

DEVELOPMENT OF SEMANTIC DATA MODELS TO  
SUPPORT DATA INTEROPERABILITY IN THE RAIL  
INDUSTRY

JONATHAN TUTCHER

A thesis submitted to the University of Birmingham for the degree of  
DOCTOR OF PHILOSOPHY

Centre for Railway Research and Education  
Electronic, Electrical, and Systems Engineering  
College of Engineering and Physical Sciences  
University of Birmingham

April 2015

UNIVERSITY OF  
BIRMINGHAM

**University of Birmingham Research Archive**

**e-theses repository**

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

Jonathan Tutchter: *Development of Semantic Data Models to Support  
Data Interoperability in the Rail Industry* © April 2015

SUPERVISORS:

Prof. Clive Roberts

Dr. John Easton

## ABSTRACT

---

Railways are large, complex systems that comprise many heterogeneous subsystems and parts. As the railway industry continues to enjoy increasing passenger and freight custom, ways of deriving greater value from the knowledge within these subsystems are increasingly sought. Interfaces to and between systems are rare, making data sharing and analysis difficult.

Semantic data modelling provides a method of integrating data from disparate sources by encoding knowledge about a problem domain or world into machine-interpretable logic and using this knowledge to encode and infer data context and meaning. The uptake of this technique in the Semantic Web and Linked Data movements in recent years has provided a mature set of techniques and toolsets for designing and implementing ontologies and linked data applications.

This thesis demonstrates ways in which semantic data models and OWL ontologies can be used to foster data exchange across the railway industry. It sets out a novel methodology for the creation of industrial semantic models, and presents a new set of railway domain ontologies to facilitate integration of infrastructure-centric railway data. Finally, the design and implementation of two prototype systems is described, each of which use the techniques and ontologies in solving a known problem.



## PUBLICATIONS

---

Some of the work that appears in this thesis have appeared previously in the following publications:

- [1] J. Tutcher. “Ontology-driven Data Integration for Railway Asset Monitoring Applications”. In: *IEEE Workshop on Large Data Analytics in Transportation Engineering, 2014 IEEE International Conference on Big Data*. 2014.
- [2] J. Tutcher, J. M. Easton, and C. Roberts. “Enabling Data Integration in the Rail Industry Using RDF and OWL: The RaCoOn Ontology”. In: *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering* (2015), F4015001.



## ACKNOWLEDGEMENTS

---

This thesis is a culmination of four years of work at the Birmingham Centre for Railway Research and Education, and significantly more time living and studying in Birmingham. My thanks go to everyone who made my time there so inspirational and enjoyable, and I am particularly grateful to the following people:

- Clive Roberts and John Easton, for their ideas and supervision, company, and expertise both inside of academia and outside.
- Gemma Nicholson, Mani Entezami, and Chris Morris, whose wise words and cooking skills kept me sane.
- Stephen Kent, Edd Stewart, and Andreas Hoffrichter, for distracting me with fuel cell trains, exploding point machines, robots, and freezers.
- Pete Nickless and Lizzie Smith Airey for the creative inspiration and company.
- Rob Myall, Mark Hargreaves, Colin Tiller and Simon Chadwick from Siemens, who were fantastic collaborators.
- The Engineering and Physical Sciences Research Council, Siemens, Network Rail, and Future Railway for funding and facilitating the work undertaken in this thesis.
- Rory Dickerson, for being an incredible friend.

Finally, I am indebted to Jo Bryant and my parents David and Hilary Tutchter, who have been an unwavering source of love and encouragement throughout.





## CONTENTS

---

List of Code Listings	xxi
<b>1 INTRODUCTION AND PROBLEM STATEMENT</b>	<b>1</b>
1.1 Project Background and Problem Statement . . . . .	1
1.1.1 Railway Fragmentation and Failure . . . . .	1
1.1.2 Information Silofication . . . . .	2
1.1.3 The Data-driven Railway . . . . .	3
1.1.4 Standardised Data Models . . . . .	4
1.2 Aims and Original Contributions . . . . .	5
1.3 Thesis Organisation and Structure . . . . .	6
1.4 Project Partners . . . . .	6
<b>2 MODELLING, ONTOLOGIES, AND THE SEMANTIC WEB</b>	<b>9</b>
2.1 Introduction . . . . .	9
2.2 Data Modelling and Knowledge Management . . . . .	9
2.2.1 Application and Domain Engineering . . . . .	10
2.2.2 Incentives for Standardised Models . . . . .	12
2.2.3 Syntax-based Models . . . . .	14
2.3 Semantic Modelling . . . . .	18
2.3.1 Flexibility in Knowledge Representation . . . . .	19
2.3.2 Preservation of Context . . . . .	20
2.3.3 Interoperability and Expressivity . . . . .	20
2.4 Knowledge Representation and Ontology . . . . .	23
2.4.1 What Is An Ontology? . . . . .	23
2.4.2 Semantic Expressivity and Ontology Languages . . . . .	24
2.4.3 Data Models, Vocabularies, and Ontology . . . . .	26
2.4.4 Ontology Types . . . . .	27
2.4.5 Ontology Reasoning . . . . .	28
2.4.6 Ontology Engineering Methodologies . . . . .	29
2.4.7 Ontology Modularity . . . . .	33
2.4.8 Validation and Evaluation of Ontology Design . . . . .	36
2.5 The Semantic Web and the Linked Data Movement . . . . .	38
2.5.1 The Semantic Web . . . . .	38
2.5.2 The Linking Open Data Movement . . . . .	41
2.5.3 Disincentives to Uptake of Linked Open Data and Enterprise Ontology . . . . .	42
2.6 Core Technical Concepts and Notation Used . . . . .	43
2.6.1 The Resource Description Framework . . . . .	43
2.6.2 RDF Schema . . . . .	50

2.6.3	Ontology Languages, The Web Ontology Language and Description Logic (DL) . . . . .	51
2.6.4	Reasoning and Inference in OWL and DL . . . . .	55
2.6.5	Terminological and Assertional Knowledge . . . . .	59
2.6.6	RDF Storage and Presentation . . . . .	60
2.6.7	Overview of Software Tools . . . . .	60
2.6.8	Querying RDF Data . . . . .	62
2.6.9	Presentation of OWL Examples and Patterns . . . . .	63
2.7	Practical Problems and Assumptions in OWL . . . . .	65
2.8	Summary . . . . .	66
3	RAILWAY DATA MANAGEMENT, INDUSTRIAL MODELS, AND NOTABLE ONTOLOGIES . . . . .	67
3.1	Introduction . . . . .	67
3.2	State of UK Rail Data Management . . . . .	67
3.2.1	Current Wheel Maintenance Workflow . . . . .	67
3.2.2	Network Rail Intelligent Infrastructure . . . . .	68
3.2.3	DARWIN and Network Rail ORBIS . . . . .	70
3.3	Transportation Data Models and Frameworks . . . . .	72
3.3.1	RailML . . . . .	72
3.3.2	TAF/TAP TSI . . . . .	73
3.3.3	RailTopoModel and National Topology Models . . . . .	75
3.3.4	Proprietary Systems and Models . . . . .	77
3.3.5	InteGRail . . . . .	77
3.3.6	Rail Functional Architecture . . . . .	80
3.4	ISO15926 . . . . .	81
3.5	Generic Asset Information Integration Standards . . . . .	82
3.5.1	MIMOSA OSA . . . . .	83
3.5.2	Siemens Ontology-based Data Access System . . . . .	85
3.6	Relevant Ontologies and Common Modelling Paradigms . . . . .	85
3.6.1	Upper Ontologies . . . . .	86
3.6.2	Approaches to Time Representation . . . . .	86
3.6.3	Approaches to Representing Quantities, Units, and Dimensions . . . . .	91
3.7	Summary . . . . .	92
4	DESIGNING EXTENSIBLE MODELS FOR LARGE COMPLEX SYSTEMS . . . . .	95
4.1	Introduction . . . . .	95
4.1.1	Introduction to the RaCoOn Ontologies . . . . .	95
4.1.2	Methodological Requirements . . . . .	96
4.1.3	Proposed Approach . . . . .	97
4.2	Stage 1: Specification and Scope Definition . . . . .	100
4.2.1	Scope Definition Methodology . . . . .	101

4.2.2	RaCoOn Stakeholder Requirements and Applications . . . . .	103
4.3	Stage 2: Architecture and Ontology Modularity . . . . .	112
4.3.1	Module Interdependence . . . . .	114
4.4	Stage 3: Knowledge Acquisition and Conceptualisation . . . . .	114
4.4.1	Top-down Knowledge Acquisition . . . . .	115
4.4.2	Initial Conceptualisation and Iteration of RaCoOn ontologies . . . . .	116
4.4.3	Knowledge Extraction from Non-Ontological Resources . . . . .	117
4.5	Stage 4: Implementation and Ontology Reuse . . . . .	120
4.5.1	Ontology Design and Implementation Best Practice . . . . .	120
4.5.2	Use of Ontology Design Patterns To Encourage Re-use . . . . .	122
4.5.3	Pattern Design vs. Reuse . . . . .	122
4.5.4	Developing Ontology Design Patterns . . . . .	124
4.5.5	Reusing Best Practice Ontologies and Patterns . . . . .	125
4.5.6	Expressivity and Reasoning . . . . .	129
4.6	Stage 5: Validation, Evaluation, and Iteration . . . . .	131
4.6.1	Logical Validation . . . . .	132
4.6.2	Ontology Coverage through Application Data Mapping . . . . .	134
4.6.3	In-use Validation . . . . .	136
4.6.4	Similarity Measurement Through Expert Knowledge Elicitation . . . . .	137
4.6.5	Iteration and Version Control . . . . .	137
4.7	Best Practice Implementation Design Patterns . . . . .	139
4.7.1	Annotation Best Practice and Naming Conventions . . . . .	139
4.7.2	Ontology Self-documentation . . . . .	140
4.7.3	Provenance, Trust, and Metadata . . . . .	141
4.8	Summary . . . . .	145
5	RACOON: PRAGMATIC ONTOLOGIES FOR THE RAIL INDUSTRY . . . . .	147
5.1	Introduction . . . . .	147
5.2	Modular Ontology Design . . . . .	147
5.2.1	Ontology Module Structure . . . . .	147
5.2.2	Key Concepts and Semantic Trade-offs . . . . .	148
5.3	The Cross-Domain Ontology . . . . .	150
5.3.1	Conceptualisation, Structure and Patterns . . . . .	151
5.3.2	Representation of Common Concepts . . . . .	155
5.4	The Rail Core Ontology . . . . .	159
5.4.1	Subdomains and Terminology . . . . .	159
5.4.2	Local Naming Pattern . . . . .	160
5.4.3	Representing Asset Capabilities and Characteristics . . . . .	162
5.4.4	Geographical Positioning and Location . . . . .	164
5.4.5	Representing Diagrammatic Network Layouts . . . . .	178

5.4.6	Navigability and Routing Across Networks . . . . .	180
5.4.7	Re-engineering Knowledge from RailML . . . . .	182
5.5	RaCoOn Ontology Evaluation . . . . .	190
5.5.1	Structural and Syntactic Validation . . . . .	190
5.5.2	Workshop Evaluation . . . . .	193
5.5.3	Measuring Ontology Fit Using Railsys . . . . .	199
5.5.4	In-use Validation . . . . .	205
5.6	Summary . . . . .	205
6	INTEGRATION OF RAILWAY REMOTE CONDITION MONITORING DATA . . . . .	207
6.1	Introduction . . . . .	207
6.2	The AMaaS Application . . . . .	207
6.2.1	Overview and Use Case . . . . .	208
6.2.2	Existing Prototype Architecture . . . . .	212
6.2.3	Proposed System Architecture . . . . .	213
6.2.4	Stages 1 & 2: Asset Monitoring System Implementation . . . . .	220
6.2.5	Stages 3 & 4: Infrastructure Integration and Reasoning . . . . .	227
6.2.6	Stages 5 & 6: Integration of Timetable Data and Inference of Rolling Stock Faults . . . . .	234
6.3	The Train Locator Application . . . . .	242
6.3.1	Motives for Second Demonstrator . . . . .	242
6.3.2	Design . . . . .	243
6.3.3	Front End Application Implementation . . . . .	248
6.3.4	Source Data and Simulation . . . . .	250
6.3.5	Live Departure Boards View & Reasoning . . . . .	256
6.3.6	Train Mapper View & Reasoning . . . . .	260
6.4	Summary . . . . .	263
7	CONCLUSIONS DRAWN AND FURTHER WORK . . . . .	265
7.1	Key Findings and Contributions Made . . . . .	265
7.1.1	RaCoOn Methodology . . . . .	265
7.1.2	The RaCoOn Ontologies . . . . .	266
7.1.3	The FuTRO Case Studies . . . . .	267
7.2	Limitations of Approaches Taken . . . . .	268
7.2.1	RaCoOn Methodology . . . . .	268
7.2.2	The RaCoOn Ontologies . . . . .	269
7.2.3	Other Limitations . . . . .	271
7.3	Planned and Possible Further Work . . . . .	272
7.3.1	Possible Extensions . . . . .	272
7.3.2	Work Currently Underway . . . . .	273
A	LIST OF CODE AND ONTOLOGIES HOSTED ONLINE . . . . .	275

