as explained in Appendix C). All classes are subclasses of the special class *owl:Thing*, which represents the set of all resources. Likewise, the empty class is represented by the special class *owl:Nothing*.

Classes can be declared to be **equivalent** or **disjoint**. Equivalent classes have exactly the same individuals (e.g. `EquivalentClasses(:Experiment :Test)`) whereas two disjoint classes cannot have any instance in common (e.g. `DisjointClasses(:Actuator :Sensor)`). Note that from a logical point of view, equivalence can be defined by subsumption relations (in description logic syntax, the equivalence $Experiment \equiv Test$ is the same as $Experiment \sqsubseteq Test$ and $Test \sqsubseteq Experiment$).

Classes can be created from more complex statements based on other classes, by means of **union** (`ObjectUnionOf`), **intersection** (`ObjectIntersectionOf`) and **complement** (`ObjectComplementOf`) of classes, e.g. an *experimental activity* is something that is either numerical or physical: `EquivalentClasses( :ExperimentalActivity ObjectUnionOf(:PhysicalExpAct :NumericalExpAct))` (equivalent to $ExperimentalActivity \equiv PhysicalExpAct \sqcup NumericalExpAct$

10 Note that, when coding OWL vocabulary in RDF, IRIs are used. The namespace “owl:” is often used as a shortcut for the IRI <http://www.w3.org/2002/07/owl#>. However, the Functional-Style Syntax, used in the examples of this thesis, does not require IRI identifiers for most of the OWL vocabulary.

11 They can be identified with *concepts*, *roles* and *individuals*, respectively, in description logics or with *unary predicates*, *binary predicates* and *constants* in first-order predicate logic.

12 Classes and individuals are normally written with a leading uppercase character whereas properties are normally written with a leading lowercase character.

in description logic syntax). Similarly, the class of *European laboratories* as the class of individuals belonging to the class *European* **and** the class *Laboratory*: `EquivalentClasses(:EuropeanLaboratory ObjectIntersectionOf( :Laboratory :European))` (equivalent to $EuropeanLaboratory \equiv Laboratory \sqcap European$ in description logic syntax). Also, numerical experimental activities can be defined as those that are **not** physical: `EquivalentClasses(:NumericalExpAct ObjectComplementOf( :PhysicalExpAct))` (equivalent to $NumericalExpAct \equiv \neg PhysicalExpAct$).

Classes can also be created from property restrictions with other classes, such as those for **existential quantification** (`ObjectSomeValuesFrom`) and **universal quantification** (`ObjectAllValuesFrom`). For example, an experiment can be defined as something that executes **at least** one physical activity: `SubClassOf(:Experiment ObjectSomeValuesFrom(:executes :PhysicalExpAct))` (equivalent to $Experiment \rightarrow \exists executes. PhysicalExpAct$, those individuals that are connected to instances of the class *PhysicalExpAct* via the *executes* property). Similarly, European projects are those that are funded **only** by European funds: `SubClassOf(:EuropeanProject ObjectAllValuesFrom(:isFundedBy :EuropeanFunding))` (equivalent to $EuropeanProject \rightarrow \forall isFundedBy. EuropeanFunding$, so all relations via *isFundedBy* have to be with instances of the class *EuropeanFunding*. Note that this also holds when there are no occurrences of the property *isFundedBy*).

The **self-restriction** (`ObjectHasSelf`) specifies the class of individuals that are related to themselves by means of a specific property. For example, an auto-powered device is something that is powered by itself: `EquivalentClasses(:AutoPoweredDevice ObjectHasSelf(:poweredBy))`.

With **inverse** properties (`ObjectInverseOf`), the direction of a property can be “reversed”. For example, European funding is something that funds European projects: `SubClassOf(:EuropeanFunding :ObjectAllValuesFrom( ObjectInverseOf(:isFundedBy) EuropeanProject))`. The inverse property changes the direction of the property (from *is funded by* to *funds*).

Individuals can belong to one or more classes (*e.g.* the individual *John* is an instance of the class *PrincipalInvestigator*). The axiom `Declaration( NamedIndividual(:John))` defines the individual *John* and `ClassAssertion( :PrincipalInvestigator :John)` declares *John* in the class *PrincipalInvestigator*. All individuals are instances of the class *owl:Thing* and, similarly to equivalent and disjoint classes, individuals can be declared **equal** or **different** via `SameIndividual`

and `DifferentIndividuals`, respectively.

A *property* specifies a relation. There are two types of properties:

- **Object properties** relate individuals to individuals. For example, *is principal investigator* of is an object property to relate two individuals, such as *John* is principal investigator of *Project1212*. In Functional-Style Syntax this would be `ObjectPropertyAssertion(:isPrincipalInvestigator :John :Project1212)`.
- **Data properties** relate individuals to data values. For example, *construction year* is a data property to relate an individual with a positive numeric value, such as *Actuator25 construction year* “1998”¹³. In Functional-Style Syntax this would be `DataPropertyAssertion(:constructionYear :Actuator25 “1998”)`. Note that data values should have a datatype defined by an IRI, as explained for Figure 5.2. The correct axiom would be `DataPropertyAssertion(:constructionYear :Actuator25 “1998”^^xsd:positiveInteger)`, but datatypes will be omitted in this thesis for simplification purposes.

Properties can be defined to have some characteristics. If a property *P* making a relation from *A* to *B* is defined as **functional** (`FunctionalObjectProperty`, `FunctionalDataProperty`), then *A* can only be related to a single *B*. For example, if the property *construction year* is functional, `DataPropertyAssertion(:constructionYear :Actuator25 “1998”)` means that *Actuator25* cannot be related to any other value by means of the *construction year* property, so the axiom `DataPropertyAssertion(:constructionYear :Actuator25 “2000”)` will entail an inconsistency (an error in the ontology). Functional is the only characteristic allowed in data properties, the rest are only allowed in object properties.

An **inverse functional** property (`InverseFunctionalObjectProperty`) is a property where an individual *B* can only be related from one individual *A*. For example, if *is biological mother* is inverse functional, then *B* can only be related to a single *A* (which means *B* cannot have more than one biological mother).

A **transitive** property (`TransitiveObjectProperty`) is a property where if an individual *A* is related to an individual *B* via *P*, and *B* is related to an individual *C* via *P*, then a reasoner can infer that *A* is also related to *C* via *P*. For example, if *made of* is transitive, then from the axioms

¹³ In this thesis, data values will be written between quotation marks to distinguish individuals from values (e.g. the individual *John* from the string value “*John*”).

`ObjectPropertyAssertion(:madeOf :Concrete :Mortar)` and `ObjectPropertyAssertion(:madeOf :Mortar :Water)`, it can be inferred that `ObjectPropertyAssertion(:madeOf :Concrete :Water)`. Note that this third axiom is not stated explicitly, it is a logical consequence from the rules of transitive properties.

OWL **property chains** implement limited *complex role inclusion*, which allows the definition of properties from the composition of other properties. An example of a property chain is the following: a device that is used in an experiment is also used in the project that conducts such experiment. This can be represented by $conductsExperiment \circ usesDevice \sqsubseteq usesDevice$. Having the following example axioms:

```
ObjectPropertyAssertion(:conductsExperiment :Project1212 :Experiment1605)

ObjectPropertyAssertion(:usesDevice :Experiment1605 :Actuator25)

SubObjectPropertyOf(ObjectPropertyChain(:conductsExperiment :usesDevice)
:usesDevice)
```

A reasoner can infer the following axiom:

```
ObjectPropertyAssertion(:usesDevice :Project1212 :Actuator25)
```

Herein, transitivity is a simpler form of a property chain. There exist some limitations in the way property chains can be used as discussed by Hitzler *et al.* (2009).

Properties can be declared to be equivalent or disjoint. Two equivalent properties relate the same individuals in the same way and two disjoint properties cannot relate the same two individuals in the same way.

5.3.3 Semantic considerations

One interesting characteristic of entailments in OWL (the conclusions that result from reasoning) is that they are based on the **Open World Assumption** (OWA), which assumes that something that is not stated in the ontology *might* be true (contrarily to standard relational databases, which assume that something that is not in the database is therefore false). Because Semantic Web technologies are supported by *monotonic logics*, knowledge is never considered complete, and this behaviour works well in a distributed environment where a node can express some knowledge

that other nodes can complete. However, this can produce confusing results when reasoning over a set of axioms if OWA is not considered. Let us use the following example:

```
SubClassOf(:Project ObjectSomeValuesFrom(:conductsExperiment :Experiment))
```

```
ClassAssertion(:Project :Project1212)
```

The first axiom states that projects are individuals that conduct at least one experiment - an individual belonging to the class of experiment ($\text{Project} \sqsubseteq \exists \text{conductsExperiment}.\text{Experiment}$, in description logic syntax). The second axiom states that *Project1212* is a *Project* but the ontology does not assert that *Project1212* conducts any experiment. Because of the OWA, this is not an inconsistency. *Project1212* might conduct an experiment but it is not known yet. Unless any other axiom explicitly states that *Project1212* does not conduct any experiment, there is no possible inconsistency.

Another interesting characteristic of entailments in OWL is the **absence of the Unique Name Assumption** (UNA), which assumes that different IRIs can actually identify the same resource. Let us consider the functional property *has biological mother* in this ontology:

```
ObjectPropertyAssertion(:hasBiologicalMother :John :Mary)
```

```
ObjectPropertyAssertion(:hasBiologicalMother :John :Maria)
```

A functional property can only have one value for the same individual, and in the example, *John* is related to two different individuals, *Mary* and *Maria*. However, because of the absence of UNA, this is not an inconsistency as it could be expected. Instead, a reasoner would infer that *Mary* and *Maria* are indeed the same individual, by creating the axiom `SameIndividual(:Mary :Maria)`. An inconsistency could only happen if another axiom states that *Mary* and *Maria* are not the same individual via `DifferentIndividuals(:Mary :Maria)`.

As it has been depicted in the previous example of UNA and the example of *rdfs:domain* in Section 5.3.1, the addition of axioms to an ontology do not reduce the number of logical entailments.

By using reasoning algorithms in OWL, new axioms can be inferred and inconsistencies can be detected. **Ontology consistency checking** is a reasoning task to verify that an ontology is logically satisfiable (it has at least one interpretation, as described in Appendix C). An inconsistent ontology is useless because it entails any possible axiom. Likewise, **class consistency checking** verifies that classes are satisfiable, so that they could have instances. An ontology with inconsistent classes can still be consistent if the inconsistent classes have no instances, and it is called *coherent*

if it only has consistent classes. **Subsumption checking** reasoning tasks verify subclass and subproperty relations. **Classification** reasoning tasks compute class and property hierarchies (all subsumption relations). **Instance checking** verifies if an individual belongs to a class and **instance retrieval** computes all individuals of a class.

OWL is an ontology language based on description logics as well as an extension of RDF. Therefore, OWL semantics have been defined in two slightly different ways:

- **RDF-based Semantics** (Schneider, 2012), which comes as a direct extension of RDF. Reasoning under these semantics is *undecidable* (see Appendix B for a definition of *decidability*), and ontologies interpreted by these semantics are in the *OWL Full* language. Due to its undecidability, there is no reasoner that correctly compute, without errors or omissions, the logical conclusions of any given OWL Full ontology.
- **Direct Semantics** (Motik *et al.*, 2012c), which is logically connected to the description logic *SROIQ* (Horrocks *et al.*, 2006) although OWL ontologies should comply with some restrictions, as discussed in Krötzsch *et al.* (2012b). Reasoning under these semantics is *decidable* in *nondeterministic doubly exponential time* (N2ExpTime; Kazakov, 2008) and ontologies interpreted by these semantics are in the *OWL DL* language. Direct Semantics can be seen as a semantically restricted subset of the RDF-based semantics.

Besides OWL Full and OWL DL, three lightweight profiles are defined, known as *OWL EL*, *OWL RL* and *OWL QL* (Krötzsch, 2012). Each profile is designed for a specific application and uses a limited range of OWL features. The benefit of using a lightweight profile is that they are designed to be tractable, so the different selection of OWL features they are allowed to use permit reasoning algorithms that run in polynomial time. Therefore, these profiles can have simpler reasoners and encourage the development of more tools.

The OWL semantics allow machines to interpret data within an application domain. However, the conception that machines can actually understand the meaning of concepts, such as “human”, is erroneous. Consider the example “all humans are mortals” and “John is a human”:

SubClassOf(:Human :Mortal)	$H(x) \rightarrow M(x)$
ClassAssertion(:Human :John)	$H(\text{John})$

A reasoner can infer from these axioms that John is mortal:

ClassAssertion(:Mortal :John)	$M(\text{John})$
-------------------------------	------------------

For the machine, semantics are not provided by real meaning of concepts but by the rules that have been defined in the system. A machine can equally interpret the previous example as: all birds are animals, a canary is a bird, therefore a canary is an animal. The axioms just establish that if something belongs to a set, then it also belongs to its superset, without making any distinction of what “something” or the “set” are, which have to be interpreted within the context of a specific application.

5.4 The Celestina Data ontology

This section describes an ontology for earthquake engineering experimental data, that has been named **Celestina Data ontology** and that provides the core component of the data services of Celestina Data.

The objective is the creation of a single, centralised ontology (what is considered a *TBox* as described in Appendix C) to support *assertional* content (what is considered an *ABox* as described in Appendix C) created by each institution. The ontology can be extended with additional terminological content by the different institutions if local applications have extra requirements or if agreement about terminological content of the ontology is not reached. The decision of which ontologies or terminological content are to be used is made by the different applications, according to their data needs.

The OWL axioms used in the description of the ontology have been defined by using Functional-Style Syntax. Additional figures are used to provide a clearer view of how assertional information is used together with terminological content. These figures follow the notation depicted in Figure 5.4, which uses a graph to represent RDF-like triples of OWL content. Instances and classes are represented by graph nodes (ellipses, which can be identified by an IRI), where bold font signifies class names and a regular font signifies individual names. Literals are represented by rectangles (which are never identified by an IRI). Properties are represented by directed named arrows, where solid arrows represent explicitly stated triples and dashed arrows represent information that can be inferred by a reasoner. For clarity, the IRIs of classes, individuals and properties under the Celestina Data domain have been abbreviated in the figures (*e.g.* *Facility* instead of *http://www.celestina-integrations.com/data#Facility*), and only *namespaces* of IRIs external to

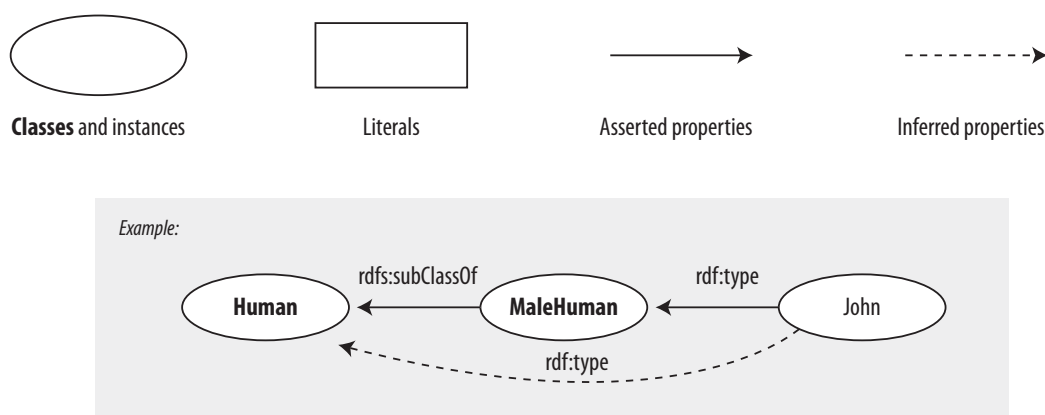


Figure 5.4. Graphical notation used for OWL examples

Celestina are shown (e.g. *rdf:type* instead of *http://www.w3.org/1999/02/22-rdf-syntax-ns#type*). However, sometimes the text will show the namespace “*celestina:*” or simply “*:*” for IRIs internal to Celestina if it makes the text easier to follow (e.g. *celestina:description*, *:description*, etc.). Note that terminological content shown in the figures does exist in the ontology, whereas *most* of the assertional content has been invented for illustration purposes.

In Celestina Data, object property names are normally prefixed with the verb “is” or “has” to establish the direction of the property clearly. For instance, using a property named “material” to define materials of a specimen is ambiguous, because it is not clear if the material should be a subject or an object. By using “hasMaterial” instead, it is made clear that the material should be an object of the property. Note that data properties do not lead to this ambiguity and no prefix will be used.

All classes, instances and properties defined in the Celestina Data ontology are described by means of *rdfs:label* and *rdfs:comment*. *rdfs:label* is used to provide a friendly name for the IRI of the elements whereas *rdfs:comment* is used to describe elements and document ontological terms in natural language. Correct use of elements and examples are provided in the description, which is of special relevance for humans to understand how to use the ontology. Despite the logical rules established by the semantics, humans can still misuse the elements of the ontology.

Many properties in the Celestina Data ontology can be used in all different classes and instances (for example, *name* can be used in instances of all classes) but some of them use a domain (*rdfs:domain*) or range (*rdfs:range*) that limits their use.

5.4.1 Ontology overview

The presentation of the ontology has been divided into different sub-groups, in a similar fashion as the hierarchies of existing data models seen in Figure 5.1. This division can be used to create different independent ontologies, which can be maintained separately and gathered together in a single ontology as imported ontology modules. The general description map is shown in Figure 5.5. The **project module**, discussed in Section 5.4.2, describes all general information about earthquake engineering projects, which have objectives, start and end dates and outcomes. The project module subtly links to the specimen and the equipment and facilities modules, but its main link is with the experimental activity module. In this sense, the ontology is experimental-activity centred, and projects can be considered as high-level groups of experimental activities.

The **experimental activity module**, discussed in Section 5.4.5, describes all physical and numerical experimental activities. Experimental activities are conducted under some loading parameters (input experimental information) and produce some results (output experimental information). This information is described in the **experimental input/output module**, discussed in Section 5.4.6.

Experimental activities are conducted over a specimen. Specimens are described in the **specimen module** discussed in Section 5.4.3, and the composition of the specimen is described in the **materials module**, discussed in Section 5.4.4.

Finally, experimental activities are supported by experimental equipment and systems. These are described in the **facilities and devices module**, discussed in Section 5.4.7.

There are two vertical modules that can complete information in the rest of the modules. The **persons and organisations module**, discussed in Section 5.4.8, describes participants for any task or activity. This can define the participation of persons and organisations in any module of the ontology, as for example, funding organisations for projects, principal investigators, test agents, people that constructed a specimen, *etc.* The **media and data module**, discussed in Section 5.4.9, describes documents, images, videos and other large data files that are used from the other modules.

This division defines a high-level perspective of experimental data, which can probably be easily accommodated to experimental data of other hazard mitigation disciplines.

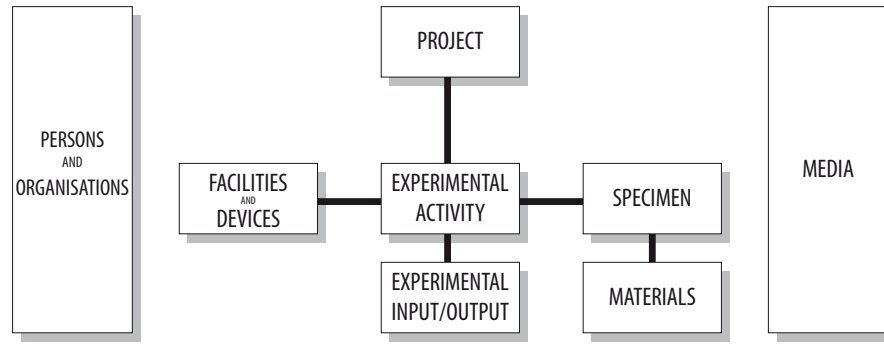


Figure 5.5. Celestina Data ontology modules division

The process to construct the ontology followed these steps, which constitute the development methodology:

- 1) **Expert of the domain enquires.** The development of the ontology required many iterations and discussion with a reduced number of experts of the domain, belonging to disparate institutions. The creation of ontological concepts can be a complex task, especially when there is no explicit agreement on the terms in a dispersed community. Existing systems proved to be beneficial as a source of data requirements, but sometimes they can imply more complexity in the development process, since accommodation of existing data must be considered.
- 2) **Community enquires.** In cases where concepts were unclear, the collaboration of a larger group of experts in the community was required. This is the case of the terminology used for experiments and simulations, where a survey was conducted to clarify the understanding of terms in different institutions of the USA and Europe, as explained in Section 5.4.5.
- 3) **Quality assurance.** Once the ontology was developed, its logical correctness was verified with the support of a reasoner, as explained in Section 5.5. This helped to detect and fix errors.

The final Celestina Data ontology consists of 1,140 axioms, 156 classes, 21 individuals, 67 object properties and 34 data properties as depicted in Table 5.1. In this table, numbers are also given per module, but note that some properties are shared between the different modules and have been counted in more than one module, so the sum of the number of properties (and therefore, the sum of the number of axioms) does not match the numbers for the full ontology.

	Nr. axioms	Nr. classes	Nr. properties	Nr. object prop.	Nr. data prop.	Nr. individuals
Celestina Data (full)	1140	156	101	67	34	21
Project module	129	2	16	7	9	3
Experimental activity module	136	13	19	8	11	0
Experimental input/output module	94	11	14	6	8	0
Specimen module	230	58	9	5	4	0
Materials module	140	22	10	5	5	0
Facilities and devices module	247	31	29	14	15	0
Persons and organisations module	51	4	9	5	4	0
Media and data module	341	15	47	29	18	18

Table 5.1. Statistical data about the Celestina Data ontology

Taxonomies (hierarchies of concepts) such as the material taxonomy discussed in Section 5.4.4, and the specimen component types taxonomy discussed in Section 5.4.3, have been kept as simple as possible, since it has been experienced that development of an extremely complete taxonomy could lead to unnecessarily complicated systems that are rejected by their own users (Bardet *et al.*, 2004). Therefore, a more simple yet useful data model is more beneficial than a complicated data model that is not used in practice. However, the Celestina Data ontology is by no means a complete and closed model. As any data model, it should be continuously developed, maintained and evolved to adapt it to current data needs in the earthquake engineering community.

As of October 2014, the ontology can be downloaded from:

<http://www.celestina-integrations.com/data>.

The following sections discuss the different ontology modules.

5.4.2 Project module

The project module describes projects as instances of a class *EarthquakeEngineeringProject*. Projects and other activities have a “status” that defines the progress situation of the project. It has been identified that only three statuses are necessary: an activity that has not started yet but it is planned to occur in the future (*StatusForeseen*), an activity that is currently conducted (*StatusInProgress*) and an activity that has been completed (*StatusFinished*). These have been modelled as individuals of the class *ActivityStatus*.

Most of the instances, classes and properties of a Celestina Data ontology make use of two properties: *name* and *description*. *name* is basically a Celestina version of the existing *rdfs:label*, and it is used in cases where a name is required for a user interface. However, *name* might be removed in future if its use is not extensive enough to justify its existence.

A graph example of a project is depicted in Figure 5.6. Project instances have no privacy property since all data in Celestina Data is considered public. Despite information being public, it is important to give *adequate* credit to the particular owner of the information, and to correctly identify the provenance of the data. This is the function of the property *isIntellectualPropertyOf*, which provides information about the institution that holds rights over the data.

The relation between projects, specimens and experiments is one of the main differences in SERIES and NEES data models. On the one hand, projects-specimens, and specimens-experiments are linked in the SERIES data model (projects use specimens, which are tested in experiments). On the other hand, Projects-experiments, and experiments-specimens are linked in the NEES data model (projects conduct experiments, that use specimens). Despite the different relation connections, both models state the same facts, and the elements involved can be interconnected as shown in Figure 5.7a.

Initially, the ontology was prepared to have asserted any of the two relations, and infer the other, but this raised some issues as explained below.

If the relation project-specimen is asserted as in the SERIES data model, the relation project-experiment has to be inferred as shown in Figure 5.7b. This can be achieved by implementing the following statement (the logic notation of this example can be consulted in Fitting, 2012):

$$\text{Project}(x) \wedge \text{usesSpecimen}(x,y) \wedge \text{usesSpecimen}(z,y) \wedge \text{ExperimentalActivity}(z) \rightarrow \text{conductsExperimentalActivity}(x,z)$$

Implementation of this as a property chain can be done by using a technique called *rolification* as explained in Rudolph *et al.* (2008) (additional information on rolification be found in Krisnadhi *et al.*, 2011 and Krötzsch, 2010), which aims to “rolify” the classes by creating a property (binary relation) based on a class (unary relation). In this specific case, the creation of an object property *rProject* is required:

```
Declaration(ObjectProperty(:rProject))
```

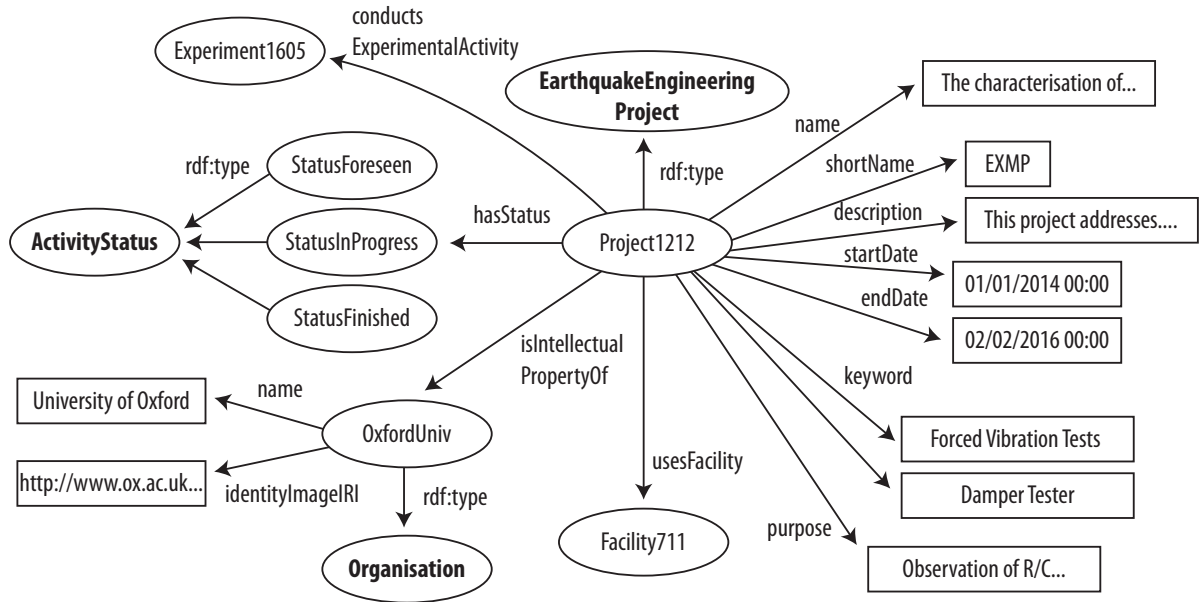


Figure 5.6. Example of the project module in the Celestina Data ontology

and the axiom that makes any individual of the class *Project* in a *rProject* relation with itself:

```
EquivalentClasses(:EarthquakeEngineeringProject ObjectHasSelf(:rProject))
```

The same process is followed for *ExperimentalActivity* and the property chain can be created to obtain the desired inference:

```
SubObjectPropertyOf(ObjectPropertyChain(:rProject :usesSpecimen
ObjectInverseOf( :usesSpecimen) :rExperimentalActivity)
:conductsExperimentalActivity)
```

This solution uses local reflexivity (*ObjectHasSelf*), which is not allowed in OWL QL and OWL RL (although there is no technical reason for OWL RL not have it, as discussed in Krötzsch, 2012) and inverse properties (*ObjectInverseOf*) which are not allowed in OWL EL.

On the other hand, if the relation project - experiment is asserted as in the NEES data model, the relation project - specimen has to be inferred as shown in Figure 5.7c. This can be done with a rather more simple property chain:

```
SubObjectPropertyOf(ObjectPropertyChain(:conductsExperimentalActivity
:usesSpecimen) :usesSpecimen)
```

Both inferences of Figure 5.7 work well separately, but reasoners will raise a cyclic dependency error when put together in the same ontology. This issue comes from the fact that one property chain creates the element necessary to trigger the other property chain, and the reasoners tested identified correctly the cyclic dependency. Herein, the Celestina Data ontology only infers the

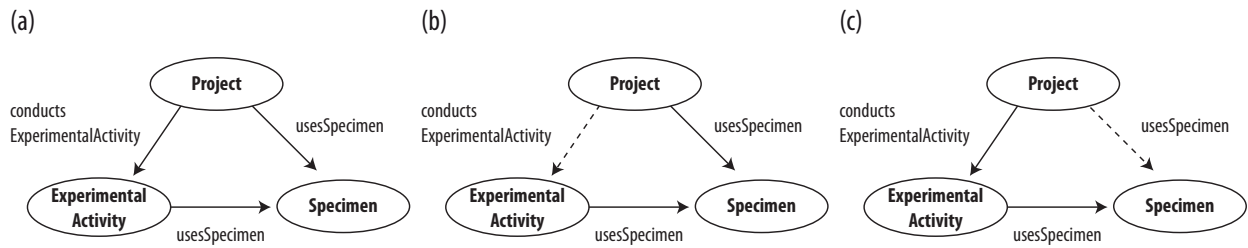


Figure 5.7. Relation project - specimen - experiment (a) desired model (b) model inferring the project - experimental activity relation (c) model inferring the project - specimen relation

axioms linking projects and specimens as depicted in Figure 5.7c, and expects the axioms linking projects and experiments to be asserted. This approach is more similar to the NEES data model.

5.4.3 Specimen module

Specimens are physical elements that are tested during a physical experimental activity or are emulated during a numerical experimental activity. They are represented by the class *Specimen* and are described by means of physical properties (mass, height, maximum weight, *etc.*), represented by the class *PhysicalProperty*. A specimen is divided into sub-elements represented by the class *SpecimenComponent*. A taxonomy of specimen component types has been created, based on the SERIES data requirements specified in Chapter 3. The identification of types of specimen components has not been found easy by some experts of the domain, and future use of the ontology will dictate the way this taxonomy should evolve.

Specimens are referenced by the property *usesSpecimen* and projects that conduct an experimental activity that uses a specimen are considered to use the specimen too, as discussed in Section 5.4.2. An example of a graph representing a specimen can be found in Figure 5.8.

5.4.4 Material module

Materials refer to the composition of the specimen, such as steel or concrete. In current earthquake engineering databases, such as the ones at SERIES and NEES, materials are used for different kinds of composition level. For instance, reinforced concrete is a block of concrete containing embedded reinforcing bars made of steel. In practice, reinforced concrete and other composite structures are

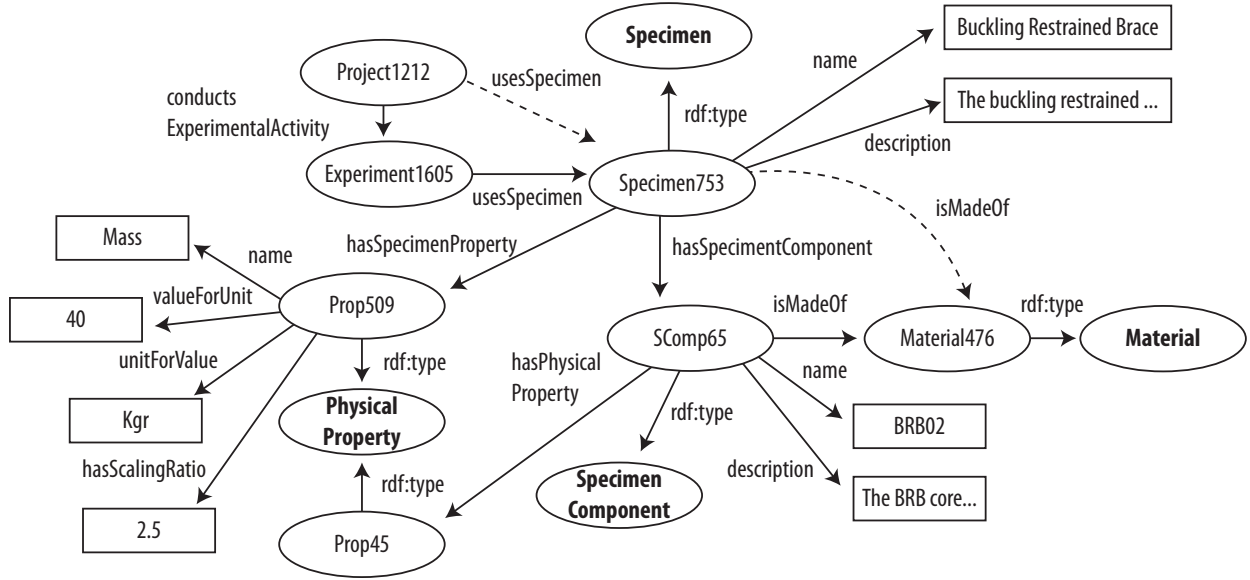


Figure 5.8. Example of the specimen module in the Celestina Data ontology

used in the same manner as *raw* materials such as steel or glass (as it happens for example in the SERIES database). A similar example is the “composite steel” used in the NEES database, that refers to steel beams acting compositely with a concrete deck.

For simplification purposes, the creation of the ontology has considered the practical use of materials in earthquake engineering research more than actual structure composition. This consideration could be changed if using different levels of composition demonstrates any benefit in the data modelling (*e.g.* the information can be expressed more efficiently or truthfully) or the inference system (*e.g.* richer information can be concluded from the data by a reasoner).

Many of the materials are also composite materials. For example, iron and carbon make *steel*, and sand, water, gravel and a binder like cement or lime make *concrete*. However, this level of specification has been considered to overcomplicate the ontology and has not been implemented.

The simple taxonomy depicted in Figure 5.9 has been created by obtaining input from experts of the domain (a *taxonomy* is a hierarchy of classes, and it has been created with axioms of the form `SubClassOf(:PrestressedConcrete :Concrete)`). Materials are considered in two different branches, those used for structural elements, and those used for ground elements. New materials are actively researched and the ontology is expected to grow accordingly. Composite materials have been defined based on the opinion of experts of the domain, and general material concepts and subclass relationships have been obtained by verifying what experts of the domain would expect to obtain in a data search. For example, even though reinforced concrete is actually made of steel and concrete, it is not expected to obtain reinforced concrete specimens when looking for

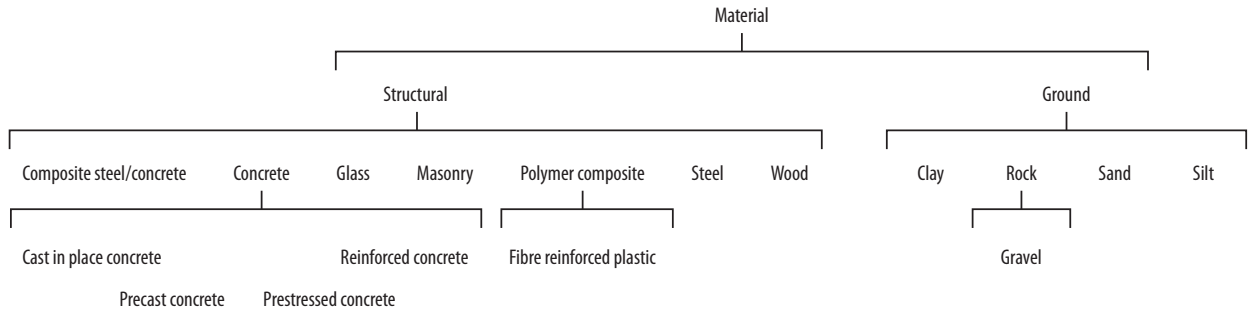


Figure 5.9. Taxonomy of materials in Celestina Data

specimens containing steel - as it would be inferred from a logical point of view.

Another implementation decision based on practical use is the classes definition of reinforced concrete. Reinforced concrete is made from steel and concrete (strictly speaking, reinforcing bars made of steel embedded in a block of concrete). If the steel bars are tensioned while casting the concrete, then it is called *prestressed concrete* (note that prestressed concrete can be obtained by other methods too), which from a logical point of view makes prestressed concrete a subtype of reinforced concrete. However, the term reinforced concrete implies in practice that there is no prestressed process in the generation of the concrete, so reinforced and prestressed concrete are defined as disjoint classes. Sometimes, researchers are interested in knowing whether the concrete was *precast* or *cast in place*. Precast concrete is sometimes used as a material (*e.g.* in the SERIES data model), although in theory it is more a fabrication procedure than a composition type. Precast concrete is normally prestressed but not always, so precast concrete has been defined as the class of materials made of concrete and precast (`EquivalentClasses (:PrecastConcreteMaterial`

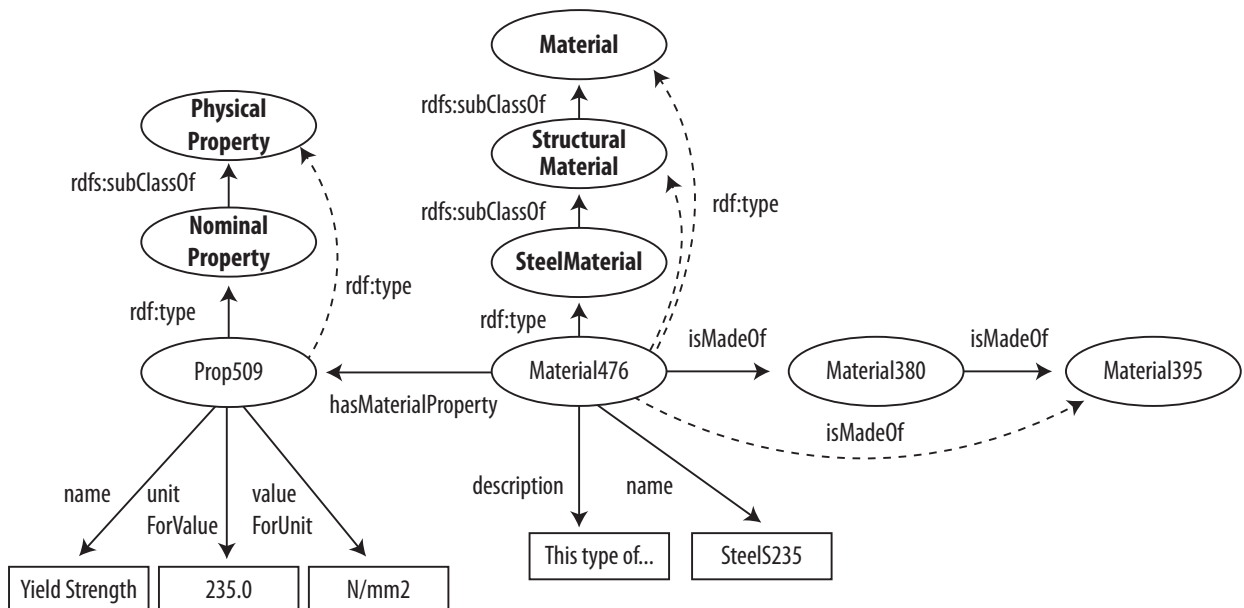


Figure 5.10. Example of the material module in the Celestina Data ontology

`ObjectIntersectionOf(:Precast :ConcreteMaterial))`). Therefore, a material can belong to either class reinforced and prestressed concrete as well as either class precast and cast in place concrete, which allows an extra degree of freedom when defining a type of concrete.

Materials are instances of the class *Material*, and they are referenced by the property *isMadeOf*. This property is transitive (as explained in Section 5.3.2) and a property chain is defined to infer that specimens are made of the materials that their specimen components are made of. Properties of materials can be specified by means of *hasMaterialPhysicalProperty* (a subproperty of *hasPhysicalProperty*), *e.g.* the yield strength of a material or the percent amount of reinforcing bar in the reinforce concrete. An example of a graph representing a materials can be found in Figure 5.10, in which *Material476* is an instance of steel, and it is made from other two materials (with no specific classes).

5.4.5 Experimental activity module

The experimental activity module is concerned with physical and numerical experiments (see Section 7.1 for a definition of physical and numerical). The first difficulty has been to find a suitable single term that embraces both physical and numerical experiments, because there is no such concept in any existing data model or in practical earthquake engineering terminology. The term *experimental activity* has been chosen for this purpose, represented by the class *ExperimentalActivity* in the ontology.

It has been experienced that the terminology used for experiments and simulations is sometimes not fully shared or understood. Note that the NEES data model uses “experiment” for physical activities and “simulation” for numerical activities whereas the SERIES data model uses “experiment” and “computation” respectively. This has motivated a survey, which has been conducted between 26 experts of the domain in earthquake engineering, 15 from European institutions and 11 from American institutions, in order to clarify what the understanding of experimental terminology is in the community. The survey consisted in the identification of several terms (experiment, simulation, hybrid simulation, *etc.*) with activities that can be purely physical, purely numerical or contain components both physical and numerical. Whereas the terms “experiment” or “simulation” might be understood ambiguously, the terms “physical” and “numerical” are interpreted without a doubt

in the community. The results of the survey are presented in Figure 5.11. The terms “experiment” and “test” were mainly identified as physical activities, whereas the terms “simulation” and “computation” were identified as numerical activities. “Hybrid simulation”, “hybrid test” and any kind of “pseudo-dynamic test” were mainly identified as activities that include both physical and numerical components, although the understanding of some pseudo-dynamic test concepts was highly dispersed.

As a consequence of the survey, five classes have been defined: *Experiment* and *Test* as physical experimental activities, *Simulation* and *Computation* as numerical experimental activities and *HybridSimulation* as an experimental activity that includes both physical and numerical components. They are all subclasses of *ExperimentalActivity*.

Judging by the survey, there is an apparent misconception over terms for pseudo-dynamic tests (PsD). They are rarely used in America and United Kingdom but actively used in some institutions in Europe and probably other regions around the world (Pegon *et al.*, 2000). PsD testing (Donea *et al.*, 1992) has been used as a synonymous of hybrid simulation by Nakata *et al.* (2014), and some comparisons between PsD with sub-structuring and hybrid simulations have been made by Pegon (2008). Since there is a motivation to use the PsD terminology, the class *PseudoDynamicTest* has been created, with two subclasses *PseudoDynamicTestSubstructuring* and *PseudoDynamicTestWithoutSubstructuring*, all of them subclasses of *ExperimentalActivity*.

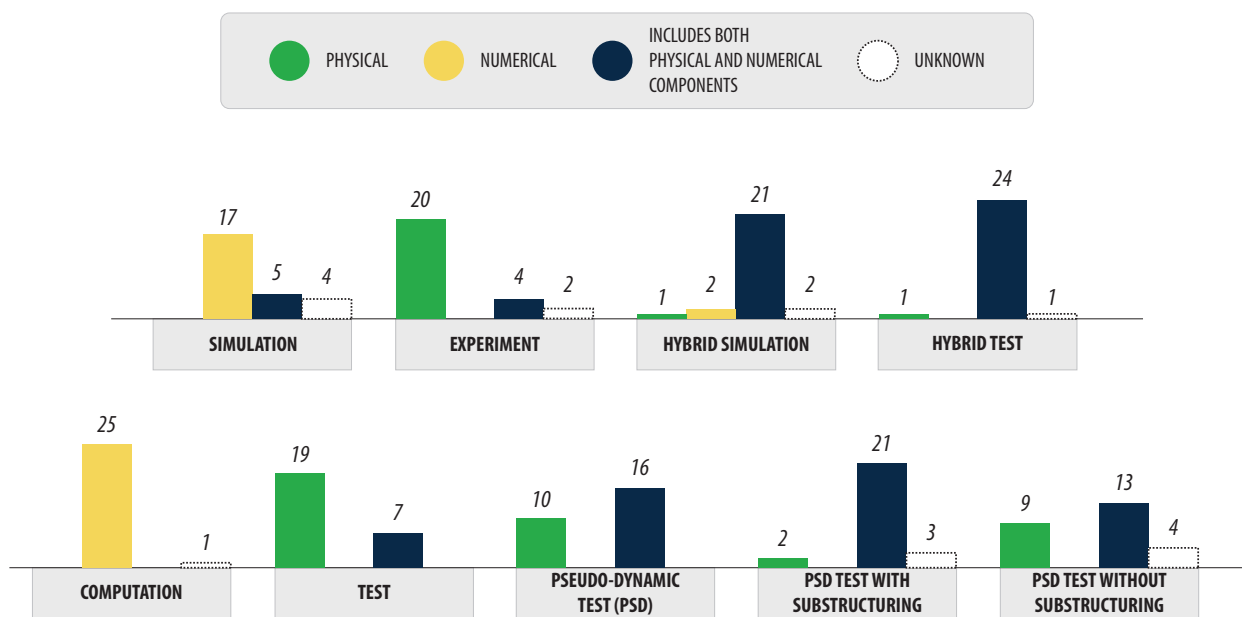


Figure 5.11. Results of the survey for different experimental-related concepts

The problem of experimental activity classification has been delegated to the classification of experimental activities into physical (by means of the class *PhysicalExperimentalActivity*) and numerical (by means of the class *NumericalExperimentalActivity*). Logical axioms have been created to infer the actual experimental classification. Experimental activities are the union of physical and numerical activities:

```
EquivalentClasses(      :ExperimentalActivity      ObjectUnionOf(
:PhysicalExperimentalActivity :NumericalExperimentalActivity))
```

Experiments and tests have been defined as something that is physical while computations and simulations have been defined as something that is numerical. This effectively means that the classes are equivalent:

```
EquivalentClasses(:Experiment :PhysicalExperimentalActivity)
EquivalentClasses(:Computation :NumericalExperimentalActivity)
```

Experimental activities can act as aggregations of other experimental activities, especially in the case of hybrid and distributed simulations, where different local experimental activities interact with one another and have to be interpreted as a whole. The property *hasExperimentalActivity* allows the creation of such a hierarchy. Experimental classification of the higher elements of the hierarchy depends on the class of dependent activities. Therefore, an experimental activity that is only related to experiments can be considered an experiment, but if it is related to both experiments and computations, can be classified as a hybrid simulation.

This model enables a reasoner to classify experimental activities automatically just by defining if the activities at the bottom of the hierarchy are either physical or numerical. However, unrestricted use of this propagation in a hierarchy imposes some complexity. Let us suppose an experimental activity *Exp1* that groups to two other experimental activities *Exp1.1* and *Exp1.2* of nature *physical* and not *numerical*. Because of the OWA behaviour discussed in Section 5.3.3, *Exp1* cannot be classified by a reasoner as *Experiment* as expected, because the information is assumed to be incomplete and it is not known if it is related to a possible third instance *Comp1.1*, which is a computation. The only way to infer the desired classification is to explicitly state that *Exp1* is **only** related to the individuals *Exp1.1* and *Exp1.2*. This creates some unnecessary complexity: (i) new axioms have to be created to explicitly “close” activity relations and, (ii) an inconsistency will be created if a third experiment or computation is related to *Exp1* in any other axiom, which

could easily happen in a distributed environment.

Thus, some restrictions have been applied in the final ontology. Firstly, the classes *PhysicalExperimentalActivity* and *NumericalExperimentalActivity* are defined as disjoint, so they cannot share any individual. This implies the nature physical and numerical cannot be propagated because hybrid simulations will raise a contradiction being physical and numerical at the same time. Another restriction is that it is assumed that only combined physical and numerical activities (not only experiments or only computations, but activities combining both physical and numerical substructures) can create a hierarchy of experimental activities, and such hierarchy must consist in a single hierarchical level.

Hybrid simulations have been defined as experimental activities that have at least one physical and at least one numerical experimental activity (HybridSimulation $\equiv \exists \text{hasExperimentalActivity. PhysicalExperimentalActivity} \sqcap \exists \text{hasExperimentalActivity. NumericalExperimentalActivity}$):

```
EquivalentClasses(:HybridSimulation ObjectIntersectionOf (ObjectSomeValuesFrom
(:hasExperimentalActivity :PhysicalExperimentalActivity) ObjectSomeValuesFrom
(:hasExperimentalActivity :NumericalExperimentalActivity)))
```

Hybrid simulations are classified by the reasoner according to this entailment, so providing an instance is related to at least one physical experimental activity and at least to one numerical experimental activity by means of the *hasExperimentalActivity* property, the instance will be classified as hybrid. Figure 5.12 shows an example of how a hybrid simulation is classified as

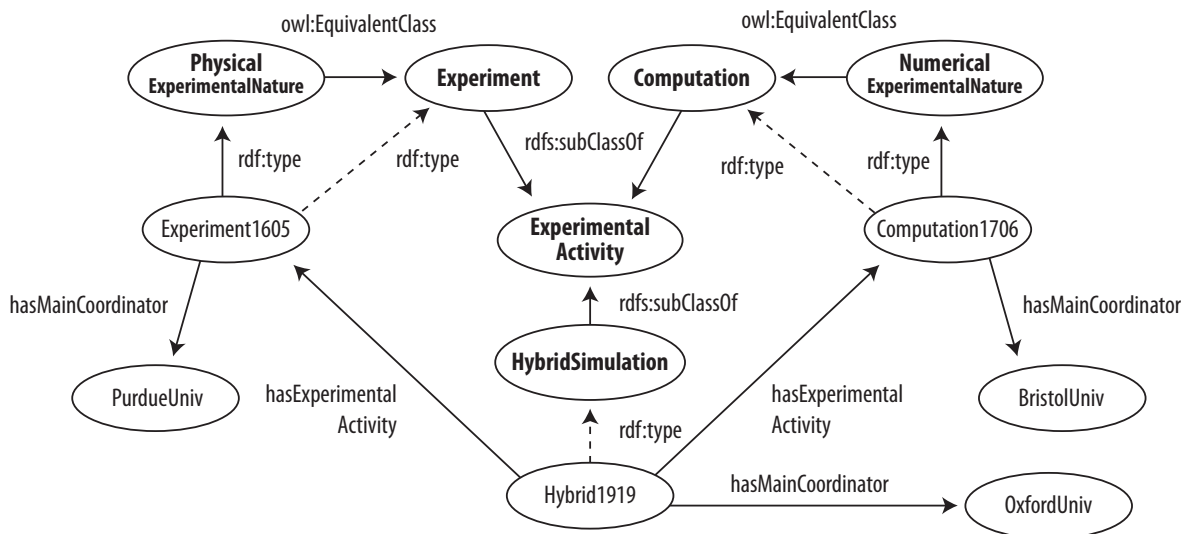


Figure 5.12. Example of a hybrid simulation entailment

such. As shown in the figure, each of the experimental activities can have their own coordinator or conductor, which is a common case in distributed hybrid simulations.

Pseudo-dynamic tests are more difficult to classify from the concepts physical and numerical. The nature of PsD tests without sub-structuring seems to be purely physical, and therefore the concept PsD test cannot be equivalent to hybrid simulation, because no different substructures are tested. A PsD with sub-structuring could be something that tests sub-structures, but the definition does not restrict the sub-structures to include both physical and numerical components. Therefore, no logical rules have been created to infer the classification of PsD tests, which has to be explicitly asserted. For the rest of the experimental activities, the classification must only involve physical or numerical classes, and a reasoner can then infer the nature of the activity.