B	REFERENCE DIAGRAMS AND LISTS	277
B.1	List of CURIE Prefixes Used Throughout Thesis . . . . .	277
B.2	ISO 15926 EXPRESS-G Notation Diagrams . . . . .	278
C	RACoon ONTOLOGY RESOURCES	281
C.1	RaCoOn Ontology Class Terms . . . . .	281
C.1.1	List of Cross-domain Ontology Terms . . . . .	281
C.1.2	List of Railway Domain Ontology Terms . . . . .	282
C.2	Validation Workshop Results . . . . .	285
C.2.1	High Level and Subdomain Concepts Elicited From RaCoOn Workshops . . . . .	285
C.2.2	Domain Interactions Elicited From Rail Core Ontolo- gies (RaCoOn) Workshops . . . . .	289
C.3	RaCoOn Railsys Validation . . . . .	292
D	FUTRO IMPLEMENTATION NOTES	303
D.1	AMaaS Track Layout Graphics . . . . .	303
D.2	Legacy Wheelchex Data Snippets . . . . .	305
D.3	AMaaS Stardog Rules and Queries . . . . .	306
	BIBLIOGRAPHY	309

## LIST OF FIGURES

---

Figure 1.1	Interactions Between Major UK Railway Stakeholders . . . . .	2
Figure 1.2	UK Railway Customer Information Architecture . . . . .	3
Figure 2.1	Interactions between Domain Engineering and Application Engineering . . . . .	12
Figure 2.2	Extract from the RSSB Rail Functional Architecture . . . . .	13
Figure 2.3	Interfaces in Complex Systems With and Without Standard Data Models . . . . .	14
Figure 2.4	Graphical Representation of Possible XML Schema for Data from <a href="#">Table 2.1</a> . . . . .	17
Figure 2.5	Example Semantic Data Model Using Data Modified from <a href="#">Table 2.1</a> . . . . .	20
Figure 2.6	Alignment of Semantic Data Models Using Mapping Axioms . . . . .	22
Figure 2.7	Example of Generalisation and Interoperability in Semantic Data Model . . . . .	23
Figure 2.8	Illustration of Example Pop Music Ontology	24
Figure 2.9	The “Ontology Spectrum” . . . . .	25
Figure 2.10	Set of Modular Ontologies Partitioned by Purpose . . . . .	35
Figure 2.11	A Wikipedia Page as Understood by Humans and Computers . . . . .	39
Figure 2.12	Screenshot of Google Shopping Results Page	41
Figure 2.13	Example Relationships in The Simpsons Family	48
Figure 2.14	Example of N-ary Relationships Pattern . . .	48
Figure 2.15	Expressivity Characteristics of OWL Profiles	55
Figure 2.16	Protégé 5.0 Ontology Visualisation View . .	61
Figure 2.17	Demonstration Graphical Representation of OWL Example . . . . .	65
Figure 3.1	Flow Chart of Current and Future Train Wheel Maintenance Workflows . . . . .	69
Figure 3.2	Network Rail Intelligent Infrastructure User Interface Components . . . . .	70
Figure 3.3	DARWIN Input Data Sources . . . . .	71
Figure 3.4	TAF/TAP TSI Implementation Tasks and Timescales for Completion . . . . .	76

Figure 3.5	Railway Infrastructure Status Dependence Pattern . . . . .	79
Figure 3.6	MIMOSA OSA-CBM Architecture Levels And Example Applications . . . . .	84
Figure 3.7	Key Components in MIMOSA OSA-EAI Standard . . . . .	84
Figure 3.8	Example of Temporally-Varying Data Represented Using Endurantist Approach . . . . .	87
Figure 3.9	Example of Temporally-Varying Data Represented Using Perdurantist Approach . . . . .	87
Figure 3.10	Representation of Time Extents Using Endurant Entities . . . . .	88
Figure 3.11	Reified Endurantist Time Extents Representation . . . . .	88
Figure 3.12	Example of 4D Fluents [225] Approach in OWL	89
Figure 4.1	RaCoOn Ontology Engineering Methodology	98
Figure 4.2	“Torchlight” Diagram Showing an Overview of the Rail Domain, and Proposed Scope of Domain Ontology . . . . .	101
Figure 4.3	Simplified Diagram of ‘Vee’ Systems Engineering Methodology . . . . .	102
Figure 4.4	Initial Rail Domain Functional Conceptualisation with Invensys Rail Group Stakeholders (2011) . . . . .	105
Figure 4.5	Interconnected Railway Data Management Application Areas . . . . .	107
Figure 4.6	Transport Data Applications Mentioned by Participants at TSC ‘Data Challenge’ Workshop . . . . .	108
Figure 4.7	Data Types Used by Participants at TSC ‘Data Challenge’ Workshop . . . . .	109
Figure 4.8	TSC Workshop Infrastructure-centric Data Integration Themes . . . . .	111
Figure 4.9	Modularity Based on Expressivity in a Set of Ontologies . . . . .	114
Figure 4.10	Diagram Showing Design Question Paths to Domain Model Construction . . . . .	117
Figure 4.11	Stages in NeON Methodology Non-ontological Resource Reuse Process . . . . .	118
Figure 4.12	Steps Towards Concept Formalisation in RaCoOn Methodology . . . . .	123
Figure 4.13	Visualisation of ‘Linked Open Vocabularies’ Datasets by Size . . . . .	125



Figure 4.14	RaCoOn Methodology Ontology Integration Process . . . . .	126
Figure 4.15	W3C RDF Validator Results for Upper Ontology	133
Figure 4.16	RaCoOn Ontology Git Commit History . . . . .	138
Figure 4.17	Documentation Pattern for Representing Content Pattern Association in OWL . . . . .	140
Figure 4.18	Example Use of Meta-modelling to Assert Provenance Information on Ontology Concepts . . . . .	144
Figure 5.1	RaCoOn Ontology Modules Structure . . . . .	148
Figure 5.2	OWLViz Diagram of RaCoOn Cross-domain Ontology . . . . .	152
Figure 5.3	Representation of Temporally Changing Data in RaCoOn Cross-Domain Ontology . . . . .	155
Figure 5.4	Example of Measurement Design Pattern in RaCoOn Ontologies . . . . .	156
Figure 5.5	Example of Composition Design Pattern Showing High Level Points Machine Components	158
Figure 5.6	Example Showing Local Naming Design Pattern . . . . .	161
Figure 5.7	Design Pattern Showing Assertion of ERTMS Capability on Track Section and Class Inference	163
Figure 5.8	Example Showing Track Characteristic Change Design Pattern . . . . .	164
Figure 5.9	Example of ‘Network Level’ Railway Route Graph . . . . .	166
Figure 5.10	Example of ‘Route Level’ Railway Route Graph	166
Figure 5.11	Example of Arbitrary Graph Directivity in Track Topology . . . . .	170
Figure 5.12	Example Showing Track Inheritance Design Pattern . . . . .	172
Figure 5.13	Track Element Positioning Design Pattern . . . . .	175
Figure 5.14	Example WGS84 Position Represented Using Location Pattern . . . . .	176
Figure 5.15	Screenshot From Invensys Westlock Interlocking Simulator . . . . .	178
Figure 5.16	Example XY Presentation Position Represented Using Location Pattern . . . . .	179
Figure 5.17	Example Representation of Network Navigability Across Routes . . . . .	181
Figure 5.18	Overview of RailML Subschemas and example elements . . . . .	183
Figure 5.19	Flowchart Showing RailML Re-engineering Workflow . . . . .	185

Figure 5.20	RailML Entity Provenance Annotation Pattern	187
Figure 5.21	Example of a RailML Signal Group, Represented Using the Collections Ontology . . .	189
Figure 5.22	Example Showing Route Profile Design Pattern Across Three Route Nodes . . . . .	190
Figure 5.23	Screenshot of Oops! Result for 3D Cross-domain Ontology . . . . .	192
Figure 5.24	Photograph of RaCoOn Validation Session at the University of Birmingham . . . . .	193
Figure 5.25	Photograph of RaCoOn Validation Session at the University of Birmingham . . . . .	194
Figure 5.26	Railway High Level Systems As Conceptualised by Groups at Edgbaston Validation Workshop . . . . .	196
Figure 5.27	Edgbaston Validation Workshop: Results of ‘Infrastructure’ Category Decomposition . .	197
Figure 5.28	Example Interaction for Validation Workshops	198
Figure 5.29	Alignment Between High Level Railway Themes Elicited in Workshops and RaCoOn Scope . .	200
Figure 5.30	XML Schema Definition of <code>line</code> from Railsys Example Documents . . . . .	201
Figure 6.1	AMaaS Modularity . . . . .	209
Figure 6.2	Data Sources And Actors In AMaaS . . . . .	210
Figure 6.3	System Design Storyboard for AMaaS System	211
Figure 6.4	Siemens Cloud-based Asset Monitoring System Architecture . . . . .	212
Figure 6.5	AMaaS Demonstrator System Architecture .	214
Figure 6.6	Block Diagram Showing Query Activity across Federated Data Stores in AMaaS . . . . .	220
Figure 6.7	AMaaS Component Topology Design Pattern	221
Figure 6.8	AMaaS Observation Pattern . . . . .	223
Figure 6.9	Hierarchy of PCM Devices in AMaaS Demonstrator System . . . . .	224
Figure 6.10	Screenshot of the AMaaS Entity Browser . .	225
Figure 6.11	AMaaS Observation Pattern . . . . .	226
Figure 6.12	AMaaS Web Application Profiles View . . .	227
Figure 6.13	Railway Layout Around Coventry Station According to OpenStreetMap . . . . .	228
Figure 6.14	Screenshot of AMaaS Track View . . . . .	230
Figure 6.15	Asset Dependency and Fault Inheritance Design Pattern . . . . .	234
Figure 6.16	Screenshot of OpenRefine RDF Mapping Plugin . . . . .	238

Figure 6.17	AMaaS Rolling Stock Design Pattern & Example Data . . . . .	240
Figure 6.18	Screenshot Showing AMAaaS Train Finder View . . . . .	240
Figure 6.19	System Design Storyboard for FuTRO Train Locator Application <sup>1</sup> . . . . .	244
Figure 6.20	FuTRO Train Locator Key Components . . . . .	247
Figure 6.21	UML Sequence Diagram Showing High Level Data Flow for Request of Live Departure Board View . . . . .	249
Figure 6.22	Train Locator Infrastructure Data Mapping Workflow . . . . .	252
Figure 6.23	UML Class Diagram Showing Structure of FuTRO Train Simulator . . . . .	254
Figure 6.24	Design Pattern Used for Train Locations in Train Locator Demonstrator . . . . .	255
Figure 6.25	Screenshot of Train Locator Live Departure Board View . . . . .	257
Figure 6.26	Demonstration of Train Location as Reported by Train Locator . . . . .	260
Figure 6.27	Screenshot of Train Locator Map View . . . . .	261
Figure 6.28	Pattern for Prioritising Knowledge in TLOC Ontology . . . . .	262
Figure B.1	EXPRESS-G Model Diagram . . . . .	279
Figure B.2	EXPRESS-G Instance Diagram . . . . .	279
Figure C.1	Railsys RaCoOn Transformation Validation Spreadsheet . . . . .	293
Figure D.1	SVG Track Diagram Used in FuTRO AMaaS Project . . . . .	303

## LIST OF TABLES

---

Table 2.1	1950s Jazz Musicians and Key Personal Information . . . . .	15
Table 2.2	Making a Phone Call . . . . .	18
Table 2.3	The 5 Stars of Open Data . . . . .	42
Table 2.4	Example RDF URIs and Literals . . . . .	44
Table 2.5	Two Examples of Semantic Web URI abbreviated as CURIE identifiers . . . . .	45
Table 2.6	Example OWL Constructs and Related DL Symbols . . . . .	53
Table 2.7	Icon Symbols Used to Denote OWL Entities .	64
Table 3.1	TAP Key Activities and Descriptions . . . . .	74
Table 4.1	Use Cases for a Standard Rail Data Model, Categorised by Area . . . . .	106
Table 4.2	Table Showing Subjective Metrics for Non-ontological Resource Reuse in RaCoOn Ontologies . . . . .	120
Table 4.3	Division of OWL constructs between core and constraints ontologies . . . . .	131
Table 5.1	RaCoOn Modules and Dependencies . . . . .	149
Table 5.2	Basic Track Topology Design Pattern OWL Constructs . . . . .	169
Table 5.3	Linear Track Positioning OWL Constructs .	174
Table 5.4	Geodesic Location Representation OWL Constructs . . . . .	176
Table 5.5	OWL Expressivity Profiles of RaCoOn Modules	191
Table 5.6	Rail Domain Subsystems Validation Workshop Consensus . . . . .	195
Table 5.7	Table Showing Example Interactions Elicited from Ontology Evaluation Workshops . . . . .	198
Table 5.8	Table Showing Disambiguation of Railsys platform element . . . . .	202
Table 5.9	Extract from Railsys Transformation Table Showing ‘Station’ Pattern . . . . .	204
Table 6.1	Functional Requirements Defined From AMaaS Storyboard . . . . .	216
Table 6.2	Comparison of Features Across Popular RDF Triplestores . . . . .	217

Table 6.3	RaCoOn Elements Used for AMaaS Infrastructure Mappings . . . . .	229
Table 6.4	Example of Asset Data Contextualisation using Asset Monitoring System Data . . . . .	231
Table 6.5	Working Timetable Attribute Mappings . . . . .	238
Table 6.6	Selected Records from CIF ‘Schedules’ Schema Table . . . . .	239
Table 6.7	Matrix of Train Locator System Behaviour Using Different Data Sources . . . . .	246
Table 6.8	Views Provided by Train Locator Application . . . . .	250
Table 6.9	Train Locator API Calls and associated functionality . . . . .	251
Table B.1	Ownership Denoted by URI Namespaces . . . . .	277
Table C.1	Railway Domain Interactions Elicited From Edgbaston Validation Workshop . . . . .	290
Table C.2	Table Showing Railway Domain Interactions Elicited From Chippenham Validation Workshop . . . . .	291