An example of a graph representing an experimental activity is depicted in Figure 5.13. Similarly to projects, experiments can have a status and be related to the specimen they test. Experiments can be related to the equipment (specific facilities and devices) that is used in their conduction. This use is propagated to the project under which the experiment is conducted by means of property chains. Provenance of the experimental activity data can be implicitly assumed to be equal to the provenance of the project data, but the property *isIntellectualPropertyOf* can be used as well on experimental activities to define the owner of a particular set of data.

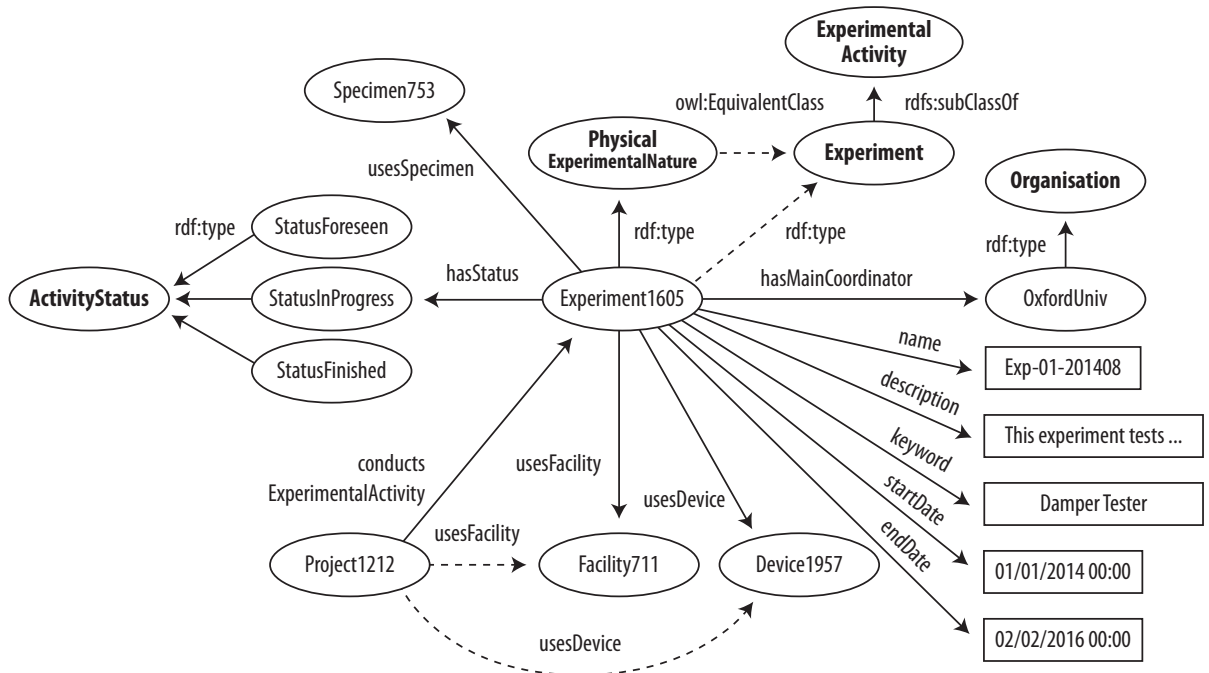


Figure 5.13. Example of the experimental module in the Celestina Data ontology

5.4.6 Experimental input / output

Experimental input refers to all data provided for the execution of an experimental activity whereas experimental output refers to all data produced from the execution of an experimental activity. The same experimental activity is sometimes re-conducted with slightly different input data in order to analyse the output produced. This concept of input and output is not a standard terminology in earthquake engineering, but represents an abstraction of what is called a *trial* (for physical experiments, the concept *run* is used for numerical computations) in the NEES data model, and *loading configurations* in the SERIES data model.

Sometimes, the same input data can generate different output data. The exact same experimental activity re-conducted under the same input data is called a *repetition*. Herein, repetitions only produce new (and sometimes equal between all repetitions) output information, but the rest of the modifiable parameters remain unchanged.

The Celestina Data ontology uses objects of the class *ExperimentalIO* that are linked to *ExperimentalInput* and *ExperimentalOutput* objects by means of the properties *hasExperimentalInput* and *hasExperimentalOutput* respectively. Each input and output object is related to data files, sometimes instances of the class *SignalData*. This class represents signals - paired vectors of values and times together with some related metadata, which are sometimes recorded by sensors. Signals might be used as input data for an experimental activity (*e.g.* the recorded accelerometer of an earthquake) or generated as output (*e.g.* the recorded vectors of forces and displacements captured by a sensor). An example of a graph representing experimental input and output is depicted in Figure 5.14. Output instances can specify the repetition number by means of the *experimentalRepetitionNumber* property. The use of facilities and devices can be specified per input and output instance. This use of devices and facilities is propagated by inference in the hierarchy, and eventually propagated to the project level as shown in Figure 5.13 (note that the propagation of *usesDevice* to *Output1*, *Execution1* and *Experiment1605* is not shown in Figure 5.14 for clarity reasons, but it is depicted in Figure 5.15). To achieve this propagation, property chains are used.

Objects of the class *DeviceDataStream* represent the use of a device that produces/uses a signal in an experiment. They are related to objects of the classes *ExperimentalInput* and *ExperimentalOutput* by means of the property *hasDeviceDataStream*. Different subclasses of

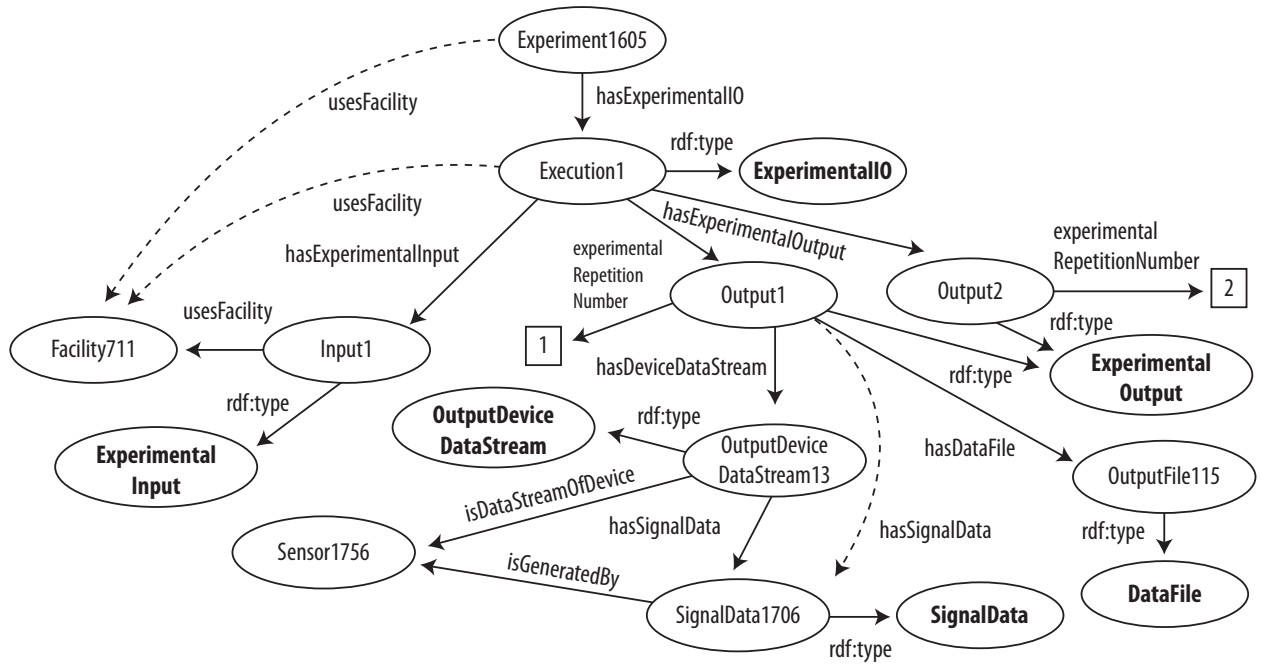


Figure 5.14. Example of the experimental input / output in the Celestina Data ontology

DeviceDataStream are defined for input and output.

Devices can have temporal information that is only valid within an experiment, such as the location and the calibration of the device. These are represented by the classes *ExperimentalLocation* and *Calibration* respectively. As depicted in Figure 5.15, this information can be related to the *DeviceDataStream* object, and its use is propagated to the actual device by inference via property chains. Location and calibration are further explained in Section 5.4.7.

5.4.7 Facilities and devices

The equipment used in experimental activities is constituted by facilities and devices. Facilities are represented by the class *Facility*, and are defined as a physical system or group of physical systems that support experimental activities, which are fundamental physical features of the experiment and work atomically (meaning that it cannot be divided into sub-systems). This concept has some subtle meaning. For example, a facility is not a device in the sense that a facility also determines or might restrict the type of experiment conducted, not like a sensor or an actuator that is a fairly more general purpose device. A facility is a physical feature of the experiment, such as a reaction wall, a shaking table, a strong floor, a centrifuge system or a supercomputer centre. However, a computer itself is not considered a facility in the ontology (as it is in the SERIES data model). A facility should be the minimum system that works atomically, which differs from the use of the

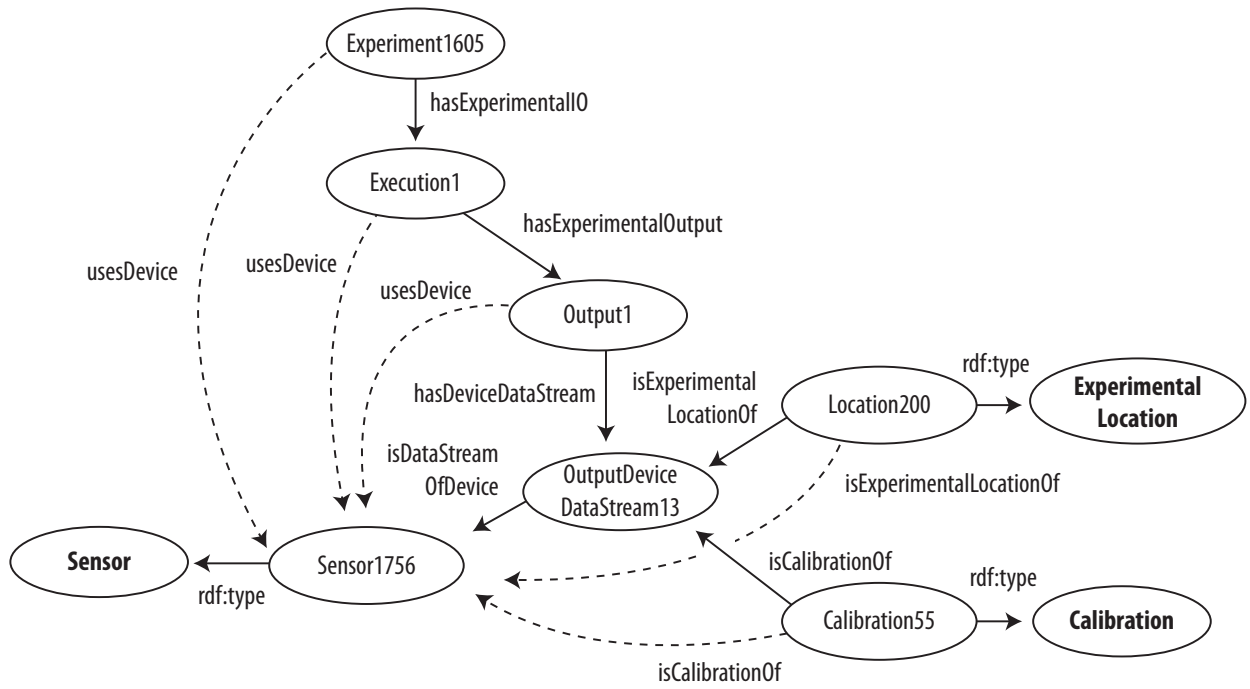


Figure 5.15. Example of device calibration and location in an experimental input / output

same concept in the NEES data model, where a whole laboratory is considered a facility (*e.g.* the Buffalo facility). This aspect is important in order to identify the exact system that was used in an experimental activity. For example, the Structural Dynamic Laboratory at Oxford is not a facility, because different sub-systems in the laboratory can be identified to work separately. On the other hand, the strong floor inside the laboratory is a facility.

This new definition of facility will require some adaptation time to users accustomed to a more general meaning of the concept, such as NEES data model users, but it will benefit the quality of the data by avoiding ambiguity and by defining explicit systems used to conduct experimental activities.

Devices, represented by the class *Device*, are physical objects used during an experiment, such as actuators, shakers or sensors. Sensors are typically treated separately in NEES and SERIES data models because of the role they play in capturing data from a physical experiment, but conceptually they are defined as another type of device in the ontology. Sensors, and other types of devices, have to be calibrated over the time, and calibration information is represented by the class *Calibration*. Calibration information is embedded in the device/sensor data in the SERIES data model, whereas it is defined in an external table in the NEES data model. In the ontology, calibration information can be defined for any type of device. Calibration information depends on time, so the same device can be used in two different experiments after two different calibrations. To achieve this, Calibration

instances are related to devices by the property *isCalibrationInformationOf* and to the experimental activity by the property *hasExperimentalEnvironment*. This avoids having to look for a specific date between all the device calibrations to obtain the calibration valid during an experiment. Calibration objects can use specific calibration properties such as *hasCalibrationSensitivity*, *hasMaxCalibrationMeasuredValue* and *hasMinCalibrationMeasuredValue*, all of them linking to blank nodes using the properties *valueForUnit* and *unitForValue*. Alternatively to the properties for measured values (that, for example, do not exist in the SERIES data model), the property *calibrationMeasuredRange* can be used. Additionally, calibrations can specify a label for a sensor channel by means of the property *sensorChannel*. However, the specification of the calibration channel is modelled by relating the calibration to an *OutputDeviceDataStream* object, as presented in Section 5.4.6.

Sometimes it is important to store the location of devices and other elements inside the experimental environment, because their location can make a difference in the experiment information *e.g.* the location of a sensor or the place where an actuator is attached to a specimen. Such information is specified by means of instances of the class *ExperimentalLocation*. Location of objects in an experimental environment are defined by position coordinates (x, y, z) and orientation

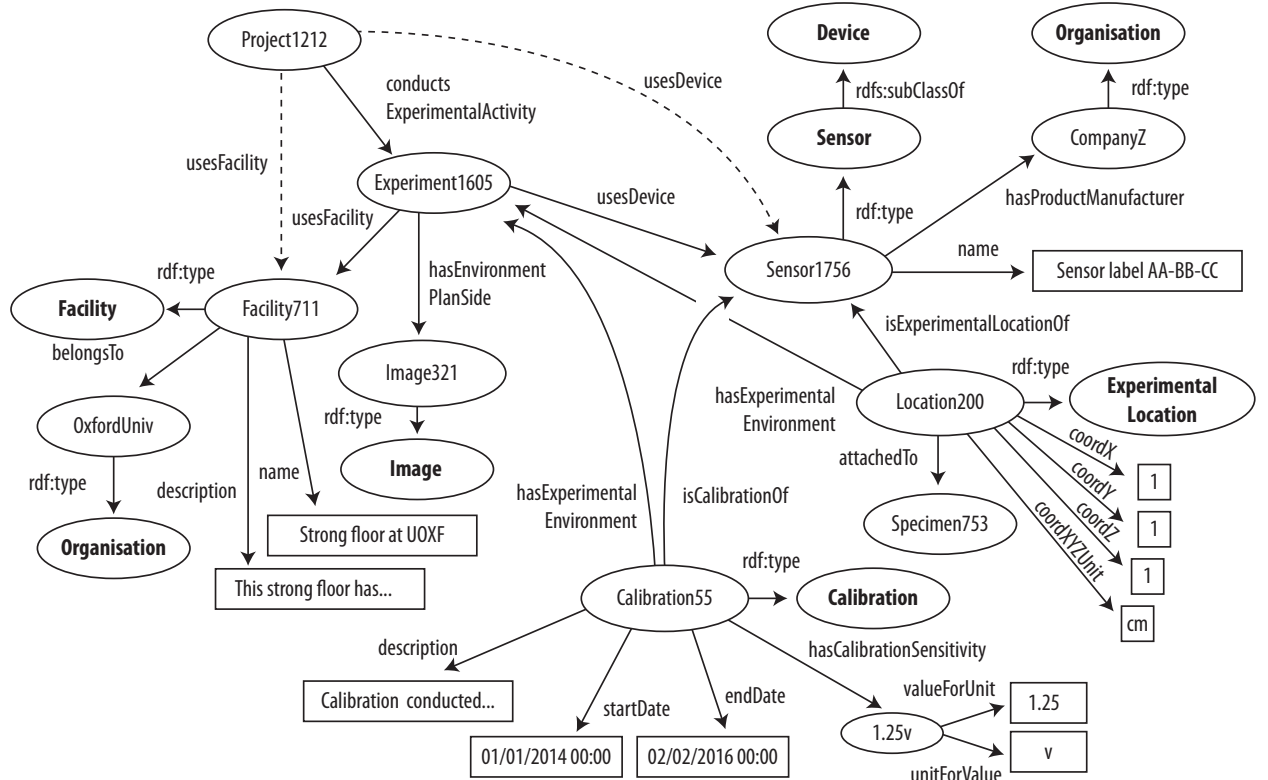


Figure 5.16. Example of the facility and device module in the Celestina Data ontology

coordinates (i, j, k). Objects attached to other objects can use the property *attachedTo* in the specific location. For example, two different *ExperimentalLocation* can be used to define the location of an actuator attached to two different specimens at two different points. The property *hasExperimentalEnvironment* relates the location object with an experimental activity. Because obtaining coordinates of objects in an experimental environment can be a complex task, the ontology supports the definition of images containing plans or blueprints of top and side views of the experimental environment. These are defined over the experimental activity by means of the properties *hasEnvironmentPlanTop* and *hasEnvironmentPlanSide* and they can show the experimental floor plan depicting all devices, facilities and specimens. By using them, tools can support the definition of relative coordinates of elements with respect to these plans. An example of a graph showing facility and device information is depicted in Figure 5.16.

5.4.8 Persons and organisations

Persons and organisations are considered a vertical module because such entities can participate in many of the activities represented in the experimental data. For example, they can fund a project, conduct an experimental activity, participate in the construction of a specimen, be principal investigators of a project, *etc.*

In a relational model, these relationships would be modelled by means of a junction table containing an attribute to store the type of relation. Typically, modelling this table in RDF would use an auxiliary node as depicted in Figure 5.17, linking a person/organisation, an activity (*e.g.* project, experiment, *etc.*) and a role (*e.g.* principal investigator, sponsor, *etc.*). By using a property chain and *rolification* (as explained before), a reasoner can infer the specific property linking a person/organisation with an activity, *e.g.* the *isTestAgentRole* property in the figure.

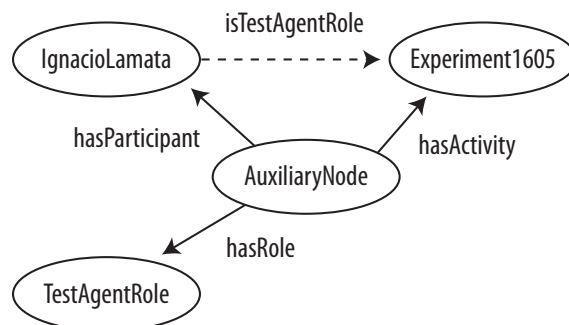


Figure 5.17. Auxiliary node representing the role of a person in an experiment

ontology under the class *DataFile*, that represents non-metadata information in the NEES data model and large files in the SERIES data model. Regardless of the class name, the way the data is stored is not limited to actual files.

The class *DataFile* has two subclasses: the class *Media*, subdivided in itself into the classes *Document*, *Image* and *Video*, and the class *SignalData*. A *DataFile* instance of any subtype represents a resource that can be accessed via Web.

DataFile instances can be referenced by different properties according to the specific use of the instance by other elements. Examples of these properties are *hasJournalPublication*, if the document represents a journal paper for a project, *hasExecutiveSummaryDocument*, if the document is an executive summary of a project, *hasConstructionDocumentation*, used for images, documents and videos about the process of the construction of a specimen, *etc.* The property *hasDataFile* is a superproperty of all properties referencing *DataFile* instances.

An example of a graph showing media and data information is depicted in Figure 5.19. As illustrated in the figure, signals contain vectors of data represented by the class *DataVector*. Instances of this class contain a vector of values, related by the property *vectorData*. In order to interpret correctly the values of the vector, the property *vectorDataFormat* is used to specify a particular format (*e.g.* binary *little endian* integers, comma separated float values, *etc.* Further

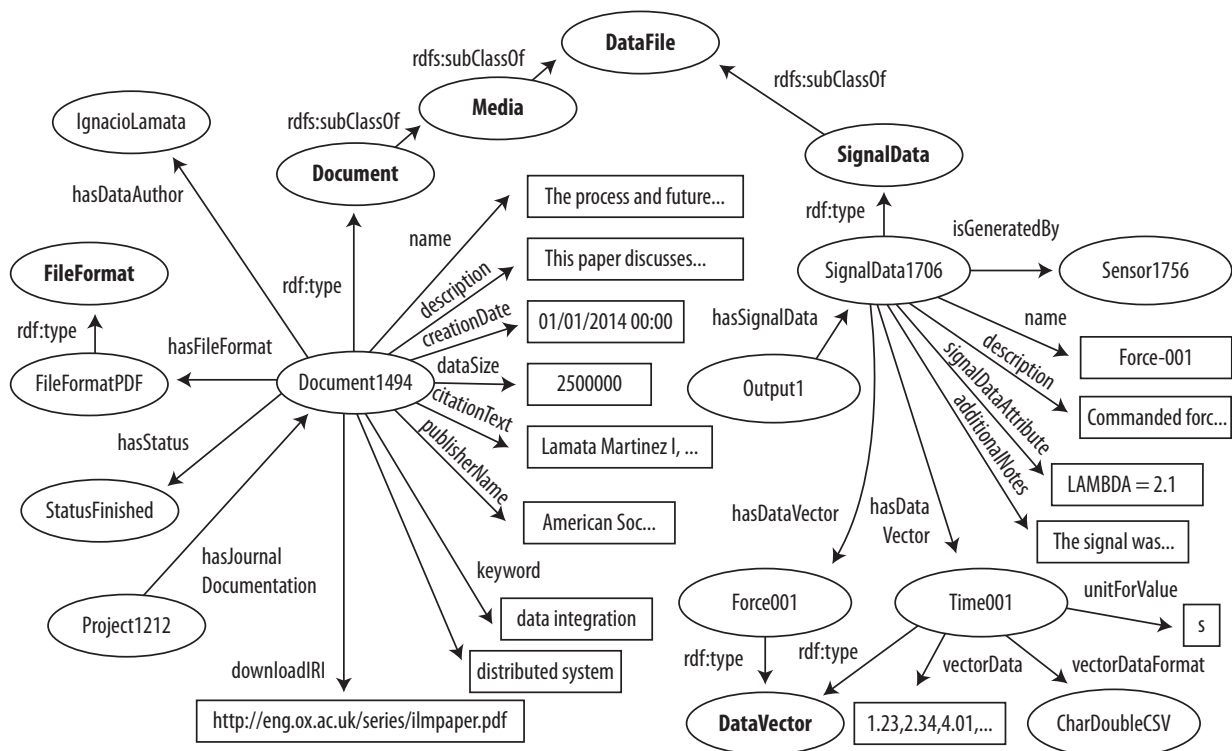


Figure 5.19. Example of the media and data module in the Celestina Data ontology

discussion about these formats can be found in Section 9.5.2). A signal can have multiple vectors of data, such as the vector of data for force values and its corresponding vector of time values when such forces were recorded.

5.5 Validation

The quality check of the ontology has been divided into two groups: (i) verification of the correctness and complexity of the ontology from a logical point of view, and (ii) verification of the suitability of the ontology as a method to store, integrate and exchange earthquake engineering experimental data. The first group is discussed in this section whereas the second group is discussed in Chapter 6, since it requires actual experimental data and the support of an additional technological infrastructure.

The correctness of the ontology from a logical point of view has been verified by means of reasoning tasks, such as *ontology consistency* and *class consistency* as discussed in Section 5.3.3. This has been conducted with the support of a reasoner. In general, OWL reasoners can be divided into two groups according to the reasoning algorithm implemented, as identified by Motik *et al.* (2012b). Those using *tableau calculi*, as explained by Motik *et al.* (2009), are more suitable for classification and other reasoning over terminological axioms (over a TBox, as defined in Appendix C). Examples of these reasoners are HermiT, Pellet, FaCT++ and Racer. Those using *materialisation*, as explained by Motik *et al.* (2014) and Horrocks (2012b), are more suitable to query assertional data (an ABox, as defined in Appendix C) and therefore this technique is typically used by reasoners of triplestores. Examples of these reasoners are Jena, Sesame and Oracle 11g.

For the validation of the ontology, *HermiT* (Horrocks *et al.*, 2012; Shearer *et al.*, 2008) has been chosen. HermiT is an OWL reasoner based on *hypertableau and hyperresolution calculi*, that can classify ontologies (compute class and property hierarchies by obtaining all subsumption relations) and perform consistency tests.

To validate the ontology, test data have been used by creating individuals for each of the classes. HermiT has been run and the ontology has been corrected until it proved to be consistent and coherent.

In terms of complexity, the ontology is in OWL DL, which guarantees that its reasoning algorithms are decidable in *N2ExpTime* time, as depicted in Table 5.2 (Kazakov, 2008). However, many parts of the ontology are in some OWL lightweight profile and some axioms in the ontology have been simplified to allow less complex reasoning. The main part preventing the ontology from being in OWL RL is related to the experimental module, specifically with the following axioms:

- `EquivalentClasses(:ExperimentalActivity ObjectUnionOf( :PhysicalExperimentalActivity :NumericalExperimentalActivity))`, which is the equivalent to $ExperimentalActivity \equiv PhysicalExperimentalActivity \sqcup NumericalExperimentalActivity$. OWL RL does not allow disjunction ($\sqcup$) at the right side of the equivalence. This axiom is not strictly necessary in current reasoning and could be removed.
- `EquivalentClasses(:HybridSimulation ObjectIntersectionOf( ObjectSomeValuesFrom (:hasExperimentalActivity :PhysicalExperimentalActivity) ObjectSomeValuesFrom (:hasExperimentalActivity :NumericalExperimentalActivity)))`, which is the equivalent to $HybridSimulation \equiv \exists hasExperimentalActivity. PhysicalExperimentalActivity \sqcap \exists hasExperimentalActivity. NumericalExperimentalActivity$. OWL RL does not allow the existential quantifier ($\exists$) at the right side of the equivalence. This axiom is necessary in current reasoning to categorise hybrid simulations properly. It could only be removed if hybrid simulations are explicitly categorised without needing the support of a reasoner.

The benefit of using OWL RL reasoning is that it can be conducted in polynomial time and that it is supported by a variety of tools, including Stardog, Jena, Oracle 11g and GraphDB (Glimm *et al.*, 2012). Note that the development of an ontology that uses logical features that allow reasoning algorithms with good computational properties (such as the OWL lightweight profiles) is a good decision, but it does not guarantee that reasoning time will meet time requirements of our application in practice.

			
OWL profile:	Lightweight profiles (EL, RL, QL)	OWL DL	OWL Full
Complexity of reasoning:	<i>Polynomial time</i>	<i>Nondeterministic doubly exponential time</i>	<i>Undecidable</i>

Table 5.2. Complexity of reasoning of OWL EL, OWL RL, OWL QL, OWL DL and OWL Full

5.6 Conclusions

This chapter has presented a new data model for earthquake engineering experimental data, developed in a novel technology. Semantic Web technologies provide a way to structure data in a standardise way, and make it accessible and shareable via Web. This technologies have shown some benefits over the traditional relational model, some of which are highlighted below:

- Behaviour of the data is declared within the data, without requiring to be created externally. This indeed makes a clear distinction between data and programming over the data. For example, it is not necessary to create software to identify that an experimental activity over both physical and numerical components is a hybrid simulation - this is declared in the data model itself.
- The technologies are designed to represent data from distributed sources and are well-prepared for sharing. For example, the use of universal identifiers by means of IRIs present a clear advantage in a distributed environment with respect to local relational model identifiers.
- Ontologies might be slightly easier to maintain than the relational model analogous schemas. Modification of the ontology just requires editing of the ontology file and the existing data can coexist with the modifications as long as there is no inconsistency. Modifications on relational databases could be more complex and require modifications on the data before being accepted in the database management system.
- Ontological data can be incomplete. This is useful in a distributed environment because not every institution has the same data policies. Also, the technologies allow the same data to be completed by different distributed data sources.
- It is a standardised way to implement a data model and share data, which might facilitate the integration with ontologies of other hazard mitigation disciplines.

Other benefits of Semantic Web technologies over the relational model will be explored in Chapter 6.

There exist tools that automate the construction of an ontology from a relational database, such as the *R2RML* language as described by Das *et al.* (2012). However, manual development of the ontology has demonstrated to provide some benefits over automatic solutions:

- It provided more accuracy and flexibility. For example in the implementation of junction tables, where manual decisions could be taken.
- The ontology does not intend to integrate existing data sources, but to serve as a data model for any earthquake engineering institution, improving quality of existing sources whenever possible and, sometimes, changing the meaning of some concepts in existing data models. This could only be achieved by manual development.

In terms of data modelling, some conclusions are presented below:

- The use of few data properties that can be reused has proved to simplify the ontology without affecting expressivity (*e.g.* a single property *name* instead of different name properties per class: *deviceName*, *projectName*, *etc.*).
- Usual junction tables from a relational model containing a single attribute for the type of relation have been modelled as properties in the ontology, which simplified the model. More sophisticated relational junction tables have to be modelled as auxiliary instances that are related to the other elements involved.
- Labels (*rdfs:label*) and names (*celestina:name*) have demonstrated to play a similar role except in a few exceptional cases. The *name* property might be removed in future if no specific functionality is demonstrated.
- Annotations and contextual information for humans have proved to be of great help for the creation of assertional information in a correct way. Every class and property has been annotated with *rdfs:comment* properties.
- The use of inverse properties has been avoided to simplify the ontology.

The ontology presented in this chapter is not a final product, and it should be extended and modified over the time according to data requirements. Some of the future work should focus on providing data with more structure (*e.g.* modelling in the ontology some data that are currently only described in documents), enabling additional features (*e.g.* the definition of a *loading coefficient* for signals, to indicate that the signal was loaded in an experiment at, for instance, 30%), and creating additional axioms to extend the support for data integration (*e.g.* the definition of instances for units - instead of using literals, and the definition of classes for physical properties).

Chapter 6

Celestina Data

As in many other scientific fields, data are one of the main resources of earthquake engineering. The ability to work with uniform data structures not only facilitates collaboration in a scientific community, but it can improve the quality of research outcomes and ease evaluation processes. Chapter 5 has presented a conceptual data model for earthquake engineering, called the *Celestina Data ontology*. This serves as the foundation for building applications to work with experimental data, but it needs an underlying technical infrastructure to organise the access to the different data sources.

This chapter discusses the design and implementation of the prototype of **Celestina Data**, an infrastructure for data interoperability of hazard mitigation disciplines, which aims to make the world-wide integration of experimental earthquake engineering data possible. Although Celestina Data is supported by the Celestina Data ontology, it can be enriched with additional hazard mitigation data models, such as those for winds and fire.

This chapter is organised as follows: Section 6.1 lists some of the requirements that Celestina Data should meet. Section 6.2 provides the overview of the Celestina Data architecture, whose implementation is discussed in Section 6.3. Section 6.4 discusses the validation conducted to evaluate the usefulness and correctness of the architecture. This has been done by means of a practical application, and some results from this validation are discussed in Section 6.5. Sections 6.6 and 6.7 discuss future concerns of the platform, such as potential applications of Celestina Data and its maintenance. Finally, Section 6.8 presents some conclusions.

6.1 Basic requirements

A list of some requirements have been identified:

- It is a requirement that participants are considered authors of their own data and have control over the data access (RQC1).
- Celestina Data must coexist with existing data systems, such as the ones at SERIES and NEES. In this sense, Celestina Data is not a replacement for the different data systems, but an additional access point of data services for the benefit of the scientific community (RQC2).
- The system must be able to scale easily, in two different aspects: (i) additions of new institutions that want to join Celestina Data, ideally without any impact on the institutions in operation, and (ii) additions of new hazard mitigation disciplines (RQC3).
- Celestina Data must promote its infrastructure as a future standard for the access of hazard mitigation data globally, as well as provide guidance for institutions to join, both those having an existing data source and those having no formalised manner to store their experimental data (RQC4).
- The system must be open to future applications (RQC5).
- All data in Celestina Data must be public (RQC6).

The following sections discuss the system architecture, that will meet this summary of basic system requirements.

6.2 Architecture overview

This section presents the technical design of Celestina Data. This infrastructure aims to be a system for the inter-communication of computers, the system is not intended to be accessed directly by humans. Use and display of data is delegated to additional systems, such as interfaces to present the data to the end user in a user-friendly way.

The general structure of Celestina Data is depicted in Figure 6.1, which presents a high-level view of Celestina Data in four layers. At the bottom layer, the data sources can be found. These are normally existing data sources, such as those of NEES or SERIES. Their data have to be extracted and sometimes translated into the Celestina Data model. This satisfies requirement RQC2, asserting that Celestina Data must coexist with existing data systems. The second layer is the Celestina

Data infrastructure, which offers a conceptual layer to extract information from the data sources in a uniform, ontology-based manner. This is the key layer that makes data integration possible. Different ontology modules are depicted in the figure, such as those for earthquake engineering and wind engineering. Note that the ontology discussed in Chapter 5 only consists of earthquake engineering experimental data, but it can be extended to other hazards. The subsequent layer is the application layer, which represents the software that uses the data. This layer is what makes sense of the data interoperability and utilises the data in a practical way, offering a benefit for the end user. Example of applications are Web interfaces to present data, or data visualisation tools. This layer is also called *Celestina Tools*, and it is further discussed in Section 6.6. Finally, the uppermost layer represents the users, primarily consisting of the scientific community, who are the consumers of the data.

6.2.1 Constituents

The main constituents of Celestina Data are the following:

- **Celestina Data nodes**, which offer data services for the applications. Each of these nodes provides data in the conceptual data model discussed in Chapter 5. Nodes are discussed in Section 6.2.3. They host the data sources of Figure 6.1.

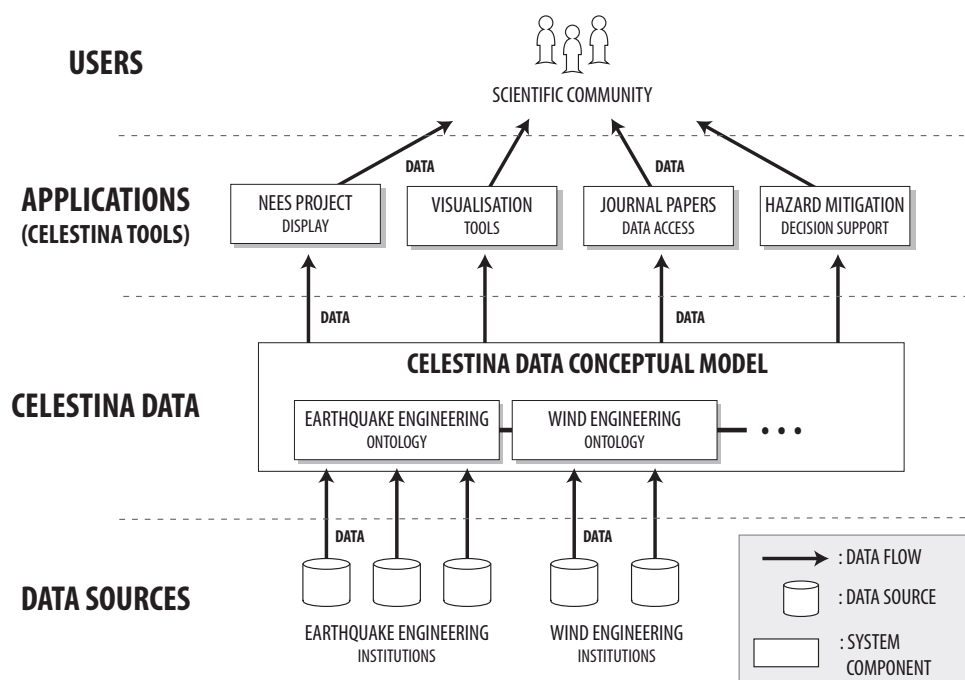


Figure 6.1. Celestina Data four layer structure

- **Celestina central site**, which is a neutral, centralised entity in charge of the different ontology modules and the official directory of Celestina Data nodes. The central site is discussed in Section 6.2.4.
- **Applications**. The third constituent are external systems, which consist of data sources that produce data and applications that consume the data. Although these systems are conceptually considered external to Celestina Data, they can be often found inside the technical infrastructure of a Celestina Data node. For this reason, data producers will be discussed within the Celestina Data node, whereas data consumers (applications) will be discussed in Section 6.2.5.

Since Celestina Data is a system for the inter-communication of computers, it has no direct human constituents. The general architecture of Celestina Data is presented in Section 6.2.2.

6.2.2 General architecture

This section presents the system interaction between the constituents introduced in the previous section. A schematic view of the Celestina Data architecture is illustrated in Figure 6.2. This shows three nodes, named *A*, *B* and *C*, the application *Z*, the Celestina central site and the end users. Celestina Data nodes provide an access point to be accessed by applications, represented by the circle in the figure. This access point is publicly available, and any external system can access it to obtain data in the Celestina Data ontology format. The figure shows how *application Z* is connected to the data access points of nodes *B* and *C*. Applications connect to the data access point directly, with no intermediaries.

The way an application knows where to find data access points (the addresses/IRIs of the Celestina Data access points) is by asking the Celestina central site, which stores a *resources ontology* containing every Celestina Data node registered in the system. Note that Celestina Data nodes do not really need to be registered in the central site to operate, but this helps to keep a centralised, official directory of nodes, where applications can find data services. The central site also keeps the official, most-updated version of the different data models (such as the one discussed in Chapter 5). Data access points are connected to local data sources that store the actual dataset.

Finally, applications provide services for the end users, who do not actually know if the data come from Celestina Data or any other system. Users can also access a Website in the central site, to obtain information and technical details about the Celestina platform.

The architecture depicted has been designed to be flexible and it only requires nodes to provide a data access point to serve data in the Celestina Data ontology format. The following sections discuss with more detail the elements of the system and how the data access point works.

6.2.3 Celestina Data nodes

A Celestina Data node is a system offering Celestina Data services. Internally, nodes can use very different hardware and software configurations. Celestina Data does not intend to replace existing (relational) data systems but to add a high-level, unified view to access heterogeneous data sources conceptually. For the nodes with existing databases, such as the ones at SERIES and NEES, this has been addressed by means of an *ontology-based data access* approach (OBDA), which satisfies RQC2. OBDA is a well-known method, actively researched, to query non-ontological sources in the Semantic Web (Poggi *et al.*, 2008; Kontchakov *et al.*, 2011). Examples of this are the projects *Optique* (Calvanese *et al.*, 2013), *ExODA* (EXODA, 2013), *REQUIEM* (Pérez Urbina *et al.*, 2009) and *MASTRO-Studio* (Civili *et al.*, 2013). The basic concept behind OBDA is to provide access to relational or other type of heterogeneous data through an intermediate ontological layer. There are two main approaches to OBDA:

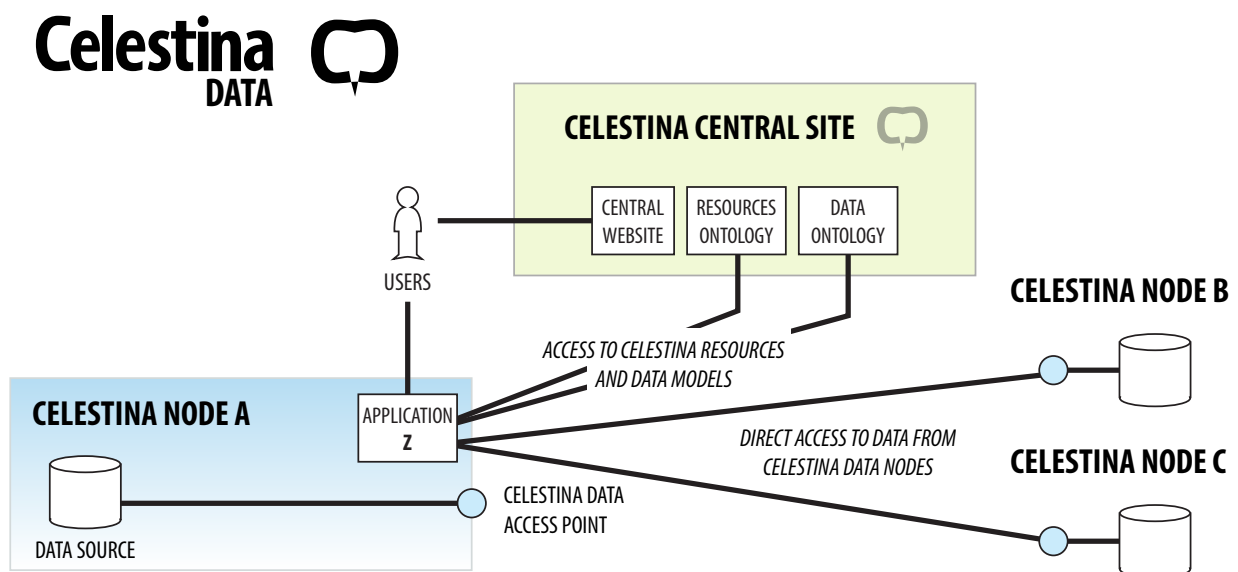


Figure 6.2. General architecture of Celestina Data

- **OBDA by data transformation.** As depicted in Figure 6.3a, the relational databases export their own data into an ontological format (serialised in RDF, see Section 5.3.1), that can be download and queried locally with an ontology-based query. The query is executed by the consumer of the data. Different methods to export relational data into RDF have been gathered by Sahoo *et al.* (2009). A standardised language to help with the export of relational data into RDF is *R2RML* (RDB to RDF Mapping Language), as described by Das *et al.* (2012). R2RML is a language that permits expressing sentences in a mapping document to translate a relational schema into RDF by using an *R2RML processor*. An alternative method is a direct mapping, as described by Arenas *et al.* (2012).
- **OBDA by query transformation.** As depicted in Figure 6.3b, the relational databases are queried with an ontology-based query. At query time, a query rewrite is conducted in order to transform the ontology-based query into a SQL query that the relational database natively understands. Systems typically use a mapping that relates data from the data source to the vocabulary of the ontology. The query is executed by the producer of the data. An example is *Ontop/QUEST* defined by Rodríguez Muro *et al.* (2012) and Rodríguez Muro *et al.* (2012b), which has support for *OWL QL* entailments (Krötzsch, 2012; see Section 5.3.2 for extended information about OWL). Similar functionality for querying “virtual” RDF graphs is offered by MASTRO, described by Calvanese *et al.* (2011), Virtuoso RDF views, described by Erling *et al.* (2013), and D2RQ (D2RQ, 2013).

Data sources that are implemented by triplestores provide ontological access natively, and do not require additional elements as an OBDA approach does, as depicted in Figure 6.3c.

The choice of one OBDA approach over the other has some subtle architectural implications, both external (the way data is accessed by external entities) and internal (the configuration of components inside the node). Externally, an OBDA by query transformation implies that the local node receives a query, executes it and returns a set of results. Processing of the query is delegated to each of the nodes. On the other side, OBDA by data transformation exposes data that can be accessed by an external system and queried at the external system once it is downloaded. Note that this difference is less significant than it might seem, since OBDA by data transformation could use additional systems to conduct queries locally at the node side, and OBDA by query rewriting could return a dataset that can be queried externally.

The solution adopted has been an OBDA approach by data transformation. This is justified by the following reasons: (i) Less complexity: the data transformation requires a new component to export relational data into RDF but, normally, existing systems can be used to serve the resulting RDF data via Web. This approach might be simpler and does not require the installation of an additional system (at the server side) to execute queries as in OBDA by query transformation. (ii) More secure: new publicly accessible systems might have security flaws that can be externally exploited. If no additional software is required, it means less exposure. (iii) Less network traffic for whole datasets: one advantage of OBDA by query transformation is that it could generate less network traffic, considering the query results do not return the whole dataset. However, many applications need to use several queries that eventually return the whole dataset, and in those cases the network traffic can be greater than serving the whole dataset at once. (iv) Independence of query processor: the query engine is located externally at each application, instead of at each node. This allows uniformity when data is queried, avoiding potential problems of the particular query engine installed at each node. (v) Data can be queried as a whole: queries can be executed over the whole merged data. This presents some advantages over executing the same query on the isolated dataset at each node, which could imply that certain queries could not be answered efficiently.

Nodes are expected to provide a Web-accessible RDF file in the Celestina Data ontology format (the different *ABoxes*, as described in Appendix C). Additional more complex data services can be provided by nodes, but those are considered optional (*e.g.* a Web-accessible query processor

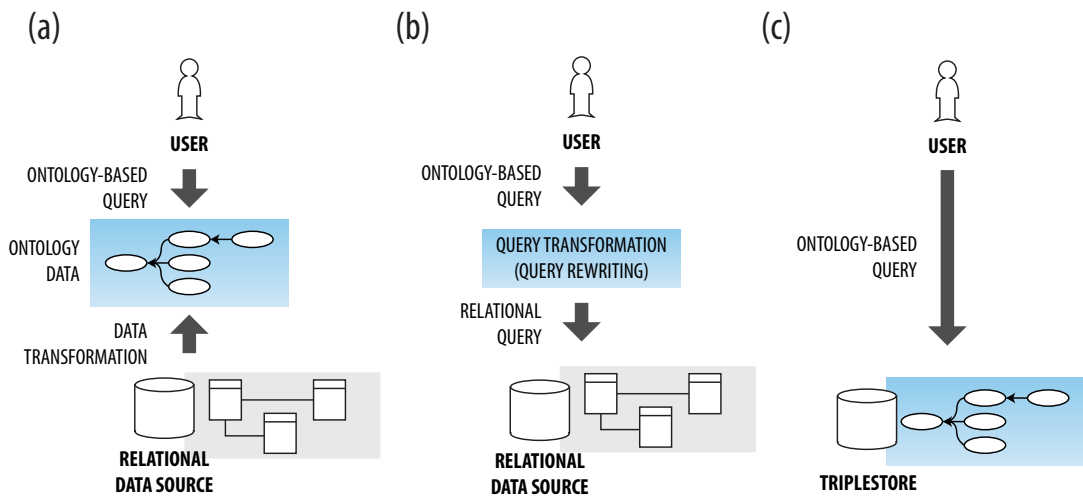


Figure 6.3. Types of OBDA approaches (a) by data transformation (b) by query transformation (c) using a triplestore

that can receive a query and will return a set of results).

6.2.4 Celestina central site

The Celestina central site plays a crucial role in the architecture of Celestina Data. Firstly, it serves as the primary information point for the whole Celestina platform, including all its subsystems, *Celestina Data*, *Celestina Tools* (discussed later in this chapter) and *Celestina Simulations* (discussed in Chapter 9). The Website has to provide policies on how the data are licenced and how they should be cited and used. These policies, together with the capacity of the ontology to describe data provenance (see Section 5.4.2), work towards the requirement RQC1, to support authorship of data.

It also contains the most updated versions of all ontological modules (the *TBox*, as described in Appendix C), such as the Celestina Data ontology discussed in Chapter 5, that can be used by applications when querying RDF data obtained from Celestina Data nodes. It should be noted that Celestina Data is based on a versioned, centralised ontology, which has less overhead and is much easier to implement by multiple institutions than multiple distributed ontologies. In the cases of disparity of data, data that are only relevant for a single institution and concept disagreement, site-specific ontologies can be developed locally at each node.

Finally, the central site maintains a list of official Celestina Data nodes and their contact details in a *resources ontology*. The decision of having a centralised directory of nodes, instead of other network structures such as the decentralised schema for Celestina Simulations discussed in Section 9.2.1, is based on management requirements: (i) Data sources are expected to be continuously available (not only at simulation execution times) and need to be monitored. (ii) If data have to be curated, a directory of data sources is more favourable. (iii) A centralised management is also more appropriate to verify global consistency in the data (see Section 6.7.2).

The procedure to include new nodes in Celestina Data will be discussed in Section 6.7. Although the first version of the resources ontology only includes Celestina Data nodes, note that the ontology is intended to be extended with more types of Celestina nodes that provide different Celestina services (*e.g.* a list of active Celestina Simulations sky nodes as defined in 9.2.1).

6.2.5 Applications

Applications are any external system that makes use of the services provided by a Celestina Data node, and constitute the gate to Celestina Data services for the end user. Applications accessing data might need to connect to the Celestina central site to obtain the list of nodes that provide data services. Then, they can decide which of these nodes they need to access, connect each of the nodes directly, collect the data, process them and present them to the users of the application in a user-friendly way. The data updating process lies on the applications, in a similar way as explained in Section 4.4 for the SVDB Central site.

Since Celestina Data is publicly accessible, applications are suggested to incorporate *politeness*, meaning that responsible access to Celestina Data nodes must be done by avoiding the generation of unlimited requests to the nodes. Note that this is regardless of the measurements that Celestina Data nodes have to implement to deal with *impolite* applications.

Applications are considered in a Celestina sub-system called *Celestina Tools*, which is further discussed in Section 6.6.

6.2.6 Security considerations

Celestina Data does not define any security policy, and delegates security measurements to the nodes themselves. Information from Celestina Data will be often transmitted through public networks, but, by definition, this information is public. Herein, confidentiality of information is not required (see Section 4.2.5 for definitions of security concepts). On the other hand, the information transmitted is subject to unauthorised modifications, and integrity of information might be desired. Nodes can decide whether to establish secure communication channels on their Celestina Data servers (for example, by means of *HTTPS*¹).

Availability of the Celestina central site is important for the system operation for two main reasons: (i) it is the centralised point where the list of Celestina services (provided by Celestina Data nodes specified in the resources ontology) is stored, and (ii) it is the site where all the most updated data models (the different ontologies) are available. Again, security measurements are delegated, and it is suggested that applications take steps to minimise the risk of an unavailable

¹ *Hypertext Transfer Protocol Secure*, a protocol for secure Web communication.

central site, for example by caching latest known ontologies (this technique is implemented in the component described in Section 6.4.1). The central site can also establish secure communication channels to guarantee the integrity of the ontological resources.

6.3 Implementation

This section discusses a practical development of the architecture presented previously. Section 6.3.1 discusses the implementation of Celestina Data nodes, using as example the creation of a Celestina Data node from a SERIES node. As in the SERIES Virtual Database, large files are directly served by Celestina Data nodes. Section 6.3.2 gives some implementation details of large files downloading from SERIES nodes in Celestina Data. Finally, Section 6.3.3 deals with the implementation of the Celestina central site.

6.3.1 Celestina Data node implementation

Due to the architecture of the system, the implementation of an arbitrary data node is fairly simple. As long as there is a Web-accessible RDF file with data in the Celestina Data ontology format, the node can be part of Celestina Data. However, there exist more complex node implementations. In this section, the implementation of a Celestina Data node from a SERIES node is discussed. It has been implemented at the University of Oxford, but the software is automatically extendable to any other SERIES node, and a similar infrastructure could be used at NEES or any other node with an existing database.

SERIES nodes have all necessary software installed to participate in the Celestina Data infrastructure, in which new components have to be developed to implement the data export required by the OBDA by data transformation. The internal components of a SERIES -based Celestina Data node are depicted in Figure 6.4. The *data transformation* component is responsible for exporting database data into the RDF file², by mapping SERIES data into the Celestina Data ontology format. The *ontology support* component simply provides TBox ontological information.

² Note that, although represented with the same symbol as the SERIES database in the figure, the RDF file is a very simple data source

The actual data export has been conducted by a direct mapping reusing the SERIES Virtual Database (SVDB) components that access the SERIES database. New code has been developed when SERIES private database tables had to be exported for Celestina Data - for example, device and sensor information is always private in the SVDB, but it is part of the public data in Celestina Data. No information explicitly marked as not public has been exported, following requirement RQC6 of Section 6.1.

The export of some SERIES data into the Celestina Data ontology has not been completely straightforward, and has been accomplished in a best-effort manner. For example, output experimental files are defined per experiment in the SERIES database, so it cannot be distinguished to which input information they belong to in an experiment with several input configurations (see Section 5.4.6 for more details). In this situation, each experimental input has been linked to all the experimental outputs within the experiment. Likewise, the relationship between input files and devices does not exist in the SERIES database, so instances of *SignalData* were used without any *DeviceDataStream* in the experimental input (see Section 5.4.6). Also, classification of materials was done by guessing the material type according to a string search, and sensors were not sub-classified because SERIES data do not provide enough hints to attempt an accurate classification.

To keep an updated version of the RDF file efficiently, the creation process of the file is triggered by external applications. The component represented by the Web server in Figure 6.4

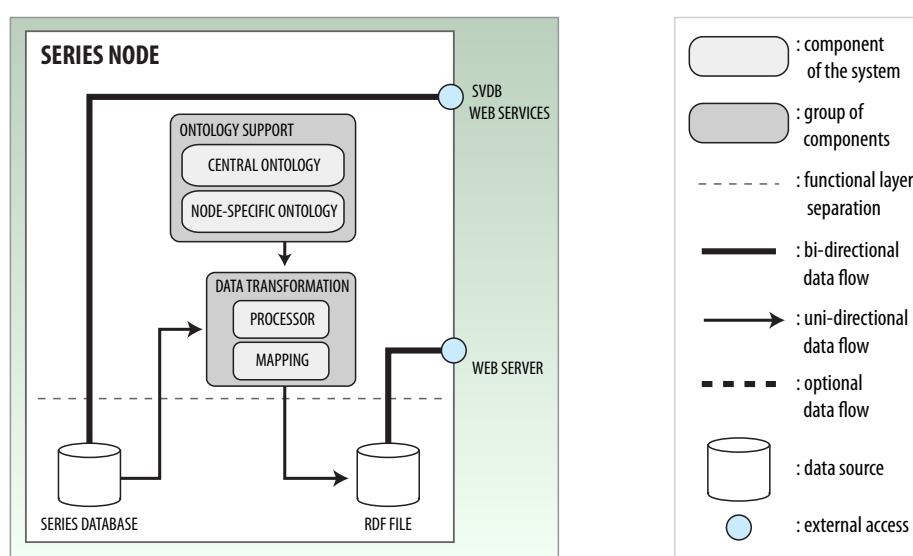


Figure 6.4. Internal configuration of a SERIES node for Celestina Data

is actually a *Java servlet* (a Java component that can receive HTTP requests) that triggers the exportation process when it is accessed. Once the export is completed, the servlet returns the RDF data transparently to the caller. A first attempt of this implementation is represented in the pseudo-code of Algorithm 6.1.

Algorithm 6.1. Inefficient RDF exportation with race conditions

```

1:  RDF _ DATA = EXPORT _ INTO _ RDF( SERIESDB )
2:  OPEN RDF _ FILE
3:  WRITE RDF _ DATA TO RDF _ FILE
4:  CLOSE RDF _ FILE
5:  REDIRECT REQUEST TO RDF _ FILE

```

This process has two main problems: (i) it is very inefficient, because the RDF has to be generated at every request, and (ii) it is bound to potential *race conditions* (explained in Section 4.4.4; a more complete explanation is given by Göetz *et al.*, 2006). To explain this, let us imagine the situation where a single caller application obtains data from the SERIES node. Every time the application accesses the node, the node processes the request, creates the RDF file and serves it to the application. This process finishes correctly. However, when two or more applications access the service at slightly different times, race conditions happen. The problem is that the code is not thread-safe, so while one application is downloading the RDF file, another application can access the node, which will run the RDF file creation process. This will overwrite the RDF file at the same time it is being read by the first application.

The solution to these two problems is to implement a cache with mechanisms that guarantee mutual exclusion³. The cache is created the first time the servlet is called, and then it is regenerated every time the cache is considered expired according to some time threshold. By using a cache, the node does not have to access the SERIES database for every request, so requests can be served very efficiently and performance is dictated by the time it takes to download the RDF file through the network.

In order to guarantee concurrent access, locking mechanisms have to be implemented, while considering the balance between efficiency and correctness. Very aggressive locking mechanisms will

³ Mutual exclusion is a locking strategy that prevents two processes to access the same resource at the same time.

serve requests correctly, but will not enable real concurrency, and application requests will queue to access the node. Very limited or incorrect locking mechanisms will accept multiple requests at the same time but will run into race conditions. Concurrent code is one of the most challenging subjects in programming because correct operation of concurrent code is very complex to verify, and behaviour of concurrent programs is difficult to predict. An example of this complexity is illustrated in a second attempt in Algorithm 6.2, which seems to work correctly but still has some problems, as discussed below.

Algorithm 6.2. RDF exportation using a cache with race conditions

```

1:  IF (CACHE NOT EXIST) OR (CACHE IS OLD)
2:      LOCK( LOCK _ OBJECT )
3:      IF (CACHE NOT EXIST) OR (CACHE IS OLD)
4:          RDF _ DATA = EXPORT _ INTO _ RDF( SERIESDB )
5:          OPEN RDF _ FILE
6:          WRITE RDF _ DATA TO RDF _ FILE
7:          CLOSE RDF _ FILE
8:          CACHE EXIST
9:      END IF
10:     RELEASE LOCK( LOCK _ OBJECT )
11: END IF
12: REDIRECT REQUEST TO RDF _ FILE

```

The algorithm shows a variable *cache* that is consulted to know if the cache exist and to determine if it has expired and needs to be renewed. The method *lock* uses a mutual exclusion mechanisms over the variable *lock_object* that guarantees that no other process in the system can lock the same object until it has been released. This way, it is guaranteed that only one process is executing lines [3] to [9] at any given time, and any further attempt to lock the object while it is not released will hold the process at line [2]. Lines [3] to [10] are called the *critical section* of the algorithm. This locking mechanism enables multiple processes to run part of the code at the same time (lines [1], [11], and [12]) but only one to run the critical section. While the cache exists and is not old, all processes seem to be able to run at the same time very efficiently.

The double check existing over the variable *cache* in lines [1] and [3] is done for efficiency reasons. Let us imagine that a process *P1* executes the code. The cache does not exist and *P1* enters the

critical section. While this process executes line [4], another process $P2$ executes the code, verifies the cache does not exist (because $P1$ is still executing line [4]) and waits at line [2] until the lock is released by $P1$. Once $P1$ finishes, it declares that the cache exists in line [8] and releases the lock. Without the double check of line [3], $P2$ (and any other subsequent process waiting) would have to erroneously regenerate the cache. This double check allows $P2$ to release the lock as soon as possible and continue operation. Although the algorithm seems correct, it has race conditions.

Algorithm 6.3. Correct but inefficient RDF exportation using a cache

```

1:  DEFINE CACHE VOLATILE
2:  IF (CACHE NOT EXIST) OR (CACHE IS OLD)
3:      LOCK( LOCK _ OBJECT )
4:      IF (CACHE NOT EXIST) OR (CACHE IS OLD)
5:          RDF _ DATA = EXPORT _ INTO _ RDF( SERIESDB )
6:          CACHE EXIST
7:      END IF
8:      RELEASE LOCK( LOCK _ OBJECT )
9:  END IF
10: LOCK( LOCK _ OBJECT )
11: SEND RDF _ DATA
12: RELEASE LOCK( LOCK _ OBJECT )

```

The first problem is that the RDF file can be modified at the same time that it is being sent to an application. Because the locking mechanism can only be applied to Java memory objects, and not to files in the file system, one process could execute line [12] and finish while a second process is allowed to execute line [6]. The solution to this is to use a memory variable instead of a file, and include in a critical section the code that reads the variable when the data is sent to satisfy the application request. One potential problem of this solution is the generation of RDF data that is too large to fit in memory.

A more obscure second problem is related to the variable *cache*. When a code is compiled, the compiler is allowed to make copies of the variables to improve performance. In this case, the variable *cache* will be duplicated locally for each process, so different processes could obtain out-

of-date versions of the same variable. The solution for this is to use a *volatile* variable⁴ (which are available in Java and many other programming languages).

Algorithm 6.3 does not have race conditions, and is valid for Celestina Data. However, it is inefficient because only one application is served at a time. The instructions that take more time are the ones from line [5] (the one that access the database and creates the RDF data) and [11] (the one that sends the data). One performance improvement is to place the code that creates the RDF data (line [5]) outside the critical section, assign the result to an auxiliary variable, allow other processes to use the “expired” *rdf_data* variable while the auxiliary variable is being created, and update the *rdf_data* variable in a critical section once the auxiliary variable is completed (this is similar to keep serving clients with the RDF file while a temporary file is used to export the database, and then replace atomically the RDF file with the temporary one). Nevertheless, line [5] is only executed when the cache has not yet been created or has expired.

The development of a component that serves data correctly and efficiently in a concurrent environment can be a complex task. The development of the extractor has been enriched with functionality to deliver compressed RDF data (for cases where the dataset is too large), to activate and deactivate the cache, to set up the expiration time of the cache and to enable a request to force a cache renewal (for testing purposes or when it is crucial to obtain up-to-date data before the next cache expiration threshold). This behaviour is depicted in the flow diagram of Figure 6.5.

6.3.2 Large files

Information of large files such as documents, images and videos (and any object represented by the class *DataFile* in the Celestina Data ontology, as described in Section 5.4.9) is divided into data and metadata. The metadata is defined as structured data in the ontology, but the actual data is only referenced by means of a download IRI in the ontology, and is normally only accessed by the end user via a Celestina Data application. Herein, similarly to the SVDB infrastructure defined in Section 4.3.2, Celestina Data nodes control their own data (which satisfies requirement RQC1) and must provide a method to download large files referenced by an IRI.

⁴ Volatile is a qualifier to make the compiler aware that the variable can be modified by an external process. Without this, the compiler can translate a high-level instruction in machine code containing unexpected behaviour and providing out of date shared data.

The method used for downloading of large files in the SVDB contains some security restrictions that can be reused on Celestina Data with slight modifications. The first modification is that SERIES nodes have to develop a component for the creation of legitimate IRIs to download large files. As described in Section 4.3.2, SERIES nodes accept download requests by an IRI, whose integrity is verified by a public-cryptography signature of a *trusted* institution. In the SVDB, this trusted institution is the SVDB central site. The second modification is that the own SERIES organisation serving the data is used as a trusted institution. Herein, when a SERIES node exports data into Celestina Data, it creates their own download links for large files and signs them itself.

Since Celestina Data is a world-wide infrastructure, *caching* will be probably a common technique to maximise performance (*e.g.* the cache described in Section 6.3.1). This forces SERIES nodes to accept download requests with longer expiration times, since a node can create a download link that is locally cached for several days and the link could be obtained and again cached by an application. By the time the download link reaches the end user, the link could have expired. By allowing longer download windows at the SERIES node, this risk is mitigated.

6.3.3 Celestina central site implementation

The Celestina central site consists of three main components: the central Website, the resources ontology and the data ontologies, which have to be web-accessible. Because of the Celestina neutrality discussed in Section 6.7, it has been decided to host the Celestina central site in an external commercial hosting.

A simple resources ontology has been developed. An example describing the Celestina Data node for the University of Oxford in this ontology can be found in Figure 6.6. The notation used in the figure is explained in Section 5.4. The resources ontology describes instances of the class *CelestinaDataNode*. These instances have a name (by means of *rdfs:label*), which must be unique, and a description (by means of *rdfs:comment*). The IRI address to their web-accessible RDF data is specified by means of the property *serviceIRI* and the syntax of this RDF content is defined by the property *serviceFormat*. Permitted formats correspond to the permitted encoding syntaxes for RDF, as defined by Schreiber *et al.* (2014), and identified by the strings *TURTLE*, *RDF/XML*, *NTRIPLES*, *TRIG*, *NQUADS* and *JSON-LD*. If a node does not specify a format, *RDF/*

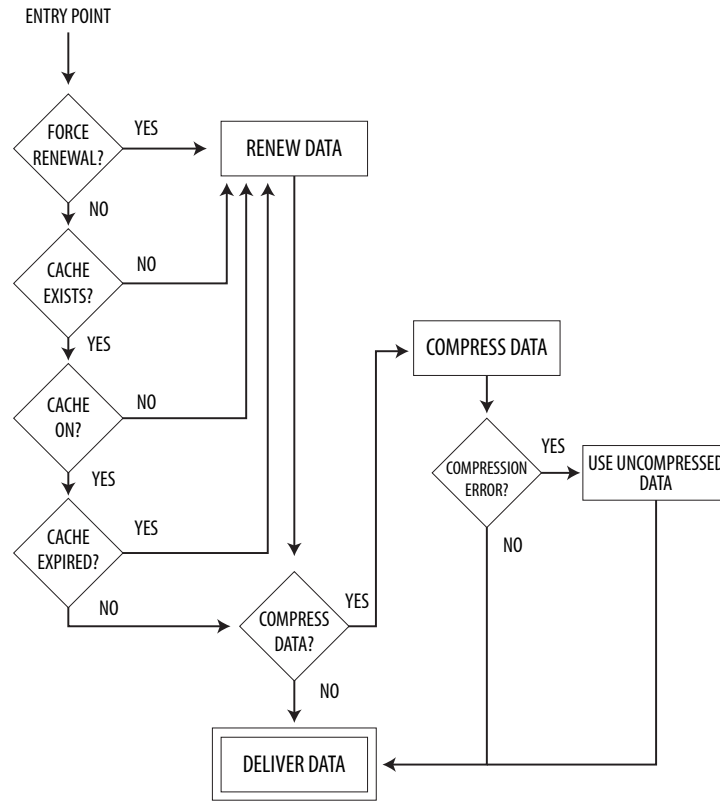


Figure 6.5. Flow diagram of the exporter from SERIES to Celestina Data

XML is assumed by default.

Celestina Data nodes can provide additional data services, such as an access point to submit a query and receive a set of results back. These services are specified by means of the properties *additionalServiceIRI* and *additionalServiceFormat*. They can be used with the support of blank nodes as defined by Cyganiak *et al.* (2014). An instance of *CelestinaDataNode* can be related to an organisation (defined by the class *Organisation* in Section 5.4.8) by means of the property *CelestinaNodeBelongsToOrganisation*.

An informative Website has been developed in PHP/XHTML/CSS (part of the home page is shown in Figure 6.7), containing information about the three current sub-systems of Celestina (Data, Tools and Simulations). The Website contains links to the resources ontology, describes the Celestina Data data model and explains the process to join in Celestina Data. The Website satisfies the requirement RQC4, of promoting Celestina as a future standard and proving guidance to institutions to join the platform.

The resources ontology and the Celestina Data ontology are simply served as Web content by a Web server. The address of the resources ontology is the only address that an application needs to know to access Celestina Data, because the Celestina Data ontology is defined inside the resources ontology as an instance of a *CelestinaDataNode* with name (*rdfs:label*) *CELESTINA_DATA_TBOX*.

Addition of new ontologies (*e.g.* data models for other hazard mitigation disciplines) can be easily done by making them Web-accessible and linking them from the resources ontology (as required by requirement RQC3 described in Section 6.1).

6.4 Validation

Before Celestina Data was completely designed, a preliminary proof of concept was conducted to integrate project-level data from the universities of Oxford and Purdue, representing SERIES and NEES data, by means of Semantic Web technologies. Using project-level data has two advantages: the project level is simple enough to allow a fairly easy integration test in a reasonable time, and at the same time it is appropriate to prove Semantic Web feasibility, define the integration environment and develop a work methodology that can be easily extended for the rest of the hierarchical levels. This proof of concept was designed in October 2012 and finally conducted in March 2013, and results encouraged a validation that not only carried out an integration of data in the background, but that had some practical interest and could demonstrate the potential of a world-wide data integration to the rest of the community.

The actual validation of Celestina Data has been conducted by creating a Celestina Data application and testing it in a real environment between the University of Oxford, Purdue University (in Indiana, United States, where the current NEES headquarters are located) and EUCENTRE (In Italy). The application consists of the presentation of data from Oxford and EUCENTRE in the NEES Web interface for the Project Warehouse, also called the *Project Display*. This application

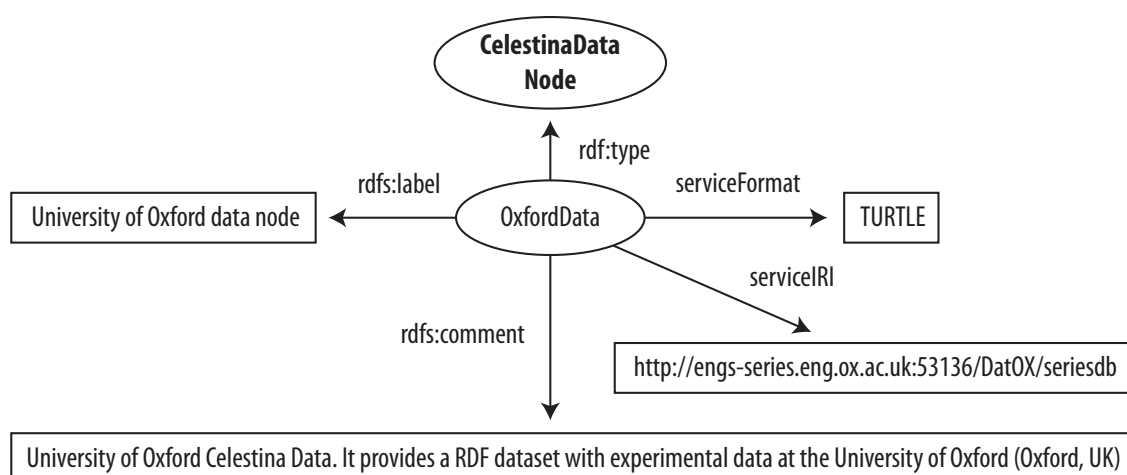


Figure 6.6. An example of the ontology resource for the Oxford Celestina Data node

provides NEES users with more data, with the advantage that users can interact with the new data by using the interface they are more accustomed to. The validation has been conducted during July and August 2014.

A simplified schema of the environment for this application can be found in Figure 6.8. The figure depicts four Celestina nodes: The Celestina central site, in a neutral location, and the respective Celestina Data nodes at NEES (Purdue University), Oxford and EUCENTRE. In this application, the nodes at Oxford and EUCENTRE provide an external access for their Celestina Data services, which are accessed by the NEES node. The latter does not need to provide any Celestina Data services for the validation so, for simplification purposes, no external access at NEES is shown in the figure. Nodes at Oxford and EUCENTRE use identical configurations with a component *C-Data Exporter*, which behaves as explained in Section 6.2.3 and is implemented as described in Section 6.3.1. The node at NEES uses a component *C-Data Accessor*, which is in charge of accessing the Celestina Central Site as explained in Section 6.2.4 and whose implementation will be explained in this section. The Celestina central site provides access to the Celestina ontologies (the resources ontology discussed in 6.3.3, and the Celestina Data ontology discussed in Chapter 5). For this validation, the Celestina central site has been located at the University of Oxford, but this has no implications in the validation results.

As depicted in Figure 6.8, the application works in this way: (i) In predefined periods of time, the C-Data Accessor contacts the Celestina central site and obtains the resources ontology. (ii) It extracts Celestina Data nodes' IRIs from the ontology and accesses each of them, including the Celestina central site to obtain the Celestina Data ontology (exceptions can be added about the nodes that are contacted, as discussed below). (iii) With all data collected, the application stores them in a triplestore that acts as a cache. This is done to minimise request time when accessing Celestina data from the Project Display, which is especially important when querying multiple geographically dispersed nodes on the network. (iv) When a NEES user accesses the public experimental data from the Project Display, the NEES Project Warehouse database (described in Section 5.2) is accessed as usual, together with the triplestore containing Celestina data, which is queried by using *SPARQL*, a query language for the Semantic Web discussed below. (v) Data from both data sources are presented to NEES users in a unified manner, via the Project Display.



Figure 6.7. Celestina Website home page

Details on this application are presented in the following sections. Section 6.4.1 describes the C-Data Accessor component, which collects data from the Celestina Data nodes and store them in a triplestore. Section 6.4.2 briefly presents the query language SPARQL, which is the language and protocol used to query the triplestore. Finally, Section 6.4.3 explains how the Project Display queries the Celestina data stored in the triplestore.

6.4.1 The Celestina Data Accessor

The Celestina Data Accessor (CDA, represented by *C-Data Accessor* in Figure 6.8) is a component developed in Java, that collects data from each Celestina Data node and store them in a local triplestore. It uses the Java library *Jena*⁵ and its operation can be divided into three phases: (i) access to the Celestina central site, (ii) access to Celestina Data nodes, and (iii) storage of data into the triplestore.

The CDA accepts the following input parameters: (i) IRI of the resources ontology at the Celestina central site, (ii) maximum time to wait to retrieve the resources ontology (central site

⁵ Jena is an open source Java framework for building Semantic Web and Linked Data applications. <http://jena.apache.org/>

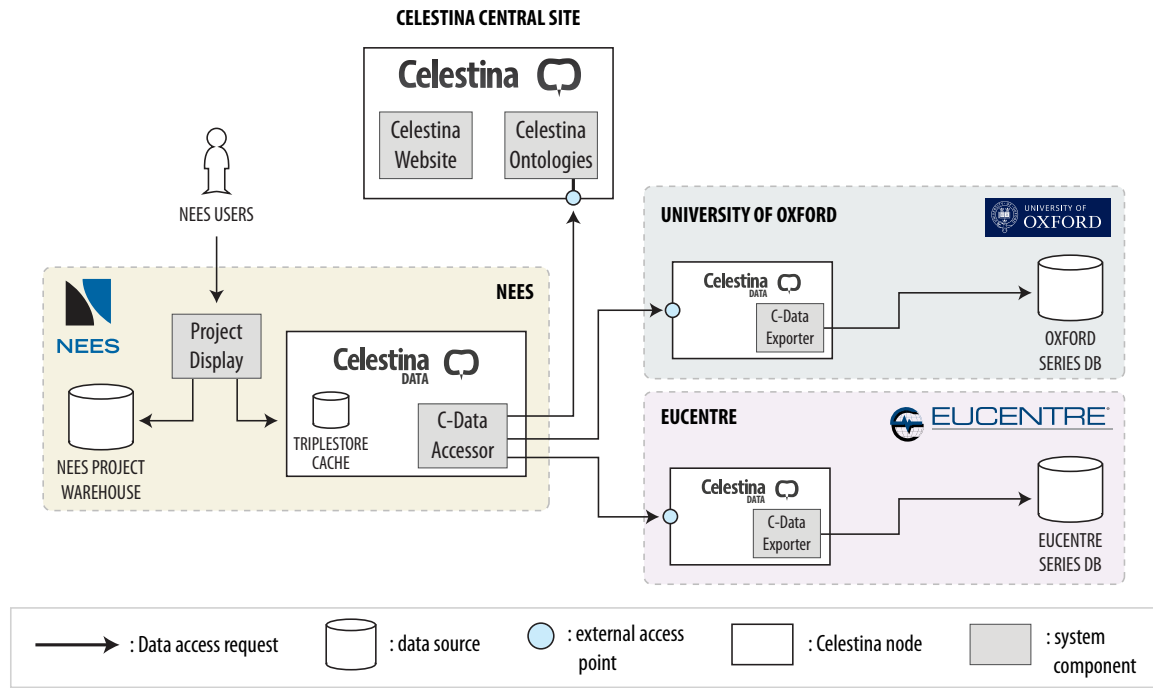


Figure 6.8. Celestina Data validation environment between the universities of Oxford and Purdue and EUCENTRE

time-out), (iii) maximum time to wait for data from each of the Celestina Data nodes (node time-out), (iv) list of Celestina Data node names that will not be accessed, and (v) configuration of the triplestore, discussed below.

The IRI of the resources ontology is firstly used by the CDA to download it, which stores a copy of the downloaded ontology in the local filesystem. If the access is unsuccessful or if the ontology is not received before the central site-time out is over, the CDA attempts to use a local copy of the resources ontology that was previously stored when the ontology could be last accessed.

Once the resource ontology is obtained, the CDA parses the Celestina Data nodes from the ontology and creates a list of nodes that have to be accessed. This list includes the Celestina Data ontology (described in Chapter 5) stored at the central site. The CDA can receive as input the nodes that have to be removed from this list. This has two main purposes: (i) some applications do not want to collect information from its own node. For example, in the application described in this section, the data coming from the NEES Celestina Data node should not be collected. (ii) Some nodes might be known to give some problems. For example, in the temporal event that data from one node makes the whole dataset to be *inconsistent* (more on this in Section 6.7).

With the list of nodes, the CDA accesses them one by one. When a SERIES node is accessed, the actions described in Figure 6.5 happen. If a node cannot be reached or the node time-out is over, the data from the node is not collected, but the CDA attempts to collect data from the rest. The existence of a node time-out is important, because errors in a node might lead to extremely

long request waiting times. The CDA parses all received RDF data, merges them and stores a copy in the local filesystem, which can help to trace potential errors.

Finally, the CDA stores the merged RDF data into a triplestore, according to a store configuration. For flexibility, this configuration uses a Java interface⁶ so different types of data stores can be used (including simple files). A store configuration is defined by: (i) the store name (which identifies the Java class used to access the triplestore), (ii) if the server has to be started by the CDA or will be running, (iii) if the triplestore database should be created in disk (persistent) or only on memory, (iv) if the input data for the database should be loaded from disk or from memory (defined in a *Jena* model), (v) the address, port, connection string, username and password to access the store, and (vi) a database name.

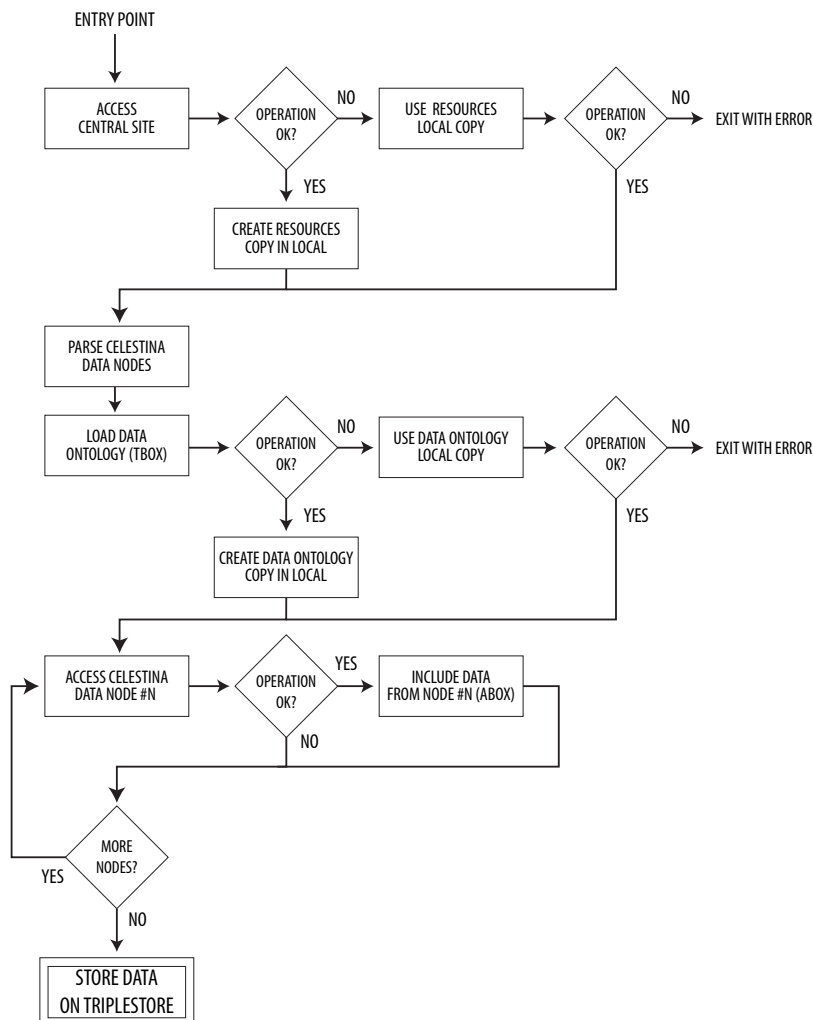


Figure 6.9. Flow diagram of the accessor of Celestina Data nodes

⁶ A Java interface is an abstract definition of methods that should be coded in all classes implementing the interface. This way, it is guaranteed that any class implementing the interface will provide methods to deal with specific functionality.

The CDA operation to access the nodes is described in the flow diagram of Figure 6.9. Because of its flexibility and capacity of configuration, the CDA is a component that can be reused by any application that needs to collect data from Celestina Data nodes. The following section explains the Semantic Web technology used to query the triplestore by the CDA.

6.4.2 SPARQL, the Semantic Web query language

SPARQL (*SPARQL Protocol and RDF Query Language*; SPARQL, 2013) is a language to query and manipulate RDF datasets as well as a protocol that defines the request and result communication with a SPARQL service over HTTP.

An example of a SPARQL query can be found below:

```
PREFIX celestina: <http://www.celestina-integrations.com/data#>

SELECT DISTINCT ?projectTitle ?projectDescription ?website
WHERE {
    ?projectId celestina:name ?projectTitle .
    ?projectId celestina:description ?projectDescription .
    OPTIONAL { ?projectId celestina:website ?website . }
}
```

The first sentence specifies the abbreviation of the URI *http://www.celestina-integrations.com/data#* as *celestina:* whenever it occurs in the query. The *SELECT* command specifies the data that will be returned by the query. The word *DISTINCT* prevents exact repetitions in the result set. The clause *WHERE* constrains the pattern, in the shape of triples, that has to be matched in the dataset. Variables are defined with an interrogation mark (?) as their first character. In the example, the line `?projectId celestina:name ?projectTitle .` will look for a triple that links an instance with a value by means of the property *celestina:name*. Whenever the triple is found, it will assign (*bind*) the subject of the triple to the variable *?projectId* and the object of the triple to the variable *?projectTitle*, and this assignment will remain for the rest of constraints in the query. The query looks for an instance that is linked to two objects by means of the properties *name* and *description*. Note that these properties could be either object properties or data properties,

so the variable at the end of the triple could be matched to an instance (such as *John*) or a value (such as the string “*John*”).

If there is an instance that is linked by means of only one of the properties (either *name* or *description*), it will not match the pattern - both relations *name* and *description* must be found in the dataset. Also, note that the fact that the variables are called *projectID* or *projectTitle*, does not imply that the pattern will match a “project”. In fact, anything that is related to two objects by means of *name* and *description* will match the pattern, regardless of its type.

Finally, the clause *OPTIONAL* specifies another triple pattern to match, but in this case, results will be returned even if this pattern is not found. As explained, this is not the case with the other two “non-optional” triples constraining of the pattern.

Results of a SPARQL query can be returned in a variety of formats, including *XML*, *JSON* and *CSV*. Also, some results use an RDF syntax (such as those for the *CONSTRUCT* command, which returns an RDF graph). A possible result returning three matches for the previous query, and coded in CSV (Comma-Separated Values; Seaborne, 2013) is the following:

```
1: SERIES Project , A consortium of 22 leading... , http://www.series.upatras.
   gr/
2: University of Oxford , The oldest University in the English-speaking
   world ,
3: A name , A description ,
```

The query returns the values for the properties *celestina:name* , *celestina:description* and *celestina:website*, as specified in the *SELECT* command, for the instances that match the query pattern. Lines [2] and [3] of the example show a result with only two elements, because the *OPTIONAL* constrain for the property *celestina:website* was not matched for those particular instances.

Graphs of triples can be queried with SPARQL even when they present unknown structures. For example, the query `SELECT DISTINCT ?p ?o WHERE { John ?p ?o . }` returns all properties and their values for the individual *John*, even if the properties related to *John* (the structure of the data) is unknown.

In SPARQL, patterns specified in the query are matched against explicitly asserted triples in a dataset by the SPARQL engine. This behaviour is called *simple entailment*. SPARQL also

defines other *entailment regimes*, which allow SPARQL to match query patterns over additional triples obtained by logic entailments of RDF Schema and OWL amongst others (Glimm *et al.*, 2010; Kollia *et al.*, 2011). Herein, triples inferred by reasoning are used when obtaining results for a SPARQL query.

Originally, only query features were provided by SPARQL. *SPARQL Update* (Gearon *et al.*, 2013) is an extension created to fill this gap, by defining language mechanisms to manipulate datasets, such as the operations *INSERT* and *DELETE* to add and remove triples, and operations such as *CREATE*, *DROP* and *LOAD* to manage graphs. Currently, many tools provide support for SPARQL Update (such as Stardog⁷, Jena⁸ and Virtuoso⁹). SPARQL Update is an interesting feature for Celestina Data nodes using a triplestore, as depicted in Figure 6.3c.

Some of the additional data services mentioned in Section 6.2.4 can be implemented by means of *SPARQL endpoints*. A SPARQL endpoint is a Web-accessible service that receives SPARQL queries over HTTP and returns the results of their execution. SPARQL endpoints can execute queries over a fixed, predefined RDF dataset or can receive generic, arbitrary datasets over which execute the query (as long as they can be accessed via Web by the SPARQL endpoint). SPARQL also enables distributed queries by querying multiple SPARQL endpoints at once with what is called a *federated query*. This enables execution of complex queries against distributed data sources with a single query. An example of an endpoint is Fuseki¹⁰ for Jena, but many triplestores, such as Stardog and Virtuoso, provide embedded SPARQL endpoint features.

SPARQL has been used to query the triplestore discussed in the following section.

6.4.3 Querying the triplestore

During the validation, the triplestore *Stardog* has been used. This has been motivated by¹¹: (i) the existence of a free version of Stardog, (ii) its reasoning capabilities, which include the lightweight

⁷ <http://docs.stardog.com/using/>

⁸ <https://jena.apache.org/documentation/query/update.html>

⁹ <http://virtuoso.openlinksw.com/>

¹⁰ http://jena.apache.org/documentation/serving_data/

¹¹ The information can be found in Stardog documentation (<http://docs.stardog.com>), specifically at <http://docs.stardog.com/owl2/>, <http://docs.stardog.com/java/> and <http://docs.stardog.com/http/>.

profiles and OWL DL, (iii) its complete documentation and responsive support, (iv) existing libraries to work with Java and Jena, and (v) an existing HTTP-based API to query Stardog databases.

The CDA discussed in the previous section, creates a Stardog database containing the Celestina Data ontology and the data from the Celestina Data nodes at Oxford and EUCENTRE. By using the HTTP access capabilities of Stardog, HTTP requests with SPARQL queries are submitted by the Project Display, which formats the results in the Project Display Web pages. Developments on the Project Display were conducted by Purdue University personnel.

The application contains a known point of failure: race conditions on the access from the Project Display to Stardog. If the Stardog database is updated at the same time the Project Display queries it, the received data could be incomplete or not accessible. This can be solved by locking the database while it is being updated, so requests are forced to wait until the process is completed. However, this solution was not implemented in this prototype .

A total of 33 SPARQL queries were created to query Celestina Data stored on Stardog. Some of them required the use of entailment regimes for RDFS - to extract information of the media and data module described in Section 5.4.9, and OWL RL - to extract information of the experimental activity module defined in Section 5.4.5. Although some data in the ontology require OWL DL reasoning (see Section 5.5), this was not necessary because the SERIES database does not explicitly use the concept of *hybrid simulation*, which is the responsible for the need of OWL DL reasoning, but the concept of *pseudo-dynamic test* (see Section 5.4.5 for a discussion on this issue). At the end of the validation, the application showed SERIES data from Oxford and EUCENTRE in the NEES Project Display, as depicted in Figure 6.10. This suggests both the suitability of the Celestina Data ontology as a means to store, integrate and exchange earthquake engineering experimental data (the remaining validation discussed in Section 5.5) and the correct operation of the infrastructure for the prototype of Celestina Data. Some results from this validation are presented in the following section.

6.5 Results

The application discussed in Section 6.4 has demonstrated the potential of a data integration between earthquake engineering institutions. The NEES Project Display has been automatically

fed with more data from distributed sources, which benefits both sides: NEES obtains more data for their users without having to conduct the experiments whereas Oxford and EUCENTRE expand their research findings to other parts of the community without the need to develop Web interfaces or attract users. Moreover, once the infrastructure has been created, the effort that NEES has to do to access data from additional Celestina Data nodes is null, which satisfies the requirement RQC3. This means that if the components used at Oxford and EUCENTRE are installed in other SERIES institutions, NEES will automatically obtain the new data without any development or reconfiguration. This means that the public subset of the 1,980 experimental activities from SERIES could join the 1,475 curated and public experimental activities of NEES¹².

Similarly, most parts of the described application can be reused at other sites. For example, if the SERIES central site wishes to show NEES data, the main development it has to tackle is querying a triplestore as explained in Section 6.4.3, and formatting the results into the SERIES DAP. The rest of components, such as the Celestina Data Accessor, can be reused. As verified in other distributed environments, errors were sometimes very complex and tedious to trace.



Figure 6.10. NEES Project Display showing a project from the University of Oxford, obtained via Celestina Data

¹² Number of experimental activities as of July 2014.

Some of the lessons learnt are presented in the following sections. Section 6.5.1 discusses the complexity of the node infrastructure based on the experience of its implementation. Section 6.5.2 discusses some considerations about the performance of the infrastructure. Section 6.5.3 describes findings about the queries executed on the triplestore. Finally, Section 6.5.4 presents some of the benefits that the Semantic Web data model has provided over the relational model used for the SVDB.

6.5.1 Complexity of node implementation

The implementation of a Celestina Data node from a SERIES node, explained in Section 6.3.1, implied two main tasks: (i) export of existing data into RDF according to the Celestina Data ontology and (ii) making the resulting RDF publicly available. Neither of these tasks resulted completely trivial.

Main problems found in the export are that some parts of the SERIES data model were not directly translatable into the Celestina Data model represented by an ontology. The ontology is indeed a more complete data model, and some of the information did not exist in SERIES data. This is not really significant since incomplete information is expected in RDF/OWL. An example of how this affects an application is that every SERIES experimental activity presented in the NEES Project Display only shows a repetition (because the SERIES data model does not store information based on a repetition level, as the NEES model does). On the other hand, the description of SERIES specimens is more structured than in NEES, and this was also visible in the Project Display.

Creating a bespoke exporter helped to refine the data presented in RDF. For example, SERIES types of experiment were translated into *celestina:keyword* properties. Some types of experiments are defined according to substructuring: “centrifuge with substructuring”, “centrifuge without substructuring”, “shaking table with substructuring”, “shaking table without substructuring”, *etc.* The bespoke exporter allowed to polish the export and divide the experimental type into two keywords, such as “centrifuge” and “with substructuring”. In this way, searches for experiments with substructuring will identify them efficiently, without having to look for all keywords containing the string “substructuring”.

Correct interpretation of the Celestina Data ontology must be done when an institution exports their existing data. Support for this comes from the central site Website and from the ontology annotations that explain the meaning of all ontology classes and their properties.

It was easy to fall into race conditions when making the resulting RDF publicly available, as presented in Section 6.3.1. This is a common and well-known problem of distributed systems, and nodes using similar approaches to serve RDF data must take care in the design of the solution.

Although the presented solution has some advantages and is perfectly valid for the implementation of a Celestina Data prototype, it is suggested that other implementations using R2RML to export data (see Section 6.2.3) and a SPARQL endpoint to serve RDF (see Section 6.4.2) are considered in future.

Nodes that do not have existing data systems do not need to consider a data exportation, which can simplify the implementation of the Celestina Data node if a triplestore is used directly (as depicted in Figure 6.3c). A triplestore containing data in the Celestina Data ontology does not require any data translation, and serving RDF can be done by means of a SPARQL endpoint, which is sometimes offered by the triplestore itself. Since Celestina Data only requires an IRI to serve Celestina data, this IRI could encode the SPARQL query `SELECT DISTINCT ?s ?p ?o WHERE { ?s ?p ?o }` to be executed by the SPARQL endpoint, which will provide all triples as a result.

6.5.2 Performance

Times have been measured at the Celestina Data Accessor located at NEES accessing both nodes at Oxford and EUCENTRE. The measurement has been done at different steps, from the start of the execution until the data is fully stored on the Stardog database, which is the process shown in Figure 6.9 (note that the Celestina central site has been simulated so the times to obtain the resources ontology and the Celestina Data ontology are unrealistic).

The following scenarios have been considered: [1] access to Oxford and EUCENTRE with the cache deactivated, [2] access to Oxford and EUCENTRE with a working cache at both nodes, [3] access to Oxford and EUCENTRE with cache activated and creating the Stardog database in memory instead of making it persistent on disk, [4] access to Oxford and EUCENTRE with cache activated, creating the Stardog database in memory but conducting the initial load of data

on Stardog from memory instead of from disk. The results are shown in Figure 6.11. The node at Oxford served a total of 4,645 axioms in one project containing 21 experimental activities, whereas the node at EUCENTRE served a total of 5,900 axioms in three projects containing 9, 6 and 5 experimental activities.

The time that it took to obtain Celestina Data nodes, access them one by one and store the data in Stardog was less than one minute in all cases, which can be considered acceptable for the system requirements. Note that this time is acceptable for the loading of data into the cache triplestore, but not as a waiting time for a user. This just corroborates that using a triplestore as a cache is necessary.

EUCENTRE response time was significantly greater than Oxford time in any scenario, but especially when the cache was deactivated - which is probably bound to the fact that data at EUCENTRE were more complex to export, judging from the comparison of response times with and without cache. Cache at EUCENTRE entailed a great difference in the time response (5 s. average with cache vs. 27 s. without cache). It must be also considered that, as data size increases in a node, the response time when caching is deactivated will increase too, whereas the response time is affected less dramatically when caching is activated.

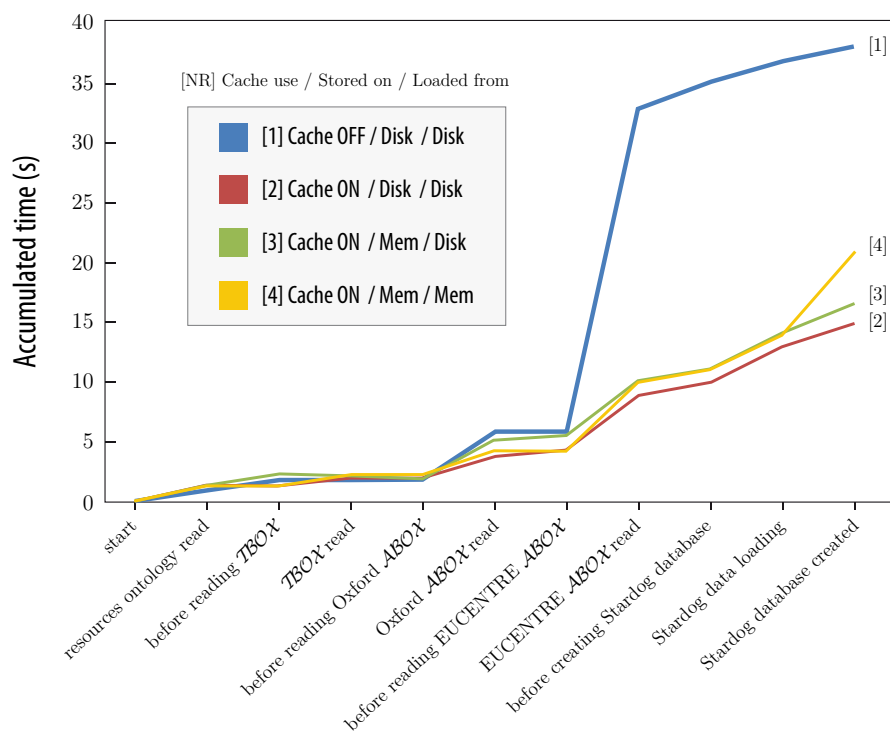


Figure 6.11. Celestina Data Accessor times for different scenarios

Stardog performed well. Contrarily to what was expected, it did not show significant difference between storing the database in memory or in disk (probably because the size of the data was not large enough to make a difference). However, loading data from memory was substantially less efficient than loading data from disk. This is an expected result according to Stardog developers.

Although the response time was considered fast, it must be considered that only a total of four projects with 41 experimental activities were sent. If Celestina Data is expanded to other nodes around the world, the total response time can grow substantially. As an example of how these data can grow, project-level data from NEES has been exported to RDF. The time required to collect only project-level data of NEES public projects, consisting of 5,728 axioms, was 19 s.

As a conclusion, it is clear that extensive use of caching techniques (on both client and server sides) is suggested to be crucial to meet acceptable response times in Celestina Data, and this need will become more patent as the number of Celestina Data nodes increases. A performance improvement that can be done in the CDA is to run several threads to access the different nodes in parallel. Although this will not benefit the network time to transfer the RDF data, it will take advantage of parallel processing at each node.

6.5.3 Query results

As mentioned before, 33 SPARQL queries were created to be executed on Stardog. Semantic Web technologies proved great flexibility, allowing the same query to be reused when data from different classes were requested. As a simple example, the following query could be used to obtain keywords of an individual, regardless of being keywords of a project or keywords of an experimental activity:

```
SELECT DISTINCT ?keyword
WHERE {
  <ID> celestina:keyword ?keyword .
}
```

In the query, the `<ID>` element can be substituted for the identifier of any individual.

Some information could be retrieved in a variety of ways. For example, documents could be obtained by asking for objects of the property *hasDocument*, which is a sub-property of *hasDataFile*. Similarly, they could be obtained by asking for objects of the property *hasDataFile*

that are instances of the class *Document*.

Despite the standardisation of SPARQL, some query results seemed to be open to the interpretation of the specific triplestore used, which produced unexpected results. For example, let us consider the following query *Q1* using RDFS entailment:

```
SELECT DISTINCT ?datafileid ?name ?type (SAMPLE(?downloadURI) AS ?download)
WHERE {
    <ID> celestina:hasDataFile ?datafileid .
    <ID> ?role ?datafileid .
    ?role rdfs:label ?type .
    ?datafileid celestina:downloadIRI ?downloadURI .
    ?datafileid celestina:name ?name .
} GROUP BY ?datafileid ?name ?type
```

The relevant part is the *WHERE* clause, that tries to match the pattern of triples, including a specific instance identifier represented by *<ID>*. Despite existing suitable data for a response, this query produced no results. However, a small modification in the following query *Q2*, returned the expected results:

```
SELECT DISTINCT ?datafileid ?name ?type (SAMPLE(?downloadURI) AS ?download)
WHERE {
    ?id celestina:hasDataFile ?datafileid .
    ?id ?role ?datafileid .
    ?role rdfs:label ?type .
    ?datafileid celestina:downloadIRI ?downloadURI .
    ?datafileid celestina:name ?name .
    FILTER (?id = <ID>)
} GROUP BY ?datafileid ?name ?type
```

Both queries, *Q1* and *Q2* are operationally identical. The second query uses the constraint *FILTER*, which, after obtaining the matching triples of the pattern, only returns those results where the *?id* variable has been bound to the value *<ID>*. In essence, *Q2* is less “correct” or a less “optimised” version of *Q1*, but both queries should produce the same results. Surprisingly, that was not the

case. This was probably bound to the fact that the triplestore interpreted ambiguously some of the first two pattern triples in *Q1* and, consequently, used a simple entailment producing no results.

The execution of some other queries failed to terminate without apparent reason. This was the case of a large query to extract calibration information of a device. When the following pattern triples were added to the WHERE clause, the query never stopped execution:

```
OPTIONAL{    ?locid celestina:coordX ?x .  
            ?locid celestina:coordY ?y .  
            ?locid celestina:coordZ ?z .  
            ?locid celestina:coordXYZunit ?xyzunit .}
```

The query was split into two different queries to solve the problem and extract calibration information. In general, SPARQL queries over Stardog executed as expected, but some special cases were found where the execution of the query produced unexpected results, and it was difficult to determine the causes of such behaviour.

6.5.4 Benefits of a Semantic Web data model

The relational model has performed very well, which questions the reason and benefit of using Semantic Web technologies instead. In fact, it has been demonstrated during the work of this thesis that, generally speaking, there is little that can be achieved using only Semantic Web technologies and cannot be done with other technologies (in a similar way to what can be done in Java that cannot be done in assembly language). However, relational databases have been widely exploited, which has shown some of their weaknesses, where a number of Semantic Web features have proved to be especially useful. Some of the weaknesses typically noted in relational database management systems are:

- Database schemas are quite rigid, and changes on a schema can have a considerable impact. Scalability and expansion of a database can lead to complex processes and maintenance problems.
- Tables with many optional attributes require an additional table to store them, which implies extra reading operations and inefficiency. The same situation also occurs when the attributes of a table are unknown beforehand or can vary over time (an example in the

SERIES data model previously defined is the *properties* of a *material*, which are difficult to define accurately in advance).

- Attributes with multiple values can also be complex to design or can imply the creation of additional structures. It is also difficult to share new attributes between different databases. For example, if a SERIES institution creates a new *nominal property*, ideally it should be shared among the rest of SERIES nodes, which might also be interested in using it. Nominal property identifiers have a local scope and the distribution of new nominal properties between all institutions can entail complex data sharing processes.
- Database replication and synchronization are also typical problems of relational databases.

Some of these issues are better addressed with Semantic Web technologies as will be discussed below. However, in the modelling aspect, working with RDF/OWL showed no much difference with respect to the creation of a relational data model. In both cases, data requirements were needed from experts of the domain, and several iterations had to be accomplished until the data model was considered accurate and stable.

Differences from the relational model, that can be potential benefits of the Semantic Web technologies, are listed below:

- Semantic Web technologies provide a common programming interface to retrieve knowledge from world-wide earthquake engineering laboratories. The information is published in a standardised way, supported by well known technologies, which makes data collaboration easier.
- Data integration is made within the data model; no external programs are required. The technology is designed to facilitate data integration from different sources, which has simplified the infrastructure of Celestina Data and supported data interoperability that could not have occurred otherwise (for example, in the case that every other Celestina Data node had to implement the SVDB Web Services).
- They enable interoperability with a neutral, standard, technology-independent framework to define data models. It resulted in being easier to import and export RDF data between different triplestores than to perform the according operations with relational data between

different relational database management systems. Despite the standardisation of the relational model, in practice, many relational database management systems still have many differences.