## LIST OF CODE LISTINGS

---

Figure 2.1	Example XML Markup of <a href="#">Table 2.1</a> . . . . .	16
Figure 2.2	Example of Facts Represented Using Terse RDF Triple Language (Turtle) . . . . .	47
Figure 2.3	Example of RDF Reification in Turtle . . . . .	49
Figure 2.4	Demonstration of Inference based on <code>rdfs:domain</code> and <code>rdfs:range</code> Restrictions . . . . .	51
Figure 2.5	OWL Markup Showing Restrictions on Train Class . . . . .	54
Figure 2.6	SPARQL Query for Railway Stations in Model	63
Figure 3.1	Using Datatype Properties to Represent Attributes . . . . .	91
Figure 3.2	Turtle Listing Showing Time Represented using the QUDT ontology . . . . .	92
Figure 4.1	Train Speed Represented in RDF Using Direct Data Property Approach . . . . .	99
Figure 4.2	Train Speed Represented in RDF Using Ternary Relations Design Pattern . . . . .	100
Figure 4.3	Ontology Validator Method Listing . . . . .	133
Figure 4.4	Example VOID Description of Upper Ontology	142
Figure 5.1	Example Assertions to Demonstrate <code>rdfs:range</code> Restrictions . . . . .	153
Figure 5.2	Example Inferences Made Through <code>rdfs:range</code> Restrictions . . . . .	153
Figure 5.3	Demonstration of Ternary Relations Measurement Pattern in Turtle . . . . .	157
Figure 5.4	Example SPARQL Location Mapping Between RaCoOn and the W3C Geo Ontology . . . . .	177
Figure 5.5	Extract from RailML Infrastructure Model Showing Complementary Elements and Types . . . . .	186
Figure 6.1	OWL Axiom Asserting Observation Relation Properties . . . . .	232
Figure 6.2	Stardog Rule for Inference of Current Asset Condition . . . . .	232
Figure 6.3	Extract of RDF Station Information from Train Locator Knowledge Base . . . . .	253

Figure 6.4	Natural Language Query for Train Forecast in Train Locator Application . . . . .	257
Figure 6.5	SPARQL Query for Train Forecast in Train Locator Application . . . . .	258
Figure 6.6	Stardog Rule to Assert Track Circuit Mileage in Train Locator Application . . . . .	259
Figure 6.7	Stardog Rule to Present Preferred Data in Train Locator Application . . . . .	262
Figure D.1	Extract from RDFa Enriched SVG Code . . .	304
Figure D.2	Example Wheelchex XML Data File . . . . .	305
Figure D.3	Stardog Rule for Deriving Current Asset Condition in AMaaS . . . . .	306
Figure D.4	SPARQL Query for Wheel Impact Load Detector Traffic Inference . . . . .	307

## ACRONYMS

---

AJAX	Asynchronous Javascript and HTML.
AMaaS	Asset Monitoring As A Service.
AMQP	Advanced Message Queuing Protocol.
API	Application Programming Interface.
ATOC	Association of Train Operating Companies.
BFO	Basic Formal Ontology.
CASE	Collaborative Awards in Science and Engineering.
CIF	Common Interface Format.
CQ	Competency Question.
CSV	Comma Separated Values.
CURIE	Compact URI.
DERI	Digital Enterprise Research Institute.
DILIGENT	Distributed Engineering of Ontologies.
DL	Description Logic.
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering.
DUL	DOLCE & DnS Ultralite.
ECML	East Coast Main Line.
ELR	Engineer's Line Reference.
ERTMS	European Rail Traffic Management System.
EU	European Union.
FOL	First Order Logic.
GIS	Geographical Information System.
GPS	Global Positioning System.
HABD	Hot Axlebox Detector.
HDT	Header, Dictionary, Triples.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
IBM	International Business Machines.
ID	Identifier.
IIP	Intelligent Infrastructure Project.
INSPIRE	Infrastructure for Spatial Information in the European Community.



IP	Internet Protocol.
IRG	Invensys Rail Group.
IRI	Internationalized Resource Identifier.
ISO	International Organization for Standardization.
ITS	Siemens Intelligent Transportation Systems.
JSON	JavaScript Object Notation.
JSON-LD	JavaScript Object Notation for Linked Data.
KIF	Knowledge Interchange Format.
LDB	Live Departure Board.
LDL	Layout Description Language.
LOD	Linked Open Data.
MIMOSA	An Operations and Maintenance Information Open System Alliance.
MVC	Model-View-Controller.
NASA	National Aeronautics and Space Administration.
NR	Network Rail.
ODP	Ontology Design Pattern.
ONTIME	Optimal Networks for Train Integration Management across Europe.
OOPS!	Ontology Pitfall Scanner.
ORBIS	Offering Rail Better Information Services.
OSA-CBM	Open System Architecture for Condition-based Monitoring.
OSA-EAI	Open System Architecture for Enterprise Application Integration.
OWA	Open World Assumption.
OWL	Web Ontology Language.
PC	Personal Computer.
PCM	Points Condition Monitoring.
PDF	Portable Document Format.
PRM	Passengers with Reduced Mobility.
QL	Query Language.
QUDT	Quantities, Units, Dimensions, and Types.
RaCoOn	Rail Core Ontologies.
RCM	Remote Condition Monitoring.
RDF	Resource Description Framework.
RDFa	Resource Description Framework in Attributes.
RDFS	Resource Description Framework Schema.
REST	Representational State Transfer.
RFA	Rail Functional Architecture.

RFC	Request for Comments.
RFF	Réseau Ferré de France.
RIF	Rule Interchange Format.
RINF	Register of Infrastructure.
RINM	Rail Infrastructure Network Model.
RL	Rule Language.
RRUKA	Rail Research UK Association.
RS	Rolling Stock.
RSSB	Railway Safety and Standards Board.
RTPI	Real Time Passenger Information.
SaaS	Software-as-a-Service.
SCADA	Supervisory Control And Data Acquisition.
SDEF	Signalling and Data Exchange Format.
SNOMED CT	Systematized Nomenclature of Medicine Clinical Terms.
SPARQL	Sparql Protocol and RDF Query Language.
SPIN	SPARQL Inferencing Notation.
SQL	Structured Query Language.
SSN	Semantic Sensor Network.
SVG	Scalable Vector Graphics.
SWRL	Semantic Web Rule Language.
TAF	Telematics Applications for Passenger Services.
TAP	Telematics Application for Passengers.
TBC	Topbraid Composer.
TCP	Transmission Control Protocol.
TLOC	Train Locator.
TOC	Train Operating Company.
TOCs	Train Operating Companies.
TOVE	Toronto Virtual Enterprise.
TRUST	Train Running System TOPS.
TSC	Transport Systems Catapult.
TT	Timetable.
Turtle	Terse RDF Triple Language.
UIC	International Union of Railways.
UK	United Kingdom.
UML	Unified Modelling Language.
UNA	Unique Name Assumption.

URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
VCS	Version Control System.
VoID	Vocabulary of Interlinked Datasets.
W <sub>3</sub> C	World Wide Web Consortium.
WILD	Wheel Impact Load Detector.
WILM	Wheel Impact load Measurement.
WTT	Working Timetable.
WWW	World Wide Web.
XML	eXtensible Markup Language.
XSD	XML Schema Definition.

## INTRODUCTION AND PROBLEM STATEMENT

---

### 1.1 PROJECT BACKGROUND AND PROBLEM STATEMENT

Railways across the world have enjoyed a revival in recent years, with passenger numbers growing in many countries as a result of increased fuel prices, traffic congestion, and a shift in public opinion to more environmentally friendly methods of transport [48]. In the United Kingdom (UK), passenger numbers have increased dramatically in the last decade [152], and this resurgence in popularity has led to continued investment in railway infrastructure and operations [150] to provide more capacity and reliability. As the returns on investment from ‘easy wins’ such as platform extensions and re-signalling diminish, and in light of recent government pressure towards efficiency savings [212], methods for making better use of existing assets are increasingly being sought. One such method is to encourage better use of data management principles and modern Information Technology systems, as noted by the 2012 UK Rail Technical Strategy [175]. A key motive for developing such systems is to allow greater *information sharing* between stakeholders, in order to support better decision making and thus encourage greater efficiency, reliability, and safety.

This thesis seeks to provide ways to encourage information sharing, by using semantic data models to provide machine-interpretable context around railway data. This first chapter overviews the current challenges and circumstances around railway data, outlines the aims and contributions of the PhD project, and provides an overview of the rest of the document.

#### 1.1.1 *Railway Fragmentation and Failure*

In spite of recent investment, the UK railway is highly fragmented. Much of its infrastructure originates from lines constructed during the ‘railway mania’ of the 1840s, during which time economic disincentives and a lack of standardisation encouraged the creation of many heterogeneous systems that are still in use today. The privatisation and subsequent re-organisation of the railways in 1993 led to heavy fragmentation of the stakeholders that operate it, with many different companies now responsible for operating, maintaining, and regulating different aspects and subsystems, as shown in [Figure 1.1](#).

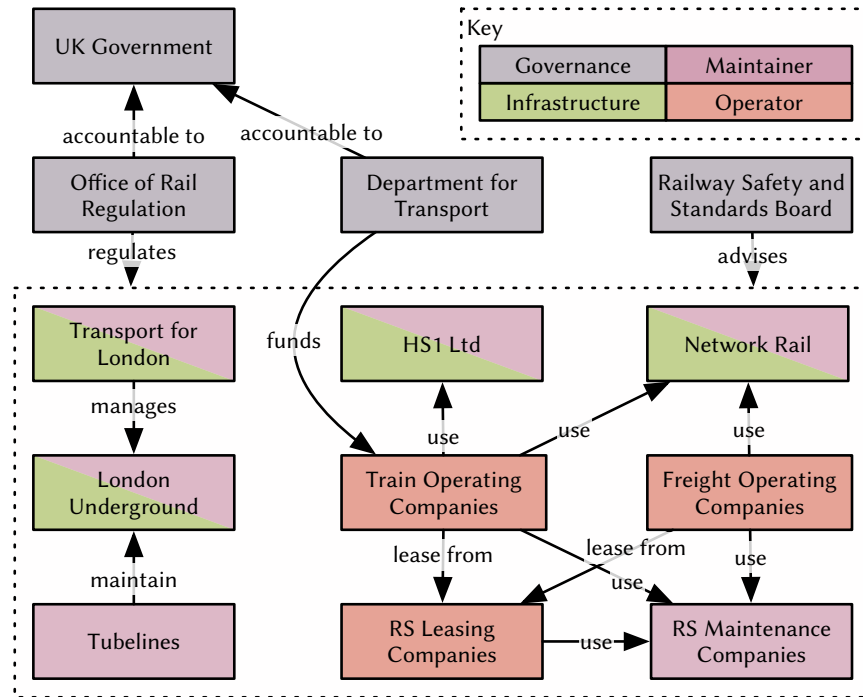


Figure 1.1: Interactions Between Major UK Railway Stakeholders (Data Taken From Office of Rail Regulation [151])

This fragmentation has led to the proliferation of independently owned and operated information systems across the industry, with little regard for exchange or sharing of knowledge between parties. Railway stakeholders that own few physical assets place immense value in the knowledge they can gather, and are not incentivised to share it unless necessary [184].

### 1.1.2 Information Silofication

In addition to business incentives for keeping information private, information sharing also faces technological and operational barriers. The long lifecycles and safety-critical nature of many railway information systems means that applications are often difficult to modify. The McNulty report [212] observes that a lack of ‘systems thinking’ has led to continued development of heterogeneous systems and bespoke interfaces between systems, often with complex, manual workflows bridging the gap. It estimates that over 1700 independent information systems are in use in the UK rail industry today, and concludes that:

“...the effectiveness of the industry’s IS [Information Systems] is inhibited by a suite of legacy systems that are expensive to run, unable to communicate with new tech-

nology and encourage users to develop a wide range of bespoke local systems to overcome limitations.” and “...many systems were felt to be inflexible, intertwined and increasingly difficult to maintain and enhance. This has resulted in additional (rather than replacement) applications and business practices being implemented to plug the gaps and to support the emerging devolved rail industry model.”

To illustrate this point, Figure 1.2 shows the current system layout of a set of passenger information systems in the UK railway.

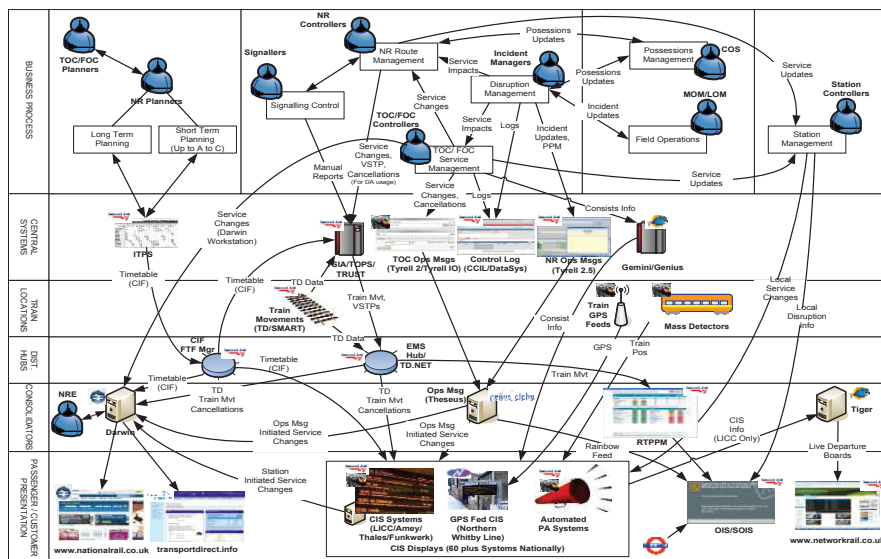


Figure 1.2: UK Railway Customer Information Architecture (Taken from [53])

These heterogeneous information sources often possess few or ill-defined system interfaces, leading to the creation of *information silos*—large repositories of potentially useful data that is inaccessible to other system, and therefore of no use in domain-wide decision making. Recent European Commission data sharing mandates such as the Register of Infrastructure [59] that aim to provide interoperability between European railways have presented difficulties to many railway undertakings, with Network Rail having identified at last 32 systems that must be modified to provide conformance [14].

### 1.1.3 The Data-driven Railway

It is not just government recommendations that provide incentives for embracing new information systems. The industry is also beginning to recognise the potential for new technologies for decision support

and automated processing within the railway, such as remote condition monitoring systems that aid with detection and diagnosis of faults before they occur [15]. Newer systems, such as Network Rail's Intelligent Infrastructure project [222] acquire large amounts of data about asset health, but currently confine its use to only its immediate application: that of thresholding and triggering alarms on faulty assets. This creates further information silos, and encourages unnecessary duplication of data.

#### 1.1.4 *Standardised Data Models*

As the number of information systems employed in the railway industry increases, so too does the need for long term approaches for management of data. Domain-wide standard data models provide a shared vocabulary with which to exchange data, and several prominent efforts have already been made in the railway domain, as discussed in [Chapter 3](#).

Semantic data models provide a way of storing information whilst preserving its context. In contrast to traditional approaches to data modelling and storage, semantic models encode the *meaning* of data by relating it to a logical conceptualisation of the world, allowing computers to more easily integrate data from multiple sources. This reduces reliance on one particular data model structure, allowing greater flexibility—a feature which is necessary considering that some systems have lifespans of 25–50 years. Consequently, semantic models provide several advantages over relational databases, eXtensible Markup Language (XML) schemas, and other syntax-dependent information exchange formats when considering the demands of the railway industry and other complex systems:

- **Preservation of data meaning** across system interfaces and upgrades [107]. The semantics of each term used in a system are defined by the relationships they share with other concepts, and can be stored alongside the data itself. This reduces ambiguity and preserves the intent of the information stored.
- **Ease of system integration:** semantic models encoded in Resource Description Framework (RDF) encode data as knowledge graphs, and so do not depend on any fixed structure beyond that of representing the facts themselves. Terms that share semantics can be mapped and queried together, facilitating easier combination of knowledge.

- **Model extensibility** [33]: semantic models can easily be extended incrementally over time, allowing new concepts and knowledge to be introduced without re-designing entire standards and systems.
- **Derivation of new knowledge** [84]. By understanding the context of data within a system, semantic models can be extended through a process of machine reasoning, to add ‘common sense’ and business rules to data stores, and materialise additional knowledge within a system.

For these reasons, semantic models have been used successfully for data integration in applications such as oil and gas engineering [226], medicine [200], and journalism [178]. The uptake of semantic modelling on the internet, in the form of the Semantic Web and Linked Open Data initiatives, has brought about a set of standardised tools and technologies on which to implement ontologies and build applications, and many of these technologies are now used widely in production environments [50].

## 1.2 AIMS AND ORIGINAL CONTRIBUTIONS

This work presented in this thesis builds on the current state-of-the-art in ontology engineering methodologies and techniques to develop new ways to aid data exchange in the railway industry. In addition to providing an overview of current literature in these areas, the thesis seeks to answer the following research questions:

- How should domain ontologies for industrial data sharing be designed and built using existing non-ontological information resources and expert knowledge
- How can such models be evaluated and validated?
- What are the key use cases and over-arching requirements for a domain ontology that supports data sharing and integration in the railway industry?
- Can Web Ontology Language (OWL) and semantic web technologies be used to create a candidate domain ontology for infrastructure-based data sharing in the railway industry, and can such a model be used to infer new knowledge from existing information?
- How can asset monitoring and Real Time Passenger Information (RTPI) systems take advantage of and implement these models, and how are such applications achieved using current off-the-shelf technologies?



To answer these questions, three principal novel contributions are presented:

1. **Chapter 4** describes a new methodology for knowledge elicitation and design of domain models for large complex industrial systems, based on current state-of-the-art.
2. In **Chapter 5**, A set of OWL domain ontologies and vocabularies for the railway domain are presented, as well as a set of corresponding design patterns for representing key railway concepts to allow future extension for specific tasks.
3. **Chapter 6** documents the development of two demonstration information systems utilising these rail ontologies and extensions to them, based on identified rail industry use cases. Documentation of novel ontology design patterns for use in other railway applications is also provided.

### 1.3 THESIS ORGANISATION AND STRUCTURE

The rest of the thesis is structured as follows:

- **Chapter 2** provides an overview of the principles of knowledge management and data modelling. It describes semantic data models, ontologies, and their uptake on the World Wide Web, which has given rise to many of the tools and technologies used to undertake this project.
- **Chapter 3** describes the current state-of-the-art in industrial data models, as well as current workflows and models used within the railway industry and other large complex systems.
- **Chapter 7** provides an evaluation of the work undertaken, outlines conclusions reached, and describes both potential future work arising from this thesis, and current or upcoming projects utilising its outputs.

### 1.4 PROJECT PARTNERS

This thesis was produced as the result of an Engineering and Physical Sciences Research Council Collaborative Awards in Science and Engineering (CASE) Studentship between Invensys Rail Group and the University of Birmingham.

Invensys Rail Group were primarily a provider of railway control, signalling, and safety systems. In 2012, the company was bought by

Siemens PLC, and became part of the Siemens Intelligent Transportation Systems group. This merger in turn led to a collaborative project (discussed in [Chapter 6](#)) with another part of the Siemens Intelligent Transportation Systems (ITS) group, based in Ashby-de-la-Zouch, Leicestershire.

For consistency, the name ‘Invensys Rail’ is used throughout this thesis to refer to the organisation based in Chippenham and Birmingham, and ‘Siemens’ is used to refer to the group based in Ashby-de-la-Zouch.



## MODELLING, ONTOLOGIES, AND THE SEMANTIC WEB

---

### 2.1 INTRODUCTION

To contextualise the aims and research questions set out in [Chapter 1](#), this chapter provides an overview of the fields of knowledge representation and semantic data modelling, and sets out the conventions adopted to document and explain concepts in subsequent chapters. [Section 2.2](#) introduces key data modelling principles and techniques, and outlines current types and approaches to knowledge management. In [Section 2.3](#), the concept of semantic data modelling is introduced, and the advantages of semantic data models in assisting data interoperability across systems are shown. [Section 2.4](#) discusses the use and characteristics of ontologies, and how reasoning can be used to infer implicit knowledge from a set of facts. The final section, [Section 2.6](#), shows the state-of-the-art in technologies for creating and using computer ontologies and linked data, and outlines technical notation used throughout the rest of the thesis.

### 2.2 DATA MODELLING AND KNOWLEDGE MANAGEMENT

Since the advent of affordable computing in the 1960s, there has been a steady increase in the number of organisations and individuals exploiting information systems to automate business processes and store information. As the information processed by these systems became more complex, ad hoc methods for representing data were no longer practical, and new techniques and paradigms for doing so gained traction. *Data modelling* is the practice of defining the structure and representation of data to be stored in such a system, with the purpose of allowing the information present to be correctly preserved and recalled. Data models provide a mapping between real life concepts and their physical representation on a computer storage device, and ensure that data quality is ‘fit for purpose’ within an application [226, p. 6]. As such, data models vary in complexity and scale depending on their requirements—from simple tabular structures that support individual tools or applications that depend on *implicit* domain knowledge for users to make sense of them, to intricate enterprise-wide models that support interoperability across an organisation by mak-

ing the meaning of data *explicit* in the models themselves. The development of the World Wide Web (WWW) has brought about a similarly diverse set of models for representing data on the internet, ranging from unstructured representations of interlinked knowledge in standardised markup formats such as Hypertext Markup Language (HTML), to machine-readable representations used by search engines and software applications, and discussed in [Section 2.5.2](#).

To provide this mapping between concepts and data storage, software engineering approaches often advocate the design of several models, each at a particular level of abstraction between the real world and the machine. Types of data model can be categorised into the following three main categories [94, p. 6], defined as follows:

- **Conceptual models** describe the concepts and behaviour of a problem domain or system. They are implementation-agnostic, and do not attempt to specify behaviour of the system itself. Conceptual models may be specified formally (using a meta-model) or informally, depending on their requirements.
- **Logical data models** describe the high level structure and relationships of the system to be implemented, according to some set of rules or expressiveness. They introduce devices to allow conceptual data to be encoded by computers (such as relationships and attributes), but are not defined by a particular implementation themselves.
- **Physical data models** are the manifestation of the logical model within a defined technological implementation. Common physical modelling formats include eXtensible Markup Language (XML) and relational databases.

In trivial cases, physical models are defined ‘ad hoc’, with no formal logical or physical representations. However, as complexity increases, the structure of such physical models becomes less intuitive. When considering larger systems, for example railway ticketing databases, formal data models reduce ambiguity and aid interoperability, and are a key component in several systems engineering methodologies [125, p. 42].

### 2.2.1 *Application and Domain Engineering*

As a result of the ambiguity inherent in many data models, many large complex systems include components that can be regarded as *information silos*: applications or data stores where useful information

is held but cannot easily be extracted. Applications using conflicting data models (at either a logical or a physical level) cannot exchange data without some bespoke interface to translate between systems, and design of such interfaces is often expensive or unfeasible[216]. This heterogeneity can be eased somewhat by the introduction of common data models, shared by several applications. Such models can reinforce the consistent meaning of concepts across multiple applications, and therefore mitigate the likelihood of information silos forming in the first place.

In the absence of wider motivations for interoperability, software design methodologies base the creation of an application's data model on pre-defined sets of functional requirements. These requirements specify what information should be represented, and rarely consider the needs of surrounding systems [169]. This *application engineering* approach can establish comprehensive models that suit one particular application, but may make assumptions that do not hold in the wider domain of discourse in order to reduce complexity and increase system efficiency [192]. The actual domain knowledge required of the system for interpretation is often ad hoc, implicit (assumed) and informal [116]. In enterprise environments, this limits re-usability and interoperability between systems, as different assumptions and requirements can lead to different approaches to modelling the same types of information.

Consider a railway maintenance system and a passenger information system. In the former, a *train* may be defined as a physical thing consisting of a number of unique carriages with serial numbers. In the latter, however, a *train* may refer to a service that can be fulfilled by any physical vehicle. Although these are both perfectly reasonable definitions to make in the domain of each system, different assumptions made in each case may be incompatible and require a translation layer to provide interoperability.

*Domain engineering* seeks to mitigate these incompatibilities by creating models that identify key concepts and interactions across a domain of discourse, rather than those confined to one application's requirements [169]. These *domain models* are agnostic of any particular application, and are often designed at a higher conceptual level than their application-specific counterparts, allowing the types of assumptions necessary in physical models to be avoided altogether. By building on domain models, applications across an organisation can encode information using shared definitions of concepts, facilitating

easier re-use and exchange of data [154].

Successful software design for interoperability utilises both approaches [94], with standard domain-specific vocabularies used to represent the semantic of key concepts, and application-specific extensions to provide additional functionality and encode these abstract concepts into logical and physical models. Figure 2.1 shows how these two disciplines may interact during development of a particular system.

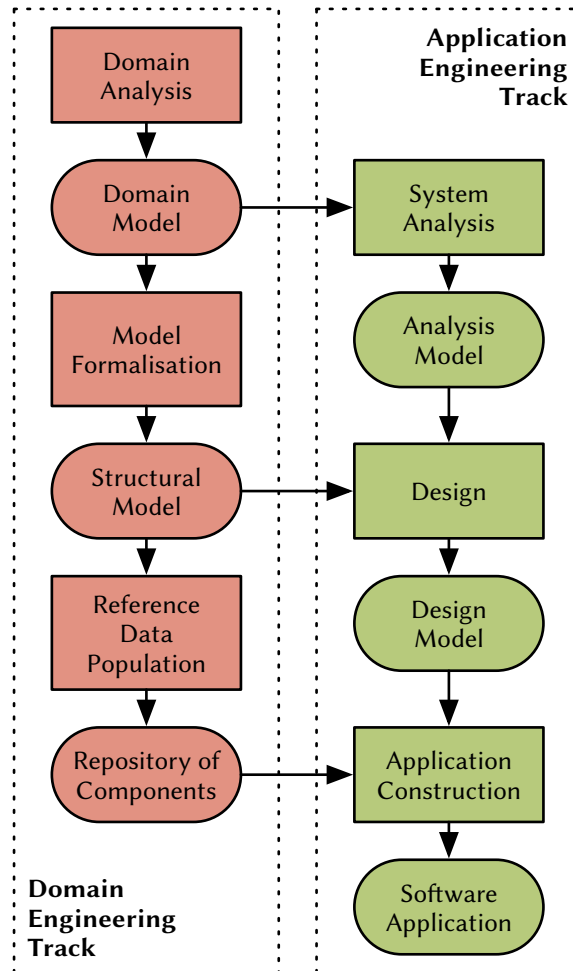


Figure 2.1: Interactions between Domain Engineering and Application Engineering, adapted from Falbo et al. [60]

### 2.2.2 Incentives for Standardised Models

The work presented in this thesis centres on the use of standardised domain models to aid interoperability across large complex systems,

by providing a shared vocabulary and conceptualisations that applications can subscribe to and use. As an example of this, an extract from the Railway Safety and Standards Board (RSSB) Rail Functional Architecture, a recently created conceptual model for functions across the rail industry, is shown in Figure 2.2 [176] and discussed in more detail in Chapter 3.