- Because of the OWA feature explained in Section 5.3.3, Semantic Web technologies are prepared to deal with incomplete data, so an institution could generate information which can be completed by another institution. On the other hand, relational databases assume that data is complete (what is not in the database is false).
- Data identifiers are universal in the Semantic Web, which is useful for example on data from distributed simulations. Part of the data can be stored in a node and be referenced from another node. This is difficult to accomplish in relational databases, because the identifiers have only a local meaning, and computers do not *identify* a similarity between the *Experiment* tables of two different databases.
- Semantic Web data do not have a rigid fixed structure. The information structure is very flexible, permitting changes and expansion. There exist no fixed attributes as in a relational table, which benefits information that is difficult to store in fixed-length database rows. This also allows experts of a domain to create data structures without the need of a technology expert to modify the data schema.
- Semantic Web technologies use reasoning to create new data from existing data. Also, data could be transmitted more compactly over the network and additional data can be inferred at the destination.
- The Semantic Web is Web based, and is very well integrated with the rest of the Web technologies. The Web has been demonstrated to be a robust platform from client to server sides.
- Semantic Web concepts can have equivalent names and help with terminology agreements. Where a relational table will be called *Computation* in the data model, a Semantic Web concept could use equivalent classes for *Computation* and *Simulation*, and users can utilise the term they are more accustomed with.

- One difference with the relational model is that constraint verifications over the data stored are applied at the time the data is introduced in the relational model, and the schema is not consulted at query time. On the other hand, the Semantic Web does not apply any constraint verification when the data is stored and the “schema” (TBox) is extensively used at query time. This difference could be an advantage of relational databases, whose queries might be answered more easily.

6.6 Other applications (Celestina Tools)

Celestina Data has partially shown its benefits with the application discussed in Section 6.4, which shows integrated data from NEES and SERIES sources. However, this is just *one* possible application, and the potential of future applications based on Celestina Data is very large. Applications have been organised in a different Celestina sub-system called **Celestina Tools**, which presents practical benefits of the Celestina Data infrastructure.

In this section, three benefits of the use of Celestina Data applications are envisioned, as a small fraction of the potential that Celestina Data can reveal in future:

- **Data publishing standard.** The scientific publishing industry is increasingly focusing on improving mechanisms that bring better experiences for its readers. An example of this is the Elsevier’s *Article of the Future* project¹³, which aims to enrich content, context and presentation of journal articles, as well as facilitating their understanding and validation. Celestina Data can support the exposition and evaluation of data presented in scientific articles, which will benefit both publishers and researchers. Providing Celestina Data becomes a standard in the earthquake engineering community, publishers can access earthquake engineering data from their articles in a unified manner, and reuse visualisation tools already developed for Celestina Data. Moreover, articles can be evaluated easier: researchers can get access to article data in a standardised format, and use over them the same tools they are more comfortable with.

¹³ <http://www.articleofthefuture.com/>

- **Sharing software systems across research activities.** Finding funding is one of the continuous tasks in the academic environment, and lack of funding can cause undesired interruptions in research projects. Sometimes, the lack of funding might prevent institutions from pursuing their research interests entirely. Celestina Data enables a more efficient management of global funding. For example, a Celestina Data node can use its expertise, time and money to create some Celestina Tools (*e.g.* a Web interface to interact with Celestina data, visualisation tools, *etc.*), which can be automatically reused on other nodes. Note that these tools could be reused by nodes *over their own repositories*, without the need to share their data with a third party. This can improve the way funding is spent globally. The development of different components can be organised globally and be reused by all Celestina Data nodes, which can spend the funding more intelligently in other research tasks.
- **More intelligent decision-support systems.** Currently, the decision making support from computers in earthquake engineering is rather limited. This is due to the lack of interoperability of seismic systems. Celestina in general, and Celestina Data in particular, aims to fill this gap. Let us imagine the situation of a building collapse after an earthquake. The earthquake accelerogram is recorded and stored in a database. The field data from the collapsed building might be recorded and stored in a different database. And research experimental data from experiments conducted in a laboratory are stored in a different database. Celestina Data can support the integration of these dispersed data sources, which can enable a decision support system implemented in a Celestina tool, to act over the whole set of data. By looking at “hidden” data patterns in large datasets, the computing tool could attempt to explain for causes of the event to prevent this behaviour in future. As a naive fictitious example, the tool could conclude from experimental activities testing column specimens of concrete, as found in the collapsed building, that they will break under seismic activities of certain magnitude, as the one recorded during the earthquake, and it could provide recommendations of what type of concrete should have been used to avoid the collapse.

Celestina Data is just the basic foundation to build the future of earthquake engineering tools, based on extensive world-wide collaborations and more supportive systems to address complex problems.

6.7 Maintenance considerations

Celestina Data addresses data integration of earthquake engineering in a world-wide scale. This presents a very different situation from the SERIES Virtual Database (SVDB), since there is a lesser level of control in a globally distributed environment. As in the SVDB, different data sources are autonomous and responsible for their respective data, however there is no real contractual obligation nor common funding body to enforce a certain level of availability or common operational policies. Thus, the occurrence of errors must be assumed as part of the distributed nature of the system, where participants of Celestina Data have absolutely no control over the rest of the participants. Every Celestina Data participant offers their public data in an “altruistic” manner (even though it might be beneficial for the institution itself, since its research is published in a world-wide scaled system), and it is also not required that an institution shares their own data in exchange of accessing Celestina Data by means of an application.

Likewise, the distributed nature of the system delegates quality assurance of data to their respective owners. The process to enrich the data until they comply with certain quality standard is sometimes called *data curation*. For example, NEES data is curated according to some policies, but SERIES data has no common curation process at all.

Despite the distributed nature of the system and the autonomy of its nodes, some common management is necessary. It is suggested a centralised management body called the **Celestina Data consortium** is created, which is discussed in Section 6.7.1. Finally, the admission of new nodes in the Celestina Data network is discussed in Section 6.7.2.

6.7.1 The Celestina Data consortium

The creation of the Celestina Data ontology is not a one-off development. As any other data model, the Celestina Data ontology must evolve with time. Data requirements in the community might change, and improvements, extensions and modifications might be conducted. In order to maintain a centralised ontology, it is suggested that a consortium is created. This consortium should be constituted by a small group of both IT specialists and civil engineers from the main participant institutions of Celestina Data, with the mission of watching over general data requirements for

the earthquake engineering community, and particularly over any required modification on the Celestina Data ontology.

One important consideration is that Celestina must be a neutral, community-driven platform, and, likewise, the consortium must be a neutral management body to represent the community. Even if world-wide data integration is technically feasible, it is probably a more complex task to deal with the social challenges implied, because many disparate institutions are involved. Celestina is by no means an infrastructure that is imposed by any institution (*e.g.* the University of Oxford or NEES), but a technical infrastructure to promote collaborations and joint efforts to address complex problems of hazard mitigation disciplines.

Main communication channels for the consortium members could be a simple mailing-list, which could serve as an informal forum of discussion to capture data requirements and new ideas. It is suggested the consortium meets on regular basis and takes the opportunity of global conferences to gather in person. The consortium should also enable channels to capture data requirements from the community and to inform participants of any change in Celestina Data. The Celestina central site Website can be used for these purposes.

The consortium must gather data requirements and take action to implement them. Note that if data requirements are discussed and an agreement is not achieved, site-specific ontologies can still be used for such disparity cases - these ontologies can also be accessed by external institutions, not only locally. The development of Celestina Tools can be also managed by the consortium, to join development efforts.

It is suggested that the consortium establishes common policies in terms of: implementation of Celestina nodes, curation, data licence and security. These should be recommendations rather than contractual obligations, and the consortium should provide support and guidance to Celestina nodes in these issues. For instance, the Celestina curation team could watch over Celestina Data and provide curation advice to institutions in order to improve the quality of the overall data.

Funding mechanisms and detailed structure of the consortium is not discussed in this thesis.

6.7.2 Joining Celestina Data

As discussed previously in this chapter, the infrastructure of Celestina Data accepts new participants (or nodes) without having dramatic impact in the existing participants. Some methods to implement a new node has been exposed, such as the implementation of an OBDA by data transformation approach or the use of a triplestore.

The Celestina central site Website establishes a first contact point to guide new institutions to join Celestina. The Celestina consortium proposed in the previous section should then be the management body that officially includes new nodes in the system, by verifying that the node has correctly implemented a Celestina Data node, and adding the relevant contact information in the Celestina resources ontology.

When a new node joins the Celestina Data network, it must be considered that ontology consistency (a *logical error* in the ontology data; see Section 5.3.3) must be satisfied globally. This means that even if the ontology consistency of the Celestina Data ontology has been proved (as discussed in Section 5.5), and the consistency of every node data is proved in isolation, an inconsistency could happen once all data come together. Automatic monitoring of global data consistency should be conducted from the Celestina central site.

Also, the central site plays a key role on identifying Celestina Data nodes that are not available. As a lesson learnt from the SVDB, parts of a distributed system will eventually fail (see Section 4.5.2), and Celestina Data should be prepared to monitor the alive nodes and automatically inform the node manager if the node becomes unavailable, its data are inconsistent in isolation or if its data make the whole Celestina dataset inconsistent.

6.8 Conclusions

This chapter has presented Celestina Data, an infrastructure for data integration in earthquake engineering. The Celestina Data architecture has been described, implemented and validated. Semantic Web technologies have demonstrated to be a feasible technological infrastructure for a global system of distributed data sources. The validation has been conducted by means of a practical application that has integrated SERIES data from the University of Oxford and EUCENTRE

into the NEES Project Display Web interface. This is the first time that earthquake engineering experimental data have been exchanged between institutions in Europe and the United States by using a formalised technical infrastructure.

Celestina Data is constituted of distributed nodes that are autonomously managed. However, the creation of a centralised management body, called the Celestina consortium, has been proposed. This consortium should be neutral and community-driven, with the ultimate objective of watching over emerging data requirements and providing guidance and support for institutions forming Celestina Data.

Celestina Data enables earthquake engineering institutions to exchange information in a unified manner, build applications on top of this data integration and improve collaboration during joint research projects, amongst many others benefits.

Future work should concentrate on these aspects:

- Creation of the Celestina consortium, to define common management directives.
- Showing the benefits and potential of a platform for the integration of data world-wide, with the hope of persuading earthquake engineering institutions around the world to join the effort.
- Extend the effort to other hazard mitigation disciplines, such as those for wind engineering and fire engineering.
- Continue development on the Celestina Data data model and implementation of new applications based on Celestina Data.

Chapter 7

Foundations of distributed simulations

In the past few years, new experimental techniques have arisen, which take advantage of developments in computing science. One of these is distributed simulation¹, which enables the use of resources at different locations to achieve a more representative experiment. Having all the resources in a single laboratory requires a large amount of space and a big economic investment, as for example, at *e-Defense* in Japan (Ohtani *et al.*, 2003). On the other hand, sharing the facilities of several laboratories makes experiments affordable, creating the illusion of using a single laboratory when several laboratories are involved. Operating several laboratories as one, presents great computational challenges as well as more complexity of experimental planning. This behaviour will be referred to as **laboratory integration**.

The integration of several laboratories is a form of distributed system, defined as “a collection of distinct processes which are spatially separated, and which communicate with one another by exchanging messages” (Lamport, 1978). Laboratories (or, strictly speaking, their facilities) work as processes that communicate with each other in the same way that a central processing unit communicates with the memory inside a computer or different government ministries communicate to each other in a country. What makes integration of laboratories a computational challenge is the time requirements of simulations, which are difficult to satisfy considering the delays involved in the communication and the data processing. Ideally, the laboratories’ facilities would be interconnected as if the communication delays between them were either null or insignificant. The real world makes this impossible with network, operating system, device and sensor delays

¹ The term “simulation” will be used interchangeably with “experiment” to refer to a numerical or physical experiment as explained later in this chapter.

amongst others. Laboratory integration must cope with these delays to minimize the impact in the quality of the experiment.

This chapter explains the basis of distributed simulation in earthquake engineering. Section 7.1 categorizes the most common types of earthquake engineering simulations and identifies which ones are the interest of this thesis. Section 7.2 presents some of the basic requirements for distributed simulation. Section 7.3 offers a global vision of a usual simulation infrastructure whereas Section 7.4 discusses the simulation software usual activities. Section 7.5 describes a practical simulation software that has been used to conduct distributed simulations successfully, with a summary of the main lessons learnt from this in Section 7.6. The chapter concludes in Section 7.7 with two main ideas:

- Simulations requiring real-time operation must be supported by a specialised computing infrastructure.
- Simulation activities must be addressed in two different layers: low-level and high-level.

This chapter focuses on simulations from a computing perspective (simulation software to command devices and transmit data over a network) and, although they are briefly mentioned, it does not deal with simulation control algorithms, testing devices, specimens, test rigs or any other aspect related to experiments.

7.1 Types of earthquake engineering simulations

Earthquake engineering simulations can be divided into two groups: **numerical** and **physical**. Numerical simulations are computer-based simulations to re-create the behaviour of physical structures. They are good for modelling linear systems and are relatively easy and cheap to implement since they run as programs in a computer. Despite these advantages, numerical simulations entail uncertainty over non-linear and rate dependent behaviours (described below). To address this, physical simulation is required. Physical simulations use physical (sub) structures so they can model real behaviour. However, physical sub-structures are very expensive and time consuming to build. Scaled model simulations are feasible but can be hard to extrapolate to full scale models.

A rate dependency system is governed by the equation of Figure 7.1, where m , c and k are key structural properties (mass, damping and stiffness respectively). $m\ddot{x}$ and $c\dot{x}$ are rate-dependent

components, and kx might be, depending on material properties. If c or k are not constant, it is difficult to simulate response without testing at the correct rate, which compels a simulation to work in real-time basis as described in Section 8.1.

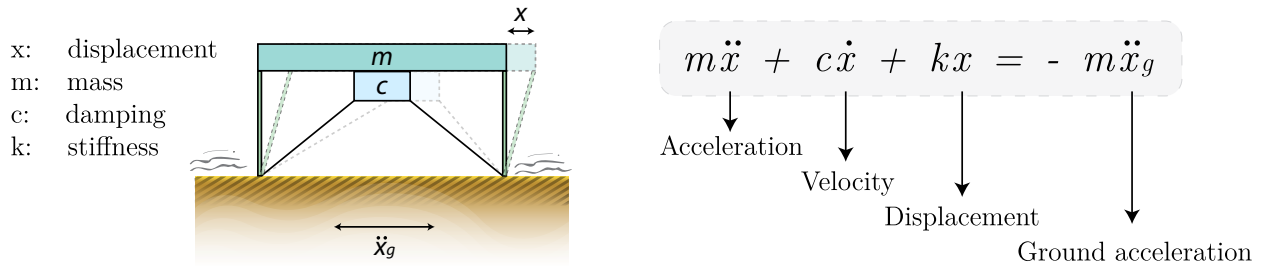


Figure 7.1. Rate dependency system

Numerical and physical simulation advantages are complementary, so **hybrid** simulations attempt to have the best of both worlds: the most important parts of a sub-structure are tested physically whereas the rest of the parts are simulated numerically. Hybrid simulations are an innovative field in earthquake engineering (Bonnet *et al.*, 2008b).

Simulations can also be categorised according to where they take place: **locally** or **distributed**. Local simulations use resources of a single laboratory whereas distributed simulations take advantage of computer networks to combine facilities of different laboratories. Joining resources of different laboratories enhances experimental simulation capabilities but brings as well the computational difficulties discussed in Chapter 8 for simulations requiring fast operation. Distributed simulations can be conducted between two sites or three or more sites (multi-site). This distinction is made because multi-site simulations involve greater complexity than two-site simulations.

This research interest is in distributed hybrid simulations: numerical and physical experiments conducted between two or more geographically located facilities. The following section discusses some of the requirements for such simulations.

7.2 Basic requirements for distributed hybrid simulations

In terms of requirements, earthquake engineering simulation is a very specialized field with a limited number of world-wide laboratories performing experiments. Therefore, the number of

simulation software users is not high. Simulations are carried out between known organizations who previously agree the terms in which the simulation will be conducted. Consequently, the experimental environment is usually well-known.

In some cases, simulations can be conducted using dedicated networks. Examples of these are the *UK-Light* dedicated link between the Universities of Oxford, Bristol and Cambridge and the *GÉANT* network² connecting many educational institutions around Europe. The reliability required for distributed simulations is widely satisfied by these dedicated networks.

One of the most challenging requirements of distributed simulations is the demand for predictable, fast simulation operation, especially when a rate dependency system is tested. Even if the networks used are considered fast for the current age, there still exist physical limitations for the speed at which data can be transmitted.

Distributed simulations in which physical devices are required (such as actuators or network devices) might impose additional speed limitations if the device cannot be commanded to achieve a desired time constraint. Hence, physical device limitations must be considered when running a simulation.

Such as in many other distributed systems, interoperation between heterogenous systems is often a requirement.

Although most of the basic simulation requirements do not imply a major problem, fast operation still remains a challenge. The following section depicts a typical distributed hybrid simulation set-up and describes the elements involved.

7.3 Simulation infrastructure overview

The core of the simulation activities consists of a data acquisition/generation, a data exchange and operations on these data. A typical simulation infrastructure is depicted in the diagram of Figure 7.2, where three main parts can be distinguished inside the local site: testing devices (which are the physical simulation components: specimens, sensors for data acquisition, devices such as actuators, controllers, *etc*), the computer (holding the simulation software: monitoring tools, programs to

² GÉANT2 network. <http://www.geant2.net>

command data to devices and network, algorithms to operate over data, *etc*) and the network (formed by the network devices to link two or more computers and the data transmission media).

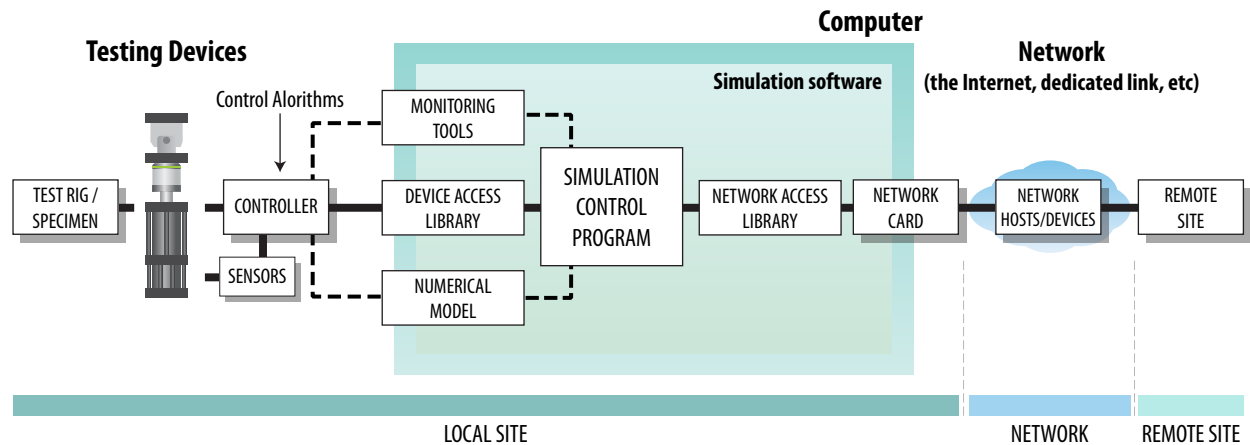


Figure 7.2. Simulation infrastructure, highlighting the simulation software

The central element in the figure is the *simulation control program*, held on a computer, which is often the main simulation software and generally consists of a program created in *C*, *Java* or any other language, that acts as a *middleware*³ connecting devices and network. In the access to physical devices, a library is normally used to connect to specific hardware (such as a board in the computer) or to a controller that ultimately commands a device. Such a device acts on the tested element which is the focus of the simulation. The library is usually a proprietary device driver that connects to very specific hardware in a low-level way, providing an abstract access layer to release the program from managing tedious hardware details.

During the process of data transmission between devices and network, monitoring tools can be used to capture all data for as-it-happens data visualization or for later analysis. To access the network, another library is used. It is quite standard to use *sockets*⁴ to manage the network communication, and normally sockets programming deals with operating system functions that call protocol drivers to finally deal with the computer's network card. The data, together with additional network payload data, are delivered as signals over a cable, radio frequency or other media, and transmitted from one network device to another until the information reaches its

³ Middleware is a software layer mediating between operating system services and applications. In this case, the middleware mediates between the computational part and the input/output operations for network and physical devices.

⁴ A socket is a transport-layer access point and associated functions for network communication on the Internet.

destination, the remote site. Similar flow of events happens on the remote site, where the received data will condition the new response: the data travel from the network interface to the physical devices, obtaining a new response to be sent to the origin. This loop is repeated for as long as the simulation runs.

The whole data exchange, as simple as it could seem moving data from one end to the other, involves many hardware devices, code developed by many different groups, protocols, technologies, specifications and algorithms. From the moment the data is acquired from a real physical structure (or simulated on a computer), it is manipulated by many computer instructions from several different libraries and transmitted or stored as electrical signals, electrons in integrated circuit capacitors of a computer's memory or as light impulses amongst others. Having a fine control over all elements involved is normally not possible, so the simulation control program has to orchestrate all the element involved and lead the data flow to meet simulation requirements.

This work only focuses on the tasks of the simulation control program, which are discussed in the following section.

7.4 Activities conducted by the simulation control program

The simulation control program is in charge of managing the communication with physical testing devices (or simulation algorithms) and the network, sometimes interpreting the data that is transmitted from one part to the other, to take specific actions. In this process, normally three simulation phases are found: initialization, data exchange and finishing. During the initialization, remote sites synchronise their laboratory activities (such as turning on the laboratory pumps or the actuators) and exchange initial data to conduct the experiment (such as a data matrix to be able to operate over the exchanged data). The tasks during this phase might not have to be completed within extreme time deadlines but they might need specific order or have some light time restrictions. On the other hand, data exchange usually needs hard time-constraints and simulation software should focus on delivering data within a time step. Generally, the simulation software will iterate continuously in a loop - similar to the one presented in Algorithm 7.1, to conduct the data exchange until the simulation is considered finished or an error arises. Reading

and writing operations order depends on the site's role: sender or receiver. On some occasions, the loop is specifically made for a client or a server (meaning that different programs are used on both sides) so there is no need to distinguish between client and server roles.

Algorithm 7.1. Typical simulation loop

```
1: DO
2:   IF (SENDER)
3:     READ n1 BYTES FROM Device
4:     WRITE n1 BYTES TO Network
5:     READ n2 BYTES FROM Network
6:     WRITE n2 BYTES TO Device
7:   ELSE
8:     READ n1 BYTES FROM Network
9:     WRITE n1 BYTES TO Device
10:    READ n2 BYTES FROM Device
11:    WRITE n2 BYTES TO Network
12: WHILE (SIMULATION IS NOT OVER)
```

The operations shown in the previous algorithm mainly deal with data, reading from and writing to experimental devices and network as a middleware. Additional control activities might be required in some simulations, so sites respond to control commands behaving differently according to the control signal received, as shown in Algorithm 7.2. In this case, the loop activities are not predictable, since they vary during the simulation in accordance to the control data. The use of control data is not recommended if extremely fast operation is required (see Chapter 8) and, whenever possible, a simulation should be defined under predictable conditions.

Simulation programs might work as merely middleware, acting as a *glue* layer between algorithms and network and hardware devices or include simulation-specific functionality such as a numerical model.

The following section provides an example of a simulation control program called DHT.

```
1: DO
2:   IF (SENDER)
3:     READ n1 BYTES FROM Device
4:     IF( CONTROL _ DATA == SITUATION _ 1)
5:       DO _ SOMETHING
6:     ELSE IF ( CONTROL _ DATA == SITUATION _ 2)
7:       DO _ SOMETHING _ ELSE
8:     ...
9:   ELSE
10:    READ n1 BYTES FROM Network
11:    IF( CONTROL _ DATA == SITUATION _ 1)
12:      DO _ SOMETHING
13:    ELSE IF ( CONTROL _ DATA == SITUATION _ 2)
14:      DO _ SOMETHING _ ELSE
15:    ...
16: WHILE (SIMULATION IS NOT OVER)
```

7.5 The DHT program

One example of a simulation control program is the DHT program (standing for *Distributed Hybrid Testing*), developed to conduct distributed hybrid simulations for the UK-NEES project between the Universities of Oxford, Bristol and Cambridge (UKNEES, 2009). The program was developed in *C* (IA32 architecture, on Windows), and it consists of two different programs: one for the client and one for the server(s), the communication being led by the client. DHT mainly works between one client and one server, but some versions were adapted for multi-site testing (Ojaghi *et al.*, 2010). The physical part of the simulation is managed by connecting to a *dSPACE PPC Controller board*⁵ which eventually commands the actuators. Hence, DHT is a middleware acting as a glue between the different hardware modules, as depicted in Figure 7.3.

⁵ More information can be found on the dSPACE Website: <http://www.dspace.com>

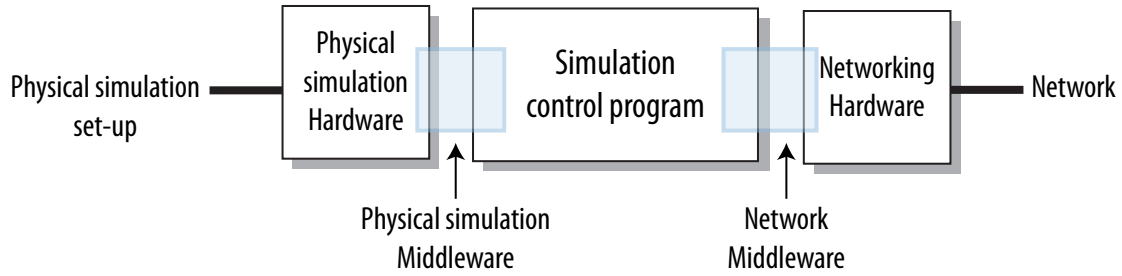


Figure 7.3. Middleware components in the UK-NEES simulations

The main program workflow is depicted in Figure 7.4a. When the program starts, a configuration file is read, which contains information of the *IP address*⁶ and *port*⁷ of the other end, the protocol to be used and specific parameters to access the dSPACE board. The server does not need any other information but the port for incoming connections and the dSPACE access parameters. Afterwards, the dSPACE board is initialised and memory is allocated to store data to run the simulation.

The protocol configuration is always done by using TCP sockets and triggered by the client. DHT allows both TCP and UDP network protocols (information on these protocols can be found in Appendix D). Surprisingly, existing simulation software does not usually enable UDP communication. If the simulation is going to be carried out over TCP, the TCP socket remains open; otherwise, the TCP socket is closed and a new UDP socket is opened. The client then pauses (ordering the server to pause too) until a start signal is received. At this stage, researchers on client and server sides perform laboratory-related activities, such as turning hydraulics on. This is normally done with support of phone communication.

Once the laboratory hardware is ready, the user at the client side activates the start-signal via the keyboard and the simulation starts. The simulation follows a predefined loop as shown in Figure 7.4b: the client reads from the dSPACE board (1), sends the data to the server (2), the server receives the data from the network, writes them to the dSPACE board (3), reads from the board (4), sends the data to the client (5), which receives the data and write them to the board (6) to start the loop again. Communication with the dSPACE board uses the *dSPACE Clib*

⁶ A unique identification address of a network interface on the Internet.

⁷ A port is a logical channel to establish a communication on the Internet, allowing multiple communication channels over a single physical channel.

library and communication with the network uses the *Winsock* library. This loop runs until the *simulation-end* signal is received: a specific value read from the board. Once the client reads the simulation-end signal from the board, it informs the server that the simulation is over. Memory is deallocated, sockets are closed and the dSPACE board is unregistered.

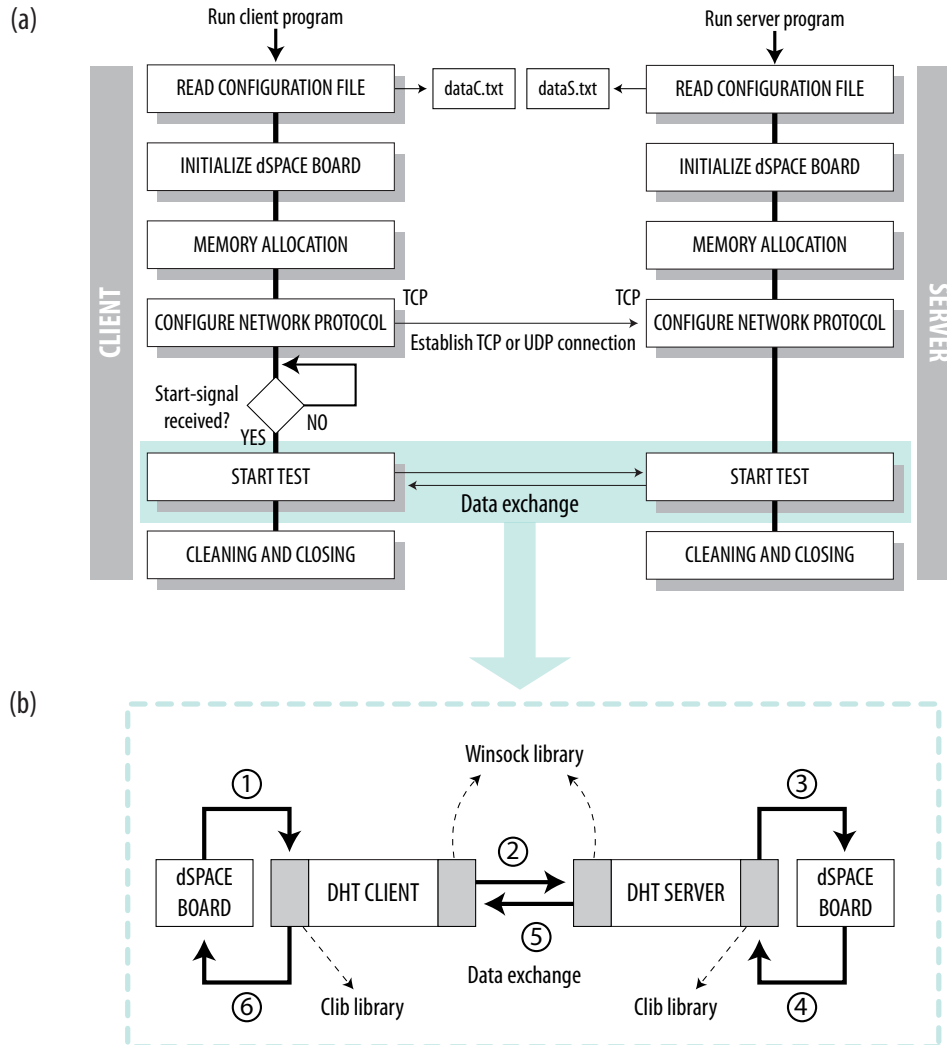


Figure 7.4. (a) DHT main workflow for client and server (b) Steps carried out during the simulation

The DHT program has been successfully used for distributed testing between the Universities of Oxford and Bristol, and occasionally used for a multi-site test including the University of Cambridge (Ojaghi *et al.*, 2010). However, DHT was developed for very specific hardware (dSPACE and Windows) to serve a very specific purpose for a single type of simulation. It was highly coupled with the simulation control algorithms and, therefore, very inflexible in terms of what the program could do. As an example, any alteration in the data exchanged required modification to the source code for it to be recompiled. This makes very tedious both code maintenance and

software versioning but it is justified by the requirement of running as fast as possible (since flexibility in compiled code normally implies additional instructions that eventually imply delays). Even though the program is intended for extreme performance and real-time (see Chapter 8), it does not really implement any real-time mechanism, and it just blocks reading from the dSPACE board until data is provided. Hence, time management is delegated to the dSPACE board. The program runs on a single thread in any of its variants. Some of the lessons learnt with DHT can be found in the following section.

7.6 Results from the DHT program

The simulations conducted between the Universities of Oxford, Bristol and Cambridge with the DHT program provided a wide understanding on earthquake engineering requirements for distributed testing. It also revealed some of the most common mistakes in the conduction of distributed simulations and highlighted important lacks in the subject. A summary of this is presented below. Additional information about the UK-NEES tests and the methodology applied can be found in Ojaghi (2011) and Ojaghi (2011b).

7.6.1 Seek for computing expertise

The first observation when running experiments was that both civil engineers and computer scientist were necessary. Civil engineers do not normally gather the required IT knowledge to address problems as efficiently as a computer scientist whereas computer scientists lack the engineering expertise required for the experiments. As soon as specialised or bespoke software is required, the involvement of computer scientists is crucial, and the communication between both roles is important for the success of the research. This is in agreement with the observations conducted by Spencer *et al.* (2008).

7.6.2 Understand operating system internals

One of the most common mistakes is to blindly rely on what it is commanded by means of a program and expect a deterministic execution. Let us use Code 7.1 as a example. The code developed by the programmer in C language is shown on the right side, with the correspondent assembly translation for a x86 processor on the left. From a programmer perspective, it might be believed that the code will be executed sequentially without interruption: adding two values, assigning the result to a variable and checking if the variable is greater than 100 to set a flag. In reality, such high-level instructions are actually translated in a sequence of processor instructions and, without taking special measures, the operating system might decide to stop execution of the code temporarily at any arbitrary time (even in the middle of a high-level instruction, *e.g.* line 3 in the code example, since the processor works with machine code). This is the way multi-tasking is implemented, since the limited processors have to be shared between all the tasks running in the system.

Code 7.1. Example of preemption (assembly code | C code)

1: MOV EAX, [4C9C98h]	iSize = (iSitesNumber + iOverhead);
2: MOV EDX, [4C9C84h]	
3: ADD EAX, EDX	
4: MOV [4C9C00h], EAX	
5: CMP DWORD PTR [4C9C00h], 64h	if (iSize > 100) {
6: JLE 71A13016	
7: MOV [4C9C72h], 1h	flag = 1;
8:	}

There is no public software to conduct distributed simulations that takes special measures to prevent this behaviour, which means that the time required to run a piece of code is not deterministic. In order to gain control of the simulation execution, it must be understood what the operating system is doing underneath and how our program instructions are actually placed on the computer processor for execution.

7.6.3 Design networks correctly

The DHT program was tested under two different networks, the UK-Light dedicated network and the public Internet. Although it might not always be the case, dedicated networks usually provide an advantage in *quality of service* (QoS) and, therefore, in the robustness of distributed simulations. The network QoS is defined in terms of reliability, delay, jitter and bandwidth (Tanenbaum *et al.*, 2010):

- Reliability measures how feasible it is to send data from a source to a destination without errors (both data losses and alterations of the data content).
- Delay or latency is the time that a network package takes to reach its destination.
- Jitter is the variation of the delay when sending multiple packages.
- Bandwidth is the amount of data that can be sent in a fixed time.

A dedicated line is usually more reliable, provides a lower latency and less jitter than a public network of networks such as the Internet. In general terms, data exchanged in a distributed simulation are small in size but need to arrive within a short period of time. Hence, distributed simulation requires low-latency and low jitter (especially in real-time simulations), it can cope with a few reliability errors and does not need high bandwidth. A reliable network helps greatly to minimise delays by enabling the use of non-reliable network protocols that typically get lower latency by avoiding error checks and error corrections. Reliability is very important but can be sacrificed to some extent if it adversely affects latency. A single isolated error could be overlooked by using data extrapolation algorithms, which can be more convenient than a data retransmission that could ruin the whole simulation. In these cases, latency takes precedence over reliability as long as errors are few during the simulation. High bandwidth is not really necessary in distributed simulation, since data exchanged during an experiment are usually quite small (in order of a few bytes for every data exchange). Therefore, it is a big mistake to design a dedicated network for distributed simulation giving priority to a high bandwidth. This is a reason some dedicated networks might offer no significant performance improvement over the Internet, and it has been the case in some of the simulations conducted with DHT under the UK-light dedicated link between Bristol and Oxford Universities (Ojaghi, 2010b).

For all the aforementioned reasons, dedicated networks are preferred for distributed simulations. However, building a dedicated network line is very expensive and not always affordable or feasible. This forces the use of public networks, normally the Internet. If simulation participants are geographically far from each other, the network reliability gets seriously affected, as well as the latency and jitter.

7.6.4 Operate in real-time = with time constraints

The DHT program was running as fast as possible with no time-step control. The program was expecting to send a new value as soon as it was provided by the dSPACE board. Even if the code gave the impression of sending data only once per loop iteration, values were continuously provided by the dSPACE board and often the same value was sent several times unnecessarily. A real-time program⁸ is expected to send a value for each time constraint as depicted in Figure 7.5a. However, monitoring the DHT program activity revealed that the first value was sent in 7 ms, then 91 values were sent in the next 7 ms and then the last value took long enough to ruin several time step deadlines (see Figure 7.5b).

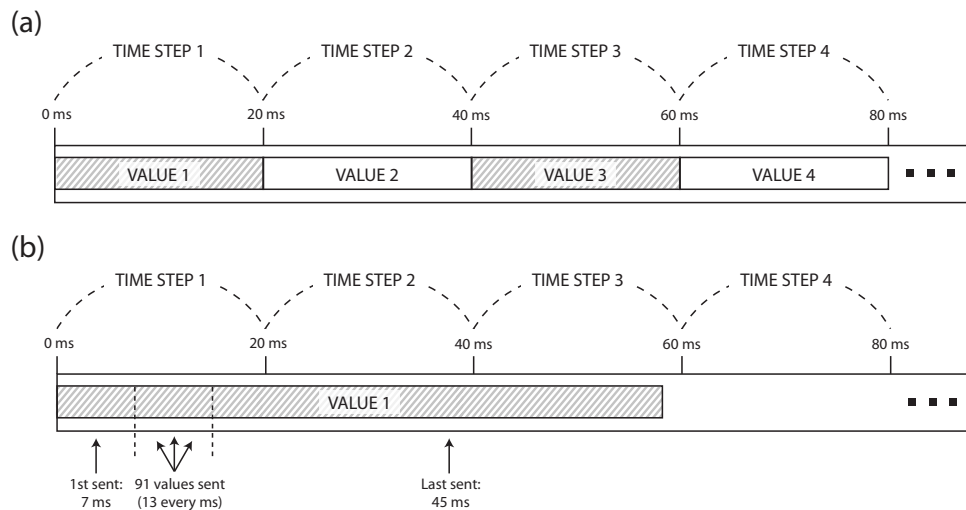


Figure 7.5. Data sent with a 20 ms time constraint (a) As expected in a real-time operation (b) As it appears to the program with no time control

Such operation is often found in software sending data on the Internet but it is totally incorrect in our case. Also, those measurements were taken from the program, which explains the distribution

⁸ The term “real-time” is defined in Section 8.1.

of values sent over the time and is quite unrealistic about what was actually happening in the network: the Windows implementation of the transport protocol was (correctly) buffering all these data to send several packets together. Network monitoring revealed that for instance a hundred sending commands were actually sent in five data segments of 4, 16, 560, 656 and 368 bytes respectively instead of in a data segment for each sending command.

Moving to UDP and blocking on hardware activity is a first step but the program must know about time-steps to send values once per time slot.

7.6.5 Choose appropriate network protocols

To solve some of the network data delivery problems exposed on the previous section, an appropriate network protocol must be used. The simulations conducted with the DHT program proved that TCP is not an appropriate protocol to deliver small data in low latency terms (at least in its standard behaviour), as it is widely known in the Internet community.

However, all the main available software for distributed simulations in earthquake engineering **only** works with TCP, which is a proven mistake for some simulations.

Many network infrastructures and its operational protocols have not been designed to satisfy the requirements of distributed simulations because they are not deterministic and they give priority to reliability over low latency. This will be further discussed on Section 8.3.3.

7.6.6 Tedious to plan and organise

Simulations conducted between the Universities of Oxford, Bristol and Cambridge demonstrated that simulations running in a distributed environment are tedious to organise. Planning an experiment and the execution of the different steps surrounded a simulation required extensive human interaction and, sometimes, tedious software configuration protocols that led to concealed errors (De la Flor *et al.*, 2010). This pointed to the necessity of a platform supporting simulation activities to facilitate researchers' work.

Such activities are not part of the actual data exchange between simulation participants (which will be called **low-level simulation activities**) but of the **high-level simulation activities**

that prepare and configure the simulation environment to run a simulation. Existing distributed simulation software does not make a clear distinction between these two groups.

7.7 Conclusions

Distributed simulations provide the main source of data to understand the behaviour of structures and other components under seismic activity. They can be divided into numerical simulations (computer programs simulating a sub-structure behaviour), physical simulations (experiments conducted over a real sub-structure) and hybrid (a combination of numerical and physical simulations). The basic activities of a simulation software have been described and an example of real simulation software has been described: the DHT program used to conduct simulations between the Universities of Oxford, Bristol and Cambridge. This software, together with the analysis of similar simulation software available for earthquake engineering, provide results that set the foundation for two new research areas:

- The analysis of distributed simulation requirements for laboratory integration, discussed in Chapter 8.
- The development of high-level standards to support earthquake engineering simulations, discussed in Chapter 9.

Chapter 8

Laboratory integration

Laboratory integration is the interconnection of different distributed laboratory facilities creating the illusion as if they were in the same laboratory¹. This often implies extreme performance and operation in real-time basis (these concepts will be clarified in Section 8.1). To the best of the author's knowledge, there exists no software for distributed simulation specifically designed for real-time simulation in the earthquake engineering field. This does not mean that existing software can not be successfully used in real-time simulations (and, actually, firm real-time feasibility has been proved with existing software; Ojaghi *et al.*, 2013) but that existing software has not been internally designed for real-time purposes.

A good example is the family of DHT programs discussed in Section 7.5, developed for distributed hybrid simulation by the author and extended by other researchers. Although the DHT programs became more sophisticated over time and implemented new techniques to try to minimise delays - sometimes with the assistance of external software, they did not internally operate in real-time basis. Simulations ran in real-time at both ends thanks to real-time controllers, but real-time operation was lost as soon as host computers and networks were involved. Real-time clock management was delegated to asynchronous input/output operations to external systems while the program operated as fast as the operating system allowed it. Most of the earthquake engineering distributed testing programs work similarly, trying to send data as fast as they are allowed without an actual time control and a proper study of both the network protocols involved and the underlying operating system scheduler.

¹ Note that this integration entails challenges in terms of control of systems rather than in semantics.

Laboratory integration is purely part of the low-level simulation activities, which are the essential part of a simulation - the actual simulation execution. This chapter discusses the basic concepts behind laboratory integration and presents some improvements made over simulation software (specifically, over the DHT program described in Section 7.5) to achieve laboratory integration.

A key aspect of laboratory integration is the operation in real-time, which often leads to the necessity of maximising the performance of the computing resources. These concepts will be clarified in Section 8.1, explaining where the necessity for real-time in simulations comes from. Section 8.2 lists some of the necessary mechanism to implement a real-time environment in laboratory integration. Section 8.3 discusses the different elements affected by real-time in the laboratory integration environment. These three sections set the basis for RT-DHT, a real-time aware program for distributed simulation that is presented in Section 8.4. Some security considerations for real-time are briefly discussed in Section 8.5. Finally, Section 8.6 presents the conclusions of the chapter.

8.1 Real-time and extreme performance

In this thesis I will distinguish between *real-time* and *extreme performance*, and the meaning of both concepts will be clarified in this section.

Real-time is when a system operation “must satisfy explicit response-time constraints or risk severe consequences, including failure” (Laplante, 2011). A system operates in real-time when it works with time constraints, regardless of the constraint being 20 milliseconds, 20 seconds or 2 hours. Correctness of a real-time system does not only depend on accurate results but also on the time in which such results are formulated. Real-time systems operate nuclear stations, traffic control, airplanes, cars, *etc.* There are two classic types of real-time: *soft real-time* (when the system can cope with some missed time deadlines without facing catastrophic consequences) and *hard real-time* (when missing a single time deadline implies total system failure)².

Extreme performance is the configuration of hardware and software necessary to maximise the performance of existing system resources, normally leading to a fast operation to reach some objective.

² Sometimes, firm real-time is also used to refer to a real-time system that holds a threshold of missed deadlines before it behaves as a hard real-time system.

In the previous definitions, real-time does not imply going fast and extreme performance does not imply working with time constraints. Both concepts are complementary. Note that in earthquake engineering, real-time is sometimes referred as “fast operation” over *implicit* time constraints (Bonnet *et al.*, 2008). The nomenclature used is not really important as long as the two concepts are distinguished.

The necessity to run a simulation in real-time is dictated by some materials involved in a test, whose behaviour is affected by velocity (especially those with a lower viscosity or lower resistance to gradual deformation) or, in other words, which are rate dependent (see Figure 7.1). If velocity is a factor, it is required to apply a dynamic load (a force that varies over time) and operate in real-time, normally with small time constraints. Conducting the simulation slowly will cause a similar effect as applying a series of static loads and the velocity factor of the material will be missed. However, materials with low time dependency, such as concrete, do not require testing in real-time nor fast operation.

In a pure numerical simulation, time could be annulled as a variable by simulating time in a longer scale. This is a common practice in some hybrid simulations, where the actual time in the laboratory differs from the accelerometer time (Pegon, 2008). Results of numerical calculations do not really depend on when events happen over physical time and, if so, time could be scaled and simulated in a computer. This practice is not valid for rate dependent physical specimens. If a sensor is not able to capture the specimen reaction at a certain time, these data will be missed. If the specimen tested is not actuated at the right time, it will relax and its behaviour will not be captured correctly. Note that real-time pure numerical simulations are merely used for proofing purposes since they do not really demand real-time operation.

Distributed simulations can be considered *firm real-time*, because missing a few time deadlines can be acceptable or fairly solvable with extrapolation or prediction algorithms. The time rate necessary to interact with the specimen is what establishes the experiment time-step and it is normally small enough (few milliseconds, typically 20 ms) to push elements in the experiment to their performance limits, especially if they are not quite designed for real-time operation. In non real-time systems, achieving extreme performance is a way to *simulate* real-time operation, normally by optimising specific hardware and using other mechanisms discussed below in this chapter. In practice, an experiment could be conducted a hundred times giving the impression to run in

real-time basis, because the time deadlines are satisfied. Nevertheless, it must be considered that a real-time system is **deterministic**: the response time of the system *can be predicted* or, more formally, for each possible state of the system and any possible input, the next state and output can be determined (Laplante, 2011); and this is not possible when no deterministic elements are part of an experiment, even if sporadically or even normally they behave appropriately.

Let us use the example of the Internet that will be discussed later. Sending data from the UK to the USA is not deterministic. During the data routing, packets might have to be delivered through an alternative route in an event of network congestion, likely causing additional data transmission delays. This event might not happen often in practice, but it could happen since the Internet protocols are designed to work this way. Such a risk must be known and considered - for instance, if a very expensive specimen is being tested. Another example of a non real-time element is a general purpose operating system such as Windows. Operation on Windows under standard conditions is not deterministic: the operating system might decide that a thread not related to the experiment has more priority or that an unrelated interruption has to be attended, stopping the execution of the simulation software at a crucial time. Again, this might not happen during a large number of experiments, but statistically it will.

The environment in which an experiment is conducted, formed by elements such as networks or operating systems used, must be analysed to understand all elements functioning. It must be considered that hard real-time simulations cannot be accomplished if a single element in the environment is not designed for real-time. However, in practice, real-time simulation can be achieved if the conditions are propitious. The following section presents some mechanisms for real-time.

8.2 Real-time mechanisms for laboratory integration

The creation of a real-time system requires orchestration of all elements involved. Real-time operation is not guaranteed by the simple fact of using a real-time operating system or real-time protocols, and detailed system design is needed.

Ideally, a real-time system for distributed simulation should be made by a microcontroller with enough memory to store exchanged data, and use a direct physical network connection managed by a simple, bespoke protocol to communicate with other ends. The communication with external

controllers managing facility devices should be driven by compiled instructions that are loaded in memory, and that are also in charge of performing deterministic operations over the data (including corrections and interpolations), interacting with the memory and interfacing with the network. Once the simulation is over, a dump of the memory is obtained. This system can provide minimum latency and deterministic operations, and is fully predictable as long as external devices (facility devices such as actuators and the network) can operate at the desired time-step.

However, although such a system would make laboratory integration robust and efficient, its construction is not always feasible. Direct network links between facilities are simply too expensive and often the existing network infrastructure is used instead. General-purpose operating systems and microprocessors are frequently used in host computers and management of facility devices and network are delegated to drivers developed by external manufacturers. All this makes nondeterministic systems that *can produce* real-time results but that do not operate in hard real-time. The challenge of laboratory integration lies in the emulation of real-time operation with nondeterministic elements involved, together with the small time-step required. In this situation, some of the mechanisms that must be applied are listed below:

- **Clock management.** Real-time is strongly related to when events happen, and the length of simulation time-steps has to be accurately measured. A clock is a mechanism to measure time, often driven by an oscillator, with properties such as resolution (minimum time interval that can be measured) and uniformity (time progression at a constant rate, which is normally affected by temperature and other factors). Mechanisms to read the local clock have to be analysed, since some of them provide an incorrect time measurement. In a distributed environment, although no clock is perfectly accurate and the uniformity of the different distributed clocks can vary, earthquake engineering simulations do not usually last long enough to appreciate such variations. Synchronization of distributed clocks is not normally necessary and time in a distributed environment can be processed with a local clock relatively based on an initial event.
- **Task scheduling.** An application working in a real-time environment is sometimes divided into different autonomous tasks. These tasks might have different importance and assigning correct priorities to them in the schedule is crucial to meet time deadlines, especially when computer resources have to be shared. Operating systems enable mechanism to

assign priorities to tasks, which are executed according to a scheduling algorithm. This algorithm is sometimes *preemptive*, stopping the execution of a task to start executing a more critical one. Swapping tasks in a processor entails several operations called *context switching*, and this might consume an unpredictable amount of time. Correct scheduling and priority assignation is crucial to satisfy time deadlines.

- **Processor instructions.** The tasks of a real-time application have to be supplied as a set of instructions to be executed by the processor. These instructions (sometimes called *machine code*) are coded in a specific structure that is valid for a processor architecture. Machine code instructions can be easily translated into an assembly language that represents the instructions in a more human understandable way. Assembly languages are low-level and processor specific, so many engineers use higher level languages, which are more comprehensive and portable. It is important to maintain control of the actual instructions that are provided to the processor, as well as produce deterministic instructions that guarantee a predictable time operation for a real-time environment.
- **Computer networks.** Laboratory integration describes a distributed system that interconnects nodes by using computer networks. Some network protocols have been designed to benefit interconnection and are not suitable for real-time, which makes networks one of the main nondeterministic elements in a distributed simulation. Herein, appropriate real-time network protocols and transmission media must be chosen. Like the processor on a multi-tasking environment, network devices are normally shared, with the inconvenience that: (i) unlike local tasks, no control can be normally done over the other elements using the shared resource and (ii) network delays are greater than processor related ones.

These mechanisms must be considered in a real-time environment, and will be applied and further developed in the following sections.

8.3 Elements involved in laboratory integration

Unlike in conventional simulations, laboratory integration must assure both output correctness and appropriate output timing. Output correctness is always considered in distributed simulations

but, for appropriate output timing, the inherent non-determinism of the different elements involved must be analysed.