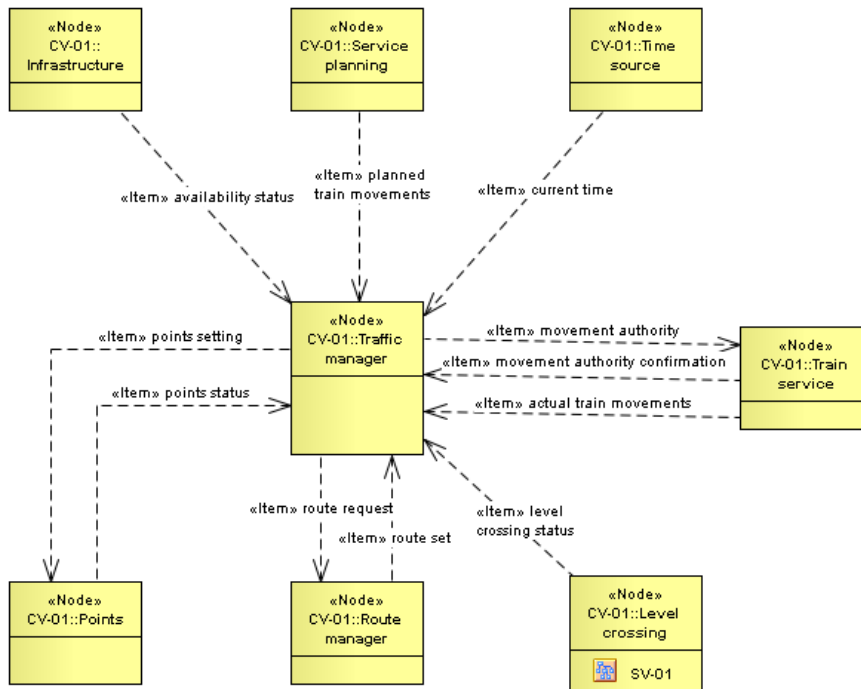


Figure 2.2: Extract from the RSSB Rail Functional Architecture

The interoperability advantages of a standardised domain model within an industry can be illustrated quite clearly by considering the need for interfaces between different systems, as shown in Figure 2.3. If we make the assumption that each pair of systems with differing representations of data require some form of reconciliation interface between them in order to communicate, then the number of such interfaces for a given complex system can be estimated.

For small applications with just two interoperating systems, only one interface is required: the one between system A and system B. If a third system must exchange data with both systems, then two more interfaces are required, so three in total. As the information landscape grows, more interfaces are required—in fact, to enable all  $N$  applications to communicate, the number of interfaces needed is  $\frac{N(N-1)}{2}$ . Furthermore, when one system is modified,  $N - 1$  interfaces also require modification!

However, if a standard data model is implemented, capable of express-



ing all information exchanged by systems, the number of interfaces drops to  $N$  interfaces—one for each system. In addition, modification of one system no longer requires alteration to many different interfaces.

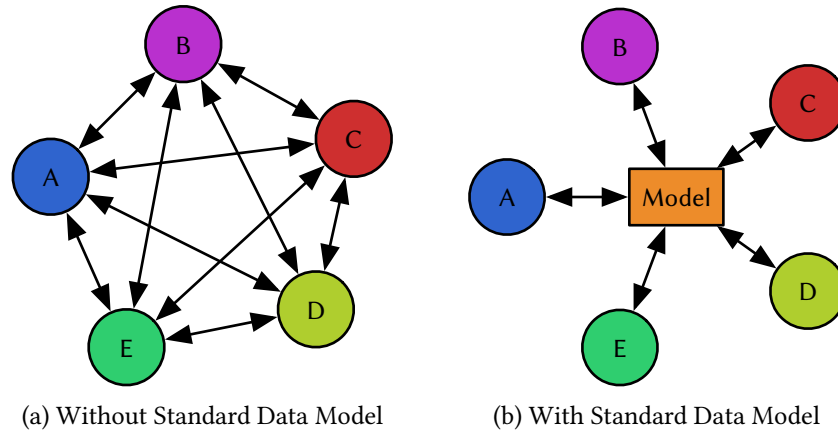


Figure 2.3: Interfaces in Complex Systems

Dickerson and Mavris [49] highlights other advantages to subscribing to standard data models within an enterprise. In addition to no longer requiring bespoke interfaces between every platform, software re-usability across systems increases: if applications are designed using common concepts, then modules that serve a particular function can be re-used in other applications. Other stakeholders also benefit from the consistent use of terminology across the domain: they must no longer learn the caveats of how terms are presented in each system.

### 2.2.3 Syntax-based Models

Implementation of both domain and application data models relies not just on an appropriate conceptual model being created, but also on the translation of this model into a representation that can be used by computers. A common mechanism for achieving this is by using a syntax-based representation, storing the context and meaning of data according to the physical structure of a document [9, ch. 5]. To define a syntax-based physical model, a logical representation is first created that encodes and documents these concepts and their attributes as an appropriate data structure. This structure is then used as the basis for a physical data model, which is defined according to the chosen implementation technology. Commonly used logical and physical modelling formats include:

- Relational databases, in which a logical model is stored in the form of a database schema. Information is retrieved by forming queries over the database according to a schema, which are used to extract and combine data from multiple tables.
- eXtensible Markup Language (XML), which encodes data using textual markers (tags) in a file. An XML Schema Definition (XSD) defines tag structure, terms, and constraints, allowing users to establish the meaning of data.
- Spreadsheets, which encode data into a tabular form, and define meaning by table headings and additional metadata.

By adhering to a known syntax, systems can deduce the meaning of a piece of data based on its position within the store's structure, and interpret it accordingly. Data shown in [Table 2.1](#), for example, can be interpreted via its table headers, which allows us to establish that the piece of information that intersects row four (*Bill*) and column three (*Primary Instrument*) is the datum that represents that Bill Evans' primary instrument is piano. [Listing 2.1](#) shows how the same information could be represented in XML form.

Table 2.1: 1950s Jazz Musicians and Key Personal Information

First Name	Last Name	Primary Instrument	Birth date	[...]
Miles	Davis	Trumpet	26/05/26	71
Cannonball	Adderley	Saxophone	15/09/28	48
Bill	Evans	Piano	16/08/29	52
Michael	Brecker	Saxophone	29/05/49	39
Bill	Evans	Saxophone	09/02/58	24

```

<?xml version="1.0" encoding="UTF-8" ?>
<Musicians xmlns="http://purl.org/ub/ex/jazz"
  ↪ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ↪ xsi:schemaLocation="http://purl.org/ub/ex/jazz
  ↪ jazzschema.xsd">
  <jazzMusician>
    <Name>
      <FirstName>Bill</FirstName>
      <LastName>Evans</LastName>
    </Name>
    <PrimaryInstrument>Piano</PrimaryInstrument>
    <BirthDate>1929-08-16</BirthDate>
  </jazzMusician>
  <jazzMusician>
    <Name>
      <FirstName>Miles</FirstName>
      <LastName>Davis</LastName>
    </Name>
    <PrimaryInstrument>Trumpet</PrimaryInstrument>
    <BirthDate>1926-05-26</BirthDate>
  </jazzMusician>
</Musicians>

```

Listing 2.1: Example XML Markup of [Table 2.1](#)

An example XML schema is shown in [Figure 2.4](#), which illustrates the logical data model to which data should adhere. Schemas are useful for specifying data structures, often provide human-readable documentation, and allow for validation of information. They do not, however, allow machine understanding of data; instead, software designers write business and application logic to extract the information they require as necessary.

Whilst XML schemas are widely used for industrial and enterprise applications, well-supported public standards also exist. Examples include Scalable Vector Graphics [42], and the Web Service Description Language [34].

XML and similar models are successful in scenarios where data concepts and features can be well-defined and do not change over time. If agreement on the nature of each term in the model is consistent among applications and developers, information can be preserved and exchanged between systems. This is, however, not always the case, as detailed in the following section.

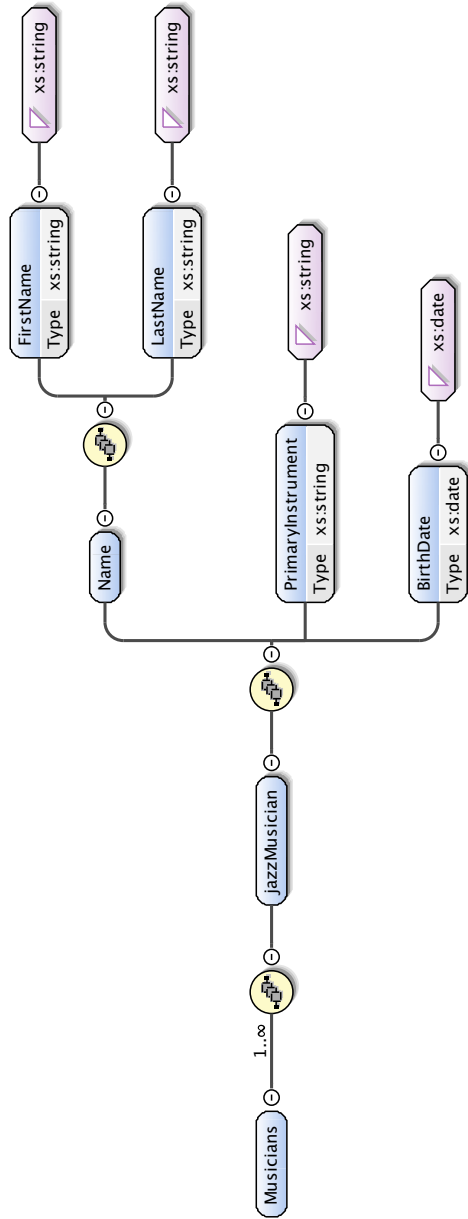


Figure 2.4: Graphical Representation of Possible XML Schema for Data from Table 2.1

## 2.3 SEMANTIC MODELLING

Semantic data models are a family of models that better allow meaning and context to be stored alongside information, independently of a fixed structure or syntax. Rather than using syntactic cues to locate and store data, they define information by its inter-relationships, much as humans think of facts, so that a concept's meaning can be derived from its relationships. In natural language, information and semantics from [Table 2.1](#) could be described as:

Bill Evans is a jazz musician. Bill Evans' primary instrument is the Piano Bill Evans' birthday is 16/08/29

Additionally, further facts can be added to describe the other concepts mentioned: **piano**, **birthday**, **musician** and so on, to create a view of the world containing this knowledge. Rules and characteristics (such as 'a piano has 88 keys') can also be expressed, allowing semantic models to further contextualise information. As such, data can adhere more closely to an initial conceptual model of the domain, without relying on any particular physical representation of the data beyond the format in which the semantic model itself is represented.

The result of creating these *knowledge graphs* is that semantic models allow computers to process data based on its *meaning*, rather than on its predefined position in a data store. An analogy to this distinction is in how the author's partner and the author's grandmother interact with technology, such as placing a call on a mobile phone. This is illustrated in [Table 2.2](#)

Table 2.2: Making a Phone Call

Jo	Diana
1. Turn on the phone	1. Press the top button
2. Navigate to 'Phone Book'	2. Press 'up', then 'left', then 'select'
3. Search for 'Jon'	3. Press '4' then 'down'
4. Press 'call'	4. Press 'select'

Jo interacts with the phone by understanding the meaning and context of the information it presents her; she could use a different phone and carry out the same task, subject to her domain knowledge being good enough. Diana, on the other hand, has no knowledge of how mobile phones work, and simply inputs a set of instructions to obtain the result. Both methods are successful, but by using semantic cues,

Jo is able to continue to carry out the task as the structural information she is presented with changes (such as on a new phone), whereas Diana is not.

The main characteristics of semantic data models are described in the following subsections.

### 2.3.1 *Flexibility in Knowledge Representation*

Many knowledge management challenges encountered by organisations occur when architectural changes occur within the information landscape, such as de-commissioning/replacement of systems, addition of features to suit new requirements, or changes from external interfaces. Often, data models must be altered to facilitate new information arising from such changes. In syntax-dependent models these changes can be difficult, particularly when complex data structures require modification. Tools such as XML, Structured Query Language (SQL) and spreadsheets rely upon the position of data to deduce its meaning and act on it; changes to the way in which information is stored to incorporate more data can require potentially expensive system modifications. Semantic models avoid this reliance on syntax—applications deduce the meaning of terms based on their relationships, regardless of how they are stored. Altering the data model of a syntax-dependent system often requires modification of all the systems that use it, whereas extending a semantic model simply requires the addition of new axioms and definitions.

To show the distinction in flexibility between syntax-reliant data models and semantic models, the example of elaborating on data provided in [Table 2.1](#) is discussed. Miles Davis does indeed play the trumpet, but specifically plays Bb trumpet, and also piano. Cannonball Adderley in fact plays both tenor and alto saxophones, and collaborates with Miles Davis.

To represent this knowledge in a syntax-based format is challenging, and requires re-definition of the schema. In a semantic model, however, new assertions can be added instead, extending the model whilst preserving compatibility, as shown in [Figure 2.5](#).

Applications based on the initial semantic model still work; answering the query ‘List all saxophonists’ will provide the same results. In the syntax-based model, the answer to this query depends heavily on implementation; an XPath or SQL query for ‘list all saxophonists’ now requires modification to facilitate the extra level of expressiveness in the data model.

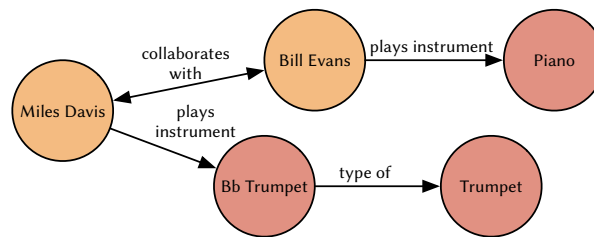


Figure 2.5: Example Semantic Data Model Using Data Modified from [Table 2.1](#)

### 2.3.2 Preservation of Context

Another important implication of the flexibility provided by taking this approach is that of the preservation of information. Rather than constraining data input to a pre-defined set of attributes and fields using a schema, knowledge bases operating with semantic models can easily convey extra information that exceeds the scope of the designed model, by providing informal extensions. For example, the fact that ‘Miles Davis visited Paris’ can be encoded in the model using some (informal or undefined) mechanism, and processed later. The rise of modern ‘NoSQL’ schema-less databases can be partly attributed to this advantage, allowing organisations to spend less time perfecting database schemas for every conceivable eventuality, and instead allowing them to evolve over time [93].

### 2.3.3 Interoperability and Expressivity

Data exchange and sharing across heterogeneous data sources requires some form of interoperability between systems. Such interoperability can exist at a number of levels, from the physical layers surrounding the systems themselves to the conceptual level discussed above. Amit Sheth [191] summarises four levels of interoperability as *system*, *syntax*, *structure*, and *semantic*:

- **System-level** or technical interoperability concerns the ability of systems to communicate physically with one another, and has been addressed since the 1960s by now ubiquitous communication standards such as TCP/IP and standardised data encoding formats. At this level, data must still be translated or aligned in some way between each system to allow information to be shared.
- **Syntactic interoperability** is the ability of systems to share and understand some type of document syntax (for example XML or

spreadsheet syntax), so that software can use concepts in these syntaxes to extract pieces of data. Systems that are only syntactically interoperable require further mappings to align information used in them to each other.

- **Structural or schema interoperability** is achieved by systems adhering to the same data model or schema, such that the categorisation of a piece of data is preserved and its meaning can be inferred by context from a corresponding schema. Many modern computing standards achieve interoperability at this level; railway-specific implementations include RailML and An Operations and Maintenance Information Open System Alliance (MI-MOSA) Open System Architecture for Condition-based Monitoring (OSA-CBM), as discussed in [Section 3.3](#).
- Full semantic interoperability allows systems communicating with each other to interpret data and its meaning independently unambiguously, without a schema to set out all valid and invalid terms. Instead, all of the knowledge required to interpret a model is defined alongside the data itself, such that systems receiving data can interpret and integrate it.

In practise, widely-used syntax-dependent technologies can only facilitate schematic interoperability, and ‘semantic clash’, where similar terms cannot be reconciled due to differently defined meanings, is common [36]. Semantic data modelling techniques provide some level of semantic interoperability to solve this problem. Complete (ideal) semantic interoperability is unfeasible—each concept requires a semantic description provided by a number of other concepts, each of which must also be defined, leading to a potentially infinite number of axioms. Thus, frameworks for semantic modelling define meta-models of standardised terms and features [89], onto which conceptual/domain models are built. Many ontology languages exist, and are discussed in [Section 2.4](#).

For systems that use the same ontology language, ease of information sharing is greatly increased over those using any kind of syntax-based model, even if they use different conceptual models. Although concepts may be labelled differently across models, if their meaning is the same, alignment of data can be done by introducing simple new rules. Given models A and B as shown in [Figure 2.6](#), asserting that ‘every musician who has composed something is a composer’ and ‘every composer is also a musician’ provides a mechanism for a computer system to align the two models.

For systems using either the same conceptual model or a derivative of it, interoperability is even easier. Those that extend a conceptual



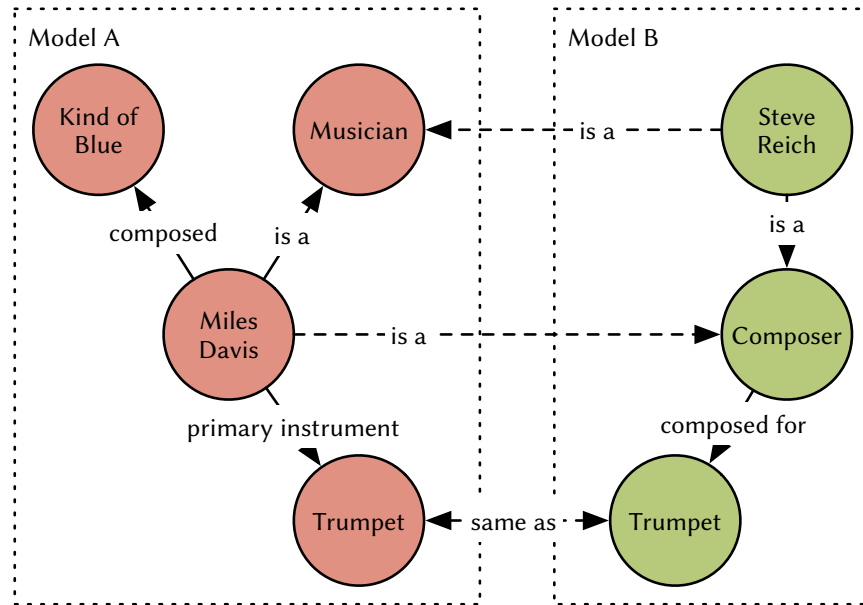


Figure 2.6: Alignment of Semantic Data Models Using Mapping Axioms

model can use underlying generalisations to allow other systems to take advantage of data, even if it is described in a more prescriptive way than the system understands. This is shown in [Figure 2.7](#). Model A now contains information about Miles Davis' place of birth, which can be exploited to infer that he is an American using another set of axioms (not shown) in model A and concepts in the common model. Queries to the set of models can now include Miles Davis in any list of American Musicians, along with those asserted as such in model B.

#### DEALING WITH AMBIGUITY IN SEMANTIC DATA MODELS

When attempting interoperability between data models, the possibility of ambiguity arises when multiple interpretations of one piece of data can be derived, as often occurs in natural language. This problem is alleviated in semantic models, as terms can be disambiguated by considering their position in the model and their relationships to other concepts, unlike in traditional database or XML systems. This characteristic has been repeatedly used to solve disambiguation problems in a number of areas, including natural language [219], entity recognition [136], and geographical mapping [218].

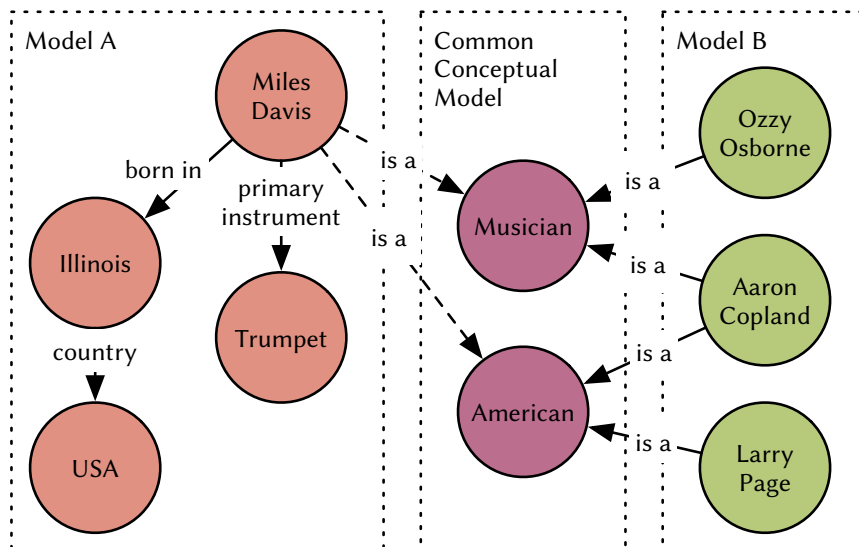


Figure 2.7: Example of Generalisation and Interoperability in Semantic Data Model

## 2.4 KNOWLEDGE REPRESENTATION AND ONTOLOGY

This section overviews the meaning of the term *ontology*, and describes ontologies, ontology engineering, and machine reasoning as used in the context of work undertaken in chapters 5-7 of this thesis. It explains key principles used in the development of semantic models, and outlines distinctions between ontologies and traditional data modelling approaches. Current technologies and notation for developing ontologies are discussed further on, in [Section 2.6](#).

### 2.4.1 What Is An Ontology?

The term ‘ontology’ is shared between two inter-related but distinct fields of research sharing the name; one in philosophy, and more recently in computer science. In philosophy, *ontology* is the study of meta-physics, or:

“[...] a philosophical theory concerning the basic traits of the world” [29]

Philosophical ontology studies the nature of being, and identifies how entities exist and interact with each other by studying their classifications and relationships between each other [28]. Within computer science, the term is used in a slightly different context. Here, the study of ontology arose from a need to understand how to formally

represent knowledge in database systems, software systems, and artificial intelligence [196], and concerns the creation of concrete models that encode knowledge about the world into a usable document with some known formalism. Thomas Gruber’s widely-cited definition states that *an ontology* is:

“[...] an explicit specification of a conceptualization.” [83]

Rather than representing an abstract philosophy about how the world works, a computer ontology can be described as a set of concepts, types, properties, and their interrelationships that in some way describe a view of a particular domain or subject area. For the purposes of discussion in this thesis, a computer ontology can be thought of as a **formal encoding of a conceptual data model**.

A basic ontology describing interactions in pop music along with some sample instance data is given in Figure 2.8, created by forming a simple conceptualisation of the domain based on facts known about entities and the relationships they have with one another.

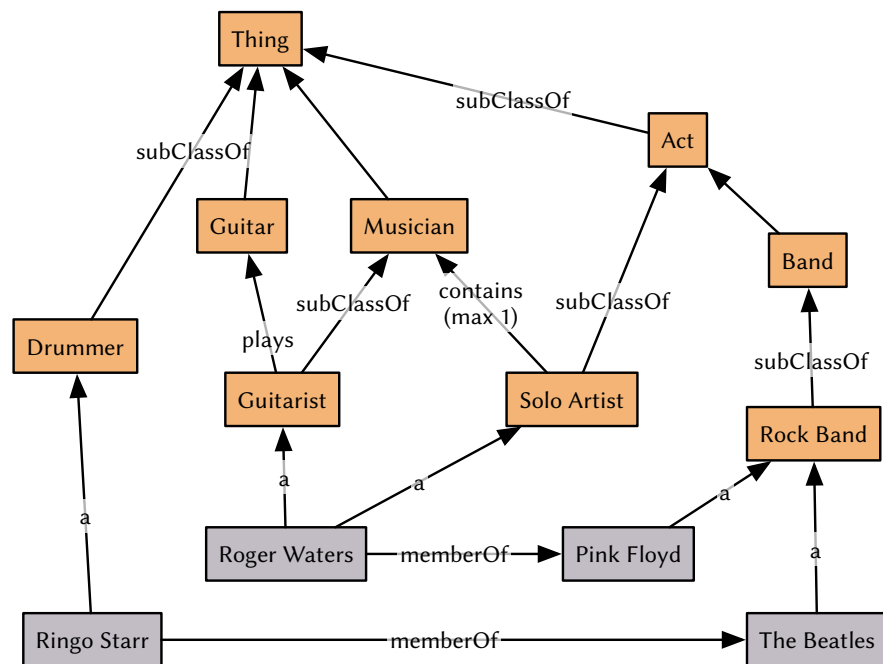


Figure 2.8: Illustration of Example Pop Music Ontology

#### 2.4.2 Semantic Expressivity and Ontology Languages

In practise, there is no concrete distinction between ‘syntactic’ and ‘semantic’ data models, and representations take a number of forms,

each of which provides some level of semantic expressivity. Computer ontologies are built using *ontology languages*: pre-defined rules and constructs that provide a semantic framework for expressing knowledge. These languages vary in the expressivity they offer and can be placed on an ‘ontology spectrum’, ranging from those that provide few assumptions and assume fixed or implicit semantics (and thus are less formal) to those that are highly expressive and provide detailed, formal semantics (and thus high interoperability). This ontology spectrum is well illustrated in Figure 2.9.