Firstly, the physical network used to transmit data must be known. The structure of the network and the intermediate devices from one experiment participant to another play an important role. This is discussed in Sections 8.3.1 and 8.3.2. Secondly, the network protocols to manage the data transmission must be appropriately chosen. Although all network protocols will deliver data from one source to a destination, their behaviour might make them inappropriate for distributed simulations, as discussed in Section 8.3.3. Thirdly, experiment physical components must be configured to work on a real-time basis according to a desirable time-step, commanding devices and capturing sensor information. Some physical devices might be too slow to achieve the desired time-step, limiting testing possibilities. This aspect is not discussed in this thesis. Finally, the middleware software in charge of dealing with the experiment devices and the network (the simulation control software for low-level testing activities as depicted in Figure 7.2) must be programmed to lead the data flow from testing devices to the other end in the network. This is discussed in Section 8.3.4.

8.3.1 Wide area physical network

The first aspect to consider in laboratory integration is the physical network that will connect the simulation participants. This will always consist of physical media to transport data (cables, radio frequencies, *etc*) and intermediate network devices to retransmit data (hubs, routers, satellites, *etc*). It is important to evaluate the network quality of service, especially in terms of reliability, delay and jitter. Reliability is not normally a problem in standard cabled networks nowadays. Regarding delays, distance between participants is an issue in two aspects:

- The speed at which data can be transmitted through the media. Using optical fibre on an imaginary straight line, the speed of light propagation delay between Oxford, UK, and California, USA, is 29.5 ms and it is 48 ms between Oxford and Tokyo, Japan. Working with 20 ms time-steps is unfeasible under these conditions.
- The number of intermediate network devices which data passes through is normally larger for longer distances. This aspect is also one of the main problems with jitter: networks

are normally shared, operating protocols that do not reserve resources, which makes a guarantee of real-time operation for laboratory integration unfeasible.

That is why laboratory integration between geographically closer participants is more likely to succeed, providing the network is efficient and well-designed.

In terms of physical networks, a dedicated network between participants is normally a better option, but it is expensive even for short distance participants and not affordable for long distances. Even in dedicated networks, resources will likely be shared with some other computers, although resource booking might be guaranteed. Ideally, the intermediate devices will just operate as switches, acting on the data link layer (which does not require routing) and simply retransmitting received signals. If a dedicated network is not a possibility, educational networks, widely used in European laboratories, are a fairly good option. They rarely fail but are shared networks and might use public Internet devices sporadically.

The public Internet is the only valid option between some participants, especially when they are geographically far from each other. The common Internet infrastructure will definitely connect experiment participants, but data transmitted might be prone to errors and be transmitted through many publicly shared network devices. The main problem with shared devices is how they normally operate in case of congestion or high workload: our data might be redirected through slower routes, divided into several data packets or simply buffered in a queue to be sent. This unpredictable aspect of the network prevents hard real-time operation (note that soft real-time operation is still feasible and currently achieved in many Internet applications).

If the network media cannot transmit data as fast as required for our simulation time deadlines, laboratory integration under such conditions will be unfeasible. Similarly, if the network devices along the path do not accept resources booking, it is not possible to guarantee real-time operation because outgoing time cannot be predicted.

8.3.2 Local physical network

The reason to divide the analysis between external and internal physical networks is that some local networking aspects might be overlooked. When a computer is in a local network, the physical media are sometimes shared. In terms of laboratory integration, this can be one of the major, most

common problems that is rarely considered in earthquake engineering. Local networks are shared resources and operate under protocols to enable media access control. If computers are connected to a switch, the data link is not normally shared, avoiding access collision problems³. However, switches normally use buffers and may still suffer from buffer overflows and non deterministic behaviour (Kopetz, 2013).

In some other networks (for example those operated by hubs), assuming that data are delivered as soon as a data transmission is commanded is a common misconception: if a cable or a radio frequency is used at the same time by two computers, their data will interfere each other. One of the traditional methods to manage network media access control is to verify if the network media is being used before transmitting and wait if it is - as in the protocol CSMA/CD⁴ (Tanenbaum *et al.*, 2010). Should a collision be detected (the media is used at the very same time or slightly before by other user) participants will wait a *random* time before trying to access the media again. This functioning will ruin any real-time operation because network latency is not deterministic. The use of wireless networks is discouraged both for this reason and for their potential instability.

To prevent such unpredictability, laboratory integration participants must assure that no network media is shared with other computers (*e.g.* by splitting network traffic with a switch) nor with host applications outside the simulation. Also, a media access control protocol enabling deterministic communication latency should be used.

8.3.3 Distributed simulation on the Internet

The Internet was born as a research network based on packet switching, highly redundant in terms of connectivity (Baran, 1964), which finally expanded to be a world-wide network of networks. The Internet was designed to be a very robust network, able to operate and re-route data in adverse circumstances (for instance if part of the physical network infrastructure disappeared). In this context, performance and determinism was not a priority over availability, integration and adaptability. This is important in order to understand the problems of real-time on the Internet. Real-time applications are becoming more and more important in the Internet and many efforts have been conducted to satisfy real-time requirements, especially for interactive audio and video

³ Strictly speaking, this is not true if half duplex cables are used (Tanenbaum *et al.*, 2010).

⁴ Carrier sense multiple access / collision detection.

applications. However, typical techniques used for these applications such as buffering, Forward Error Correction or interleaving (Tanenbaum *et al.*, 2010) are just not suitable or useful for real-time distributed testing.

Internet protocols must be adequately chosen for distributed simulation on the Internet, and a brief description of these protocols can be found in Appendix D. The TCP protocol must be avoided for laboratory integration, for which UDP is preferred because of its lower overhead, the lack of error correction and error detection in practice. The disadvantage is that UDP's lack of reliability must be compensated by a very stable network link. An alternative to TCP/UDP is to use a bespoke transport protocol over IP, specifically designed for simulation purposes.

Hard real-time on the Internet does not seem really feasible nowadays since current protocols operation is not deterministic. This will change in future with resource booking in network paths. However, there are still physical speed limitations to interacting at rates such as 20 ms time-step between far away test participants (from 3,000 km). On the other hand, soft real-time on the Internet is perfectly possible and many applications work this way successfully.

8.3.4 The simulation control program

The simulation control program (SCP) is the element of the environment that can be more easily controlled and tuned by simulation participants, especially if bespoke software is developed. The first thing that must be considered is that there is always a compromise between performance and flexibility⁵ of the software, and that prioritising one can act to the detriment of the other. In general terms, performance should be prioritised for laboratory integration, making the work easier for the processor at execution time and harder for the programmer at development time.

Regardless of the language in which the SCP is coded, when the program is executed is no more than a set of machine code instructions for the processor. This brings three considerations: (i) Some languages *per se* are more suitable for real-time because they generate more efficient, less over-bloated code. (ii) Some languages allow easier performance optimisations. (iii) The compiler used to compile or translate the code is important because it is the actual generator of final code.

⁵ Defined as portability and ease of development and maintenance.

Coding in a language close to low-level such as C or assembly is recommended - although the use of high-level, object-oriented languages such as Real-time Java or C++ is increasing according to some authors (Laplante, 2011), who also discourage the use of low-level languages. Nevertheless, low-level languages still have benefits in terms of performance prediction and provide a finer control. Inherent characteristics of object-oriented programming might generate unnecessary instructions (especially in CALL instructions) and features such as the Java automatic garbage collection⁶ or the dependency with a Java Virtual Machine interpreter might have negative consequences. Despite the efforts to reduce high-level programming problems, it is recommended that C is used, with potential assembly optimisations for a more direct control of the hardware in the critical sections of the program. Note that by using this method, portability to other hardware and software is minimised.

Regardless of the SCP programming language, the actual code must be deterministic, and unbounded loops and recursion must be avoided. If extreme performance is desired, general well-coding rules can be avoided, *e.g.* by means of *demodularisation* (by using inline functions and code duplication to save function calls and stack use) and *loop unrolling* (the loop instructions are explicitly copied in multiple instructions).

The SCP will normally use libraries to access testing device hardware and network hardware. Such libraries might contain general-purpose code that is not deterministic. Very little optimisation can be done regarding this and a dramatic solution is the creation of a device driver for the hardware which can be a complex task.

The operating system running the SCP must be studied in order to determine how the SCP code is going to be executed. General purposes operating systems are not prepared for real-time operation and any effort to create a deterministic code might be spoiled if the program instructions do not reach the processor in time.

Finally, the SCP environment and the additional programs running must be considered. There is no point in optimising the SCP and develop assembly critical sections for extreme performance just to reduce a few milliseconds of execution time if, in parallel, other simulation parts execute via not optimised or nondeterministic code (as it could be some Matlab-generated code).

⁶ The process of deallocating objects from memory automatically in Java. This process occurs also in other languages and it is not deterministic. However, efforts have been conducted to make a deterministic garbage collection in the Real-Time Specification for Java.

The considerations discussed in this section have been applied to the DHT program and the results will be presented in the following section.

8.4 RT-DHT: DHT program improvements for real-time simulation

The DHT program discussed in Chapter 7 was coded as a standard, non real-time aware application, having very little control of when events happened.

During some experiments, it was not possible to achieve extreme performance because different architectures and limited physical systems were involved, and laboratory integration had to acknowledge the presence of large delays. Herein, data integration and understanding was prioritised over a fast data exchange. In the UK-NEES simulations, a client led the communication with the rest of the participants, which acted as servers. In this operational architecture, the client managed the communication in a similar way to an active centralised manager. The objective was to connect physical devices (such as actuators) from different distributed laboratories to simulate accurately the effects of seismic activity over building sub-structures. From a computational point of view, the challenge lay in connecting different hardware elements (devices, computers, network interfaces) trying to minimise the delays that happen when transporting and processing hardware signals.

Initial results proved the complexity of achieving a robust, stable distributed simulation with minimum delays. Under very specific conditions, some very fast distributed simulations were achieved in a two-site test between the Universities of Oxford and Bristol. Repetition of these simulations is bound to a very particular hardware and software environment that is not always possible to replicate and, even under same circumstances, DHT nondeterministic behaviour will not guarantee real-time operation.

Results from these experiments and the analysis and monitoring of the program operation, made the foundation for the development of RT-DHT (Real-Time Distributed Hybrid Testing). The methodology followed during this analysis is listed below:

- Analysis of network hardware and network topology existing between simulation participants.
- Analysis of hardware of the host computers to conduct the simulations.

- Analysis of the suitability of physical devices (such as actuators) to conduct the simulation in the required time constraints.
- Analysis of the additional software (besides the SCP) necessary for the simulation.
- Analysis of network protocols and operating system mechanisms for real-time (time control, scheduling, *etc*).
- Optimisation of SCP code.

The following sections summarise the main improvements in RT-DHT coming from the application of the previous methodology. Section 8.4.1 presents main changes in the network protocol used. Section 8.4.2 describes the mechanism used for time control. Section 8.4.3 discusses the scheduling mechanism used and the division of RT-DHT instructions in three independent tasks executed in parallel. Section 8.4.4 shows a code optimisation methodology to minimise execution time, and Section 8.4.5 presents some results of RH-DHT performance, tested in three different scenarios in an Oxford laboratory and with the University of Kassel, in Germany. RT-DHT has been developed for Windows⁷, and many considerations presented are specific for this operating system.

8.4.1 Discarding TCP as a network protocol

As discussed before, TCP is not a suitable protocol for real-time operation. TCP buffering and Nagle's algorithm⁸ (Tanenbaum *et al.*, 2010) increase performance when sending data requiring high bandwidth, but they might negatively affect data requiring low latency. Monitoring the network activity showed the TCP protocol was sending five data segments for a hundred data values instead of one data segment for each of the hundred values, which proved the adverse impact of data buffering on latency.

TCP was discarded in favour of UDP. Other efforts were conducted towards the development of a new connection-less transport protocol: EELTP (*Earthquake Engineering Light Transport Protocol*), which works on the top of IP substituting TCP and UDP. The protocol has an 8-byte header containing specific control information such as relative time stamping and packet numbering.

⁷ RT-DHT works under Windows for the equivalent reasons as DHT: the dSPACE library required Windows.

⁸ A congestion control algorithm to increase performance and reduce the bandwidth used.

EELTP was developed with raw sockets, and Windows security restrictions (at the time of the testing, the version XP SP2 was used) created barriers that made it difficult to use in a real distributed simulation.

8.4.2 Measuring time

Access to a hardware clock must be enabled for the program to be aware of timing steps. Applications demanding high-resolution timers typically obtain time from a counter (an increasing variable), which should access a non-decreasing monotonic clock (meaning that every reading value from the clock is always greater than any previous value read, increasing at a fixed rate).

Some functions provided by operating systems do not use a monotonic clock and are unsuitable to obtain accurate time. Besides this, the correctness of the hardware clock depends on its uniformity (time progression at a fixed rate) and its accuracy (how close the desired frequency and the real frequency of the clock oscillator are). To obtain time on Windows, two options were considered:

- Using the *Time Stamp Counter* (TSC), a 64-bit processor register for the 80x86 Intel architecture. Its aim is counting the number of processor clock ticks since reset (which is closely related to the processor clock frequency) and it can be read by using the “RDTSC” instruction. It is hardware dependant but considered a reliable way to have a high precision timer.
- Using the *QueryPerformanceCounter* Windows function (QPC). This function uses a high-resolution counter (the resolution depends on the hardware) to measure the elapsed time between two moments. How to obtain the time is actually managed by the kernel (called via the *system call* 0A5h). This method provides higher resolution than functions such as *GetTickCount* - which might only get time updates every 15 ms approximately (the usual length of the Windows clock interval on multiprocessor computers).

TSC is faster than QPC since it does not require a system call (QPC might internally use the *RDTSC* instruction too). However, TSC is not monotonic and can provide erroneous data - *e.g.*, in a multiprocessor computer, timers in different processors are not synchronized. If the executed thread is moved to a different processor, which is a frequent event, two different timers will be

used. To avoid this, threads should be locked to execute in a single processor. In theory, QPC does not have such problems but in practice they might occur, depending on the underlying hardware or BIOS layers.

RT-DHT uses QPC (as recommended by Microsoft) and forces code dealing with timing to be executed on the same processor. Note that QPC is still not fully reliable: the counter might not be accurate if a *thread context switch* takes place (because a higher priority thread requires the processor), a *hardware interrupt* occurs or the thread *quantum* finishes in the middle of time measurement⁹.

Newer and more robust options to the RDTSC instruction, such as HPET (Intel Corporation, 2004), were not considered because they were not available for the operating system used at the time this research was conducted.

8.4.3 Operating system scheduling

Windows executes tasks by means of threads, which are a sequence of executable instructions of a program. All processes run as one or more independent threads, which share the same process resources. As a multitasking system, Windows schedules threads and decides which one is next to execute according to priorities in a *preemptive* way (moving a thread out from the processor if a new thread with higher priority appears) by using a *Round-Robin* algorithm¹⁰. To determine how long a thread runs on the processor before the operating system checks if another thread with the same priority has to run instead, a time length called “quantum” is used, which actual length varies depending on clock interrupt and processor frequencies (Russovich *et al.*, 2012). The quantum length of the computer used for RT-DHT testing was 0.325 ms, which means that Windows was running a thread for 1.95 ms¹¹ as long as it was not preempted, an input/output operation was not required and the processor was not voluntarily abandoned with a “sleep” function.

Threads in Windows have a priority assigned from 0 to 31 (typical user threads run on priority 8 to 15 and can have their priorities dynamically adjusted by Windows) running on a low *Interrupt*

⁹ The concepts thread, quantum and interrupt will be clarified in Section 8.4.3.

¹⁰ An algorithm that assigns equal processor time to threads, which run in turns.

¹¹ This datum comes from a 1,995 MHz processor frequency and a 0.977 ms clock interval, with Windows XP program performance priority (instead of services).

Request Level (IRQL), typically 0 or 1. IRQL defines a processor-dependant priority list which is used to identify priorities between hardware interrupts¹². Herein, Windows user threads will not block hardware interrupts, which have a higher IRQL.

All DHT versions run in a single thread. However, RT-DHT divides its functionality into three different threads: one for time control (TC), one to read/send data (RS) and another one to receive/write (RW). Considering that RSRW operations should run in strict sequence, as illustrated in Algorithm 7.1, the reason to divide RS and RW operations in different threads is to enable continuous operation: if an RS operation was delayed, it would not block any RW operation and vice versa. Time synchronization is independently done by using timing information provided by the TC thread and *volatile* memory variables¹³. There are two reasons for having an independent thread controlling time: (i) time control is done in a single processor, to avoid issues with different processor timers and (ii) the RS and RW threads can concentrate on their duties without checking the hardware timer. To avoid interference from other threads running in the system, all threads are given high priority (above 16, without interfering with crucial kernel threads). However, this will not block hardware interrupts, which run in a higher IRQL as stated before. The TC thread is locked in a single processor (this is called *processor affinity*) to guarantee monotonic clock use. If the computer has enough processors, each thread is locked in a different one to enable real multi-thread operation¹⁴. In other case, higher priority is given to the RW thread over the two other threads, since its operation is led by the RS thread on the other end. This priorities can be dynamically adjusted by the TC thread if necessary.

The three threads operate as follows: the TC is continuously checking the high-resolution performance counter (coming from the hardware timer) and updating the current time step for the rest of the threads. To know when the next step is coming, the number of counts per time-step is precalculated and added to the base counter at each time-step. The RS thread operates

12 An interrupt is “a hardware signal that initiates an event” (Laplante, 2011). Strictly speaking, there also exists software interrupts, so it can be better described as a request to use the processor, which leads to suspend current code execution and transfer control to an interrupt manager code.

13 *Volatile* is a qualifier to make the compiler aware that the variable can be modified by an external process. Without this, the compiler can translate a high-level instruction in machine code containing unexpected behaviour and providing out of date shared data.

14 Note that threads with lower priorities can still run on these processors, for example when the threads stop execution waiting for an input/output operation to finish.

in a loop; it first waits for the RW thread to complete its task before proceeding - this was done for testing purposes, and was removed for validation, since in practice RS can just assume that RW completed its part in time. Should the RW miss a deadline, the RW itself releases the RS from this wait. Then, the RS waits for the next time step to come, which is informed by the TC thread, before it reads from the board (the device) and sends the read data. This guarantees that RS does not send more than one value per time-step. The RW thread also runs in a loop, waiting for data to arrive. It uses non-blocking network reading operations, and if no data arrives before the time step ends, RW increases a counter for package dropping. This is fine in a highly reliable environment, but in normal conditions the received packet number has to be considered (since packet delivery is not guaranteed with UDP or EELTP). When data arrives, RW writes on the board, informs RS if necessary and waits for the next data to arrive.

Threads work on defined time-constraints, delivering data once per time slot. Windows is not really designed for real-time, and hardware interrupts will preempt RT-DHT threads - even if they are assigned a high priority. This hinders deterministic operation, since little control is owned on how efficiently all drivers in the system handle their interrupts and, therefore, threads might be stopped for an unknown period of time.

8.4.4 Code optimisation

Code optimisation can be performed both on high-level and on low-level code instructions. While high-level instruction optimisation can make a dramatic performance difference, there is still a compilation/interpretation process afterwards, and low-level instruction optimisation allows greater control on the final instructions that will be executed by the processor. However, low-level optimisation is more challenging to apply and it only addresses specific hardware, which eventually leads to non-reusable, difficult to maintain and non-scalable code. The code becomes very inflexible, needing re-compilation if a different behaviour is required or if a different hardware will execute the code.

Optimisations typically focus either on reducing code size or on improving execution speed and it should be noted that there is no point in extreme code optimisation if other poorly optimised or general code is also used during the testing (such as Simulink/Matlab automatically generated

programs). Ideally, all computational needs (numerical model, control algorithms, correction algorithms, *etc*) should be implemented into and controlled within the SCP.

When a program source code is compiled, the compiler makes decisions to generate machine code for the target machine. By rewriting critic sections of the code directly in optimised assembly, the compiler does not have to make compilation decisions on these parts and code can be exhaustively tuned for a specific processor. However, note that sometimes, final instructions provided to the processor might be translated by the processor itself into equivalent micro-ops¹⁵ as a reordering optimisation technique (*e.g.* stack instructions can be translated into a subtraction/addition and a movement of data to/from memory). If the processor believes that execution optimisations are possible, it might conduct variations on the order of the instructions, change the registers that are used or divide the instruction into different sub-instructions.

Code optimisation was conducted over the original DHT code used in RT-DHT. Firstly, some high-level optimisations were conducted, consisting of: (i) loop unrolling, (ii) moving functions code to inline code and (iii) improving write operations speed by minimizing disk access and storing all data in memory. The latter requires a large amount of memory in the host computer depending on the length of the simulation and the size of the exchanged data - note that dynamic memory allocation requests *during* the simulation should be avoided, since it is a nondeterministic operation.

Secondly, low-level optimisations for speed boost were conducted. To analyse the old DHT generated machine code, it was reverse engineered both with a disassembler and with a kernel-mode debugger¹⁶ (the latter was necessary to understand the interaction of the program with other system components). An analysis on how long every micro-op takes was done, according to micro-op latency (number of clock cycles¹⁷ needed to execute all micro-ops of an instruction) and throughput (number of clock cycles to accept the same instruction again) (Intel Corporation, 2013). However, accurate calculation of the required time from the instructions sequence does not only require a deep understanding of the processor, but it can be extremely complicated because of instruction parallelism, micro-op caching and dependance with surrounded instructions.

¹⁵ A micro-operation is a reduced, elementary low-level operation that can construct more complex instructions.

¹⁶ A disassembler obtains a “dead-list” of assembly instructions from machine code instructions (ex. IDA) whereas a debugger allows to “freeze” the processor and interact with instructions being currently executed (ex. NuMega SoftICE).

¹⁷ Smallest unit of processor activity.

Let us consider the small piece of C code at Code 8.1 as an example. This old code belonged to the TC thread originally, and it sets a flag in order to notify the RS thread that it can sleep until the next time-step arrives, as long as the RS thread has finished its duties for the current time step. The code is naive and poorly programmed, which makes it ideal to explain the optimisation methodology. Line [5] sets the flag providing that the flag is not already set (checked in line [2]), the data for the current timestep has been already sent (first comparison found in line [3]) and the current time is not greater than a predefined threshold for the beginning of the next time step (second comparison, found in line [4]). The only importance of this code is to illustrate the methodology of the optimisation, not the code operation. The code works correctly, but it is by no means efficient.

Code 8.1. TC thread example code (poorly programmed)

```

1:  do {
2:    if (sleepFlag != 1) {
3:      if ( (dataNumber >= CurrentStep()) &&
4:          (CurrentMomentMs() < (timeStep * 1000 * dataNumber)
5:            - threshold) ) {
6:        sleepFlag = 1;
7:      }
8:      QueryPerformanceCounter(&after);
9:    } while ( ((after - startProgram) / frequency) < (timeStep *
              CurrentStep()) );

```

The compilation of Code 8.1 has been partially disassembled as shown in Code 8.2, where the C instructions can be seen on top of their corresponding assembly instructions. The first (high-level) optimisation that can be conducted is related to the functions *CurrentMomentMs* and *CurrentStep*, called in lines [3] and [6] respectively of Code 8.2. The former returns a *double* corresponding to the number of milliseconds passed from the start of the simulation, whereas the latter returns an *integer* containing the step number in which the simulation currently is. They have been correctly programmed as independent functions, but in this context they both repeat the same task, creating unnecessary delays: they obtain the time by calling the *QueryPerformanceCounter* function (QPC). This code can be optimised by inlining instructions in the main loop and calling QPC only once.

The debugger showed that QPC uses the Windows system call *0A5h* to enter privileged mode and run the kernel code (by means of the *SYSENTER* instruction), so the function call could be bypassed (by mirroring its code in our program) for extreme performace. However, this is not recommended since the improvement in time would not justify it.

Code 8.2. Partial assembly code corresponding to Code 8.1

```

do {
  if (sleepFlag != 1) {
    1: .text:0043D7E6      CMP      DWORD PTR [sleepFlag], 1
    2: .text:0043D7ED      JZ       SHORT 43D823

      if ( (dataNumber >= CurrentStep())

    3: .text:0043D7EF      CALL     j _CurrentStep
    4: .text:0043D7F4      CMP      [dataNumber], EAX
    5: .text:0043D7FA      JL       SHORT 43D823

      if (CurrentMomentMs() < (timeStep * 1000 * dataNumber) - threshold) ) {
    6: .text:0043D7FC      CALL     j _CurrentMomentMs
    7: .text:0043D801      FLD      [EBP + timeStep]
    8: .text:0043D804      FMUL     DS: __real@408f400000000000
    9: .text:0043D80A      FIMUL    [dataNumber]
    10: .text:0043D80D     FSUB     [EBP + threshold]
    11: .text:0043D810     FCOMPP   ST(1), ST(0)
    12: .text:0043D812     FNSTSW   AX
    13: .text:0043D814     TEST     AH, 41h
    14: .text:0043D817     JNZ      SHORT 43D823

      sleepFlag = 1;
    15: .text:0043D819     MOV      DWORD PTR [sleepFlag], 1

      }
  }
}

16: .text:0043D823      ...

```

The instructions in lines [7] to [12] use the float point unit to work out the operations. However, considering the instructions are just comparing two counters, this complexity is unnecessary. Firstly, the multiplication by 1000 is intended to obtain the operational time-step in milliseconds. A high-level optimisation can be to precalculate this number in non-critic parts of the code and assign it to a variable *timeStepMsNumber*. Secondly, central processor units execute additions faster

than multiplications. The code can run faster by transforming the multiplication by *dataNumber* (which here represents the current step number) into an addition between *timeStepMsNumber* and an accumulation of *timeStepMsNumber* values from the previous steps. To distinguish if the step has changed and the addition is required, an *XOR*¹⁸ operation can be done. However, this optimization can be improved by using a matrix of precalculated threshold values for each time step, as shown in Code 8.3. This also helps to eliminate additional delay due to data type conversion in the floating point instructions, since *timeStep* is a double and is operated with *dataNumber*, which is an integer.

Code 8.3. Creation of a vector with millisecond threshold values for each step

```

1: base = startTimeCount - thresholdCount;
2: for (int i = 0; i < thresholdStepSize; i++) {
3:     thresholdStep[i] = ( timeStepCount * (i + 1) ) + base;
4: }
```

A possible optimised piece of code for a 64 bit architecture can be found in Code 8.4, which is mainly based on high-level optimisations. The estimation of the code execution is 17 clock cycles in an Intel Core processor (Intel Corporation, 2013; Fog, 2013), plus the time t_{QPC} required by the `QueryPerformanceCounter` function. This estimation might differ in a real execution for a variety of factors, such as reordering of instructions. Some instructions might be executed in parallel, like perhaps lines [3] and [4] - note that the register *EDI* is not modified by the QPC function. On the same processor, the old code is estimated to require around 328 clock cycles plus $2 * t_{\text{QPC}}$, plus the time of 7 other functions that are called inside lines [3] and [6] of Code 8.2. Producing the same output, the optimised code is not only more efficient but also more deterministic. This big difference is mainly because the old code was poorly programmed and the compiler did not produce very efficient machine code either.

Measurement of instruction time execution by means of instruction latency and throughput might overlook many factors and differ from actual execution time due to complex internal functioning of specific processors, the use of cache memories, *etc.* If more accurate measurement is required,

18 Exclusive OR operation, which returns 0 if both operands are equal or 1 otherwise.

time estimation should be done by running the code or with the help of a profiler¹⁹.

Manual low level optimisations were not finally included in RT-DHT since they proved not to be significant in a distributed simulation once the network was involved - the communication entailed delays that could not be overlooked even after optimisation. The main advantage of assembly optimisation lies in the more complex mathematical parts of the numerical calculations and interpolation/prediction algorithms requiring intensive processor activity, where low-level optimisations can make a difference. Although these parts are not integrated within RT-DHT, this section illustrates the optimisation methodology that can be applied over them.

As a final note, compilers have often an option to generate code with automatic optimisations which, sometimes, not only complement but surpass manual optimisations.

Code 8.4. A possible code optimisation for Code 8.2

```

1:      CMP      DWORD PTR [sleepFlag], 1
2:      JZ       ENDIF
3:      MOV      EDI, [dataNumber]
4:      PUSH     currentCounter ;direct address such as 1212120h
5:      CALL     QueryPerformanceCounter
6:      MOV      RCX, [currentCounter] ;MOVSD XMM0, MMWORD PTR
      [currentCounter] could be used instead, if SSE is available
7:      LEA      ESI, [thresholdStep + 8 * EDI]
8:      CMP      [ESI], RCX
9:      JLE      ENDIF
10:     MOV      DWORD PTR [sleepFlag], 1
11:  ENDIF:

```

8.4.5 Results from RT-DHT

RT-DHT has been tested in a simulated environment with a client and a server. Performance of reading and writing from the dSPACE board was measured during 520,000 samples distributed into different scenarios of 8 repetitions, which gave an approximation that a read or write operation took around 0.41 ms (based on mode values, with low spread). Herein, during RT-DHT testing,

¹⁹ A tool to support monitoring software behaviour.

the board data provision was emulated with code. The client implemented the TC, RS and RW threads working as described in Section 8.4.3 and illustrated in Figure 8.1; this Figure depicts threads operation in 10 ms time-step assuming around 6 ms of network and server delay. The server ran as a process, reading from the network continuously, slightly manipulating the received data, adding a small processing delay and sending the data back through the network. No special mechanism was considered to solve missed deadlines, so data arriving out of its time-step was simply discarded. In order to measure time, the QueryPerformanceCounter was used, which provided extra delays that would not exist in a real simulation. All results were kept in memory and computed once the testing was over to avoid additional time distortion.

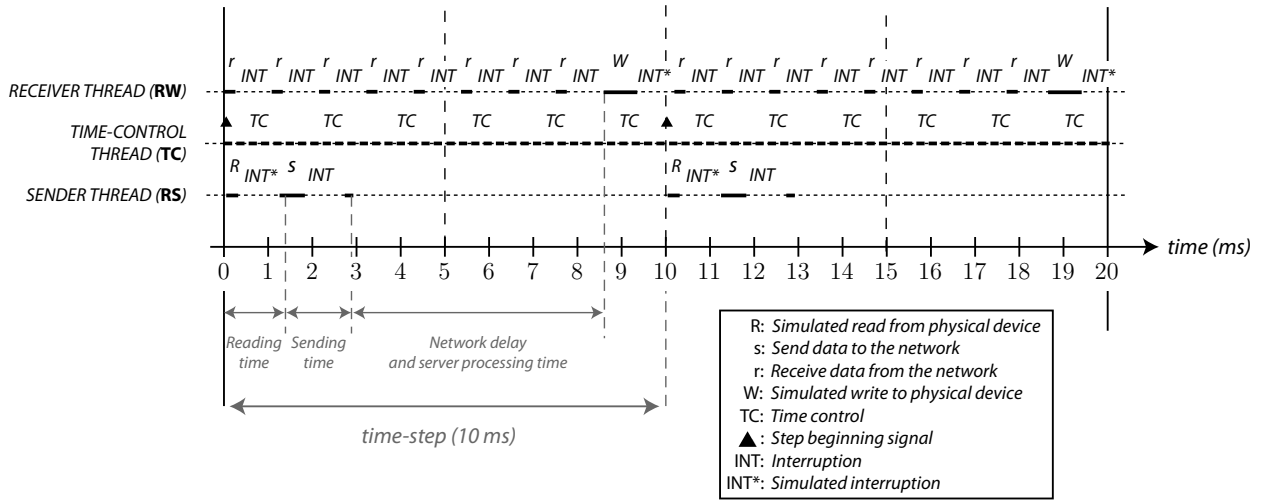


Figure 8.1. Simplified diagram of RT-DHT operation with a 10 ms time-step

Three similar series of testing were conducted, sending 3,000 values at different time-steps of 50, 40, 36, 33, 32, 30, 28, 25, 22, 20, 17, 15, 13, 10, 5 and 3 ms. The first series were conducted between two computers in the same room in an Oxford laboratory, over an isolated network with minimum delay, with no code optimisation (Section 8.4.4) and where the RS thread waited for the RW thread either to finish its task for the time-step or to acknowledge a missed deadline (as described in Section 8.4.3). The second series mirrored this configuration but included two main code variations: (i) code optimisations and (ii) the RS thread assumes the RW thread finishes its task in time, so they work independently. The third series were conducted with a real network environment, connecting computers in Oxford and Kassel, in Germany.

Under these conditions, stable simulations with no single missed deadline were conducted at 20 ms time-step in the first series, and fairly stable simulations could be run down to 15 ms. At 13 ms time-step or shorter, testing showed increasing amount of missed deadlines.

Step Nr:		1	2	3	4	5	6	7	8
50 ms	TC	0	49.9999	99.9999	149.9998	199.9998	249.9997	299.9996	349.9996
	RS [R]	0.2847	50.268	100.2691	150.2712	200.2729	250.2741	300.2755	350.277
	RS [s]	0.3302	50.2755	100.274	150.2751	200.2784	250.2786	300.2801	350.2812
	RW [r]	2.2767	52.2528	102.2859	152.2288	202.2625	252.2715	302.2455	352.2916
	RW [W]	3.1141	53.0977	103.0988	153.1003	203.1026	253.1035	303.1083	353.1073
30 ms	TC	0	30.0001	60.0002	90.0003	120.0003	150.0004	180.0005	210.0006
	RS [R]	0.9156	30.9322	60.9172	90.9001	120.9008	150.9018	180.9028	210.9035
	RS [s]	0.9617	30.9507	60.9341	90.9071	120.9053	150.9066	180.907	210.9074
	RW [r]	2.6582	32.6607	62.6602	92.4671	122.6586	152.6581	182.6572	212.6561
	RW [W]	2.8342	32.8313	62.8142	92.8152	122.8158	152.8171	182.8178	212.8188
20 ms	TC	0	19.9999	39.9997	59.9996	79.9994	99.9993	119.9991	139.999
	RS [R]	0.8682	20.9036	40.9744	60.9052	80.8869	100.8874	120.8881	140.8889
	RS [s]	0.9137	20.923	40.9826	60.9221	80.8908	100.891	120.8917	140.8922
	RW [r]	3.0344	23.0191	43.0148	63.0028	82.961	103.027	123.0293	143.0183
	RW [W]	3.8542	23.8628	43.857	63.8607	83.8404	103.8444	123.8431	143.842
3 ms	TC	0	2.9999	5.9999	8.9998	11.9998	14.9997	17.9997	20.9996
	RS [R]	0.1422	4.1422	7.1582	10.1394	12.1456	16.1429	18.1458	22.1428
	RS [s]	0.2086	4.1594	7.1654	10.1566	12.1625	16.1508	18.153	22.1603
	RW [r]	2.4925	*	*	*	14.2867	*	20.289	*
	RW [W]	*	*	*	*	14.9548	*	20.955	*

Table 8.1. Sample of RT-DHT threads times for different time-steps (* means a missed deadline)

The synchronization of RS and RW threads only proved to lead to small RS delays and additional missed deadlines. Therefore, as explained, the RS thread operation was decoupled from RW in the second series of testing, so the three threads operated independently. With code optimisation and real thread independence, RT-DHT proved to reach its limit at a much smaller time-step, and most of the steps were completed within 4 ms. Stable tests were conducted at 5 ms, and 3 ms time-step was shown to be the performance limit with very unstable results: when RS finished after the first 1.5 ms, missed deadlines went from 1% to 98%. A small sample of 8 step times for 50, 30, 20 and 3 ms is represented in Table 8.1, showing: times at which the TC thread marked the start of the step (TC), times at which RS read from the simulated board (RS [R]), times at which RS sent the data over the network (RS [s]), times at which RW received the data sent by the server (RW [r]) and times at which RW wrote to the simulated board (RW [W]). For example, in the second step of the 50 ms test, the TC thread established the 49.999 ms as the start of the step. The RS thread read from the board at 50.268 and sent the data to the other end at 50.2755. Then RS waited for TC to mark the next step. Data was slightly processed at the

server and the RW thread received the data back at 52.2528 and wrote it to the board at 53.0977.

Main conclusions from this testing are listed below:

- The TC thread marked the steps with great accuracy according to the computer clock, which proved the suitability of the Windows *QueryPerformanceCounter* as a high-resolution performance counter. Duration of time-steps was constant but acknowledged a small time (positive or negative) accumulation in some time-steps, in the order of 100 ns. This is explained by the way the length of time steps is calculated, that rounded the number of counts of the counter frequency (for example, 3,312,734 counts/s were rounded to 3,313 counts/ms). However, this time accumulation is not appreciated in short simulations of a few minutes, and is barely noticeable at some other time rates (being 3 ns per time-step).
- The RS thread was able to finish its task within the first 5 ms in many cases of the first series and remained waiting for the next step to start. However, it sporadically acknowledged some missed deadlines. In the second series, the RS thread completed its task within the first 2 ms in most of the cases. Missed deadlines were isolated to the performance limit of the program and only acknowledged by the RW thread. Most deadlines were already missed at the receiving operation (RW [r]), so both RW operations acknowledged the same missed deadlines in the majority of the cases.
- Both optimisations and thread operation independence resulted crucial to push RT-DHT to its performance limit and make simulations more stable and predictable.
- In early testing, the RT-DHT threads voluntarily left the processor by means of a *sleep* function when their task was completed for the time slot and the current time was within a safe time threshold, and asked Windows to be woken up for the next time step. Tests revealed that the Windows *sleep* function is not totally accurate and Windows can wake threads up not only a few milliseconds after the requested sleep time - which was expected, but, occasionally, also before the requested time. Herein, using the sleep function is discouraged and it is suggested instead to keep threads running continuously if enough processors are available. In the cases where processors have to be shared between the threads, the sleep function can be used with the amount of 0 ms, meaning that the

thread will leave the processor voluntarily before its quantum is complete, to allow other threads with similar priority to run.

- Tests confirmed that user threads can preempt kernel threads (what matters is the priority within the IRQ).

The third series of testing used a real network environment with the University of Oxford as client and the University of Kassel as server. There were 15 network devices in between²⁰, belonging to four main networks where the data passed through: the institutional network of the University of Oxford, the research network Janet (in UK), the pan-European GÉANT network, the educational network DFN (in Germany) and finally the institutional network of the University of Kassel. The testing took place at 1 pm (UK), when networks are normally busy. The RT-DHT code used was the same as in the second series, with optimisations and thread operation independence. Some conclusions are summarised below:

- The network delay between the two hosts was approximately 28.5 ms including data processing in the server. Testing at time rates lower than 30 ms was not feasible.
- Stable tests were conducted at 32 ms, providing the initial conditions of the simulation environment. Acceptable tests were conducted at 30 ms.
- Even with large 50 and 40 ms time-steps the testing acknowledged missed deadlines. This was found to be due to network instability and possibly missed UDP packets in both directions (either RS [s] or RW [r]) rather than to timing reasons as in the first and second series of testing. Despite this issue, UDP proved to be a viable alternative to TCP for firm real-time in simulations involving European institutions with well-connected computer networks, and hence is applicable to other environments with similar conditions.

Figure 8.2 shows the percentage of missed deadlines for the three series of testing (Oxford local, Oxford local with optimisations and Oxford-Kassel respectively) at each time-step. Lines move dramatically to 100% missed deadlines once RT-DHT reaches its performance limit within its network conditions. The smoother transition to 100% missed deadlines in the first series of testing

²⁰ This was the “usual” route. Alternative paths can route the packets due to congestion and other reasons.

(represented by *Oxford Local* in the Figure) shows a more unpredictable behaviour of the program as well as possibility of optimisation.

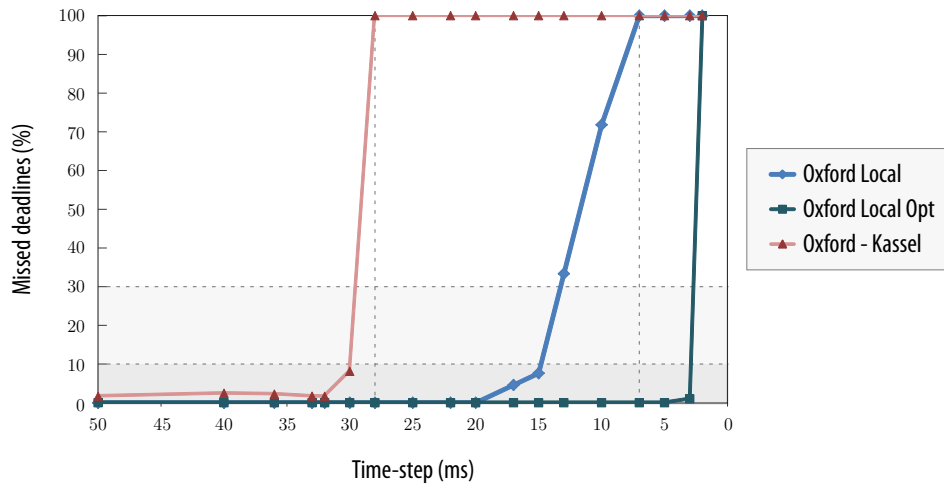


Figure 8.2. RT-DHT missed deadline percentage for the three series of tests (in a laboratory in Oxford, in a laboratory in Oxford with code modifications and between Oxford and Kassel)

8.5 Security in laboratory integration

During a simulation, the most sensitive information is the data exchanged between participants. Despite exchanged data might look a stream of meaningless bytes, it is feasible to decipher its meaning if someone in the middle eavesdrops the communication. Nevertheless, confidentiality of the information is not normally a big issue in simulations, where integrity of the data is more relevant if someone modifies exchanged values maliciously - the same problem exists if network errors happen, since UDP implementations rarely verify data integrity.

Both confidentiality and integrity of the exchanged data can be solved by using public key cryptography (see Appendix A; note that safeguard of values commanded to device controllers must be implemented additionally). However, cryptographic algorithms are computationally demanding in terms of processing time, leading to additional delays and, most likely, impossibility to satisfy tight time deadlines.

Hence, it is recommended not to use security during the data exchange of a test if laboratory integration is required. If security was strictly needed, a simple *XOR* operation could be done to encrypt data (using a pre-agreed key that is shared via public cryptography) for data confidentiality,

while sending additional values to validate data integrity once the test is finished. Algorithm 8.1 shows a possible implementation. The XOR operation has as advantage that (i) it is executed in one processor clock cycle in most processors and (ii) the same key is used for both encryption and decryption.

Algorithm 8.1. Simulation encryption algorithm

```

1:  GENERATE RANDOM KEY K AND RANDOM MODIFIER M
2:  ENCRYPT K, M = KM' WITH PUBLIC KEY OF DESTINATION
3:  SEND KM' TO DESTINATION
4:  WHILE SIMULATION IS NOT OVER
5:      READ VALUE R FROM Device
6:      STORE R FOR POST-SIMULATION INTEGRITY CHECK
7:      R' = R XOR K
8:      WRITE R' TO Network
9:      K = K + M
10:   READ W' FROM NETWORK
11:   W = W' XOR K
12:   WRITE W TO Device
13:   K = K + M
14: END WHILE

```

8.6 Conclusions

In this chapter, a novel approach to earthquake engineering simulation software for real-time has been presented. Laboratory integration has been defined as the ability to interconnect spatially separated facilities as if they were in the same laboratory, which entails great challenges. The concepts of real-time and extreme performance have been clarified, explaining why it is necessary to conduct simulations in real-time in earthquake engineering. The creation of a real-time application requires orchestration of every element in the system, for which an analysis and a methodology have been discussed, leading to the development of a software called RT-DHT.

The detailed operation of RT-DHT has been discussed, and a methodology for optimising code for extreme performance has been described. This analysis and methodology for real-time simulations has never been approached before with this level of technical detail in the earthquake

engineering field. Three lessons can be extracted from the optimisation methodology discussed in this chapter: (i) optimisations can make a big difference in program behaviour and are required for real-time and extreme performance, (ii) manual low-level optimisations are complex, so the effort of applying them must be justified by the benefit obtained; they should be one of the last optimisation resources to use after the rest of the elements have been optimised. (iii) Manual low-level optimisation should be conducted over code that is highly likely to benefit from them, such as mathematical operations using extensive processor power.

The use of UDP over a stable network between geographically close participants (providing the network has a fairly straight route), and the consideration of predictive algorithms for when data do not arrive in time, has been suggested for laboratory integration. UDP has been proved to be a valid option for firm real-time in regional distributed simulations.

Finally, although security measurement in simulations can act in detriment of the simulation performance, an efficient algorithm to guarantee confidentiality and detect integrity of the data exchanged has been introduced.

Future work for RT-DHT should focus on further development, exploring other real-time mechanisms for distributed simulations and extending and adopting EELTP (Earthquake Engineering Light Transport Protocol) as a standard network protocol. This can be done by developing RT-DHT under *Linux* or as a *Windows Device Driver*, running on privileged or *kernel* mode as the operating system itself. This option has not been explored in this research but is worth investigating in future, since it might allow greater control on the hardware and improve low latency by avoiding context switch overhead between user and kernel modes. Finally, RT-DHT should be validated in a full distributed simulation environment using physical facilities.

Laboratory integration remains challenging. RT-DHT has revealed that regardless of performance improvements conducted on the SCP, there is still a (sometimes physical) limitation to satisfy small time-step deadlines in distributed simulations using computer networks. Future work for real-time distributed simulations should concentrate in two directions: (i) reduction of network delay, by researching more efficient ways to deliver low latency data from one source to a destination and (ii) integration of all host computer activity into the SCP, where optimisations can be conducted.

Chapter 9

Celestina Simulations

In Chapter 7, the importance of dividing simulation activities into two different levels was introduced. These two levels are in charge of different parts of the simulation and have disparate requirements:

- **Low-level simulation activities**, in charge of developing best computational methods to exchange data between two or more simulation participants. They are the essential part of a simulation and dictate *how* data is exchanged. Sometimes, they present challenges in terms of control of systems, especially in those requiring real-time operation between spatially separated participants. These activities have been widely discussed in laboratory integration in Chapter 8.
- **High-level simulation activities**, in charge of supporting organisational tasks such as simulation management and planning. They dictate *what* data are exchanged and facilitate the management and execution of simulations. These activities do not require real-time operation and they entail challenges in terms of integration of services.

Existing software to conduct earthquake engineering distributed simulations (*i.e.*, SCP software such as RT-DHT) does not make such distinction, and it mainly focuses on low-level activities. This is primarily because (i) low-level activities are the essential part of a simulation, whereas high-level activities just support them, and (ii) existing simulation software expects to communicate to its homologous software in the remote location, and no integration of different software has been approached yet.

This has three direct consequences:

- There exist no standards and specifications to organise and plan simulations, and the different systems for distributed simulations lack inter-operability.
- There exists no support for high-level activities. As discussed in Section 7.6.6, organizing and planning distributed tasks entails much more complexity than is involved in a single laboratory, besides the difficulty of tracing errors caused by the distributed environment.
- Balance between flexibility and performance of SCP software is not efficiently addressed. Flexible software can run on different platforms, allows greater degrees of configuration and customisation and is easier to maintain and extend in future. On the other hand, software designed for performance is highly optimised for a specific platform and hardware configuration, permitting very few operational variations. This allows the software to get the maximum from the computational environment it runs on, with the disadvantage of being difficult to reuse and reconfigure for a different type of simulation or experimental hardware.

This chapter discusses a system to address these issues, by means of a high-level framework specification for distributed simulations called **Celestina Simulations**. This system facilitates the execution of the experimental workflow in an organized manner, enables inter-operability by means of integration of services and support organisational tasks in a distributed simulation, by promoting high-added value in experiments for humans.

The Celestina Simulations framework acts in the high-level activities layer, being in charge of all experiment oversight and the external details of the simulation itself, such as simulation planning, network addresses of the simulation participants and compatibility check. It should be platform independent and be easy to extend and maintain. The framework defines the role of a manager that coordinates experimental activities between distributed facilities as depicted in Figure 9.1, while leaving the execution of low-level activities to specialised software, such as RT-DHT.

The chapter is organised as follows: Section 9.1 defines the typical simulation workflow from the initial planning to the result collection. This workflow establishes the operational steps for the coordination of simulations. Section 9.2 discusses the Celestina Simulations framework, under three main groups of services: networking, definition and testing. Section 9.3 presents a possible implementation of the Celestina Simulations framework, which is validated in Section 9.4 by

conducting distributed simulations between the Universities of Oxford and Kassel. A discussion about how to use Celestina Simulations is presented in Section 9.5, which includes potential solutions to be adopted by existing software for simulations. Finally, Section 9.6 presents future work and conclusions.

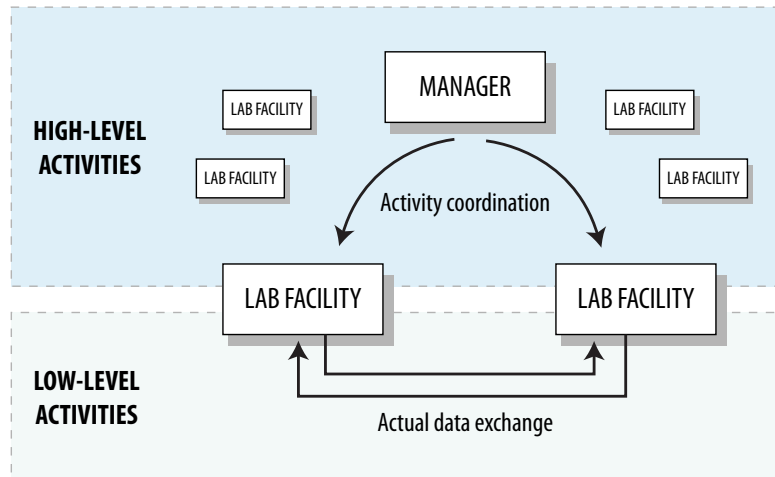


Figure 9.1. High and low-level simulation activities

9.1 Phases of an experiment

An experiment normally goes through a number of clearly separated phases, as depicted in Figure 9.2. In this Figure, the participation of researchers, computers and laboratory facilities is shown for each experiment phase. In most of the stages, human interaction is required. The Figure contains all the elements of a real experiment, but it is an idealization since experiment phases might not occur in such a clear workflow in practice. The description of the phases is given below:

- During the **experiment organization**, participants decide the objective and the resources that will be involved in the experiment. They also have to agree on the data structures to be exchanged during the simulation and to exchange networking information.
- Once resources needs of the experiment are clear, a **resources booking** phase takes place. This guarantees that the resources will be available when the experiment is conducted.

- When all participant laboratories are ready, an **experiment preparation** phase might take place. During this phase, the resources are prepared by the participants and a fairly formal protocol might be followed.
- The simulation is run in the **experiment execution** phase. Sometimes, no user interaction is required or desired at this stage, with the exception of monitoring. At this phase, devices and machines execute the orders given until the simulation is finished. This is the only phase that is uniquely connected to low-level simulation activities.
- Results are collected by the researchers and a **result interpretation** is made once the simulation is over. This can take an arbitrarily long time.
- When the results and conclusions are ready, they can go through the **result storage/sharing** phase. Storing and sharing results allows the re-use of work by others, enables machines to operate on the data and establishes a working methodology, as discussed in previous chapters of this thesis.