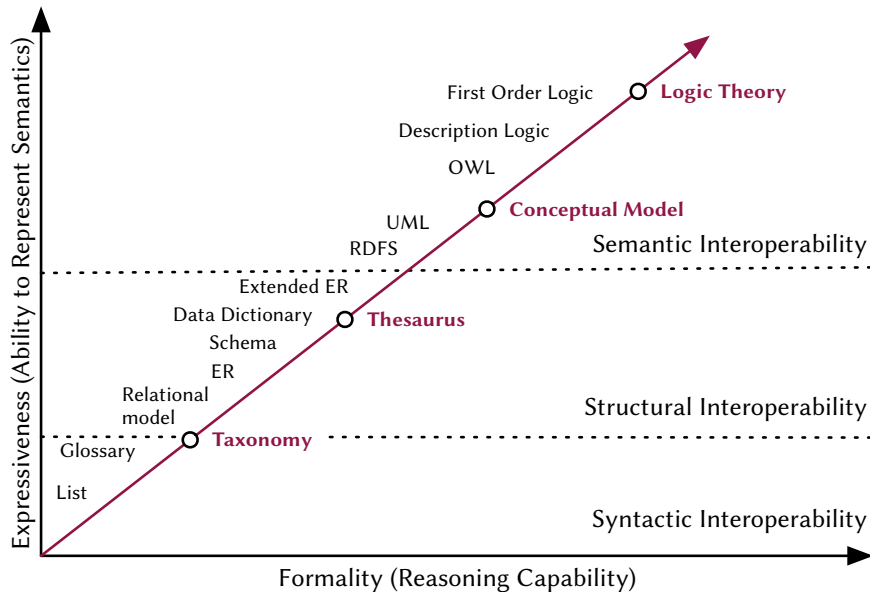


Figure 2.9: The “Ontology Spectrum”, taken from Obrst [149]

On the left hand side, informal data catalogues simply provide system designers with agreed symbolic ways of representing concepts, such that information can be exchanged using a known, controlled vocabulary. Glossaries additionally give natural language definitions for terms to aid disambiguation, but no machine-readable data semantics are captured. Thesauri extend upon this basic syntactic interoperability by encoding basic relationships, such as synonyms (equivalence) and associations across concepts. Taxonomies (trees) provide more expressivity by allowing ‘type of’ hierarchies to be represented in some way, such that computers can entail additional information about terms.

Example taxonomies in widespread use include WordNet [137], which provides semantics for lexical analysis of words, and Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT)<sup>1</sup>, which is

<sup>1</sup> <http://snomed.org/>

used for electronic healthcare and prescribing.

In order to represent richer information about the world, further flexibility is required. For instance, representing the fact ‘Roger Waters plays guitar’ in a taxonomy is difficult; we could represent his membership as a ‘guitarist’, but the fact that guitarists play guitar must be stored implicitly or in natural language. Ontologies expressed in *description logics* or *frames* provide this extra level of expressiveness, introducing a distinction between classes and individuals (concepts, and instantiations of concepts) and allowing arbitrary relations to be expressed between concepts. The characteristics of these relations can also be expressed in a machine-readable way, allowing machines to infer information based on the relationships linking concepts. Description Logic (DL) ontologies can provide a good trade-off between expressive power and computational complexity [7], and are described further below and in [Section 2.6.3](#).

Higher level ontology languages allow expression of increasingly complicated interactions between concepts, but become expensive to compute. DL-based languages, for instance, only allow binary relations (those that have one subject and one object), so certain real-world concepts (such as time) are difficult to properly represent. Encoding ontologies using full First Order Logic provides greater expressivity, but can be intractable [23, p. 329]; these and more expressive languages can be used for data exchange, but cannot be reasoned over.

#### 2.4.3 *Data Models, Vocabularies, and Ontology*

The approaches to data representation outlined so far can be outlined into three categories: traditional application-centric conceptual, logical, and physical data models as described in [2.2.3](#); controlled vocabularies with informal semantics; and formal ontologies with varying expressive capabilities as described in [Section 2.4.4](#). The main differences between these categories are as follows:

- **Data models** are designed for a particular application or suite of applications at inception. They are tightly coupled to these applications, and specified to represent the exact types of information and assumptions needed. They do not encode formal data semantics, but often enforce constraints in order to carry out data validation. Conceptual modelling techniques such as Unified Modelling Language (UML) can provide very rich data semantics and facilitate interoperability, but do not provide machine-interpretable ways of describing information context.

- **Controlled vocabularies** are, like ontologies, designed to model a domain of discourse itself, as opposed to modelling application data flow directly. They aim to abstract the representation away from a particular application with a view to encouraging data re-use. Controlled vocabularies are sets of terms that have been enumerated and defined unambiguously, such that each term can be interpreted correctly by systems with prior knowledge of the vocabulary's definitions. They may include detailed descriptions of the meaning and semantics of each term, but do encode these semantics informally and rely on logic encoded in an application to contextualise and interpret information.
- **Ontologies** are controlled vocabularies expressed in a formal ontology representation language, and capture the semantics of the domain of discourse in a machine-interpretable way, such that the meaning of data can be inferred through axioms present in the ontology itself, rather than relying on application logic. These axioms express how concepts in the vocabulary interact, so they can be used by a machine to infer new knowledge on information expressed using the ontology. Ontologies are again application-agnostic, and designed to model interactions in a particular domain or subject area, rather than data in a particular application.

Examples of the differences between controlled vocabularies and ontologies can be observed in the evolution of the semantic web and linked data movements, as described in [Section 2.5.1](#). Vocabularies such as Dublin Core<sup>2</sup> are widely used to unambiguously represent data on the semantic web and in other fields, but do not themselves provide ways for machines to make inferences based on the data; this is left to the application developer. In contrast, some web ontologies do exist and facilitate machine reasoning over data, such that further inferences based on the content presented directly can be made without the need for additional application logic. Examples include Geonames<sup>3</sup> and the Music Ontology [179].

#### 2.4.4 *Ontology Types*

In this thesis, ontologies are used predominantly for domain modelling and data exchange between practical information systems. Since this is not only one use of ontologies, it is useful to classify ontologies

---

<sup>2</sup> <http://purl.org/dc/elements/1.1/>

<sup>3</sup> <http://www.geonames.org/ontology/documentation.html>

and methodologies present in the literature. Firstly, they can be classified by purpose or intent:

- **Highly formal ontologies** are used in domains where exact semantics are required. Models designed for these purpose tend to be highly expressive, and heavily draw upon philosophical concepts in order to correctly represent terms.
- **Web ontologies** are typically far less expressive, and make assumptions about data types and applications in order to provide a concise schema-less data exchange format with some expression of semantics.
- **Domain-specific ontologies** are usually designed to be expressive enough to allow extensibility across a domain, but make assumptions in order for representation of information to remain practical. Some example domain ontologies are discussed in [Section 3.3](#).

Ontologies can also be classified according to a hierarchy based on use, as observed by Guarino [86].

- **Upper level ontologies** describe generic concepts common to all domains, such as space and time, a few examples of which are described in [Section 3.6.1](#).
- **Domain ontologies** may extend upper level ontologies, and represent knowledge of a particular area of interest.
- **Task ontologies** further specify domain ontologies by creating patterns according to a particular (cross-application) task.
- **Application ontologies** provide concepts and terms related to a specific application, such as information on views and configuration.

This hierarchy aligns well to the disciplines of domain and application engineering discussed in [Section 2.2.1](#).

#### 2.4.5 *Ontology Reasoning*

By encoding information and meaning in a formal way, facts that are included in ontologies and semantic data models can be exploited by computers in order to establish new knowledge, in a similar way to

how humans reason over information. This process is known as *logical inference*, and can also be used to aid data integration between systems and align models, as shown by the example given in [Section 2.3.3](#). Additionally, by implementing some logic about the world that the model shown in [Figure 2.8](#) might adhere to, it is possible to infer new facts about the model. For example, it could be said that:

- If  $A$  is a member of  $B$ ,  $B$  has member  $A$ .
- If  $A$  plays  $B$ ,  $B$  is played by  $A$ .
- If  $A$  is a subclass of  $B$ , then anything of type  $B$  is also of type  $A$

A reasoner using this logic could then infer that:

- Pink Floyd are an act, a band, and a rock band.
- Pink Floyd has member Roger Waters.
- Roger Waters is a musician.
- Roger Waters plays guitar, and guitar is played by Roger Waters.
- The Beatles have Ringo Starr as a member.

Furthermore, if we introduce another axiom into the ontology stating that ‘bands only have members that are musicians’, the fact that ‘Ringo Starr is a musician’ can also be inferred, even though the fact that drummers are also musicians is omitted from the ontology.

The extent to which this inference can be performed depends on the *expressivity* of the ontology, which is itself dictated by the ontology language chosen. More expressive representations provide greater potential for inference [23, p. 327], at the cost of computational efficiency. Ontology languages are discussed in [Section 2.6.3](#).

#### 2.4.6 *Ontology Engineering Methodologies*

Formal ontology engineering methodologies help to guide ontology designers into creating successful models, particularly at the early stages of a project. Drawing upon software engineering techniques, many such methodologies have been published, with most describing creation of an ontology in several distinct stages. These are summarised by Pinto and Martins [162] and explained as follows:

- **Specification.** Identification of the purpose and scope of an ontology—definition of motivations, requirements, and system boundaries



- **Conceptualisation.** Design of the ontology itself, by specifying vocabulary and relationships and formulating exact knowledge fragments to be represented. Often undertaken in natural language or using diagrams.
- **Formalisation.** The encoding of a conceptual model into expressive formal language such as First Order Logic or UML.
- **Implementation.** Creation of final ontology based on formal model. Commitment to a technology is only needed at this stage.

Additionally, Pinto and Martins also suggest three tasks to be carried out throughout the process:

- **Maintenance.** Keeping the ontology up to date and relevant throughout design, implementation, and use.
- **Knowledge Acquisition.** The use of either automatic /bibliographic techniques or domain experts to assemble an accurate representation of the domain in ontology.
- **Evaluation and Documentation.** The continual measurement of an ontology's fitness-for-purpose, and natural language documentation to aid reusability.

The most commonly-used [31, 193, 194] methodologies can be categorised into four groups, based on their characteristics and intent:

- Early 'monolithic' ontology engineering methodologies that assume a single, non-iterative design process and emphasise choice of modelling language and how knowledge is formalised. These include Uschold and King [214] and the methodology used in creating the Toronto Virtual Enterprise (TOVE) ontology [85].
- Iterative ontology engineering methodologies that place less emphasis on initial formal specification of a model, and instead advocate testing, refinement, and ontology re-use, such as the widely-used METHONTOLOGY [63] and On-To-Knowledge [204].
- 'Post semantic web' methodologies such as the NeON Methodology [203] and Distributed Engineering of Ontologies (DILIGENT) [164] that place emphasis on collaboration and flexibility, and provide more pragmatic approaches to ontology creation than earlier methods.
- Ontology learning methodologies that focus on using automated or semi-automated tools to re-engineer knowledge, such as GenTax [99], ROD [232],

Three widely-used methodologies influence the work undertaken in this thesis, and in particular the methodology described in [Chapter 4](#). They are discussed below:

#### 2.4.6.1 METHONTOLOGY

METHONTOLOGY [63] was one of the first ontology engineering frameworks to consider the activity primarily in terms of software engineering principles, rather than in philosophical terms. It defines seven stages of the process (specification, knowledge acquisition, conceptualisation, integration, implementation, evaluation and documentation), and encourages the use of graphical ‘filing cards’ with key attributes to document the process as well as the resulting ontology. Methontology introduces several important concepts in ontology engineering:

- Recognition of ‘incremental’ and ‘evolving’ ontology engineering techniques. Whilst traditional software engineering methodologies at the time favoured monolithic approaches to system design, Fernandez et al. recognised that incremental ontology design is a valid and sensible option for creating ontology models.
- Description of guidelines/tooling for following a methodology. METHONTOLOGY provides not just abstract instructions about how to design an ontology, but practical suggestions and techniques for undertaking each stage too.
- Encouragement of ontology reuse at the *integration* stage. The authors suggest that existing ontologies should be surveyed and integrated before modellers create their own conceptualisations.

Despite its age, METHONTOLOGY is still in widespread use in recent projects [31, 194].

#### 2.4.6.2 DILIGENT

The DILIGENT methodology [164] is a collaborative approach to ontology design that is aimed at designers of practical ontologies for semantic web applications. Thus, its emphasis takes advantage of more contemporary tools and resources to describe a distributed technique for creating ontologies, that allows domain experts rather than knowledge engineers to do the bulk of the modelling work. From the perspective of this piece of work, it has several relevant features:

- The assumption of collaboration and rapid evolution of ontologies. Rather than seeing models as entities which are implemented and then left or minimally maintained, DILIGENT promotes an ‘evolutionary’ approach to development, where models are released but then modified and adapted as new use cases emerge.
- The recognition that conceptualisation and formalisation of ontologies can take place simultaneously. In the methodology, Pinto et al describe a scenario where groups of ontology engineers and experts build ontologies collaboratively, and in small stages.
- The use of change management techniques to create versions of ontologies. DILIGENT allows users to make changes to local ontologies at no risk to other users, and then submit them for review with a centralised panel of judges, who decide whether to publish these changes.

The iterative approach taken by diligent allows knowledge elicited by one person to propagate to other users, which prompts them to consider these changes and further specialise them. This ‘snowball effect’ is an effective way of achieving domain coverage.