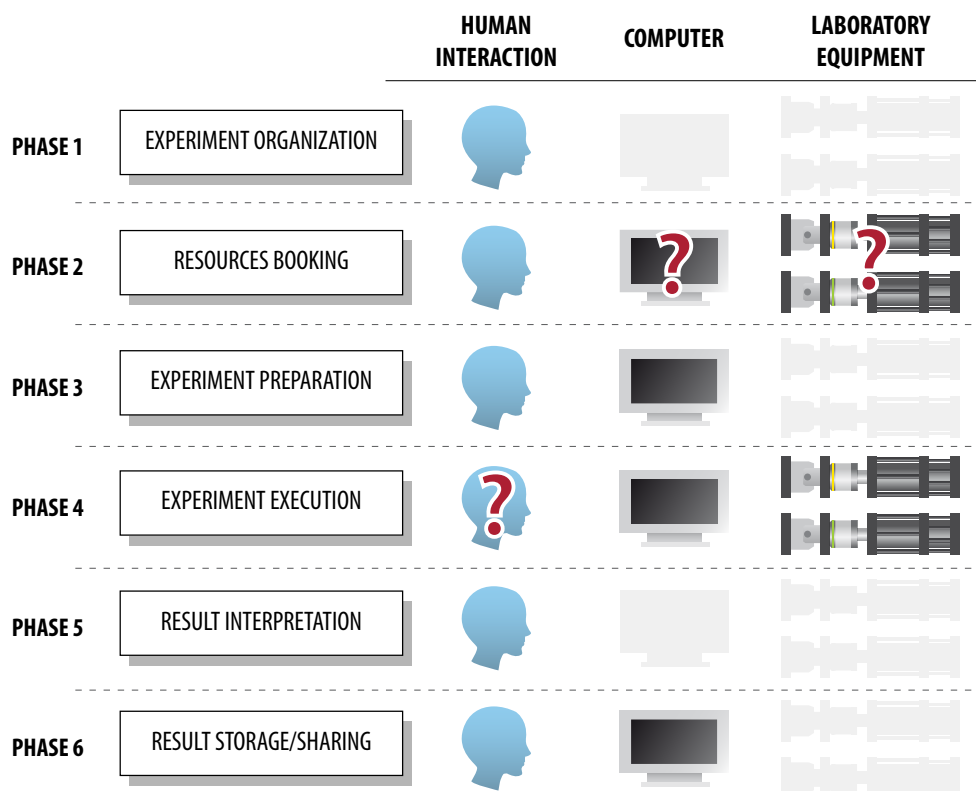


Figure 9.2. Phases of an experiment

A simulation framework for high-level activities must be aware of these phases, which are the base of the simulation workflow. The Celestina Simulations framework is defined considering these phases, and it is described in detail in the following section.

9.2 The Celestina Simulations framework

Celestina Simulations (CS) provides an understanding framework to conduct the experiment workflow, by defining an abstract specification that is platform independent. Herein, CS does not force any specific implementation, which can be independently developed and used under this framework. Likewise, the framework does not restrict to any specific way to conduct the actual data exchange of low-level testing activities.

High-level activities in support of the experiment process include identification and location of participants, compatibility verification of participants, experimental planning and results collection. These have been brought together and systematised in CS by means of a set of services, which are divided into three groups:

- **Networking services**, which deal with operations to locate and discover new simulation participants in the network (also known as nodes), as well as identifying them and the resources they share.
- **Definition services**, which deal with preliminary tasks and compatibility checks to verify the simulation is feasible and can be achieved. It involves connectivity checks, data types compatibility (since CS should work in heterogeneous systems) and experiment plan approval.
- **Testing services**, which deal with the actual simulation execution. This involves communication establishment, control flow transfer to the low-level module conducting the data exchange and results collection.

Figure 9.3 depicts the role of each service. In the figure, laboratory facilities provide services for a simulation coordinator represented as a manager. These two entities are experiment participants, which are called Celestina nodes within the framework.

Networking services do not interfere in any testing activity and they serve for node identification, resources discovery and to know how to contact other Celestina nodes. Actual testing activities

are performed under the Definition and the Testing services.

Definition services advise of the potential success of a simulation, by verifying participant agreement with the simulation that is going to be conducted and by assuring there are no technical barriers preventing the simulation execution.

Testing services support the specification of well-defined orders for what to do in the low-level activities. However, C-S guides the simulation process and defines in a formal way *what* has to be done, but *how* the data exchange is done is dictated by external modules. Since the Testing services define what has to be done in a neutral way, different low-level software (SCP) can be used at each node. This enables inter-operability and integration of simulation services.

However, to obtain full benefit of this inter-operability, SCP software should be partially integrated with CS, to be able to understand the orders received from a manager. If a SCP is not integrated with CS (such as existing software), it can be still used without having many of the advantages of the Celestina Simulations framework - such as compatibility verifications or result collection.

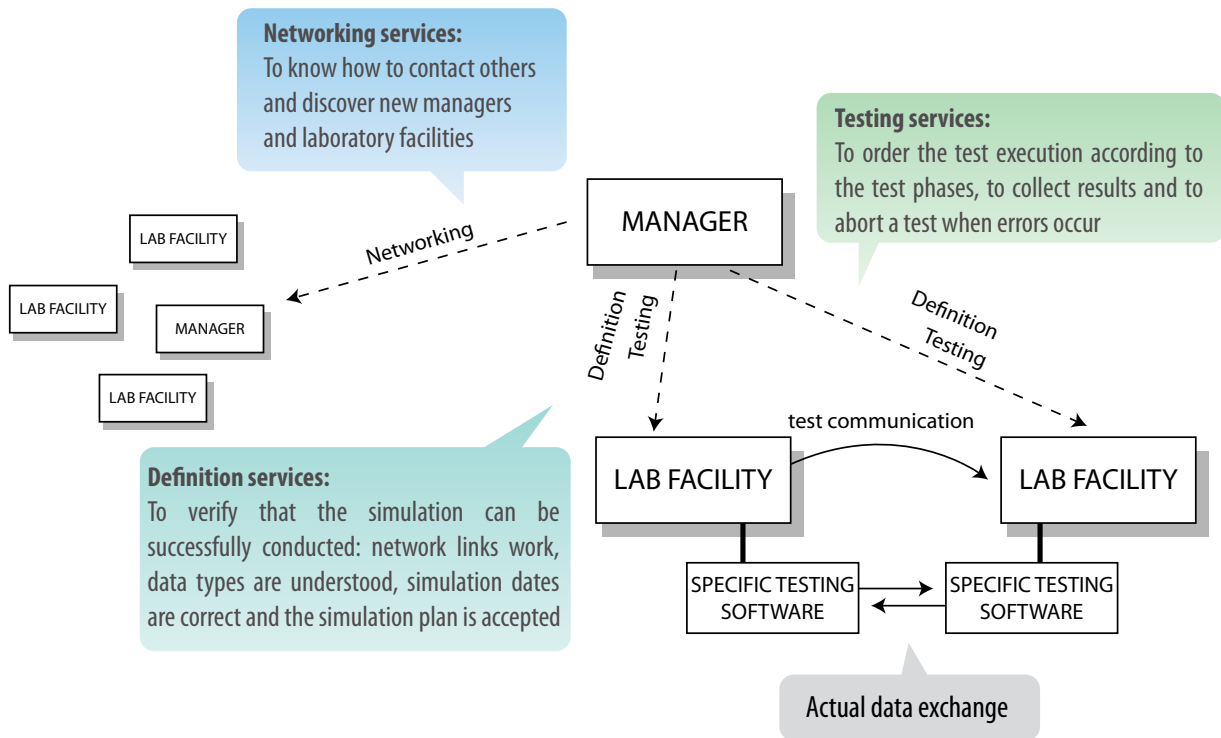


Figure 9.3. Celestina Simulations services overview

The following Sections 9.2.1, 9.2.2 and 9.2.3 describe the Networking, Definition and Testing services separately. Section 9.2.4 provides an overview of how the services relate to each other. Finally, Section 9.2.5 discusses semantic considerations in Celestina Simulations.

9.2.1 Networking

The Celestina Simulations framework is configured in a hybrid peer to peer network (P2P) operated above the Internet infrastructure. A peer to peer network is a network where all the machines within it share some of their hardware resources and where any machine is accessible by any other machine in the network without intermediaries (Schollmeier, 2001). The elements of the network are called nodes or peers. In pure P2P networks, all peers are equal, having the same function within the network. However, as shown in Figure 9.4, the CS hybrid P2P network is divided into **sky nodes**, which are pure managers in charge of controlling the experiment plan, and **ground nodes**, which are in charge of the simulation execution (corresponding to the manager and the lab. facility represented in Figure 9.1). However, note that sky and ground are actually just simply roles, since the whole network can be specified in terms of sky nodes.

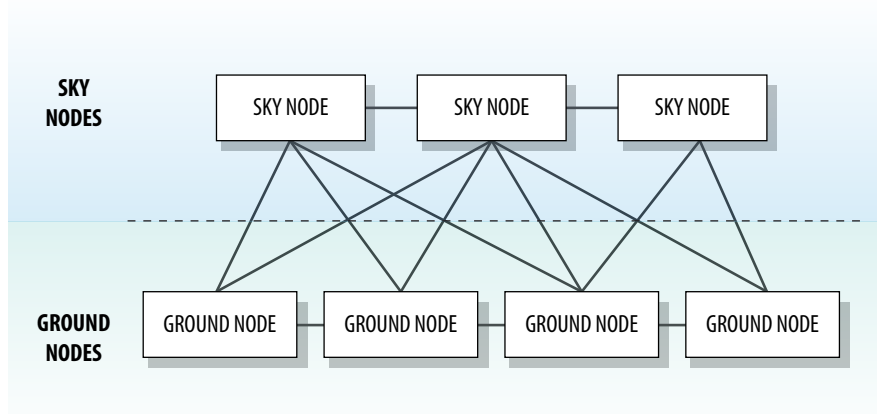


Figure 9.4. P2P Celestina Simulations network divided into sky and ground nodes

The network is unstructured, with no defined topology nor central institution, and is dynamically changeable. Nodes can join in and out at any time, providing robustness to the network: if a sky node is not available, the system can continue its operation, planning and supervising an experiment with any other available known sky node. However, if a ground node is not available, its resources become unavailable too, because ground nodes share hardware resources (normally, laboratory facilities) which can not be duplicated over other nodes. Despite the dynamic aspect of the network, sky nodes are expected to be permanently available whereas ground nodes are expected to be available only when conducting an experiment.

The most basic element of the CS network is the node, represented by a *CelestinaNode* object, which has the following attributes:

- Node ID: the unique identifier of the node.
- Node name: a name to represent the node.
- Node address: service contact information of the node. Basically consisting of an IP address and a port.
- Node description: basic description of the node.
- Resources description: brief description of the resources (experimental facilities) that the node provides. This helps to identify suitable nodes to conduct certain types of simulations but it is not specified in any formal representation.
- Node type: specifies if the node operates as sky or as ground.

Peer discovery is based on lists of known nodes, which are exchanged by using a simple asynchronous *gossip-style* protocol. The justification of this solution emerges from the fact that the network is totally unstructured with no defined node topology. The process is simple: a list of known nodes - also called “friend” nodes, is kept by every node. This list is called the “friendbook”, and is exchanged when two nodes get in contact (which happens during simulation planning or simple availability checking). When two nodes exchange node information, they become “friends”. *Friendbook* maintenance is actually not demanding since the CS network is very specific, with only a few potential nodes. Also, a directory of active sky nodes can be kept in a centralised directory, as discussed in Section 6.2.4.

A *Friendbook* object is a collection of *CelestinaNode* objects. When *friendbooks* are exchanged, complete *CelestinaNode* information is sent, and the receiver must eliminate sender’s known nodes in the exchanged data to avoid sending redundant information over the network. A performance improvement is to exchange only node IDs and provide complete node information if requested. This is only necessary in the unlikely case the CS network becomes very large.

Due to the low number of potential nodes, deletion of nodes from the *friendbook* is not suggested, although it is permitted. Deleting ground nodes from a *friendbook* is not a recommended practice unless the node does not exist any more (ground nodes are expected to be disconnected from

the CS network most of the time). However, sky nodes might be considered to be deleted from a *friendbook* if the sky node remains unavailable after several unsuccessful connection attempts over a period of time. When exchanging a *friendbook*, nodes can work in a proactive way, contacting all unknown nodes that are received, as shown in Code 9.1. This makes visible the node to any other node on its *friendbook*. Herein, proactive nodes are reciprocally known by all their *friendbook* nodes, whereas no-proactive nodes might not be known by their own *friendbooks*.

Code 9.1. Friendbook exchange in a proactive fashion

```

1: RECEIVE received_friendbook
2: SEND SET ( own_friendbook - received_friendbook )
3: FOR EACH UNKNOWN NODE IN received_friendbook
4:     CONTACT unknown_node

```

Proactive nodes enable quick transmission of the whole network topology. The disadvantage of this situation is a possible network overload due to high traffic exchange (problems typically attached to flooding algorithms, where a node tries to contact any other node in the network; Tsoumakos *et al.*, 2003), so it is suggested only sky nodes work proactively.

An example of *friendbook* exchange is depicted in Figure 9.5, where both sky and ground nodes are represented as plain nodes without distinction. Initially, two sub-networks exist in the Figure, and nodes *B*, *C*, *D* and *E* all know each other, probably because either they are all proactive or they performed a simulation together.

Following the Figure, in the other sub-network, node *A* became friend with node *F* and, later on, node *G* became friend with node *A*. By exchanging their *friendbooks*, node *G* got to know node *F*, although because node *G* is not proactive, it did not contact node *F*, who does not know node of *G*'s existence yet. If node *G* had been proactive, it would have contacted node *F* and all nodes in the sub-network would have known each other.

The inclusion of a new node *H* in the network is depicted in Figure 9.5b. The only thing a new node needs to join in the network is to have the contact details of another node in the network (a sky node preferably). Node *H* has exchanged *friendbooks* with node *A* and gets to know nodes *G* and *F*. Intuitively, the next step is the connection of the two sub-networks, for example if node *F* contacts node *B*. If nodes *F* or *B* are proactive, this will lead to many other contacts between

the two sub-networks, making the whole network more robust.

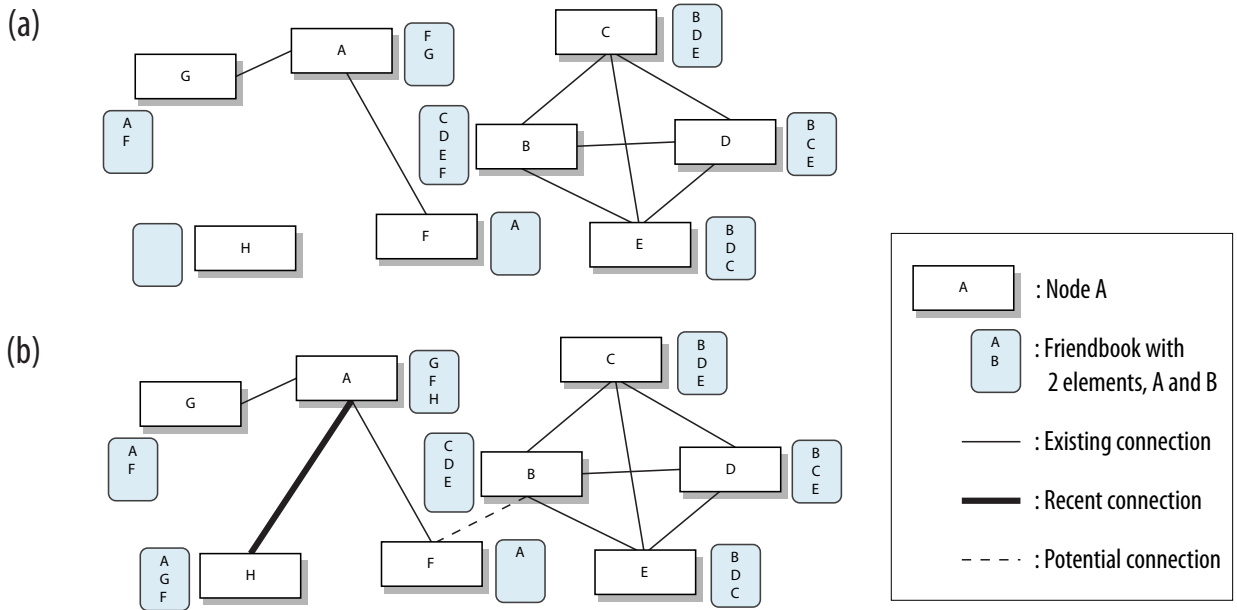


Figure 9.5. Friendbook exchange example (a) Initial situation (b) A new node joins in

Object location has been greatly studied on P2P networks as a way to search and locate shared resources in a network. Much of P2P object location research is based on file-sharing networks, where files can be duplicated and distributed within the nodes of the network. Most of these techniques are not applicable to the CS network: the main shared resources in the network are hardware resources that are inherent to the node and can not be digitally duplicated. Hence, object location is actually linked to peer discovery in the CS network.

The specification for the Networking services is listed below:

- **version()**: returns a string to identify the version of the Networking CS service that is running. This helps to check compatibility.
- **hi()**: returns the status of the node to verify it is alive and running a CS implementation.
- **hello(CelestinaNode)**: this service exchanges only node information of the two nodes getting in contact.
- **beFriend(Friendbook)**: this service exchanges the whole *friendbook* of both nodes getting in contact.

9.2.2 Definition

The definition services aim is to verify that a simulation is feasible between its participants and the simulation terms are accepted by all of them. Most of the existing distributed simulation software (SCPs) assumes that they will communicate with the same software installed in the other end (perhaps in client and server variations). However, Celestina Simulations aims to put in contact different types of software based on the fact that they will pre-agree on the semantics and the way the experiment will be conducted. Herein, a compatibility check between participants is required.

The first verification to take place is that the network link between the participants is working, ideally verifying how long it takes to reach each other. The link verification is commanded by a sky node, but executed from ground node to ground node, with no sky node involved. Some network or configuration errors as well as security restrictions could prevent the communication from taking place. When testing a link, it is important to do it from the lowest network layers to be able to identify the cause of a possible connection error. This is the case of commands such as *ping*, that use network-layer functions to determine if a system is reachable and calculate response times (further information of layers in networks can be found in Appendix D). However, ping command results are only for guidance because of these reasons:

- A destination node or intermediate network devices in the route can have security restrictions that prevents answering a ping request (based on a *ICMP ECHO* request; Postel, 1981a), even if the destination node is available for other services, such as *TCP*.
- Data delivered between two nodes on the Internet might be conducted through different routes each time, depending on network availability and flow control mechanisms.
- The response time for a ping command is only illustrative. Many additional factors affect in the communication between two nodes, and normally the instructions performed for a ping command greatly differ from the instructions actually performed during an experiment.

A second verification is that exchanged data will be correctly interpreted by any simulation participant, which must agree on the semantics of the exchanged data and the steps in which the simulation will be conducted. Although semantics can be agreed from a theoretical point of view, an actual practical attempt is desirable, especially considering the possible heterogeneous

nature of the participants. Different systems might operate with data in different ways, and such an example is the *little endian* and *big endian* byte order¹.

In order to test the data exchange compatibility of two nodes, a real data exchange is conducted in the Definition services. To validate a correct understanding between two nodes, one of the nodes sends two numerical values (preferably, decimal numbers) and an operation that applies to two parameters (such as addition or multiplication). The other node must perform the operation between the numbers and return the result, which is then checked by the first node. If the understanding is correct, both nodes should have come to the same result value.

Endianness, shown in Figure 9.6, can become an understanding problem when systems interpret values in different ways. Even if the described data compatibility check was successful, the experiment plan enables the specification of endianness for the simulation - which must be especially used when the data compatibility check was not conducted by means of the instructions that will actually conduct the data exchange during the real simulation.

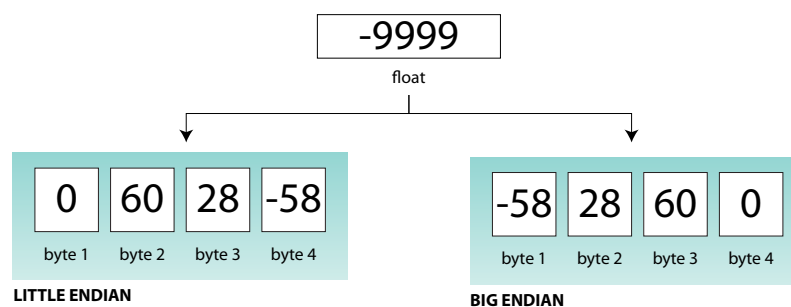


Figure 9.6. Endianness: different ways to store and interpret a four-byte float value

Endianness should not only be detected, rejecting experiments with different participants endianness, but some action should be taken to implement compatibility (for example, forcing to work in *little endian* order, by translating *big endian* values). Managing data **in terms of bytes** is suggested by the CS framework, specifying the interpretation of these bytes separately (*e.g.*, if they are *little* or *big endian*, if they have to be treated in groups of four bytes, if they represent a decimal number, a character, a float number, *etc*).

Finally, a third verification is the agreement of participants on the simulation plan. The simulation plan contains information about nodes that will take part in the simulation, data to

¹ When storing multiple bytes, *big endian* systems store the most significant byte in the lowest address of the memory where *little endian* systems store the least significant byte. An example can be seen in Figure 9.6, where the same float value can be stored in two ways.

be exchanged and the specific steps that will occur during the data exchange, and its attributes are described below:

- Simulation name: formal name of the simulation to be conducted.
- Start and end dates for the experiment. This helps to determine availability of the resources that will be used in the experiment.
- Coordinator: *CelestinaNode* information of the sky node that is in charge of managing the simulation.
- Frequency: time-step intended to be achieved during the data exchange.
- End-of-test or end-value: which is normally helpful to detect the end of the data exchange.
- Experiment type: category of the simulation to be conducted, for example multi-site, peer-to-peer and others.
- Experiment key: a key used to lock and unlock resources.
- List of steps: formal representation of the testing sequence to follow during the simulation. Simulation steps will be translated into testing instructions for each node. Every individual step defines a list of **commands** to be exchanged and the nodes involved in this exchange, indicating whether they act as a client (communication senders) or as a server (communication receivers) for the command. Commands are defined with a common name (such as “force”), a data type and a size in bytes. Data types of commands are defined semantically by using a formal vocabulary. Additionally, the command includes a flag to determine if it should be recorded and sent to another node or only recorded. This might be useful in situations where local signals are wanted to be recorded but they are not needed by any other distributed node. Figure 9.7 illustrates this structure.
- List of initialization steps: similarly to the list of steps, there exists a list of initialization steps. Some simulations might need an initial data exchange prior the simulation starts. Initialization steps are different from the common list of steps because they are carried

out just once and before the simulation data exchange, whereas the common steps run continuously in a loop during the simulation.

- Endianness: type of *endianness* to be used during the data exchange, as explained previously in this section.

A participant can accept or reject a simulation plan based on their capacity to conduct the proposed simulation. Inconvenient dates, an un-achievable frequency, unknown data types or incompatible simulation types are main reasons to reject a simulation plan.

Compatibility verifications of the Definition services do not force a simulation abortion if something fails. Nodes are free to carry on with the simulation even if it was rejected in the definition phase.

The specification for the Definition services is as follows:

- **version()**: returns a string to identify the version of the Definition service that is running. This helps to check compatibility.
- **testLink(list of node addresses)**: performs a link test with every address specified in the list of addresses. It is suggested executing a *ping* command and returning the results from it, but the link can be tested by using any other method.
- **testData(node address, operation)**: performs a data compatibility check. Two numbers are sent to be computed according to the “operation” specified. The service returns if the data compatibility test was successful or not.
- **acceptDataTest(node address, operation)**: accepts a data test. This is the corresponding **testData()** operation on the other end. If the operation specified is understood, the node waits to receive two numbers, which are computed according to “operation” and the result is returned.
- **acceptExperimentPlan(ExperimentPlan)**: receives a simulation plan and an exception is returned if the simulation plan cannot be carried out in terms of dates, data exchange sequence, simulation type, expected time-step or data types.

- **isLittleEndian()**: returns true if the computer interprets bytes in *little endian* order or false otherwise.

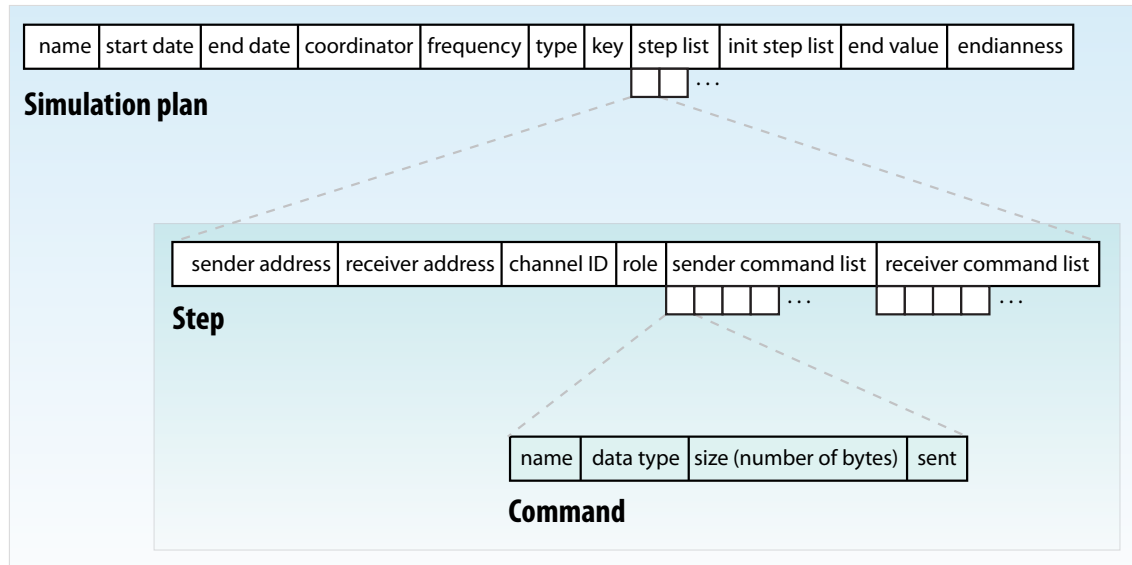


Figure 9.7. The simulation plan structure

9.2.3 Testing

Once the simulation has been designed, the compatibility has been checked and the simulation plan has been approved, the actual data exchange can start. Several sub-phases are required: establishment of the communication between the participants, subsequent preliminary steps before the data exchange (such as doing laboratory procedures to activate hydraulic systems), actual data exchange between the participants according to the simulation plan, and final result collection. In other systems, an implementation of this workflow has been developed by using a traffic light system and permanent communication over a phone line (De la Flor *et al.*, 2010). Celestina Simulations Testing services intend to release researchers from additional phone communication.

To identify the current testing sub-phase, a test state object is used, which has two purposes: firstly, it helps services to keep track of the current state of the node within the simulation workflow and, secondly, it prevents a service collision, especially considering the stateless nature of the services. Otherwise, two services could be randomly executed in sequence to run different experiments at the same time over the same laboratory facilities. Different services update the state object in a different way, from one state to another as depicted in the states diagram of Figure 9.8.

As depicted in the Figure, a node is initially in a state *Available*, meaning that no test is in process and requests to start a test are accepted. When the start of a test is requested by a sky node, a locking-test service is used to block the node and avoid collision with other test requests. A key is used in the locking process, and any subsequent service request must be validated according to the locking key. The node is moved to a *Not Ready* state, where preliminary experimental activities are in progress by calling a service called “prepare”.

Although CS does not force any testing methodology, typically SCPs provide the middleware components depicted in Figure 7.3, so in this state, network communication channels are usually opened and devices are initialised². Since the manager (the sky node in charge of the simulation) needs to have control of the whole communication, the establishment of network connections is done before the actual data exchange takes place. The manager rearranges the data exchange workflow, sending orders to nodes either to accept or to establish network connections from and to other nodes. These connections are opened and remain “in hold” until the order to run the data exchange is given by the manager.

Once all the preliminary tasks have been finished, the node moves to a *Ready* state, indicating it is prepared to run a simulation. When the simulation execution starts via the “run” service, the node moves back to the state *Not Ready*, both to prevent an error from running again the same test at the same time and to prevent running a test without going through the previous preparatory phase.

During the data exchange, nodes should record data to be collected by the manager once the data exchange is over. There are several ways to indicate the data exchange end:

- **By a simulation-end value.** Since CS (or, more specifically, the sky node) has not real control over the data exchange, simulation plans include a pre-agreed end-value to know when a test finishes, as seen in Figure 9.7. This value can be used by SCP software to acknowledge the end of the data exchange and to liberate communication with all participants in an organised, not-abruptly way. Usually, the end value will be delivered at some point by the device hardware of the leading participant (one of the nodes normally leads the communication with the rest of the participants). For loose coupling it is preferred that device hardware acknowledges the end of a test by delivering either 0 bytes when trying

² The words *network* and *devices* will be used for communication with computer network hardware and communication with testing device hardware respectively.

to read from it or by raising an exception. It is also possible that an error happens in any of the participants, having to inform the rest that the test cannot continue. In any case, it is the SCP's responsibility to inform all nodes about the end of the test. The sky node knows the test is finished because the “run” service has finished at all nodes.

- **By a waiting state.** In this case, as soon as the end of the test is captured by the leading node, its “run” service is terminated without informing any other node. Since the leading node is the one triggering the distributed testing loop, the rest of the participants will continue waiting to receive new data from the network, which will never happen. It is then the sky node's responsibility to inform all participants about the end of the test. The same situation can occur if an error happens, where one or more participants will finish the service abruptly, without waiting.

Independently of the participants being informed by the SCP software or not, as soon as the sky node is informed of the end of the test without errors (by one or more participants finishing the “run” service), it will attempt to collect the results with the “collectResults” service. Every ground node is then called by the sky node to provide a list of output signals, with at least one vector of data per each command managed by the node during the simulation. Output signals contain the following information: signal label, attributes (settings), an array of bytes for the signal data, the array data type (to know how to interpret the array of bytes) and the data type size (implicit in the type, a shortcut to know how to distribute the bytes of the array).

The real end of the test is done when the sky node calls the “abort” service. This will liberate all resources, closing network channels and hardware connections, unlocking the test, and setting the node in the *Available* state again. “Abort” must be always called by the sky node to make sure every ground node finishes in a stable state. If an error prevents the sky node from informing all ground nodes (for example, the sky node goes off-line), the aborting process should be started automatically by the ground node after some time of inactivity in the communication with the sky node.

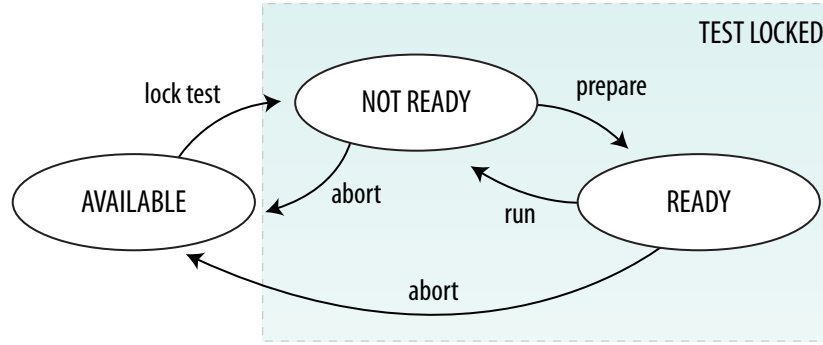


Figure 9.8. State diagram of Celestina.Simulation testing

The Testing services are defined below:

- **version()**: returns a string to identify the version of the Testing service that is running. This helps to check compatibility.
- **lockTest(lockingKey)**: by using a key, the Testing services are locked for a single experiment to avoid collisions. The rest of the services will only operate within the booked experiment.
- **unlockTest(lockingKey)**: unlocks the Testing services to allow new test requests. This service is not normally called, since it is preferred to liberate all resources and unlock the node by using the “abort” service.
- **acceptConnection(lockingKey, node address)**: informs a node that a network connection should be accepted to run the experiment.
- **establishConnection(lockingKey, node address)**: provides the node a network address to connect to.
- **prepare(lockingKey, experiment plan)**: makes all the preliminary tasks prior the data exchange takes place. Normally, it will require a translation of the simulation plan steps into computational instructions that can be processed by the middleware elements.
- **run(lockingKey)**: carries out the data exchange according to the instructions of the simulation plan.

- **collectResults(lockingKey)**: returns a list of output signals, normally one for each command managed by the device hardware connection.
- **abort(lockingKey)**: frees all engaged resources and clear the testing workflow. This service is as well called when something goes wrong in another experiment participant and a global restart is needed.

9.2.4 Services workflow overview

As discussed before, every node in Celestina Simulations communicates to any other node by using the CS specification, which provides an understanding layer and common language for all nodes (as represented in Figure 9.9a), coming to fruition by a set of services. Most of these services (classified into Networking, Definition and Testing) should not be used independently, but in a specific order. This order matches the experiment phases discussed in Section 9.1 and creates a workflow in the way services are sequentially called. For example, the Networking services can be called at any time but they are mainly used before the experiment takes place, to exchange node contact details when the experiment participants are defined.

The workflow tasks and their relationship with the services are depicted in Figure 9.9b:

- The workflow starts by setting the participants of the experiment and the communication channels between them. Sources and destinations of the data flows are defined in these channels. During these tasks, Networking services might be called.
- Once participants and channels have been defined, a link test is conducted. Methods for connectivity test are not forced by the CS specification, although a *ping* command is suggested because of its low-level network characteristics.
- A data test is conducted afterwards, to ensure data compatibility and full system understanding during the actual data exchange.
- And finally, the simulation plan has to be approved. During these three previous tasks, Definition services are called. The order of these tasks in the workflow could be altered,

although common sense suggests doing the connectivity test first, then the data test and finally the simulation plan approval.

- Testing services are called in the next workflow tasks. Locking requests are sent by the sky node to avoid any external interference on the simulation participants.
- Afterwards, orders are sent by the sky node and connections are made according to the previously defined channels. Network connections are opened and remain in hold until the experiment is started.
- Any additional task is done during the preparation workflow stage (“prepare”), which leads to the simulation running (“run”), the result collection (“collectResults”) and the final test liberation (“abort”). All these tasks have to be done in sequence, with no possible order alteration. Therefore, Testing services follow a strict calling order.

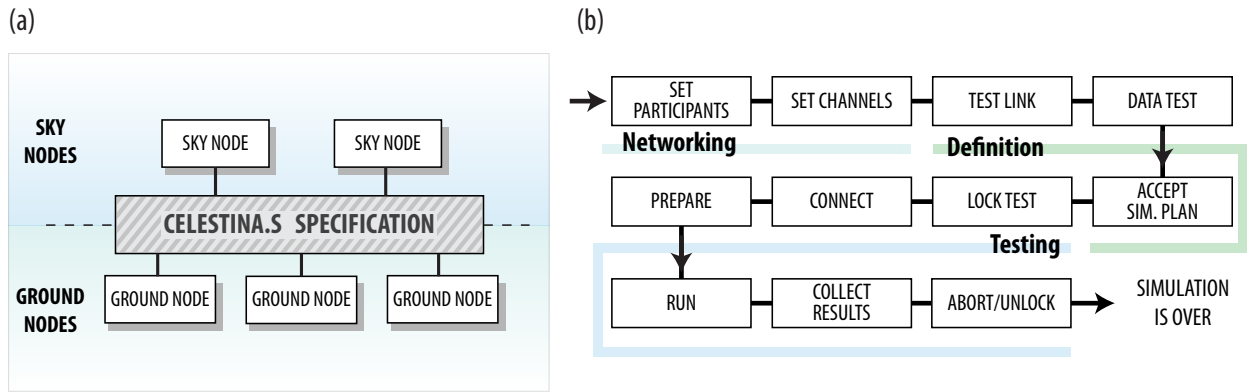


Figure 9.9. (a) Celestina Simulations specification as a common language for nodes (b) Workflow established by the service call sequence

9.2.5 Semantics in Celestina Simulations

Due to the distributed nature and the heterogeneity of the framework, semantics play a key role. When a simulation is planned, participants must pre-agree on the terms the simulation will be conducted, which is particularly important to guarantee understanding between them. Also, during the test, participants exchange bytes of data which need to be correctly interpreted by any of them. For instance, a node could send 4 bytes of data, but they could be interpreted either as

an *integer* or as a *float*, *signed* or *unsigned*, in *little endian* order or in *big endian* order. Such interpretation must be done according to the semantics of the data.

The CS framework encourages the use of some Semantic Web features to specify semantics, such as the use of universal identifiers described in Section 5.3, which is especially useful when different participants describe data of a common simulation. Not all Semantic Web features are strictly necessary or desirable (such as *reasoning*, defined in Appendix C), but the formalisation of an OWL vocabulary in a standard way enables semantics on these contexts:

- Definition of data types for data exchange, in order to provide specific information on how a (typically numerical) value must be interpreted. For example as integer, float, double, *etc.*
- Definition of operations between numbers. During the data compatibility verification of the Definition services, a service is called to verify participant understand each other data types. In this service, two numbers are provided together with an operation, such as addition or multiplication. While these are easily understood concepts, the “operation” should be specified in a formal way to avoid any misunderstanding and to enable machine automatization.
- Definition of simulation types. In the simulation plan, simulations can be defined as *peer-to-peer*, *multisite*, *hybrid*, *numerical*, *etc.* These terms must be defined in a common vocabulary to understand that a *hybrid test* in UK refers to the same concept in another location.
- Definition of *endianness*. This formalises the *endian* representation to allow machine interpretation.

Note that the vocabulary formally defines concepts, but it is always the simulation participant's task to know how to interpret such concepts.

9.3 Implementation

In order to validate the framework, the CS specification has been implemented in functional software and tested in a distributed environment with the University of Kassel, in Germany. A

possible Celestina Simulations implementation is described in this section, and the validation will be discussed in Section 9.4.

The development of this particular implementation has been done under the Web model using the Java programming language. Different implementations have been done for sky and ground nodes. The sky node has been implemented as a standalone application that contains input information for the simulation and manages the communication between the simulation participants accordingly. Ideally, input information should be obtained from a Web interface, which is more user-friendly.

A complete ground node has been implemented, which includes both the high-level layer (Celestina Simulations-related) and the low-level layer (the SCP, responsible for the actual data exchange). The three groups of Celestina Simulations services have been developed and their implementation is discussed in the following sections.

9.3.1 Implementation of Networking services

Networking services make use of two main classes: *CelestinaNode* and *FriendBook*. The *CelestinaNode* class contains all details about a node: a pseudo-random identifier that is assigned to every node in the network, a node identification name, network location information (IP4/6 address and port), type of the node (ground or sky) and textual information to describe the node itself and the experimental resources it provides. The *Friendbook* class is just a collection of *CelestinaNodes*. It also provides all operations to exchange *friendbook* information (working with them as sets), add new friends, *etc.* In order to make all this information persistent, a database is used. Communication with the database is managed by additional data access classes.

9.3.2 Implementation of Definition services

Definition services implement compatibility and verification commands according to the specification. The *testLink* service has been implemented by running a *ping* command, whose answer has not being especially treated and it is directly sent to the sky node. To run the command, a configuration file is used, since *ping* commands are not called the same way on different operating

systems. This provides flexibility but should be used with caution - nothing prevents users from configuring the execution of any other command.

Data compatibility is implemented in the *testData* service for the client and *acceptDataTest* for the server, which are both called by the sky node in charge of the simulation. The ground node acting as server is directly accessed by the client ground node by using *sockets*, with no intermediate parties. In this CS implementation, two operation types are accepted: addition and multiplication. The client chooses two numbers and sends them to the server, which performs the operation on them and sends the result back. The client compares the received and expected result and sends the compatibility conclusion to the sky node. The two numbers the client chooses are either of type *BigDecimal* or *byte*. The latter type is preferred since it is the one used in the data exchange.

The simulation plan is defined as a series of *steps* containing *commands* as defined in Figure 9.7. In this implementation, no special restriction is applied and all simulation plans are accepted by the service.

9.3.3 Implementation of Testing services

Testing services are the most complex of the three groups, and their call flow is depicted in the diagram of Figure 9.10, where a summary of operations that are carried out surrounding the data exchange is shown.

A password is used in the services *lockTest* and *unlockTest* to prevent other nodes from accessing any Testing service (note that a confidential exchange of this password would require public key cryptography). This password is verified by every other Testing service before doing any operation. Concurrent access to locking and unlocking services is not allowed, to prevent concurrent users from locking a test at the same time. The node status is changed to *Not Ready* by the *lockTest* service, as depicted in Figure 9.8.

Network sockets to start the communication are opened by the *acceptConnection* and *establishConnection* services. They are called by the sky node one-at-a-time for each defined network channel: firstly calling the *acceptConnection* service in the server and then the *establishConnection* service in the client. When this happens, client and server connect directly to each other by opening

a socket on the port specified by the sky node. Network connection references for each network channel are organised and stored in a *Stream* object.

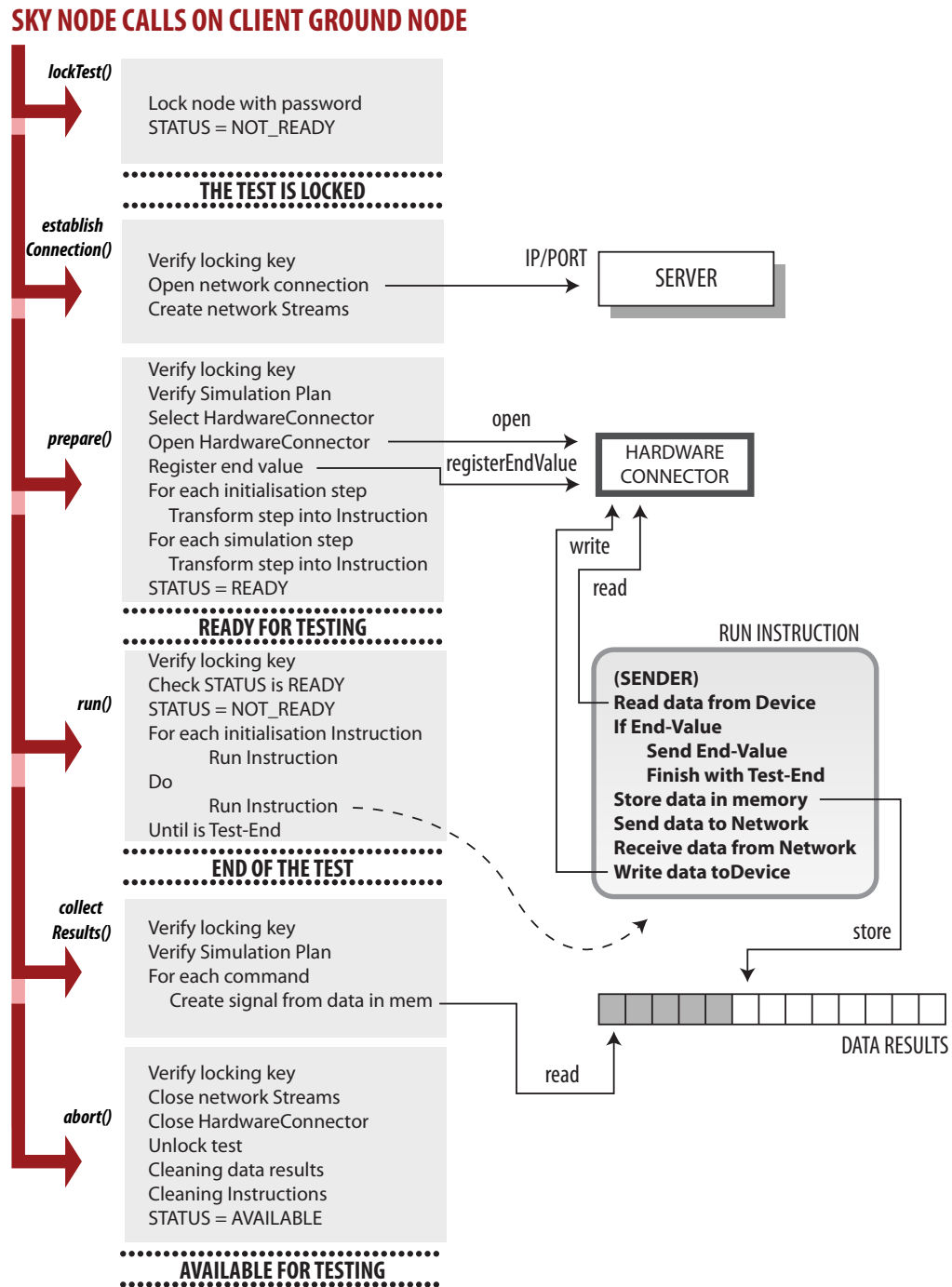


Figure 9.10. Diagram of the Testing services called by the sky node

Once all network channels have been opened by means of sockets, the sky node will call the *prepare* service on each participant, which will create in advance the appropriate data structures in the computer's memory to increase performance during the data exchange loop. These data structures include:

- Channels to communicate with device controllers (or otherwise with device emulators). For flexibility, all device controllers must implement a Java interface³ called *HardwareConnector*, which defines operations to open, read bytes, write bytes and close a hardware device as well as to register a value to acknowledge the end of the data exchange (simulation end-value). This provides a high abstraction level: the type of device that is accessed (a file, the computer memory, an external controller, *etc*) and the way these operations are implemented are completely open, and details to deal with the device are left to the specific classes implementing the *HardwareConnector* interface. The *HardwareConnector* to be used is set in a configuration file, so no re-compilation is needed to alter the node behaviour. Examples of *HardwareConnector* can be found in Sections 9.4.2 and 9.4.3.
- Set of *Instruction* objects. To conduct the simulation loop efficiently (as presented in Algorithm 7.1), each step in the simulation plan is translated into an *Instruction* object. The simulation loop is considered a set of deterministic operations in this CS implementation: read from a device, send data through the network, receive data from the network, write on a device, read from a device... *etc*. Herein, an *Instruction* always consists of a four-operation set (read, send, receive, write: RSRW), and contains all necessary information to run an iteration during the simulation loop: references to access the device input/output, references to access the network input/output, sizes of the data to be read/write and if the node is expected to send or to receive data (which determines the order the RSRW operations are conducted).

The node status is change to *Ready* by *prepare*, which enables execution of the next step in the experimental workflow.

Once simulation data structures have been prepared, the sky node calls *run* in the ground nodes, which is the service triggering the simulation data exchange. The initialisation instructions are executed first, followed by the simulation instructions, which are executed in a loop until an *End-of-test* exception is raised (when the simulation end-value is read) or an error happens. During a data exchange, all data operations work in bytes to provide maximum flexibility. Herein, bytes

³ A Java interface is an abstract definition of methods that should be coded in all classes implementing the interface. This way, it is guaranteed that any class implementing the interface will provide methods to deal with specific functionality.

can be interpreted as integers, decimals, *etc* but the CS implementation deals with simple groups of bytes.

Data read from the hardware are stored in memory sequentially (for performance reasons) for their later collection. However, not all data might be sent: it is possible a node wants to store some data for internal use without sending them to another node. This is indicated in each *Command* of the simulation plan (as seen in Figure 9.7) and translated into the corresponding *Instruction*. For speed reasons, all commands are read as a group from the device (*e.g.*, reading 20 bytes at once instead of 4 bytes in five independent operations for five commands). A restriction in this CS implementation is that those commands that should not be sent on must be placed before the bytes of the commands that have to be transmitted, to improve performance by allowing fast separation of those bytes that have to be sent and those which do not.

When the *End-of-test* exception is raised at any node, the other nodes are informed and the *run* service is finished. The sky node is informed and ground nodes remain waiting for next orders. The sky node then calls *collectResults* at each node, which returns a serialised signal for each *Command* the node dealt with during the data exchange. Signal data vectors are created by iterating over the array of bytes that was stored in memory during the data exchange (containing all data read from the device). These iterations are fairly complex when several commands are involved, since this array is not structured to avoid additional delays during the data exchange, and the stream of bytes has to be traversed to extract and distribute bytes into independent arrays of bytes for each signal, according to step and command sizes.

With all results collected, the sky node calls the *abort* service at every ground node to close all used resources and leave the node in an *Available* state.

9.4 Validation

The implementation described in Section 9.3 has been used to conduct a series of simulations between the Universities of Oxford and Kassel during December 2012. A simplified schema of the environment for this validation can be found in Figure 9.11. The ground node software was installed on both sides whereas the sky node was installed at Oxford. The sky node sent orders to

give instructions on operations to be carried out at each node. The physical network connecting both sites was the *GEANT* European research network.

Objectives of the validation conducted are listed below:

- Testing of general CS services, such as *Friendbook* exchange in the Networking services and link verification and data compatibility in the Definition services.
- Demonstration of flexibility by using different *HardwareConnector* components.
- Verification of correctness in the data exchange conducted.
- Measurement of performance of the data exchange conducted.

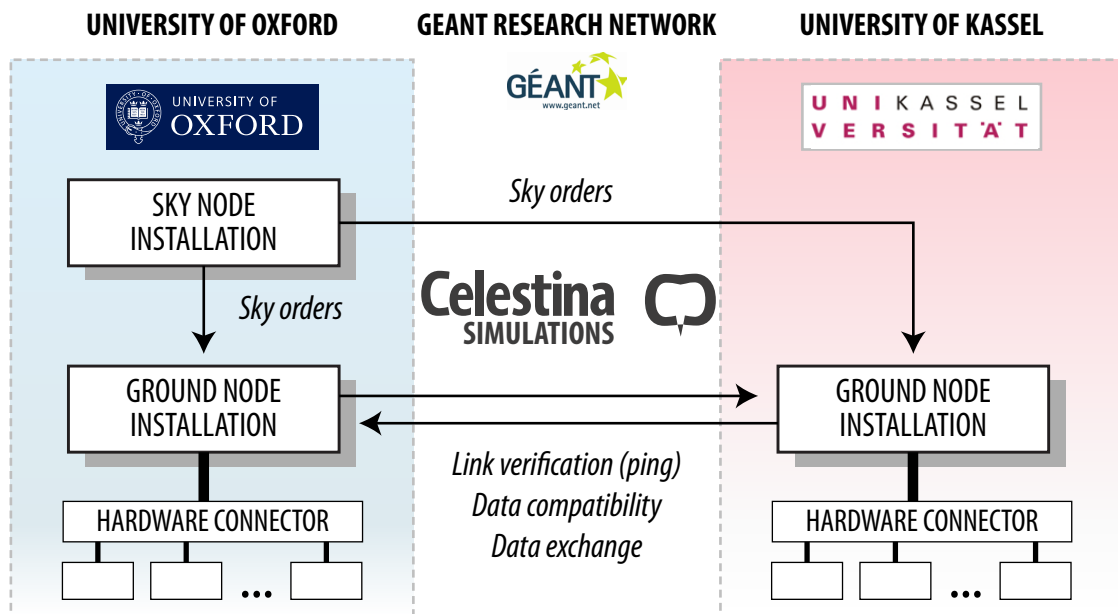


Figure 9.11. Celestina Simulations validation environment between the universities of Oxford and Kassel

Some of these objectives are related to the low-level activities, but the validation had to prove low-level operation feasibility under a CS implementation.

Specific functionality has been tested to validate CS, and the description and results of these testing are presented in the following sections. The first tested functionality was the main CS services, discussed in Section 9.4.1. The use of a very flexible *HardwareConnector* based on files is discussed in Section 9.4.2, whereas Section 9.4.3 presents the use of a more versatile *HardwareConnector* based on TCP connections. The validation of CS as a distributed hybrid simulation tool is discussed on Section 9.4.4, which describes the replication of an earlier hybrid

experiment conducted by other authors. The collection of results is discussed in Section 9.4.5, and the acquisition of data from the *HardwareConnector* that has to be recorded but not send to the other end is discussed in Section 9.4.6. Three methods to acknowledge the end of the data exchange are discussed in Section 9.4.7, and *endianness* issues experienced are discussed in Section 9.4.8. Finally, Section 9.4.9 gives an overview of the performance improvements that were conducted on the CS implementation and the minimal computational overhead that supposed the conduction of the simulation through CS.

9.4.1 Celestina Simulations services operation

Networking services were tested by introducing the Oxford and Kassel nodes, adding new nodes, exchanging *Friendbook*, creating pseudorandom unique IDs, *etc.* These did not always involve the sky node.

Definition services were tested by delivering orders from the Sky node. Data compatibility was verified by sending two numbers from the Oxford node to Kassel and checking the answer. Link test orders were sent to both ground nodes, and *ping* commands from Oxford to Kassel and from Kassel to Oxford were done. The simulation plan was validated for different restrictions such as multi-site and single-site simulations as well as requesting simulations at a specific date.

Testing services were firstly validated with no *HardwareConnector* involved (which was validated later). Locking and unlocking services were successfully implemented, including rejecting requests that did not provide the correct locking key when a node was already locked. Network channels connections were verified and additional network channels were opened on different TCP ports to simulate a multi-site environment. Attempts to run a simulation without having done the preparation phase were successfully rejected. The preparation phase was tested by checking the transformation of different step schemas into *Instructions*, and a basic data exchange through the network was conducted. The *Abort* service was ordered on both ground nodes and all resources were released leaving the nodes in a stable state. This validation required multiple tests that led to the discovery and correction of several errors in the CS implementation.

9.4.2 The HardwareConnector: FileConnector

The most basic of the *HardwareConnector* tested is the file connector, where ground nodes read simulated device commands from a local file and send them to the other side. The other side receives the data, stores them in a file, reads from a local file and sends data back. The file connector purpose is to interact with those devices for which there does not exist a direct library, interface or clear method of connection. In this case, software controlling the testing devices can read and write to files as a way to communicate with the CS implementation.

Accessing files from a drive is a slow operation compared to memory access (especially in mechanically-driven hard disks) and should be only used if no time requirements are needed. Although the file connector is not fast, it is very flexible since every operating system and program is able to deal with files. The use of files has already been used as a solution to communicate different programs with no clear data interface during a distributed test (Ojaghi *et al.*, 2010). During the testing, different simulation plans were used (sending one command of 1 byte and receiving another command of 1 byte, sending three commands of 2, 4 and 8 bytes respectively and receiving two commands of 3 and 6 bytes, *etc*) and this was done by modifying the simulation plan in the sky node, without requiring any code modification in the ground nodes. This proved the flexibility of the code, which was able to deal with any number of commands of arbitrarily any size.

9.4.3 The HardwareConnector: NetworkTCPConnector

A second set of tests involved a network connector which, instead of using a file as intermediary, uses a TCP port for communication. This communication method is found in existing earthquake engineering simulation software (Schellenberg *et al.*, 2010). The *NetworkTCPConnector* can be configured as active (the connector establishes the communication) or passive (the connector waits passively for a connection).

For the testing, an additional piece of software was used, which was developed at the University of Kassel and consisted of two C programs for both client and server. By using the *NetworkTCPConnector*, the C programs communicated with the CS implementation by means of TCP ports.

Testing was done by using different number of commands and data types, float (4 bytes) and double (8 bytes), and only configuration in the sky node was required to accommodate these changes. This proved the flexibility of the CS framework.

9.4.4 Validation of CS as a distributed hybrid simulation tool

To evaluate the use of the CS implementation in a distributed hybrid simulation, tests were conducted that replicated numerically an earlier hybrid experiment by Bayer *et al.* (2005). They were repeated as a purely numerical simulation, substructured between the University of Oxford and the University of Kassel, with the CS implementation in charge of controlling the simulation plan, the data exchange and the results collection. The client and server *C* programs mentioned in Section 9.4.3 implemented the numerical model, and they were programmed by Ferran Obón at the University of Kassel.

In the original experiment, a reference test was performed on a four degrees of freedom system under vertical excitation, representing the third stage separation of the *Arianne IV* rocket launch, depicted in Figure 9.12. The tests have been repeated in the validation as a purely numerical simulation, sub-structured between the University of Oxford and the University of Kassel.



Figure 9.12. Test set-up and model for the real-time substructure tests.

In order to evaluate the impact of Celestina, the numerical hybrid simulated tests were conducted locally (without data exchange) and in a distributed fashion using Celestina with the first two degrees of freedom simulated at the University of Kassel and the payload (the remaining two

degrees of freedom) simulated at the University of Oxford. The results were also compared to those obtained experimentally for validation purposes. As can be seen in Figure 9.13, the use of Celestina Simulations does not influence the results (blue and red curves). Further details of the numerical integration scheme can be found in Lamata Martínez *et al.* (2014).

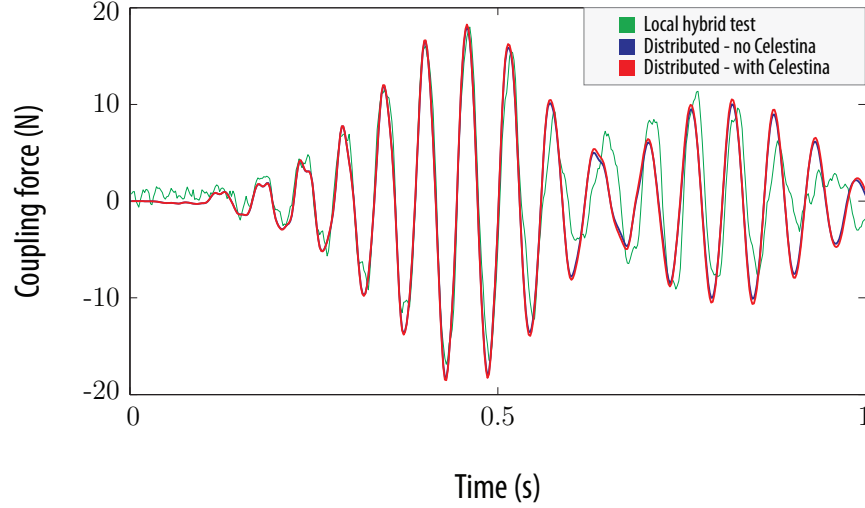


Figure 9.13. Comparison between numerical (with and without Celestina) and experimental results

The simulation is illustrated in Figure 9.14 from the Celestina perspective. Figure 9.14a depicts the steps conducted in the high-level activities: the sky node delivers orders to both client and server sides (at the University of Oxford and the University of Kassel respectively), starting from a network link verification call at the client and finishing with a cleaning and closing call at both client and server. Figure 9.14b presents a more detailed view of the tasks conducted during the execution of the simulation, which belong to the low-level activities and are carried out in the “run simulation” step - the *run* service.

In Figure 9.14b, the interaction between the CS implementation and the *C* programs is explained: **(A)** the client *C* program generated a force that was provided to CS by means of a TCP connection, **(B)** CS sent the data to the CS implementation at the other end by using the Internet, through the *GEANT* network, **(C)** the force was then passed to the server *C* program which ran the numerical model; **(D)** a displacement and another force command were provided to the CS implementation in the server and then **(E)** passed over the network to the client, which **(F)** provided the data to the client *C* program to start the simulation loop again.

During the simulation, the numerical model in the *C* programs communicated with CS by means of the *NetworkTCPConnector*. In a physical experiment, the CS implementation could interact with the physical device directly, by replacing the *NetworkTCPConnector* of Figure 9.14b for another connector able to deal with the device. Thanks to the *HardwareConnector* interface, this is transparent for ground node implementations.

As a result of these simulations, it was found that the CS implementation allowed great degree of flexibility, but performed very poorly in practice. This led to the creation of a more restrictive CS implementation and another *HardwareConnector* which are discussed in Section 9.4.9. The initialisation steps were also tested during the simulations, which led to the creation of zero byte long commands for when no initialisation step is required at only one of the two nodes.

9.4.5 Collection of results

The result collection was tested after conducting several simulations. Output signals were successfully created for each command used at each ground node, and were collected by means of the *resultCollection* service by the sky node.

A potential issue arose when the data exchange was finished with a simulation end-value that was not verified in the CS implementation (for example if the implementation considers that it is exclusively the SCP task to finish the data exchange). In that case, every output signal will contain the end-value as the latest datum of the signal data, which is not correct. Implementations following this approach should remove this latest datum.

9.4.6 Recording data that is not transmitted

Some testing focused on data that is read from the board and must be recorded but must not be sent to any other node. A new command was defined in the simulation plan with the “sent” flag cleared (see Figure 9.7). Simulations were successful and the ground node worked with the specific commands during the data exchange, providing also the additional output signal at the result collection.

9.4.7 The end of the data exchange

The following three methods to finish the data exchange were validated:

- **Managed by the SCP.** The C programs were exclusively in charge of controlling the end of the data exchange, without any CS implication. When the data exchange was over, the C programs closed the communication with the *NetworkTCPConnector*, which ended

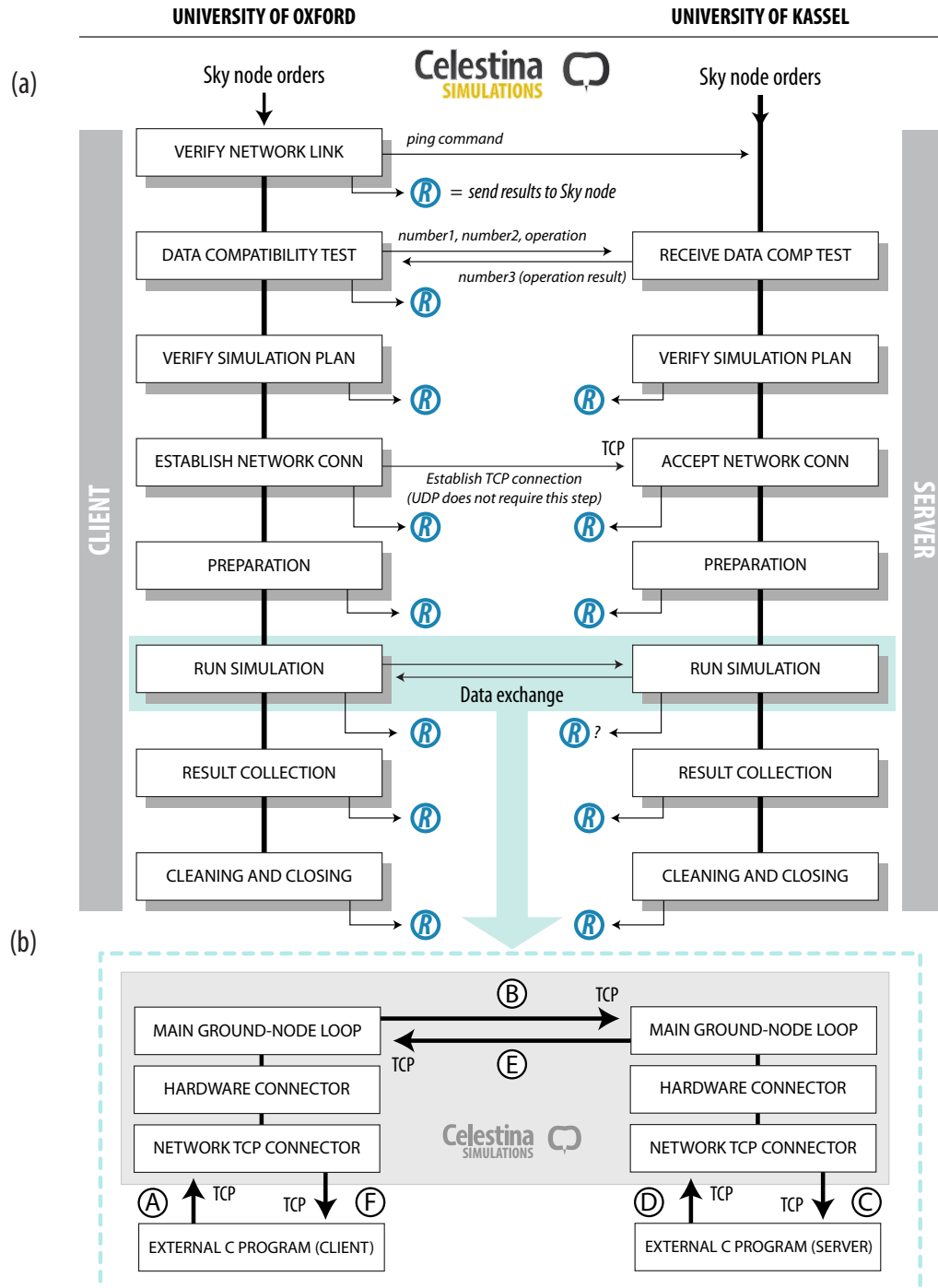


Figure 9.14. Operation of the numerical simulations between Oxford and Kassel using a network connector:

(a) High-level activities (b) Low-level activities

the *run* service informing the sky node. This method is clean and valid but delegates the management of the data exchange end outside CS.

- **Managed by the leader node.** The leader node is the node that acts as the sender that triggers the start of the data exchange. The leader node was the ground node at Oxford, who raised an *End-of-test* exception when 0 bytes were read from the *HardwareConnector*. The simulation loop was then exited, the sky node was informed and the other ground node remained waiting for data to arrive from the network. The sky node collected results from every node and called the *abort* service at each of them, which indicated the end of the data exchange - except for the leader node. This might be misunderstood by a network error if the SCP is not designed to work this way. A similar procedure is followed to finish the data exchange when an uncontrolled error is found by any node before the end of the data exchange is acknowledged.
- **Managed by the CS implementation.** By using the end-value in the simulation plan (see Figure 9.7), the CS implementation controlled when the value was read from the *HardwareConnector*, informing the other end and raising an *End-of-test* exception.

The third method proved to be more organised, providing better control to the CS implementation. However, the first and third method must be designed for multi-site simulations. The second method is not affected by this, since every node is simply ordered to finish by the sky node.

9.4.8 Endianness

Endianness has to be considered to interpret data correctly by the CS implementation - note that this is not necessary if such implementation simply moves data from one end to the other without performing any operation on the data.

For example, when verifying the end-value as explained in Section 9.4.7, both this datum and the data read from the *HardwareConnector* must be understood by the implementation. Data interpretation problems were experienced because Java (used by the CS implementation) worked in *big endian*, whereas the *C* client and server programs worked in *little endian* (so does Linux on the Intel processor used). To address these problems, the Definition service *isLittleEndian*

was created as a mechanism to determine *endianness* of a system, and the CS implementation forced the default operation in *little endian*.

9.4.9 Performance and payload

As a high-level simulation tool, Celestina Simulations has a requirement for flexibility rather than for high performance. However, time measurement were taken to demonstrate the suitability of CS to be used by low-level simulation software (SCPs such as RT-DHT).

Time was measured from the *C* programs of Section 9.4.4, during the execution of the *run* service. Several simulations were conducted during 4,906 steps, exchanging different number of commands of different sizes. Table 9.1 briefly summarises the results. Simulations conducted through the usual *NetworkTCPConnector* performed very poorly, at 188 ms/step. The reason for that is the great degree of flexibility that was allowed in this CS implementation, where simulations could arbitrarily use any number of commands of any size. This proved again the way flexibility compromise performance. Extensive debug logging was done as well, to support error detection.

Such poor performance led to the development of a more restrictive CS implementation using a special network connector called *FastTCPConnector*. Four requirements constrained the implementation: (i) no multi-site allowed, the test must be conducted between only two participants, (ii) there must be only one testing *Instruction* that is repeated during the whole simulation loop (implicitly, it means that one participant always acts as a client and the other as a server and this roles cannot vary during the simulation), (iii) the CS implementation only allows *FastTCPConnector*, and (iv) it is not possible to record data that is not sent, everything that is read from the hardware device must be sent. These four requirements were satisfied by the existing simulation plan. This new implementation proved a dramatic change in performance, at 36 ms/step.

Time was also measured for the direct simulation between the *C* programs, with no CS interaction at all, which demonstrated the small payload that CS implied, as shown in Table 9.1.

	Celestina.Simulations (NetworkTCPConnector with debug)	Celestina.Simulations (FastTCPConnector)	Direct Communication
APPX. AVERAGE (ms/step)	188.23	36.62	36.13

Table 9.1. Time measurement for two CS implementations and without CS

Two aspects must be considered in the way time measurement was conducted:

- The results were measured in the *C* client program using the Linux function “gettimeofday”. The function is non-monotonic (described in Section 8.4.2) and might not be perfectly accurate. No testing was conducted to prove suitability of this function as a high-resolution timer.
- Usual network delay between Oxford and Kassel was measured to be 28.5 ms approximately. However, sporadic increase of speed was experienced on some of the days, when simulations were conducted at around 28 ms/step. Variability in network delays might be due to many factors and these measurements are not considered in Table 9.1, since it is not supplementary to the discussion.

9.5 Using the Celestina Simulations framework

Celestina Simulations acts as a high-level organisational layer over the low-level activities. By defining simulation plans, enabling semantics of data and using specific *HardwareConnector* at each node, integration in the low-level layer (*i.e.*, integration between different SCP), is achieved.

The integration price is that SCP software is prepared to interact with CS implementations, to take full advantage of the framework. Adoption of the CS framework by existing software implies conforming with the CS services: Networking, Definition and Testing. Promoting the adoption of frameworks by a dispersed community of users might require efforts and considerable amount of time.

However, this does not mean that non-adapted software could not be integrated into Celestina Simulations to some extent. A potential solution is discussed in Section 9.5.1, consisting of a ground node implementation that calls an external program. This increases the possible uses of the framework, which are summarised in Section 9.5.2.

9.5.1 Light adoption of Celestina Simulations in existing software

This section explores the possibility of a provisional solution to use the Celestina Simulations framework in existing software for simulations (SCP) that has not adopted the framework. The solution is based on the use of a ground node implementing the CS services and executing an external program when the data exchange is requested. Some CS functionality, such as the data compatibility verification, cannot be implemented if the underlying SCP software does not provide the functionality itself.

This ground node implementation works this way:

- Services to establish network connections do not perform any networking task, and they simply store every node information in memory. This can be used when calling the external program for data exchange.
- The *prepare* service just obtains the command that has to be executed from a configuration file (the SCP execution command). The command can be called with parameters, which can be either specific fixed parameters (*e.g.*, “-c” or “5”) or parameters dynamically parsed by the CS implementation (*e.g.*, whenever the string “%ADDRESS%0” appears, it is replaced by the first node of the simulation. Likewise, “%PORT%2” is replaced by the port number of the third node).
- The *run* service simply executes the command via the operating system.
- The *resultCollection* service needs to work in a different fashion. Since the external program and the ground-node implementation do not share computer memory, an alternative way has to be used to obtain the results. A possible option that has been tested is by means of output signals serialised in an *XML* file: the service reads an *XML* file generated by the external program (according to a formalised *XML Schema*) and translates its data into CS output signals. This option requires some internal adaptation by the external program.

This solution was successfully tested between the universities of Oxford and Kassel by calling an external program and, although it is by no means ideal, it allowed partial use of the CS framework. Networking services worked normally but the data compatibility functionality of the Definition

services was missed. A source of potential problems was related to permissions in the operating system - the ground node implementation must have right access to execute the external program.

9.5.2 Possible uses of the framework

Currently, there are four ways to use Celestina Simulations:

- **By creating a new CS implementation.** This implies the implementation of the CS services in a similar way as it has been presented in Section 9.3 (via *Web Services* in any programming language).
- **By adapting existing software to implement the CS specification,** which makes the software to become part of the CS network automatically and be able to communicate with any other CS implementation.
- **By using an existing CS implementation,** such as any of the implementations discussed in this chapter. Communication with device controllers or numerical models can be done through the *FileConnector*, *NetworkTCPConnector*, *FastTCPConnector* or any bespoke *HardwareConnector*.
- **By using a CS implementation that executes a program externally,** such as the one discussed in Section 9.5.1.

9.6 Conclusions

Distributed testing enables experiment collaboration between different institutions as it implies great challenges. To address these challenges, experimental activities have been analysed in two different groups: high-level and low-level activities. In support of high-level activities, a framework called Celestina Simulations has been developed, which defines three sets of services.

Celestina Simulations's objective is to define a common understanding framework for laboratories conducting distributed earthquake engineering simulations, aiming both services integration and assistance to conduct distributed tasks. The success of Celestina Simulations depends on how

much it is adopted by simulations software, but also on raising the necessity of a common testing communication language for the earthquake engineering community, that can evolve to a future standard.

Celestina Simulations has been implemented and validated by conducting distributed simulations between the Universities of Oxford and Kassel. Such validation demonstrated the feasibility to conduct flexible general-purpose simulations as well as high performance simulations under the framework.

The promotion and adoption of the framework by the earthquake engineering scientific community has been envisaged to be a major challenge. Many Celestina Simulations aspects can be improved and evolved to inspire the path that leads, ideally, to a future standard.

Future work should focus on the following areas:

- Validation of Celestina Simulations in a greater variety of scenarios, such as in real-time and multi-site simulations.
- Promotion of the framework to encourage implementation of Celestina Simulations in existing software, such as RT-DHT.
- Implementation of a user-friendly Web interface for *sky nodes*. This aspect is key to demonstrate direct benefits for researchers and to promote the framework successfully.
- Integration of laboratory equipment inventories to enable resource booking under the Celestina Simulations framework.

Chapter 10

Conclusions

This thesis has discussed novel computing methods to integrate earthquake engineering resources such as experimental data, simulations and laboratory facilities, in order to make them make interoperable in a distributed environment. This integration has required the exploration of a great variety of computing topics.

The importance and requirements for experimental data have been discussed, and two data models have been presented: the *SERIES database*, that uses a traditional approach to store data, and the *Celestina Data ontology*, that uses a more novel approach to store and share data based on Semantic Web technologies. Advantages of the second data model over the first one have been explained.

Distributed systems to integrate experimental data have been discussed. The *SERIES Virtual Database* has been presented, as the integration solution adopted in Europe. It is a novel infrastructure in earthquake engineering using a centralised access to distributed data sources, which has been implemented with a traditional approach based on Web Services. More suitable methods to establish a worldwide system for data integration have been explored, resulting in *Celestina Data*, a Semantic Web based system to enable data interoperability of the hazard mitigation disciplines. With *Celestina Data*, the first data integration using a formalised infrastructure between earthquake engineering institutions from the United States and Europe has been accomplished.

Task forces work, flexible development methodologies and the conduction of proofs of concepts have been crucial for the success of the implementation of these systems, tackled within large projects involving multiple institutions. Social challenges have demonstrated to be as problematic as technical challenges in large projects.

The complexity of component management in a distributed environment and the importance of user-friendly interfaces have been highlighted. Also, both the importance of extensive use of

caching techniques and the importance of flexible and scalable systems in a globally distributed environment have been demonstrated. Public-key cryptography has proved to be crucial to secure communications in an insecure environment.

The foundations of software activities for distributed simulation have been discussed, and a novel approach that divides simulation activities into low-level and high-level activities has been explained. A software to conduct hybrid simulations called *DHT* has been described, and the importance of operate in real-time and explicitly implement appropriate real-time mechanisms has been discussed. This resulted in *RT-DHT*, which optimises *DHT* by using novel computing techniques in earthquake engineering. Joint work from computing and civil engineers has proved to be necessary to efficiently address complex problems. The conduct of real-time distributed simulations remains challenging, and *RT-DHT* has revealed that regardless of performance improvements conducted on the software, there is still a (sometimes physical) limitation to satisfy small time-step deadlines in distributed simulations using computer networks.

A framework to integrate distributed software has been presented, which also highlights the importance of releasing humans from tedious and repetitive tasks where automatized methods can be applied. This has resulted in *Celestina Simulations*, a system that supports the integration of distributed testing activities and facilitates the planning, definition and conduction of the experimental workflow.

Some of the impact of these conclusions is discussed in Section 10.1, global lessons learnt are summarised in Section 10.2, and future work is reviewed in Section 10.3.

10.1 Impact of the research

The impact of this research goes beyond the fact that experimental data can be exchanged between the leading earthquake engineering institutions in Europe and the United States. It has opened a method to make worldwide data interoperability in earthquake engineering, which means that institutions around the world can share data efficiently, but also that data requirements can be analysed globally for the whole community, and policies of data preservation, data curation and authorship and citation of data can be jointly addressed.

This integration enables more transparency and accuracy, because experimental data can be reviewed and evaluated more transparently, especially if the effort is extended to journals in the publishing industry.

The integration of distributed simulation software, by enabling loosely coupled, modular software, permits the use of heterogeneous distributed simulation software that works more efficiently at each participant. Implications of this are that software that is able to use rare hardware facilities at one laboratory is enabled to communicate with more general software at other facilities, without requiring the redevelopment of complicated hardware drivers in the general software. This also implies less systems to maintain.

Integration of data and services facilitates the more efficient management of efforts and funding within the whole community in order to address common problems. Herein, two institutions that have integrated systems might not need to invest time and money on the same system task, each institution can focus on a different task and both can benefit from the results. Moreover, the resulting output can be automatically adopted by all institutions that are part of this system integration.

The aim of this work is to facilitate and improve collaborations in earthquake engineering. The integration of scientific resources can generate great benefits for the whole community, and it can entail a massive impact in the way collaborations, management of funding and publishing of research outcome currently work. However, although many technical challenges are addressed in this thesis, social issues naturally intrinsic to human collaboration will always remain challenging.

10.2 Global lessons learnt

This thesis has presented specific results for the topic discussed. However, some of the lessons learnt from the work described can be applied in other similar situations. A summary of them is presented below:

- The importance of user-friendly interfaces should not be underestimated. Humans should be provided with appropriate systems to interact with technical infrastructures and to support automation of tedious tasks. Failing to do so might result in systems that are not used in practice.

- Some problems might look simple in principle, while they hide a high complexity. This is the case of the RT-DHT program, where an apparently simple exchange of data resulted in great complexity. In this situation, it is important to understand the exact behaviour of the elements involved, and to seek for appropriate expertise from other fields that might address the problem more efficiently. For instance, distributed simulations require close collaboration between civil and computing engineers.
- Novel technologies should not be used without an appropriate motivation and a practical application that reports benefits to the users. Even in cases where a new technology provides clear technical advantages, if this cannot be transmitted to the users in terms of a practical application, it might result in a waste of resources.
- Considering there exists a motivation, the exploration of alternative technologies can report many benefits, such as using Semantic Web technologies instead of the relational model, or the UDP network protocol instead of TCP, as described in this thesis.
- International large projects are normally complex to manage. The use of small task forces, flexible methodologies and proofs of concept can be of great help.
- Management of distributed, autonomous systems should be addressed appropriately. Tracing errors in distributed components can be very complex and time consuming. When distributed systems are accessed remotely, a mechanism should be enabled to automate the reporting of errors to the relevant institution.

10.3 Future work

The work presented in this thesis should be continued in the following aspects:

- Further development on the Celestina platform, including organisational aspects such as the *Celestina Data Consortium*. Celestina Data should be extended to the other SERIES institutions as well as to other earthquake engineering institutions in Japan, China, South America and other parts of the world. Agreements with the publishing industry should be tackled and the expansion to other hazard mitigation disciplines should be pursued.

- Further development on the Celestina Data data model, especially to provide existing unstructured data with more structure. There exists a great source of data described in natural language that machines cannot efficiently process. Data structuring will facilitate the automated extraction and management of data, and greater computer support to produce useful information to support smarter decision making.
- Further development on Celestina Tools, to create new and novel applications on top of Celestina Data.
- User experience in earthquake engineering should be improved. This entails the design or improvement of user-friendly interfaces, including the interface for Celestina Data access, the Celestina Simulations sky node interface, the SERIES central Website (also called the DAP) and the local interface at each SVDB node. The DAP and the SVDB node interface should be probably unified.
- Celestina Simulations should be further developed, including booking capabilities for inventory resources in a laboratory. It should also be promoted to be implemented in existing software for distributed simulations, including RT-DHT.
- RT-DHT should be tested in a real distributed simulation and further improved from research on real-time techniques on the Internet. New real-time mechanisms for RT-DHT should be researched, and the EETLP network protocol should be further developed and adopted by RT-DHT.
- The use of less general and more specialised systems (probably embedded systems) and dedicated networks is worth investigating to address real-time operation in distributed simulations.

Appendix A

Public-key cryptography

The primary idea behind public-key cryptography (also called asymmetric cryptography) is to enable the exchange of secrets in an insecure environment (Diffie *et al.*, 1976). To achieve this, cryptographic systems use trapdoor functions, and the process to encrypt a message requires a different key from the process to decrypt it. This differs from symmetric cryptography, where the same key is used to encrypt and decrypt a message, so the key has to be exchanged in a secure environment before any communication is established (Schneier, 1996).

Having an original message M , and a pair of keys (e, d) representing the encryption key (also called *public key*) and the decryption key (also called *private key*) respectively, there exists an encryption function C that generates an encrypted message M' :

$$C(M, e) = M'$$

Similarly, there exists a decryption function D that reverses the process to obtain the original message M :

$$D(M', d) = M$$

The benefit of this system is that, providing the pair of keys are mathematically bound and that it is not feasible to obtain one key knowing the other in a reasonable time, the encryption key can be shared through an insecure channel. This makes these types of algorithms ideal to implement confidentiality and integrity of information in an insecure, publicly shared media such as the Internet.

Two entities A and B that need to exchange data must be in possession of the other entity's public key. When A sends a message to B , A encrypts the message with B 's public key. The encrypted message can only be decrypted with B 's private key, which only B possesses. This enables A and B to exchange information secretly. Because public-key encryption algorithms might be computationally time consuming, in practice some systems encrypt the message by using a random symmetric key, which is then encrypted via public-key cryptography and attached to the message. Herein, asymmetric encryption is used to solve symmetric *key exchange*, a common process in secure Internet communication.

Another functionality of public-key cryptography is the *digital signature*, which provides a guarantee that some data have not been altered. In this case, A computes a hash function (a function that maps data into a value of fixed size that is ideally unique, in the sense that two different input data never produce the same value) over some data D , which returns a hash value H . H is then encrypted by A with its private key, producing H' . A sends D and H' to B , who can decrypt H' with A 's public key, compute the hash function over D and verify if both hash values coincide.

Public-key cryptography can be implemented by means of different algorithms, based on trap functions from factoring, the discrete logarithm or elliptic curves amongst others (Swenson, 2008). This appendix will briefly explain *RSA*, a well-understood factoring-based algorithm that is one of the most widely used nowadays. Some informal mathematical notions are given in Section A.1. The *RSA* algorithm is described in Section A.2. Note that this is an informal explanation and some details have been omitted for simplification purposes.

A.1 Mathematical background

This section provides a very informal review of some mathematical concepts necessary to understand the RSA algorithm. RSA security resides in the complexity of factorisation of very large prime numbers, as it will be explained in this section.

Two integers a and b are in a *congruent relation* modulus n , written $a \equiv b \pmod{n}$, if a / n and b / n have the same remainder (*i.e.* $a - b$ is multiple of n). For example, $12 \equiv 2 \pmod{10}$. This can be seen as a way to map any integer into the range defined by the modulus (*e.g.* in the previous example, any number maps into a number from 0 to 9).

The *multiplicative inverse* of a number a in modular arithmetic, written a^{-1} , is a number x which, when multiplied by a , gives 1 as a result in the specific modulus: $a^{-1} \equiv x \pmod{n}$, so $a * x \equiv 1 \pmod{n}$. For example, $4^{-1} \equiv 3 \pmod{11}$, since $4 * 3 = 12 \equiv 1 \pmod{11}$. Likewise, $6^{-1} \equiv 16 \pmod{19}$, since $6 * 16 = 96 \equiv 1 \pmod{19}$.

RSA uses modular arithmetic to encrypt a message. Let us suppose the encryption of a number m , by using the number e (the encryption key) in modulus n . The encrypted number, m' , is calculated by multiplying e times the original number m and "hiding" the result by finding its congruent relation in modulus n : $m^e \equiv m' \pmod{n}$. In order to reverse the process, the number d (the decryption key) is required, which is the inverse of e in modulus n : $e * d \equiv 1 \pmod{n}$. Herein, $m'^d \equiv m \pmod{n}$.

It is clear that applying both the encryption and decryption keys, results in the original number: $(m^e)^d \equiv m^{e * d} \equiv m^1 \equiv m \pmod{n}$. This also emphasises that, in fact, mathematically there is no difference in using d for encryption and e for decryption - what one key encrypts, the other decrypts¹.

As far as it has been explained, the security of the system relies on the possibility to obtain the decryption key d , from e and n , or, in other words, the ability to calculate the inverse of e modulus n . Therefore, e and d must be created in some special way that makes the problem difficult to solve.

A prime number is a natural number that only has 1 and itself as positive divisors. The *fundamental theorem of arithmetic* states that every integer is either prime or the product of prime numbers, and there is exactly one single way to factorise a number into prime numbers. Two numbers are *relatively prime*, also called *coprime*, if they do not have common factors except 1. By definition, any two prime numbers are relatively prime. For example, 6 and 25 are not prime ($6 = 3 * 2$ and $25 = 5 * 5$) but they are relatively prime (they do not share common factors).

The complexity of the multiplication of two numbers a and b , is a problem that computers can solve in an acceptable amount of time in practice - even on increasingly large numbers, considering some limit. However, the time required to reverse the operation, meaning obtaining the prime factors that caused the multiplication result (as stated by the fundamental theorem of arithmetic), grows much faster on increasingly large numbers, to the point that a multiplication that took seconds to calculate, can take ages to factorise with currently known algorithms (Swenson, 2008). This constitutes the trapdoor function for RSA, so both keys, e and d , are created in terms of this trapdoor. To do so, *Euler's theorem* is used.

¹ This is indeed how the keys are used in the digital signature previously explained. The private key (decryption key) encrypts the hash value that is decrypted with the public key (encryption key).

Euler's Totient function, written $\phi(n)$, calculates the number of integers between 1 and $n - 1$ that are relatively prime to n . For example, $\phi(6) = 2$ (numbers 1 and 5 are relatively prime to 6). If n is a prime number, then $\phi(n) = n - 1$, because n does not have common factors except 1 with any number between 1 and $n - 1$. If n has exactly two different prime factors p and q , then $\phi(n) = \phi(p * q) = (p - 1) * (q - 1)$.

Calculating $\phi(n)$ when n is the product of known prime numbers, is reduced to a simple multiplication. If those prime numbers are not known, calculating $\phi(n)$ is equivalent to the problem of factorisation, for which no efficient algorithm exist, as stated before.

Euler's theorem relates $\phi()$ with modular arithmetic.

Euler's theorem states that if two numbers m and n are relatively prime, being $m < n$, then $m^{\phi(n)} \equiv 1 \pmod{n}$. For example, considering $m = 2$ and $n = 5$, then $2^{\phi(5)} = 2^4 = 16 \equiv 1 \pmod{5}$. Likewise, with $p = 6$ and $q = 7$, then $6^{\phi(7)} = 46656 \equiv 1 \pmod{7}$.

Due to the way modular arithmetic works, $m^{\phi(n)}$ can be multiplied by itself as many times as desired, and still keep the same congruent relation: $m^{\phi(n)} * m^{\phi(n)} * m^{\phi(n)} * \dots \equiv 1 * 1 * 1 * \dots \pmod{n}$, so multiplying it k times, results in $m^{(k * \phi(n))} \equiv 1 \pmod{n}$.

Following this equation, if both sides of the congruent relation are multiplied by m , the following can be asserted: $m^{(k * \phi(n))} * m \equiv 1 * m \pmod{n}$, so $m^{(k * \phi(n) + 1)} \equiv m \pmod{n}$ (note that we initially assumed that $m < n$). This states that if we have a number m , and the number is multiplied by itself $k * \phi(n) + 1$ times, the same number is obtained in modulus n , which is exactly what it was defined previously in the encryption schema: $m^{e * d} \equiv m \pmod{n}$.

Herein, considering $e * d = k * \phi(n) + 1$, the difficulty to obtain d resides in solving $d = (k * \phi(n) + 1) / e$. As it was explained before, calculating $\phi(n)$ is relatively simple when n is the product of known prime numbers. Note that, since $e * d = k * \phi(n) + 1$, $e * d - 1 = k * \phi(n) \equiv 0 \pmod{\phi(n)}$, so $e * d \equiv 1 \pmod{\phi(n)}$.

RSA uses two prime numbers p and q to obtain n , and calculates $\phi(n)$ which remains secret. The larger p and q are, the more difficult will be for an external system to factorise n to find them, in order to easily resolve $\phi(n)$.

Some of the most efficient algorithms to factorise large numbers into their prime factors are based on the *general number field sieve* algorithm, as discussed by Pomerance (1996). While no more efficient algorithm is found, RSA security still depends on the size of the numbers that have to be factorised. Boneh (1999) has analysed some potential attacks to RSA.

A.2 RSA algorithms

RSA is a cryptographic algorithm that was firstly introduced by Rivest *et al.*, 1978 (actually, it can be seen as two algorithms, one for encryption and another one for digital signatures). The algorithm that obtains the public and private keys is represented in Algorithm A.1. As explained in the previous section, p and q are pseudo-randomly created to obtain n . $\phi(n)$ can be easily calculated by means of a multiplication. e and d are created by satisfying the equation $e * d \equiv 1 \pmod{\phi(n)}$ with some additional restrictions. The public key is constituted by (e, n) and can be freely published, whereas the private key (d, n) is maintained in secret.

Algorithm A.1. Generation of public (E, N) and private (D, N) keys

```
1: CHOOSE LARGE PRIME NUMBER P
2: CHOOSE LARGE PRIME NUMBER Q ( $Q \neq P$ )
3:  $N = P * Q$ 
4:  $T = \phi(N) = (P - 1) * (Q - 1)$ 
5: CHOOSE NUMBER E AND D SUCH THAT  $E * D \equiv 1 \pmod{T}$  AND E, D ARE
   RELATIVELY PRIME TO T AND E IS RELATIVELY PRIME TO D
6: RETURN PUBLIC KEY  $(E, N)$  AND PRIVATE KEY  $(D, N)$ 
```

The process to encrypt a message is shown in Algorithm A.2. The first step is to encode the message into a numeric representation, which can use numbers larger than n , used as the modulus. Once encoded, the encryption process calculates $m^e \equiv m' \pmod{n}$ to obtain the encrypted message. The encrypted message is sent to the destination, protected by the capacity of an external attacker that intercepts the message, to resolve d in a reasonable time by knowing m' , e and n . The decryption follows the inverse process, applies $m'^d \equiv m \pmod{n}$ and reverse the encoding process to obtain the original message.

Note that RSA implicitly only authenticates the receiver, not the sender. To authenticate the sender, the sender has to compute a hash value from the message and encrypt it with the sender's private key. By using the sender's public key, the receiver can decrypt the hash value and verify if it is correct. This process will show the receiver that the message was created by the sender (because it is the only entity that has its private key) and that the message was not altered during its transmission (providing both hash values coincide).

Algorithm A.2. RSA encryption of message M with key (E, N)

```
1: ENCODE M INTO  $M_1 \dots M_k$ 
2: FOR ALL  $M_i$  IN M DO
3:    $M'_i = M_i^E \pmod{N}$ 
4: END FOR
5: RETURN  $M'$ 
```

Appendix B

Computational complexity

This Appendix gives an informal insight of computational complexity and explains the concepts of *decidability* and *tractability*. The concept of *algorithm* is formally defined by the abstract model provided by Turing machines as described in Section B.1. A small introduction of some ways to measure the efficiency of an algorithm is given in Section B.2 and, finally, decidability and tractability are defined in Section B.3. This appendix presents a brief introduction to the subject, and a more complete discussion can be found in Sipser (2012).

B.1 Algorithms

An algorithm is a procedure defined in an ordered number of steps in order to solve a problem. Algorithms can be simulated in what is called a *Turing machine* (TM), which provides an abstract model of computation. Turing machines are defined by (i) a set of **possible states**, where some can be marked as *accept* or *reject* states, (ii) a set of **alphabet symbols** that the machine can read and write, and (iii) a set of **transition rules** that define the behaviour of the machine. This is formally defined by a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

Q : is the finite set of *states*

Σ : is the finite set called the *input alphabet*

Γ : is the finite set called the *tape alphabet* (the input alphabet plus the blank symbol)

δ : is the *transition function*, $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{l, r\}$

q_0 : is the *start state*, $q_0 \in Q$

q_{accept} : is the *accept state*, $q_{\text{accept}} \in Q$

q_{reject} : is the *reject state*, $q_{\text{reject}} \in Q$

Graphically, a Turing machine in execution can be seen as a device that has a control box and a tape as depicted in Figure B.1. The tape contains a symbol in each of its cells, which can be read and written by a head. When the control box reads a symbol, a series of actions happens: the control box might change its current state, it might write some symbol on the tape and it might move the head right or left over the tape to position the head for the next step. The action to take is defined by transition rules that are activated according to the current state and the input symbol read from the tape.

At any time that a Turing machine is running, its current situation is defined by a *configuration*, consisting of (i) the current state of the TM, (ii) the position of the read/write head on the tape, and (iii) the content of the tape.

Based on an initial configuration, the TM keeps reading the tape and operating according to the transition rules until it reaches an *accept* or *reject* state, where the machine halts. A configuration whose current state is the accept state is called an *accepting configuration*, whereas a configuration in the reject state is called a *rejecting configuration*. If there are no possible transitions according to the transition rules, the current state and the symbol read, the TM halts

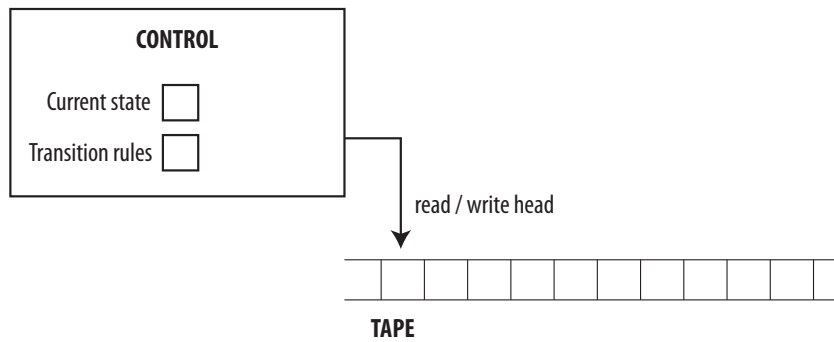


Figure B.1. A Turing Machine representation

and rejects. The definition of a TM presented here is for a *deterministic Turing machine*, and there exist other variants as discussed in Sipser (2012).

Essentially, any algorithm can be simulated in a TM, so Turing machines provide a formal model for the definition of algorithm. The TM presented here will run an algorithm to produce a boolean answer (a *yes* or *no* answer represented by the output *accept* and *reject*). Such Turing machines are also called *deciders* and solve decision problems (those whose answer is *yes* or *no*). Other types of TM solve different problems and therefore produce different type of results. For example, a TM could simulate an algorithm that reverses the digits of a number, so receiving an input “1234” in its tape, it will stop execution with the content “4321” in the tape. The output of this algorithm is not a boolean result, but the content of the tape once the TM stops.

B.2 Complexity of algorithms

Often, the efficiency of an algorithm is measured in terms of time (number of steps before the TM halts) or space (number of tape cells used by the read/write head before the TM halts), and such measurement typically varies according to the size of the input data that is provided to the algorithm.

Considering n as the finite size of the input data (the number of tape cells with symbols required to define the input), an algorithm that performs one operation per symbol in the input will require n operations to finish. If additionally the algorithm loops n times performing n operations per loop, it will require $n^2 + n$ steps to finish. Algorithms that require a number of steps that can be define by a polynomial of this type (e.g. $5n^3 + 2n^2 + n + 4$) are said to execute in *polynomial time*. Polynomial time is represented as n^k , being k the non-negative integer power of the highest-order term of the polynomial (e.g. n^3 , in the previous example). Herein, specific times of algorithms (e.g. $5n^3 + 2n^2 + n + 4$) are estimated asymptotically (e.g. n^3), in order to determine the execution time of an algorithm on increasingly large inputs. On small inputs this estimation might not be accurate, but on large inputs, as the size of the input approaches infinity, the highest-order term of the polynomial prevails over the rest. Also, for the complexity estimation of algorithms, the *worst-case* situation is considered, so the input data that produce the maximum number of steps to finish the execution of the algorithm is used. Polynomial-time algorithms are often considered “efficient”. The class of decision problems that can be solved in polynomial time is called P .

Execution time of algorithms with different efficiency is defined by other functions - such as for example $2^{(n^k)}$, which is called *exponential time*. Note that, as the size of the input data increases, the time that exponential-time algorithms require to finish grows much rapidly than the time required by polynomial-time algorithms.

The study of the complexity of algorithms is important because it provides an estimation on how much time or space will be required in practice to run an algorithm on increasingly large input data. Some algorithms under specific input data sizes might require so much time to finish that their execution is unfeasible in practice.

For example, algorithms to solve *propositional logic satisfiability* (see Appendix C) require exponential time in the worst case (2^n). For a propositional sentence with three propositional letters, 8 possibilities have to be verified in the worst case (*true* and *false* values for each of the three letters). Note that this estimation does not really determine the actual time that is required to execute the algorithm in practice. The input data might not require to explore the worst-case situation, and the time that it takes the system to run each step of the algorithm is bound to the system capabilities. However, time complexity of algorithms will show the behaviour of algorithms as the input data size increases and, by studying the complexity of algorithms, the acceptable size limit for the input data can be determined. For instance, satisfiability of a propositional sentence with 100 propositional letters will take 2^{100} steps in the worst-case, which might take an unacceptable amount of time according to the system requirements.

B.3 Decidability and tractability

A problem is *decidable* if it can be solved algorithmically or, in other words, if a Turing machine can be built that accepts the problem encoded as symbols in its tape together with any possible encoded input data for the problem, for which the TM will always halt and give an answer without looping forever. Note that the definition of decidability considers all possible input data, but an algorithm used on an undecidable problem might complete execution successfully for some particular input data. A decidable problem is just a guarantee that the algorithm used to solve it will always finish for any possible input data.

A problem is *tractable* if it is not only decidable but it can also be solved in practice because solving it does not require an “unacceptable” amount of time or space (such as memory or disk). In terms of time, tractability is linked with the class of problems that can be solved in *polynomial time* in the worst case.

For example, *first-order predicate logic satisfiability* (see Appendix C) is an undecidable problem, meaning that there exists no algorithm or Turing machine that receives a set of first order logic sentences as input and always terminates returning *accept* if the set of sentences is satisfiable or *reject* if it is not.

On the other hand, *propositional logic satisfiability* is decidable, but intractable. There exists an algorithm or a Turing machine that receives a set of propositional logic sentences as input and always terminates returning *accept* if the set of sentences is satisfiable or *reject* if they are not, but the time required to solve the problem is exponential in the worst case (2^n).

Appendix C

Logics for knowledge representation

Logic is sometimes used as a mathematical foundation for knowledge representation, which permits to express knowledge in a formal way and to avoid the ambiguity of natural language. Different logics provide different expressivity (*e.g.* mechanisms for expressing facts, relationships between them and restrictions about which facts can be expressed).

This appendix presents briefly the set of logics called *description logics* and explains some important concepts of knowledge representation with logic. The concept *reasoning* is explained in Section C.1, and typical reasoning tasks are listed in Section C.4. One of the most important reasoning task, *satisfiability*, is explained in Section C.3 for the logic called *propositional logic* described in Section C.2. Propositional logic is rarely used for knowledge representation but it is simple enough to be used in the explanation of reasoning concepts. Finally, description logic is defined in Section C.5, which is the logic formalism on which *OWL* (Web Ontology Language) is based.

C.1 Reasoning

Logical reasoning is the process of acquiring information from some known information. Examples of acquisition of information include new information that was not known, information that was already known (redundant information) or the discovery of contradictory information in the knowledge (affirmation and negation of the same facts). By using reasoning, new information that was not explicitly expressed can be gained and errors within the information can be detected automatically, which otherwise could be too complicated to discover manually.

In general, reasoning can be divided into (i) *deductive reasoning* (obtaining conclusions based on some premises and rules), (ii) *inductive reasoning* (obtaining rules based on some premises and conclusions) and (iii) *abductive reasoning* (obtaining premises based on some rules and conclusions). In this thesis, reasoning refers to deductive reasoning, *e.g.* *all humans are mortal* (rule), *Socrates is human* (premise) *therefore Socrates is mortal* (conclusion).

Reasoning is conducted by means of reasoning algorithms for a specific logic, which in the previous example could have deduced that Socrates is mortal based on the rule that every human is mortal and the premise that Socrates is human. Additionally, if our knowledge contained the premise *Socrates is not mortal*, a reasoning algorithm could detect that there is a contradiction - Socrates cannot be mortal and immortal at the same time.

The following section introduces *propositional logic*, which will be used to explain some typical reasoning tasks.

C.2 Propositional logic

Propositional logic consists of the following elements:

- **Propositions:** sentences that can be true or false and are represented with *propositional letters* (*e.g.* P: The earth is flat, S: Socrates is human).

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
$\top$	$\top$	$\perp$	$\top$	$\top$	$\top$	$\top$
$\top$	$\perp$	$\perp$	$\perp$	$\top$	$\perp$	$\perp$
$\perp$	$\top$	$\top$	$\perp$	$\top$	$\top$	$\perp$
$\perp$	$\perp$	$\top$	$\perp$	$\perp$	$\top$	$\top$

Table C.1. Boolean connectives for propositional letters P and Q

- The symbols for **truth** ($\top$) and **falsehood** ($\perp$).
- Boolean connectives: **negation** ($\neg$), **conjunction** ($\wedge$), **disjunction** ($\vee$), **implication** ($\rightarrow$) and **bidirectional implication** ($\leftrightarrow$), as defined in the truth table of Table C.1.

Propositional formulae can be built with propositional letters, $\top$, $\perp$ and connectives. An example of some propositional formulae is: if it is cold (C), then John is unhappy (U) and John is at home (M). If John is unhappy (U) then it is false that John is happy (H).

$$\begin{aligned} C &\rightarrow U \wedge M \\ U &\rightarrow \neg H \end{aligned}$$

Propositional logic will be used in the following section to explain the concepts of *interpretation*, *model* and *satisfiability*. Extended information on propositional logic can be found in Fitting (2012).

C.3 Interpretations, models and satisfiability

To explain the concepts of interpretation, model and satisfiability, the following problem is used¹:

Let us define a world where people are divided into sane (all their beliefs are true) and mad (all their beliefs are wrong). In this world, Alice believes that both she and Charles are mad. What can we say about Alice, is she sane or mad? What about Charles?

Intuitively, one will look for possible solutions to the problem: supposing Alice is sane, what she believes must be true, but this will imply a contradiction, since she believes she is mad which contradicts the initial assumption that she was sane. Therefore, Alice must be mad. But if she is mad, what she believes must be false, so not to fall into a contradiction, Charles must be sane, because otherwise what she believes would be true.

This mental process is about trying different interpretations to satisfy the problem. An **interpretation** is an assignment of values. The first interpretation attempted was that *Alice is sane* was *true*, which could not satisfy the problem because it led to the contradiction of *Alice is sane* being true and false at the same time. The only interpretation that satisfied the problem was *Alice is sane* is *false* and *Charles is sane* is *true*. When an interpretation satisfies a set of formulae (meaning that it makes them being true without having a contradiction), it is called a **model** of such set. A set of formulae is **satisfiable**, if it has at least one model, meaning that there exists at least one interpretation that makes the set of formulae be true. An unsatisfiable set of formulae is useless because it means there exists no possible assignment that makes the set to be true.

¹ Based on: Smullyan R. M. (1984) *Alice in Puzzle-land*. Penguin Books Australia. ISBN: 978-0140070569.

A	C	$\neg A$	$\neg C$	B ($B = \neg A \wedge \neg C$)	$A \rightarrow B$	$B \rightarrow A$	M ($M = A \leftrightarrow B$)
$\top$	$\top$	$\perp$	$\perp$	$\perp$	$\perp$	$\top$	$\perp$
$\top$	$\perp$	$\perp$	$\top$	$\perp$	$\perp$	$\top$	$\perp$
$\perp$	$\top$	$\top$	$\perp$	$\perp$	$\top$	$\top$	$\top$
$\perp$	$\perp$	$\top$	$\top$	$\top$	$\top$	$\perp$	$\perp$

Table C.2. Truth table for the Alice and Charles problem

The previous problem can be proved in propositional logic with the truth table shown in Table C.2, for the following propositional letters: Alice is sane (A), Charles is sane (C), they both are mad (B) and Alice believes that they both are mad (M). The only interpretation $\mathcal{I}$ that satisfies M (written as $\mathcal{I} \models M$) is when $A^{\mathcal{I}} = \perp$ (false) and $C^{\mathcal{I}} = \top$ (true), which is the assignment that makes $M = \top$ in Table C.2.

Reasoning tasks of propositional logic (also shared in other logics) are listed in the following section. Extended information can be found in Brachman *et al.* (2004)

C.4 Reasoning tasks of propositional logic

Main reasoning tasks of propositional logic are:

- **Satisfiability checking:** determining whether a set of formulae α has a model or not.
- **Validity checking:** determining if every possible interpretation $\mathcal{I}$ is a model of α .
- **Entailment checking:** determining if every model of a formula α is also a model of a formula β .
- **Equivalence checking:** determining if a formula α has exactly the same models as a formula β .

Reasoning algorithms normally focus on satisfiability checking because the rest of the tasks can be reduced to satisfiability: α is valid iff $\neg\alpha$ is unsatisfiable, α entails β iff $\alpha \wedge \neg\beta$ is unsatisfiable and α is equivalent to β if α entails β and β entails α .

Known algorithms to solve the problem of satisfiability checking in propositional logic require exponential time in the worst-case, since there exist 2^n possible interpretations for a formula with n propositional letters (*true* or *false* values for each of the propositional letters in the formula).

C.5 Description logics

Propositional logic is very limited in terms of what can be expressed (*e.g.* it is not possible to make statements about groups of elements, or statements that are valid for all or for some of the elements). This led to the investigation of more expressive logics for knowledge representation, such as *first-order predicate logic*, which has some expressivity features that propositional logic lacks. First-order predicate logic reasoning tasks are the same as propositional logic reasoning tasks. However, a first-order logic formula can have an infinite number of possible interpretations, and satisfiability checking in first-order logic is undecidable (see Appendix B for a definition of

decidability). This has motivated the creation of more restricted logics that have enough expressive power, while avoiding logic features that make reasoning tasks undecidable (Lifschitz *et al.*, 2008).

Description logics are a family of logics based on subsets of first-order logic for which exist decidable reasoning problems (Rudolph, 2011; Baader *et al.*, 2008; Horrocks, 2005). The *signature elements* (non-logical symbols) of description logics are individuals, concepts and roles, as described below:

- **Individuals:** constants in a set (*e.g.* Ignacio, John, Earth, *etc.*).
- **Concepts:** relations of arity one that map individuals to a value *true* or *false* (a unary predicate). They can be seen as sets of individuals (*e.g.* Students, representing the set of individuals that are students).
- **Roles:** relations of arity two that map individuals to a value *true* or *false* (a binary predicate). They can be seen as relationships between individuals (*e.g.* studiesAt, representing the relation between students and their educational institutions).

For example, the individual *John* can be categorised in the concept *Student*:

```
Student(John)
```

And the role *is student of* can relate *John* with his professor *Mary*:

```
isStudentOf(John, Mary)
```

There exist two special concepts, **top** ($\top$) and **bottom** ($\perp$), which represent the set of all individuals, concepts and roles, and the empty set respectively.

The *logical symbols* of description logics are the following:

- Boolean operators: **negation** ($\neg$), **conjunction** ($\sqcap$) and **disjunction** ($\sqcup$).
- **Universal** ($\forall$) and **existential** ($\exists$) quantifiers, to express concepts and roles that affect to all or to some individuals respectively. Informally, if r is a role and C is a concept, $\exists r.C$ is a concept that holds for an individual δ if it is related to an individual δ' that belongs to the concept C by means of the role r (*i.e.* there exists at least one relation r linking δ and δ' , and the latter belongs to the set C). Similarly, $\forall r.C$ is a concept that holds for an individual δ if whenever δ is related to another individual δ' , δ' belongs to the concept C (*i.e.* the only relations of δ via the role r are with individuals belonging to the concept C). Note that the universal quantifier $\forall$ holds for δ if it has no relations via r .
- Concept and role **inclusion** ($\sqsubseteq$), concept and role **equivalence** ($\equiv$), role composition ($\circ$), individual **equality** ($\approx$) and individual **inequality** ($\not\approx$).

Concept expressions can be created by using logical symbols. For example, the concept of persons that are civil engineers and university professors is represented by:

```
CivilEngineer  $\sqcap$  Professor
```

Relationships between concepts can state that earthquakes are a type of hazard:

```
Earthquake  $\sqsubseteq$  Hazard
```

Other complex concepts can be created, such as *cast in place concrete* is concrete that is not precast:

`CastInPlaceConcrete  $\sqsubseteq$  Concrete  $\sqcap$   $\neg$ Precast`

The concept of *hybrid simulation* can be described as something that tests at least one substructure physically and at least one substructure numerically.

`HybridSimulation  $\equiv$   $\exists$ testsPhysically.Substructure  $\sqcap$   $\exists$ testsNumerically.Substructure`

Simulations and *computations* can be defined as the same concept, and the *University of Oxford* and *Oxford University* as the same individual:

`Simulation  $\equiv$  Computation`

`UniversityOfOxford  $\approx$  OxfordUniversity`

Each of the previous examples is called an *axiom*. Extended examples of description logics features can be found in Krötzsch *et al.* (2012b).

There are different description logics, each of them enabled to use a specific set of features (Baader, 2003). A feature or group of features is represented by a letter ($\mathcal{A}$, $\mathcal{E}$, $\mathcal{H}$, $\mathcal{O}$, *etc.*), and these letters name specific description logics. One of the most basic description logics is $\mathcal{ALC}$, whose features include individuals, concepts, roles, negation, conjunction, disjunction and the universal and existential quantifiers. Some other examples are: $\mathcal{S}$, which represents $\mathcal{ALC}$ features plus *transitive roles*, *e.g.* if a beam is made of concrete, and concrete is made of gravel, then the beam is made of gravel; $\mathcal{H}$ represents *role inclusion* (hierarchy of roles); $\mathcal{I}$ represents inverse roles, *e.g.* *has children* is the inverse role of *has parent*; $\mathcal{O}$ represents *nominals*, which enable the definition of concepts based on an enumeration of individuals or based on specific role values, *e.g.* the concept *gender* is defined by the individuals *male* and *female*; $\mathcal{N}$ represents *cardinality restrictions*, that allows to constraint the number of values for a particular role, *e.g.* a large family represents the concept of a family with more than three children (more than three *has child* roles); $\mathcal{Q}$ represents *qualified cardinality restrictions*, which not only allows to constraint the number of values for a particular role but also its specific value, *e.g.* a solid table is a table with at least four legs made of wood.

Since $\mathcal{ALC}$ contains some of the most basic features, it is frequently used to introduce description logics. Its syntax can be defined as follows:

$\mathcal{ALC} ::= \top \mid \perp \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C$

Where C and D are concepts and R is a role, and then, $\top$, $\perp$, $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$ and $\forall R.C$ are also concepts.

Model-theoretic semantics of description logics are given by interpretations, which provides a way to interpret description logic axioms unambiguously. By complying with this semantics, different reasoning algorithms can guarantee that the same logical consequences are extracted, no matter how the algorithm is actually implemented. An interpretation is a 2-tuple $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ where Δ is a non-empty set called the *interpretation domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function* that maps: each individual c to an object $c^{\mathcal{I}} \in \Delta$, each atomic concept C to a set $C^{\mathcal{I}} \subseteq \Delta$, and each atomic role R to a binary relation $R^{\mathcal{I}} \subseteq \Delta \times \Delta$. For example, concepts in $\mathcal{ALC}$ are interpreted in the following way:

$$\begin{aligned}\top &= \Delta^{\mathcal{I}} \\ \perp &= \emptyset\end{aligned}$$

$$\begin{aligned}
(\neg C)^I &= \Delta \setminus C^I \\
(C \sqcap D)^I &= C^I \cap D^I \\
(C \sqcup D)^I &= C^I \cup D^I \\
(\exists R.C)^I &= \{x \in \Delta \mid \exists z \in \Delta \text{ such that } \langle x, z \rangle \in R^I \text{ and } z \in C^I\} \\
(\forall R.C)^I &= \{x \in \Delta \mid \forall z \in \Delta, \langle x, z \rangle \in R^I \text{ implies that } z \in C^I\}
\end{aligned}$$

A description logic knowledge base can be defined as a 2-tuple $\mathcal{KB} = \langle \mathcal{T}, \mathcal{A} \rangle$ where:

- $\mathcal{T}$, also called the *TBox*, contains all terminology axioms, such as concept inclusion, role inclusion, *etc.* For example, the axioms `Earthquake  $\sqsubseteq$  Hazard` and `CastInPlaceConcrete  $\sqsubseteq$  Concrete  $\sqcap$   $\neg$ Precast` belong to the TBox.
- $\mathcal{A}$, also called the *ABox*, contains all assertional axioms, such as concept and role assertions (those that relate concepts and roles with individuals) For example, the axioms `Student(John)` and `isStudentOf(John, Mary)` belong to the ABox.

However, this distinction is purely a way to categorise axioms in a description logic, and has no influence on the semantics of the logic.

Description logics are actively researched and are attractive in knowledge representation for their expressive power and the computational properties of their reasoning problems (which are normally decidable). They are important in this thesis for being closely related to *OWL Direct Semantics*, and normally the different OWL profiles can be semantically related to some description logic (for example, *OWL DL* can be semantically related to *SRQIQ*; Horrocks *et al.*, 2006).

Appendix D

The Internet protocols

Computer networks require protocols for machine communication. Protocols define rules, allowing computers to understand each other.

Due to the complexity of network design, networks are divided into functional layers. Each layer is in charge of some networking functionality and provides services for the adjacent layers. Layers communicate with their equivalent layer on the other side by means of a protocol, as depicted in Figure D.1a. In the Internet, there is no formal layer division, but its protocols can be categorized into the following layers (see Figure D.1b):

- **Application layer**, in charge of managing all final application data. This layer only understands about the final data exchange and usage, but has no conception of the way packets are transmitted from one node to another within the network. For example, the Web is an application that uses application layer protocols such as HTTP.
- **Transport layer**, which normally (but not necessarily) controls the flow of data and provides reliability to the communication. The basic aim of the transport layer is to offer reliability to lower network layers. The access point for an application is established in the concept of ports. A port is a logical channel to establish a communication on the Internet, allowing multiple communication channels over a single physical channel. Main protocols in this layer are TCP and UDP (discussed later).
- **Network layer**, which deals with data routing and computer network addresses. Its objective is to move data packets from a source to a destination. This might require to go through many intermediate network devices, and routing decisions have to be taken with the help of a routing protocol. Main protocols in this layer are IP and ICMP (discussed later).
- **Host to network layer**, which deals with the way the physical network is accessed, conducting data frames from one point of the physical network to the other. One of the most used protocols in this layer is the standard IEEE 802.3 (discussed later).

The core Internet protocols are TCP and IP, commonly known as TCP/IP (Stevens, 1994; Stevens, 1995). It is important to understand how these protocols work to properly design the data exchange on a distributed simulation. A brief description of the Internet protocols is provided below.

D.1 IP: Internet Protocol

IP is the Internet network protocol (Postel, 1981b; Deering, 1998). It enables identification of network devices by means of network addresses and sub-network configurations to route data packets from one source to a destination. Frequently, when two computers exchange data, the data has to be divided into smaller data packets that are delivered to the network. This normally happens if the size of the data is greater than 64 Kb (in practice, around 1500 bytes; Tanenbaum *et al.*, 2010).

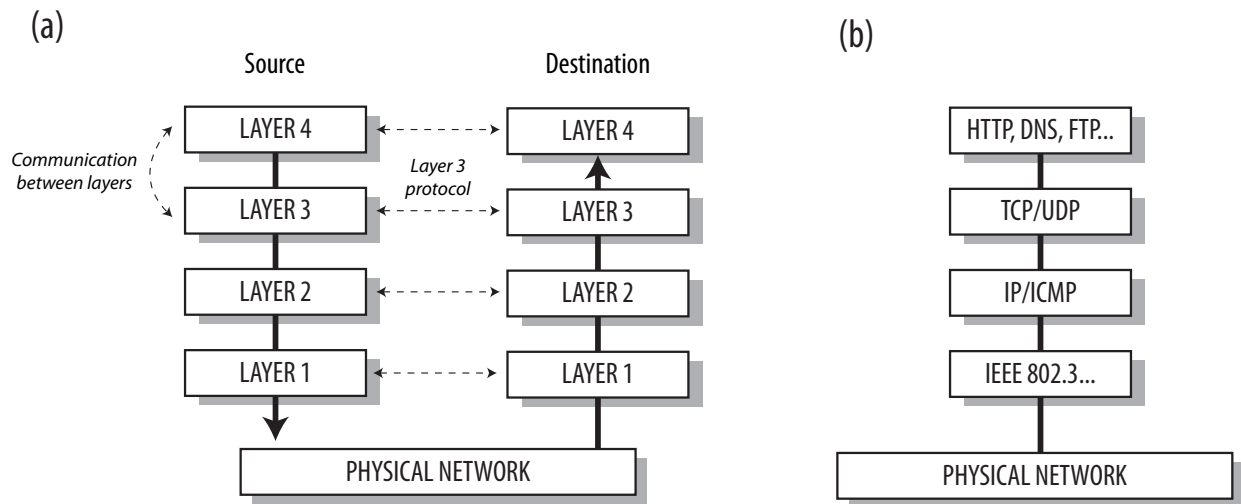


Figure D.1. (a) Generic network layer division (b) The Internet protocol stack

Routers are the devices that control packet routing on the network layer and are in charge of dealing with IP packets. Sometimes, during the routing of packets, new packet division is done by intermediate sub-networks. Once the packets are divided they are not reassembled until they reach the destination. Data packets are independently routed over the network, which means that there is no guarantee that packets will arrive in order or will arrive at all. IP is considered unreliable and does not check data integrity nor notify errors - which are delegated to higher network layers. To fill the gap of error notification, the ICMP protocol is used (Internet Control Message Protocol; Postel, 1981a). ICMP adds some data over IP to provide information such as unreachable destinations, packet life time expirations and echo requests.

IP is supported by additional protocols to translate network addresses into addresses of the layer underneath (Plummer, 1982).

D.2 TCP: Transmission Control Protocol

TCP is the main Internet transport protocol and offers data exchange reliability to IP (Cerf *et al.*, 1974; Postel, 1981c). It provides a connection-oriented service, which means that connections between two nodes are always agreed and consist on three phases: (i) establishment of the communication, (ii) data exchange and (iii) release of the communication. Moreover, TCP is reliable, meaning that TCP reorders all arriving data packages (since they are independently routed) and is in charge of requesting re-sending of a packet if it has been missed or has arrived with integrity errors.

All TCP connections are duplex: data can go both ways at the same time. It is not required TCP sends data as soon as it is provided by the application, and likewise it is not required TCP provides data to the application as soon as it is received, so TCP can use data buffers for sent and received data.

Different control algorithms, such as the *Nagle and Clark* algorithm (Tanenbaum *et al.*, 2010), improve data exchange performance of non-interactive applications, but might not be beneficial for applications requiring low latency. TCP uses several timers to detect lost data flows (and proceed to resend them) and avoid deadlocks where two computers wait endless for each other.

D.3 UDP: User Datagram Protocol

UDP is the alternative transport protocol on the Internet (Postel, 1980), which, contrarily to TCP, is very light and does not guarantee a reliable communication.

UDP is connection-less, so communications between two computers are not pre-agreed. As a unreliable protocol, sent packets might arrive unordered or not arrive at all, although it provides checksum data to detect integrity errors in a packet (which is barely used in practice).

Packet lost is never acknowledged, as in IP. Packets are not divided in UDP, which can have a maximum data size of 64 Kb.

UDP gives priority to latency over reliability, which makes it an ideal protocol for interactive applications sending control data.

D.4 IEEE 802.3

IEEE 802.3 defines a specification to access the physical network (Tanenbaum *et al.*, 2010). A computer on the Internet must use the TCP/IP protocols to communicate with the rest of the nodes, but there is no requirement to use the standard IEEE 802.3 to access the physical network in order to be part of the Internet. However, in practice it is one of the widely protocols used and knowing its operation to improve distributed simulations performance is necessary (especially considering that some of its functioning might affect real-time communication).

Access to the shared physical network in IEEE 802.3 is controlled by CSMA/CD (Carrier Sense Multiple Access with Collision Detection; Tanenbaum *et al.*, 2010). Over the years, IEEE 802.3 has evolved to a switched network with star topology to avoid security and collision problems of the protocol.

Hubs and switches are devices operating at this network layer, and they do not understand about IP network addresses or routing IP packets. Hubs simply connect electrical wires in the same physical media, which means that *all data* transmitted over the network is received by every computer connected, whether the computer is its destination or not. However, switches enable network traffic segmentation, and are able to conduct frames of IEEE 802.3 data only to the right destination.

References

- Abiteboul S., Hull R., Vianu V.** (1994) *Foundations of Databases*. Addison-Wesley. ISBN: 978-0201537710.
- AGILE** (2001) *Principles behind the Agile Manifesto*. <http://agilemanifesto.org/principles.html> (date last accessed: October 2013).
- Arenas M., Bertails A., Prud'hommeaux E., Sequeda J.** (2012) *A Direct Mapping of Relational Data to RDF*. W3C Recommendation. <http://www.w3.org/TR/rdb-direct-mapping/> (date last accessed: October 2013).
- Baader F., Horrocks I., Sattler U.** (2008) *Description Logics*. In van Harmelen F., Lifschitz V., Porter B., eds. Chapter 3 of *Handbook of Knowledge Representation*. Elsevier. ISBN: 978-0444522115.
- Baader F.** (Ed) (2003) *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press. ISBN: 978-0521781763.
- Baran P.** (1964) *On Distributed Communications Networks*. IEEE Transactions on Communication Systems, CS-12 (1), 1-9, Mar 1964.
- Bardet J. P., Liu F., Mekkaram N.** (2010) *A metadata model for collaborative experiments and simulations in earthquake engineering*. Frontiers of Architecture and Civil Engineering in China, June 2010, 4 (2), 133 - 153. Springer. DOI: 10.1007/s11709-010-0036-z.
- Bardet J. P., Liu F., Mekkaram N.** (2004) *A metadata model for the George E. Brown Jr. Network for Earthquake Engineering Simulation*. Report, Civil Engineering Department, University of Southern California.
- Bayer V., Dorka U. E., Füllekrug U., Gschwilm J.** (2005) *On real-time pseudo-dynamic substructure testing: algorithm, numerical and experimental results*. Aero. Sc. and Tech., 9, 223-232.
- Berners-Lee T.** (1999) *Weaving the Web: The Past, Present and Future of the World Wide Web by its Inventor*. Orion Business. ISBN: 978-0752820903.
- Berners-Lee T., Hendler J., Lassila O.** (2001) *The Semantic Web*. Scientific American Magazine, 17th May 2001.
- Berners-Lee T., Fielding R., Masinter L.** (2005) *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. <http://tools.ietf.org/html/rfc3986> (date last accessed: March 2014).

Boneh D. (1999) *Twenty Years of Attacks on the RSA Cryptosystem*. Notices of the American Mathematical Society, 46 (2), 203-213.

Bonnet P. A., Williams M., Blakeborough A. (2008) *Evaluation of numerical time-integration schemes for real-time hybrid testing*. Earthquake Engng. Struct. Dyn. 2008; 37:1467-1490. DOI: 10.1002/eqe.821.

Bonnet P. A., Blakeborough A., Williams M., Taylor C. A. (2008b) *Real time substructuring in the UK*. In Hybrid Simulation: Theory, Implementation and Applications, Saouma V., Sivaselvan M.V. (Eds), Taylor and Francis, New York, 145-154.

Booth D., Haas H., McCabe F., Newcomer E., Champion M., Ferris C., Orchard D. (2004) *Web Services Architecture*. W3C Working Group Note. <http://www.w3.org/TR/ws-arch/> (date last accessed: October 2013).

Bosi A., Pegon P (2009) *The ELSA database and what can be done regarding SERIES networking activities*. JRC56446. EUR Report 23931 EN, Luxembourg (Luxembourg).

Brachman R., Levesque H (2004) *Knowledge Representation and Reasoning*. The Morgan Kaufmann Series in Artificial Intelligence. ISBN: 9781558609327.

Braden R. (1994) *T/TCP -- TCP Extensions for Transactions Functional Specification*. RFC 1644. <http://tools.ietf.org/html/rfc1644> (date last accessed: April 2013).

Brickley D., Guha R. V. (Ed.) (2014) *RDF Schema 1.1*. W3C Recommendation. <http://www.w3.org/TR/rdf-schema/> (date last accessed: March 2014).

Burleson D. K., Celko J., Cook J. P., Gulutzan P. (2003) *Advanced SQL Database Programmer Handbook*. Rampant Techpress. ISBN: 9780984428243.

Burtica R., Mocanu E. M., Andreica M. I., Tapus N. (2012) *Practical application and evaluation of no-SQL databases in Cloud Computing*. Systems Conference (SysCon), 2012 IEEE International.

Calvanese D., Giese M., Haase P., Horrocks I., Hubauer T., Ioannidis Y. E., Jiménez-Ruiz E., Kharlamov E., Killapi H., Klüwer J. W., Koubarakis M., Lamparter S., Möller R., Neuenstadt C., Nordtveit T., Özçep Ö. L., Rodríguez Muro M., Roshchin M., Savo D. F., Schmidt M., Soylu A., Waaler A., Zheleznyakov D. (2013) *Optique: OBDA Solution for Big Data*. ESWC (Satellite Events), pp. 293-295.

Calvanese D., De Giacomo G., Lembo D., Lenzerini M., Poggi A., Rodriguez-Muro M., Rosati R., Ruzzi M., Savo D. F. (2011) *The Mastro System for Ontology-based Data Access*. Semantic Web journal, 2 (1), 2011. IOS Press.

CSS (2014) *Cascading Style Sheets*. <http://www.w3.org/Style/CSS/Overview.en.html> (date last accessed: September 2014).

Celko J. (2004) *Joe Celko's Trees and Hierarchies in SQL for Smarties*. Morgan Kaufmann Publishers. ISBN: 978-1558609204.

Celko J. (1999) *Joe Celko's Data and Databases: Concepts in Practice*. The Morgan Kaufmann Series in Data Management Systems. ISBN: 978-1558604322.

- Cerf V., Kahn, R.** (1974) *A Protocol for Packet Network Interconnection*. IEEE Trans. on Commun., COM-22, 637 - 648, May 1974.
- Chang F., Dean J., Ghemawat S., Hsieh W. C., Wallach D. A., Burrows M., Chandra T., Fikes A., Gruber R. E.** (2006) *Bigtable: A Distributed Storage System for Structured Data*. Google Inc. OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.
- Chinnici R., Moreau J.-J., Ryman A., Weerawarana S.** (2007) *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Recommendation. <http://www.w3.org/TR/wsdl20/> (date last accessed: October 2013).
- Civili C., Console M., Lembo D., Lepore L., Mancini R., Poggi A., Ruzzi M., Santarelli V., Savo D. F.** (2013) *Mastro Studio: a system for Ontology-Based Data Management*. OWLED, 2013.
- Codd E. F.** (1970) *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, 13 (6), June 1970.
- Codd E.F.** (2000) *The Relational Model for Database Management, Version 2*. Addison-Wesley. ISBN: 978-0201141924.
- Cygniak R., Wood D., Lanthaler M.** (2014) *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. <http://www.w3.org/TR/rdf11-concepts/> (date last accessed: September 2014).
- D2RQ** (2013) *The D2RQ Platform: Accessing Relational Databases as Virtual RDF Graphs*. <http://d2rq.org> (date last accessed: October 2013).
- Das S., Sundara S., Cygniak R.** (2012) *R2RML: RDB to RDF Mapping Language*. W3C Recommendation. <http://www.w3.org/TR/r2rml/> (date last accessed: October 2013).
- DATAWEB** (2011) *W3C Data activity*. <http://www.w3.org/2013/data/> (date last accessed: March 2014).
- De Haan L., Koppelaars T.** (2007) *Applied Mathematics for Database Professionals*. Apress; 1st edition (June 19, 2007). ISBN: 978-1590597453.
- De la Flor G., Ojaghi M., Lamata Martínez I., Jirotko M., Williams M., Blakeborough A.** (2010) *Reconfiguring practice: the interdependence of experimental procedure and computing infrastructure in distributed earthquake engineering*. Philosophical Transactions of the Royal Society A, 368, 4073 - 4088.
- Dean J., Ghemawat S.** (2004) *MapReduce: Simplified Data Processing on Large Clusters*. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA.
- DeCandia G., Hastorun D., Jampani M., Kakulapati G., Lakshman A., Pilchin A., Sivasubramanian S., Voshall P., Vogels W.** (2007) *Dynamo: Amazon's Highly Available Key-value Store*. Proceedings of the 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, October 2007.

- Deering S., Hinden R.** (1998) *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460. <http://www.rfc-editor.org/rfc/rfc2460.txt> (date last accessed: March 2013).
- Degermark M., Nordgren B., Pink S.** (1999) *IP Header Compression*. RFC 2507. <http://tools.ietf.org/html/rfc2507> (date last accessed: April 2013).
- Diffie W., Hellman M. E.** (1976) *New Directions in Cryptography*. IEEE Trans. on Info. Theory, IT-22, 644 - 654, Nov. 1976.
- Donea J., Jones P. M., Magonette G. E., Verzeletti G.** (1992) *The pseudo-dynamic test method*. In Recent Advances in Earthquake Engineering and Structural Dynamics, Davidovici VE (ed.). Ouest Editions, Nantes, France, 1992; 769-780.
- Dong L., Mathew I., Rodgers G. P.** (2014) *PEN 2.4*. <https://nees.org/resources/pen>. (date last accessed: September 2014).
- Duckett J.** (2011) *HTML and CSS: Design and Build Websites*. John Wiley & Sons. ISBN: 978-1118008188.
- Duerst M., Suignard M.** (2005) *Internationalized Resource Identifiers (IRIs)*. RFC 3987. <http://www.ietf.org/rfc/rfc3987.txt> (date last accessed: March 2014).
- Duke M., Braden R., Eddy W., Blanton E.** (2006) *A Roadmap for Transmission Control Protocol (TCP) Specification Documents*. RFC 4614. <http://tools.ietf.org/html/rfc4614> (date last accessed: April 2013).
- Elmasri R., Navathe S. B.** (2010) *Fundamentals of Database Systems*. Addison-Wesley; 6 edition. ISBN: 978-0136086208.
- Erling O., Mikhailov I.** (2013) *Mapping Relational Data to RDF in Virtuoso*. <http://www.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSSQLRDF> (date last accessed: October 2013).
- EXODA** (2013) *ExODA: Integrating Description Logics and Database Technologies for Expressive Ontology-Based Data Access*. <http://www.cs.ox.ac.uk/projects/ExODA/index.html> (date last accessed: October 2013).
- Farrel A.** (2004) *The Internet and its protocols: A Comparative Approach*. Morgan Kaufmann. ISBN: 978-1558609136.
- Ferrant J. L., Gilson M., Jobert S., Mayer M., Ouellette M., Montini L., Rodrigues, S., Ruffini S.** (2008) *Synchronous Ethernet: A Method to Transport Synchronization*. IEEE Communications Magazine, September 2008, 126 - 134.
- Fitting M.** (2012) *First-Order Logic and Automated Theorem Proving*. Springer. ISBN: 978-1468403596.
- Fog A.** (2013) *Instruction tables Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs (Vol 4)*. http://www.agner.org/optimize/instruction_tables.pdf (date last accessed: September 2014).
- Gearon P., Passant A., Polleres A.** (2013) *SPARQL 1.1 Update*. W3C Recommendation. <http://www.w3.org/TR/sparql11-update/> (date last accessed: October 2013).

Glimm B., Hogan A., Krötzsch M., Polleres A. (2012) *OWL: Yet to arrive on the Web of data?*. Proceedings of the WWW2012 Workshop on Linked Data on the Web (LDOW-12). CEUR Workshop Proceedings 2012.

Glimm B., Krötzsch M. (2010) *SPARQL Beyond Subgraph Matching*. Proceedings of the 9th international Semantic Web conference, November 07-11, 2010, Shanghai, China.

Göetz B., Peierls T., Bloch J., Bowbeer J., Holmes D., Lea D. (2006) *Java Concurrency in Practice*. Addison-Wesley Professional. ISBN: 978-0321349606.

Guarino N., Oberle D., Staab S. (2009) *What is an ontology*. Handbook on Ontologies, Staab S., Studer R. (eds.), Springer-Verlag, 2nd ed., 2009, pp. 1-17.

Hacker T. J., Eigenmann R., Bagchi S., Irfanoglu A., Pujol S., Catlin A., Rathje E. (2011) *The NEEShub Cyberinfrastructure for Earthquake Engineering*. Computing in Science & Engineering, 13(4), 67-78.

Haslhofer B., Momeni E., Schandl B., Zander S. (2011) *Europeana RDF Store Report, The results of qualitative and quantitative study of existing RDF stores in the context of Europeana*. Technical Report. Vienna. http://eprints.cs.univie.ac.at/2833/1/europeana_ts_report.pdf (date last accessed: September 2014).

Highsmith J. (2002) *Agile Software Development Ecosystems*. Addison Wesley. ISBN: 978-0201760439.

Hitzler P., Krötzsch M., Parsia B., Patel-Schneider P. F., Rudolph S. (2012) *OWL 2 Web Ontology Language Primer (Second Edition)*. W3C Recommendation. <http://www.w3.org/TR/owl2-primer/> (date last accessed: October 2013).

Hitzler P., Krötzsch M., Rudolph S. (2009) *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC. ISBN: 9781420090505.

Horrocks I. (2013) *What are ontologies good for?*. Evolution of Semantic Systems, pages 175-188. Springer.

Horrocks I., Motik B., Wang Z. (2012) *The HermiT OWL Reasoner*. Proc. of the OWL Reasoner Evaluation Workshop (ORE 2012), volume 858 of CEUR Workshop Proceedings, Manchester, UK.

Horrocks I. (2012b) *Semantics $\sqcap$ Scalability $\models \perp$?* Journal of Zhejiang University - Science C, 13(4):241-244, 2012.

Horrocks I. (2011) *Tool Support for Ontology Engineering*. In Dieter Fensel, editor, Foundations for the Web of Information and Services, pages 103-112. Springer.

Horrocks I., Patel-Schneider P. F. (2010) *Knowledge Representation and Reasoning on the Semantic Web: OWL*. In Handbook of Semantic Web Technologies, chapter 9. Springer.

Horrocks I., Kutz O., Sattler, U. (2006) *The Even More Irresistible SROIQ*. In Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006), pages 57-67. AAAI Press.

Horrocks I. (2005) *Description Logics in Ontology Applications*. In B. Beckert, editor, Proc. of the 9th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods, number 3702 in Lecture Notes in Artificial Intelligence, Springer.

Howlett G. A., Catlin A. C., Nasi K., Howell R. (2009) *inDEED*, <http://nees.org/resources/indeed>. (date last accessed: July 2011).

Huston G. (2006) *Gigabit TCP*. CISCO The Internet Protocol Journal, 9 (2).

Huston G. (2000) *TCP Performance*. CISCO The Internet Protocol Journal, 3 (2).

IEEE 1394-2008 (2008) *IEEE Standard for a High-Performance Serial Bus*. ISBN: 978-0738157719.

IEEE 1588-2008 (2008) *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. ISBN: 978-0738154015.

Intel Corporation (2004) *IA-PC HPET (High Precision Event Timers) Specification*, version 1.0a. <http://www.intel.co.uk/content/www/us/en/software-developers/software-developers-hpet-spec-1-0a.html> (date last accessed: March 2013).

Intel Corporation (2013) *Intel 64 and IA-32 Architectures Optimization Reference Manual*. <http://www.intel.co.uk/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html> (date last accessed: February 2014).

Ioannidis I., Lamata Martínez I., Pegon P. Fidas C., Bousias S., Avouris N. (2012) *SERIES Deliverable D2.5 - 2nd version of Distributed Database and of Data Access Portal including user manual, documentation and guidelines*. http://www.series.upatras.gr/sites/default/files/Deliverable_D2.5_public.pdf (date last accessed: September, 2014).

Ioannidis I., Fidas C., Bousias S., Avouris N., Lamata Martínez I., Pegon P. (2012b) *User manual of Distributed Database and of Data Access Portal*. http://www.dap.series.upatras.gr/pfiles/DAP_User_manual.pdf (date last accessed: June, 2012).

Kazakov Y. (2008) *RIQ and SROIQ are Harder than SHOIQ*. In Proc. of KR '08. Sydney, NSW, Australia. DOI: 10.1.1.143.1562.

Kollia I., Glimm B., Horrocks I. (2011) *SPARQL Query Answering over OWL Ontologies*. In Proc. of the 8th European Semantic Web Conf. (ESWC 2011), Lecture Notes in Computer Science. Springer.

Kontchakov R., Lutz C., Toman D., Wolter F., Zakharyashev M. (2011) *The combined approach to ontology-based data access*. Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11). pp. 2656-2661. AAAI Press/IJCAI.

Kopetz H. (2013) *Real-Time Systems: Design Principles for Distributed Embedded Applications (Real-Time Systems Series)*. 2nd Edition. Springer ISBN: 978-1461428664.

Krisnadhi A., Maier F., Hitzler P. (2011) *OWL and Rules*. Proceedings of the 7th international conference on Reasoning web: semantic technologies for the Web of Data, p.382-415, August 23-27, 2011, Galway, Ireland.

- Krötzsch M.** (2012) *OWL 2 Profiles: An introduction to lightweight ontology languages*. Reasoning Web Summer School 2012. Springer.
- Krötzsch M., Simančík F., Horrocks I.** (2012b) *A Description Logic Primer*. In CoRR abs/1201.4089 (Technical report).
- Krötzsch M.** (2010) *Description Logic Rules*. Studies on the Semantic Web, Vol. 008, IOS Press. ISBN: 978-1-60750-654-6.
- Krötzsch M., Vrandečić D., Völkel M., Haller H., Studer R.** (2007) *Semantic Wikipedia*. In Journal of Web Semantics 5 (4), 251 - 261.
- Kwon O., Elnashai A., Spencer B.F.** (2008) *A framework for distributed analytical and hybrid simulations*. Struct. Engng & Mechanics, 30 (3), 331-350.
- Lakshman A., Malik P.** (2010) *Cassandra - A Decentralized Structured Storage System*. SIGOPS Operating System Review, 44 (2), 2010.
- Lamata Martínez I., Obón Santacana F., Williams M., Blakeborough A., Dorka U.** (2014) *Celestina-Sim: A Framework to Support Distributed Testing and Service Integration in Earthquake Engineering*. J. Comput. Civ. Eng., ASCE.
- Lamata Martínez I., Ioannidis I., Pegon P., Williams M., Blakeborough A.** (2013). The Process and Future of Data Integration within the European Earthquake Engineering Laboratories. J. Comput. Civ. Eng., 10.1061/(ASCE)CP.1943-5487.0000308
- Lamata Martínez I., Pegon P., Bosi A., Dietz M., Bousias S., Fidas C., Wenzel H.** (2010) *Distributed DataBase: Review of beneficiary current data formats - Specifications for the common data format and Data Access Portal*. SERIES Deliverable D2.1.
- Lamport L.** (1978) *Time, Clocks, and the Ordering of Events in a Distributed System*. Communications of the ACM. 21 (7), July 1978.
- Laplante P. A., Ovaska S. J.** (2011) *Real-Time Systems Design and Analysis: Tools for the Practitioner Wiley-IEEE Press; 4th edition*. ISBN: 978-0470768648.
- Lifschitz V., Morgenstern L., Plaisted D.** (2008) *Knowledge representation and classical logic*. In van Harmelen F., Lifschitz V., Porter B., eds. Chapter 1 of *Handbook of Knowledge Representation*. Elsevier. ISBN: 978-0444522115.
- MacKenzie C. M., Laskey K., McCabe F., Brown P. F., Metz R. [Eds]** (2006) *OASIS Reference Model for Service Oriented Architecture 1.0*. <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf> (date last accessed: October 2013).
- Mills D., Delaware U., Martin J. [Eds]** (2010) *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905. <http://tools.ietf.org/html/rfc5905> (date last accessed: April 2013).
- Mitra N., Lafon Y.** (2007) *SOAP Version 1.2 Part 0: Primer (Second Edition)*. W3C Recommendation. <http://www.w3.org/TR/soap12-part0/> (date last accessed: October 2013).

- Mosqueda G., Stojadinovic B., Hanley J., Sivaselvan M., Reinhorn A. (2008) *Hybrid seismic response simulation on a geographically distributed bridge model*. J. Struct. Engng, ASCE, 134(4), 535-543
- Motik M., Nenov Y., Piro R., Horrocks I., Olteanu D. (2014) *Parallel OWL 2 RL Materialisation in Centralised, Main-Memory RDF Systems*. Proc. of the 28th AAAI Conference on Artificial Intelligence.
- Motik B., Patel-Schneider P. F., Parsia B. (2012) *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*. W3C Recommendation. <http://www.w3.org/TR/owl2-syntax/> (date last accessed: July 2014).
- Motik B., Horrocks I., Kim S. M. (2012b) *Delta-Reasoner: a Semantic Web Reasoner for an Intelligent Mobile Platform*. In Proc. of the Twentyfirst International World Wide Web Conference (WWW 2012). ACM.
- Motik B., Patel-Schneider P. F., Cuenca Grau B (2012c) *OWL 2 Web Ontology Language Direct Semantics (Second Edition)*. W3C Recommendation. <http://www.w3.org/TR/owl2-direct-semantics/> (date last accessed: July 2014)
- Motik B., Shearer R., Horrocks I. (2009) *Hypertableau Reasoning for Description Logics*. Journal of Artificial Intelligence Research, 36:165-228, 2009.
- Nakata N., Dyke S., Zhang J., Mosqueda G., Shao X., Mahmoud H., Head M. H., Bletzinger M., Marshall G. A., Ou G., Song C. (2014) *Hybrid Simulation Primer and Dictionary*. <http://nees.org/resources/7702> (date last accessed: September 2014).
- NEESDB (2014) *NEES Databases*. <http://nees.org/resources/databases> (date last accessed: September 2014).
- Ohtani K., Ogawa N., Katayama T., Shibata H. (2003) *Construction of E-Defense (3-D full scale earthquake testing facility)*. National Research Institute for Earth Science and Disaster Prevention, Tsukuba, Japan. <http://www.bosai.go.jp/hyogo/ehyogo/research/pdf/09.pdf> (date last accessed: March 2013).
- Ojaghi M., Williams M., Dietz M., Blakeborough A., Lamata Martínez I. (2013) *Real-time distributed hybrid testing: coupling geographically distributed scientific equipment across the Internet to extend seismic testing capabilities*. Earthquake Engineering & Structural Dynamics. DOI: 10.1002/eqe.2385.
- Ojaghi M. (2011) *Development of the UK-NEES test middleware, early tests, web services approach, network usage, security, and usability issues in distributed hybrid testing*. OUEL Departmental Report no 2322/11. University of Oxford, Department of Engineering Science.
- Ojaghi M. (2011b) *Supplementary test results from UK-NEES, real-time Internet distributed hydraulic actuator control and real-time distributed hybrid tests between Oxford and Bristol*. OUEL Departmental Report no 2323/11. University of Oxford, Department of Engineering Science.
- Ojaghi M., Lamata Martínez I., Dietz M., Williams M., Blakeborough A., Crewe A., Taylor C., Madabhushi G., Haigh S., Ali A. (2010) *UK-NEES - Distributed*

Hybrid Testing Between Bristol, Cambridge and Oxford Universities: Connecting Structural Dynamics Labs to a Geotechnical Centrifuge. 9th U.S. National and 10th Canadian Conference on Earthquake Engineering (in press paper 1024).

Ojaghi M. (2010b) *The Development of Real-Time Distributed Hybrid Testing for Earthquake Engineering*. DPhil. Thesis. University of Oxford.

Pan P., Tomofuji H., Wang T., Nakashima M., Ohsaki M., Mosalam K.M. (2006) *Development of peer-to-peer (P2P) Internet online hybrid test system*. Earthquake Engng Struct. Dyn. 2006; 35:867-890. Wiley InterScience. DOI: 10.1002/eqe.561.

Park J., Yoon Y. (1997) *An extended TCP/IP protocol for real-time local area networks*. Control Engineering Practice, 6 (1), 111 - 118.

Pegon P., Pinto V. (2000) *Pseudo-dynamic testing with substructuring at the ELSA Laboratory*. Earthquake Engng Struct. Dyn. 29, 905 - 925.

Pegon P. (2008) *Continuous PsD Testing With Substructuring*. Modern Testing Techniques for Structural Systems, pp 197-257. Springer Vienna. ISBN: 978-3-211-09444-0.

Pejša S., Dyke S.J., Hacker T. (2014) *Building Infrastructure for Preservation and Publication of Earthquake Engineering Research Data*. The International Journal of Data Curation, 9, 2, 83 - 97.

Pejša S., Hacker T. (2013) *Curation of Earthquake Engineering Research Data*. <https://nees.org/resources/5751> (date last accessed: September 2014).

Pejša S. (2012) *NEEScomm requirements for curation and archiving of research data*. <https://nees.org/resources/4759> (date last accessed: December 2012).

Pejša S., Weisman D. (2011) *Data Sharing and Archiving Policies*. <https://nees.org/resources/2811> (date last accessed: September 2014).

Peng J., Law K. (2004) *Reference NEESgrid Data Model*. Report TR-2004-40, NCSA, University of Illinois at Urbana-Champaign.

Pérez Urbina H., Motik B., Horrocks I. (2009) *A comparison of query rewriting techniques for DL-lite*. In Proc. of DL 2009, volume 477 of CEUR, ceur-ws.org.

Pfeifer L., McAfee J. (1973) *Real-time data transmission on the Arpanet*. RFC 508. <http://tools.ietf.org/html/rfc508> (date last accessed: April 2013).

Plummer D. C. (1982) *An Ethernet Address Resolution Protocol*. RFC 826. <http://www.rfc-editor.org/rfc/rfc826.txt> (date last accessed: February 2014).

Poggi A., Lembo D., Calvanese D., De Giacomo G., Lenzerini M., Rosati R. (2008) *Linking data to ontologies*. Journal on Data Semantics X, 133-173.

Pomerance C. (1996) *A Tale of Two Sieves*. Notices of the American Mathematical Society, 43 (12), 1473 - 1485.

Postel J. [Ed] (1980) *User Datagram Protocol*. RFC 768. <http://www.rfc-editor.org/rfc/rfc768.txt> (date last accessed: March 2013).

- Postel J. [Ed]** (1981a) *Internet Control Message Protocol*. RFC 792. <http://tools.ietf.org/html/rfc792> (date last accessed: September 2012).
- Postel J. [Ed]** (1981b) *Internet protocol*. RFC 791 <http://www.rfc-editor.org/rfc/rfc791.txt> (date last accessed: March 2013).
- Postel J. [Ed]** (1981c) *Transmission Control Protocol*. RFC 793. <http://www.rfc-editor.org/rfc/rfc793.txt> (date last accessed: March 2013).
- Rivest R. L., Shamir A., Adleman L.** (1978) *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, 21 (2), 120-126.
- Robinson I., Webber J., Eifrem E.** (2013) *Graph databases*. O'Reilly Media. ISBN: 978-1449356262.
- Rodríguez Muro M., Calvanese D.** (2012) *Quest, a System for Ontology Based Data Access*. Proc. of the 9th Int. Workshop on OWL: Experiences and Directions (OWLED 2012).
- Rodríguez Muro M., Calvanese D.** (2012b) *Quest, an OWL 2 QL Reasoner for Ontology-Based Data Access*. In Proc. of the 9th Int. Workshop on OWL: Experiences and Directions (OWLED 2012).
- Rubenstein D., Kurose J., Towsley D.** (1998) *Real-Time Reliable Multicast Using Proactive Forward Error Correction*. Proceedings of NOSSDAV, 279--293, U. Mass, Amherst.
- Rudolph S.** (2011) *Foundations of Description Logics*. 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures. Springer. ISBN: 978-3-642-23031-8.
- Rudolph S., Krötzsch M., Hitzler P.** (2008) *All Elephants are Bigger than All Mice*. In Proceedings of the 21th International Workshop on Description Logics (DL-08). CEUR Workshop Proceedings 2008.
- Russinovich M. E., Solomon D. A., Ionescu A.** (2012) *Windows Internals 6th Edition, part 1*. Microsoft Press. ISBN: 978-0735648739.
- SAE** (2006) *IEEE-1394b Interface Requirements for Military and Aerospace Vehicle Applications*. SAE Aerospace Standard.
- Sahoo S. S., Halb W., Hellmann S., Idehen K., Thibodeau T., Auer S., Sequeda J., Ezzat A.** (2009) *A Survey of Current Approaches for Mapping of Relational Databases to RDF*. W3C Report. http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf (date last accessed: September 2014).
- Schellenberg A., Kim H., Mahin S. A.** (2010) *OpenFresco*. <http://nees.org/resources/openfresco> (date last accessed: July 2011).
- Schneider M.** (Ed) (2012) *OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition)*. W3C Recommendation. <http://www.w3.org/TR/owl2-rdf-based-semantics/> (date last accessed: July 2014).
- Schneier B.** (2012) *Liars and Outliers: Enabling the Trust that Society Needs to Thrive*. John Wiley & Sons, 2012. ISBN: 978-1118143308.

Schneier B. (1996) *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition*. Wiley. ISBN: 978-0471117094.

Schollmeier R. (2001) *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*. First International Conference on Peer-to-Peer Computing (P2P'01).

Schreiber G., Raimond Y. (2014) *Resource Description Framework (RDF) 1.1 Primer*. W3C Recommendation. <http://www.w3.org/TR/rdf11-primer/> (date last accessed: March 2014).

Schulzrinne H., Casner S., Frederick R., Jacobson V. (2003) *RTP: A Transport Protocol for Real-Time Applications*. ISOC. RFC 3550. <http://tools.ietf.org/html/rfc3550> (date last accessed: April 2013).

Seaborne A. (2013) *SPARQL 1.1 Query Results CSV and TSV Formats*. W3C Recommendation. <http://www.w3.org/TR/sparql11-results-csv-tsv/> (date last accessed: September 2014).

Shearer R., Motik B., Horrocks I. (2008) *HermiT: A Highly-Efficient OWL Reasoner*. In Proc. of the 5th Int. Workshop on OWL Experiences and Directions (OWLED 2008), volume 432 of CEUR.

Shirey R. (2000) *Internet Security Glossary*. RFC 2828. <http://tools.ietf.org/html/rfc2828> (date last accessed: July 2012).

Sipser M. (2012) *Introduction to the theory of computation, Third edition*. Wadsworth Publishing Co. ISBN: 978-1133187790.

SPARQL (2013) *SPARQL 1.1 overview*. The W3C SPARQL Working Group. W3C Recommendation. <http://www.w3.org/TR/sparql11-overview/> (date last accessed: October 2013)

Spencer B. F., Butler R., Ricker K., Marcusiu D., Finholt T. A., Foster I., Kesselman C., Birnholtz J. P. (2008), *NEESgrid: Lessons learned for future cyberinfrastructure development*. MIT Press: pg. pg: 331-348, Scientific Collaboration on the Internet, Boston, MA.

Stevens R. (1994) *TCP/IP Illustrated, Volume 1: The protocols*. Addison-Wesley, 1994, ISBN: 0-201-63346-9.

Stevens R. (1995) *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995, ISBN: 0-201-63354-X.

Studer R., Benjamins R., Fensel D. (1998) *Knowledge engineering: Principles and methods*. Data & Knowledge Engineering, 25(1-2):161-198, 1998.

Swenson C. (2008) *Modern Cryptanalysis: Techniques for Advanced Code Breaking*. Wiley. ISBN: 978-0470135938.

Tanenbaum A. S., Wetherall D. J. (2010) *Computer Networks, Fifth Edition*. Prentice Hall. ISBN: 978-0132126953.

Tsoumakos D., Roussopoulos N. (2003) *A Comparison of Peer to Peer Search Methods*. Proceedings of the sixth WebDB workshop.

UKNEES (2009) *UK Network for Earthquake Engineering Simulation*. <http://www-civil.eng.ox.ac.uk/research/structdyn/presentations/uknees.html> (date last accessed: July 2011).

Yang Y. S., Hsieh S. H., Tsai K. C., Wang S. J., Wang K. J., Cheng W. C., Hsu C. W. (2007) *ISEE: Internet-based Simulation for Earthquake Engineering-Part I: Database approach*. Earthquake Engng Struct. Dyn. 2007; 36:2291-2306. Published online 2 July 2007 in Wiley InterScience. DOI: 10.1002/eqe.730.

Yang Y. S., Hsieh S. H., Tsai K. C., Wang S. J., Wang K. J., Cheng W. C., Hsu C. W. (2007b) *ISEE: Internet-based Simulation for Earthquake Engineering-Part II: The application protocol approach*. Earthquake Engng Struct. Dyn. 2007; 36:2307-2323. Published online 27 June 2007 in Wiley InterScience. DOI: 10.1002/eqe.729.

Zhang L., Berson S., Herzog S., Jamin S., Braden R. [Ed] (1997) *Resource ReSerVation Protocol*. RFC 2205. <http://tools.ietf.org/html/rfc2205> (date last accessed: April 2013).

Zhang L., Deering S., Estrin D., Shenker S., Zappala D. (1993) *RSVP: A New Resource ReSerVation Protocol*. IEEE Network Magazine, 7, 8-18, Sept./Oct. 1993.