#### 2.4.6.3 *The NeON Methodology*

The NeON methodology, proposed by Suárez-Figueroa, Gómez-Pérez, and Fernández-López [203], is a scenario-driven ontology engineering framework that describes nine approaches to model creation, based on factors such as potential for ontology re-use, resources available, and intended application. Each approach involves completion of several activities, many of which are based on approaches described in existing ontology engineering methodologies, and as such provide a huge number of possibilities for ontology development in different situations. These activities are described in detail by a published glossary [202, p. 70–74]. The NeON methodology advocates creation of ontology ‘networks’: groups of focussed ontologies that are linked together to achieve a task. Four of the nine proposed scenarios are listed below, taken directly from Suárez-Figueroa, Gómez-Pérez, and Fernández-López [203, p. 12–13], owing to their relevance to the new approach described in [Chapter 4](#):

- Scenario 1: From specification to implementation. The ontology network is developed from scratch, with no re-use of existing components.

- Scenario 2: Re-using and re-engineering non-ontological resources. This scenario covers the need to re-engineer and re-use knowledge which is currently in non-ontological form
- Scenario 4: Reusing and re-engineering ontological resources. This covers scenarios where knowledge must be re-used and modified from existing ontologies.
- Scenario 7: Reusing ontology design patterns (ODPs). Ontology developers access repositories of ontology design patterns to reuse them.

Other methodologies for ontology creation and design also exist, and this section has focussed predominantly on those approaches that are relevant to work described in [Chapter 4](#). Methodologies not described above include collaborative approaches such as RapidOWL [6], ANEMONE [155], and UPON Lite [45], the latter of which has no requirement on expert ontologies engineers for implementation. Recent surveys such as Corcho, Poveda-Villalón, and Gómez-Pérez [40] indicate a tendency towards ‘ad-hoc’ ontology creation for the web, with varying resulting ontology quality [231].

#### 2.4.7 *Ontology Modularity*

As ontologies become large, they also become more difficult to manage and to re-use [84]. Ontology modularisation is the practice of splitting models into a number of smaller chunks, or modules. Stuckenschmidt and Schlicht [201] summarises some of the key motivations and advantages of designing modular ontologies:

- **Understandability and documentation:** By separating ontology modules by task or subdomain, users can examine and understand each module easily rather than having to trawl through one monolithic representation of the entire problem. Each module can be annotated and documented independently, providing more granular provenance and context.
- **Ease of reuse.** Modularisation allows easier re-use of knowledge by other parties. Following the principle of ‘minimum ontological commitment’, users of domain models may wish to only commit to a certain part of a conceptualisation, to aid computational efficiency or to re-assert parts of a domain in other ways.

- **Scalability.** The performance and computational expense of ontology reasoning (discussed in [Section 2.4.5](#)) is often affected by the number of axioms a reasoner must consider when making inferences, with many reasoners performing particularly well over small knowledge bases but poorly over large ones. Rather than reasoning over a whole knowledge base, modularisation provides the possibility of reasoning over only those aspects required for a particular application, and thus helps aid performance as a model grows [[180](#)].
- **Maintenance and validation.** As in software engineering, the maintenance of small modules is inherently easier than that of a single, monolithic, model. Multiple ontology modules may be simultaneously authored by several individuals, and can more easily take advantage of software engineering practises such as unit testing and source control. Additional validation is also easier, as modules can be tested against known requirements for a particular use case or brief.

Methods for modularising existing ontologies are widely discussed in the literature, a recent overview of which is provided by D’Aquin [[43](#)]. These methods seek to break up large monolithic models into smaller modules for the purposes outlined above (‘module extraction’), and accomplish this through manual or automatic means. In this thesis, methods for constructing modular ontologies from scratch (‘modular development’) [[181](#)] are of more relevance, and it is from this point of view that several strategies for ontology modularisation are considered. Some desirable properties of ontology modules are summarised from the literature by Pathak, Johnson, and Chute [[161](#)]:

- **Size.** An ontology module should be as small as possible, to allow for scalability and efficient reasoning.
- **Correctness.** An ontology module should only contain information from the model it was extracted from; it should not be possible to infer any further or different knowledge to that which could be inferred from the original ontology.
- **Localized Semantics.** An ontology module should preferably be able to stand alone from other modules, such that a global model is not required to integrate individual modules.
- **Correct reasoning.** Reasoning over a collection of individual modules should produce the same logical consequence as reasoning over the original large ontology.

To modularise ontologies in order to achieve these goals, two main approaches are suggested in the literature, and are discussed below.

#### SEMANTICS-DRIVEN MODULARITY

Semantics-driven modularity[201] identifies particular subdomains or subject areas within the domain of discourse, and uses these to divide a model into several disjoint subject-specific modules. The scope of a module is thus very intuitive to designers and users, as recommended by Rector [180], and each module imports other modules that it depends upon, often in a hierarchical fashion (as shown in Figure 2.10). Modules containing commonly-used high level concepts are inherited by increasingly subject specific sub-modules[181].

This semantics-driven approach has the disadvantage that it discourages axioms that cross domain boundaries, as such axioms would require the inclusion of another subdomain module for use, diminishing the advantages of creating modules in the first place. Approaches to automated semantic modularisation of existing ontologies by subdomain are limited, but graph-based approaches that exploit model hierarchies can produce similar results [52, 187, 188].

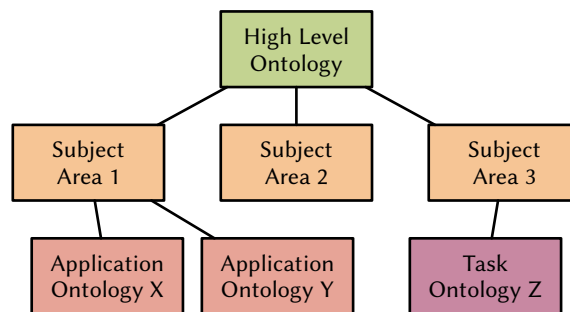


Figure 2.10: Set of Modular Ontologies Partitioned by Purpose

#### STRUCTURE-DRIVEN MODULARITY

Here, the term ‘structure-driven’ is taken to mean creation of ontology modules based on structure-based traits, such as how tightly a set of concepts and relationships are interconnected. This form of modularity is well-suited towards module extraction, where automated implementations can segment an existing monolithic ontology by its expressivity or size using logic-based methods, and can guarantee a resulting series of modules with guaranteed characteristics [78, 124]. This strategy lends itself to addressing the scalability problem addressed above owing to this guarantee, but is less well-suited to addressing

maintenance, ease-of-use, or documentation challenges that may present themselves in a large ontology.

These application of these approaches to create a set of modular domain ontologies is discussed in [Chapter 4](#).

#### 2.4.8 *Validation and Evaluation of Ontology Design*

The need for validation and evaluation of ontologies has long been recognised in the literature [63, 72, 116, 162, 165]. Validation is important in gaining acceptance of newly-created models, and many different approaches to doing so have been suggested in several different ontology engineering methodologies. Whereas validating the design of traditional data models in software engineering can often be established by directly checking a model against a set of functional (and quantified) requirements made by an application, this approach falls short when applied to ontologies. As domain models are intentionally divorced from the requirements of any particular application, testing against such requirements only guarantees that the model can represent information—it does not necessarily evaluate the quality, style, accuracy, or coverage of the model. Several techniques for ontology validation relevant to work undertaken in [Chapter 4](#) and [Chapter 5](#) are described in the following sections.

##### 2.4.8.1 *Qualitative Approaches to Validation*

Qualitative evaluation methods such as OntoMetric [131] and Burton-Jones et al. [30] entail asking domain experts to examine and rate models subjectively to a set of metrics. This approach can produce good indications of ontology coverage, quality, and correctness, but has several drawbacks. Firstly, choosing the correct set of users is difficult—if domain experts are used, model logic and semantics may be misunderstood, and if modelling experts are used, domain concepts may be misunderstood. Secondly, rating criteria are necessarily highly subjective: OntoMetric requires that users rate various factors from ‘very low’ to ‘very high’, but cannot provide any reference points to establish the meaning of these ratings.

##### 2.4.8.2 *Formal Metrics for Ontology Evaluation*

The use of formal or automated approaches to ontology validation allow for more objective results, based on parts of a model that can be evaluated by machines. These metrics include:

- RDF syntax validators, to check conformance of an ontology file to the correct syntax
- Graph metrics such as ‘connectedness’ that measure the characteristics of an underlying RDF graph, in methodologies such as AKtiveRank [2]
- Logical consistency checkers, such as those provided by the OWL Application Programming Interface (API) to prove the expressive profile and consistency of a model.
- Ontological evaluators, that check the semantics of an ontology for quality and consistency, such as those based on OntoClean [87] and the Ontology Pitfall Scanner [168]. These approaches provide an indication of how well-designed an ontology is, but cannot completely evaluate the intended semantics.

#### 2.4.8.3 *Similarity and Data-driven Measurement*

Comparison of a candidate ontology to a known ‘gold standard’ can provide indicative measures of coverage and similarity that serve to validate its design. Such methods are outlined in Maedche and Staab [132] and Burton-Jones et al. [30], but are not considered in detail here. When considering ontology design for interoperability across large complex systems, the existence of a ‘gold standard’ is unlikely.

An alternative approach is to use a known corpus of domain-related text to compare with the candidate ontology. If such a corpus aligns with the needs of the ontology, entity extraction techniques can be used to check the alignment and coverage of a candidate ontology to it. Such an approach is suggested by Brewster et al. [24], who use disambiguation and clustering approaches to additionally validate the structure of an ontology based on the similarity of distances between concepts in both the domain model and the corpus.

#### 2.4.8.4 *Application-based Validation*

‘In-use’ validation provides another partial way of validating ontologies. By attempting to use a domain model in a real-world task, the coverage, ease of use, and quality of a model in the application area can be assessed. This technique is discussed by Porzel and Malaka [167] and Seremeti and Kameas [189], and its use in evaluating semantic web ontologies are examined in Sabou et al. [186].

The validation techniques described in [Chapter 4](#) and [Chapter 5](#) draw on several of these techniques in validating domain ontologies.



## 2.5 THE SEMANTIC WEB AND THE LINKED DATA MOVEMENT

The data integration challenges discussed above are not confined to enterprise applications and private systems. Like modern enterprise systems, the Internet connects together many diverse and heterogeneous data sources, and there is a widespread desire to take advantage of data made available by this interconnectivity, as envisaged by Berners-Lee and Fischetti [18]. The ubiquity of the World Wide Web demonstrates this desire, and has influenced the development of many technologies now in widespread use in enterprise applications.

Efforts to provide greater machine interoperability over the web (the *Semantic Web*) have led to the rapid development of ontology-based technology stacks and toolsets which have in turn become available for other purposes. The development of these technologies is discussed in the following subsection.

2.5.1 *The Semantic Web*

The WWW has undoubtedly made data sharing over the internet extremely easy, and the simplicity of making content available may have aided its huge success [18]. Much of this content on the web is still presented in plain text or HTML, which contains very little machine readable markup—web pages are inherently incomprehensible to automated systems. Search engines that allow users to query information across the web developed ways of deducing semantics through natural language processing and graph-based algorithms such as PageRank [156], and these remain the primary method of accessing web data to the present day.

The lack of machine-readable data on the web restricts the ability of computers to retrieve and process information. Figure 2.11 illustrates this by obfuscating the text of a wikipedia article to simulate a computer's comprehension of it: without a way of understanding the human-readable text, machines are confined to extracting only metadata such as links, page layout, and text analysis. The *Semantic Web* aimed to address this limitation by providing a way of representing such data on the web, so that search engines and other agents may be able to carry out vastly more complex and useful queries. As an example, imagine a train booking system with a user aiming to:

‘Get me from Edgbaston, Birmingham, to Splott, Cardiff, this Friday evening in time to get a table for dinner at Mario’s’

This screenshot shows the English Wikipedia page for "The Simpsons". The page includes the Wikipedia logo, navigation links, and the main article text. The article describes the show as an American adult animated sitcom created by Matt Groening for the Fox Broadcasting Company. It mentions the main characters: Homer, Marge, Bart, Lisa, and Maggie, and notes that the show is set in the fictional town of Springfield. A prominent image of the Simpson family is displayed on the right side of the page.

(a) Human Comprehension

This screenshot shows the Polish Wikipedia page for "The Simpsons". The page content is largely machine-generated, with nonsensical characters and words replacing the original English text. For example, the title is "Tob Wbwzkdag" and the main text contains strings like "Cnu Vksxkugx wb pg Ccllyztn uttwc vfrkhbaa". The image of the Simpson family is still present on the right side of the page.

(b) Machine Comprehension

Figure 2.11: A Wikipedia Page as Understood by Humans and Computers

On the conventional web, the user must first look up when tables are free at Mario's. Assuming no prior knowledge, they must then consult a myriad of websites to determine the possible ways of making the journey. They must note down bus times for each end of the journey, and determine which train or bus to get from Birmingham to Cardiff. This is a considerable amount of work, despite all of the information being present on the web.

On the semantic web, it should be possible to automate the whole process, or to only present the user with a set of solutions:

1. A software agent queries a route planning service to list of transport routes and providers between Edgbaston and Splott, determining likely routes and times.
2. It submits a web search for Marios' restaurant, obtains the restaurant's Uniform Resource Identifier (URI), and then queries its availability for Friday.
3. Exact travel times and costs are obtained from each provider's data store
4. Itineraries are compiled and presented to the user.
5. Upon choosing an itinerary, transport arrangements and restaurant bookings are made, and corresponding bookings placed in Marios' and the travel operators' data stores.

This automation is achieved by retrieving not just HTML over the web, but machine-readable facts too, based on web ontologies. By assigning each entity on the web a unique identifier and linking these identifiers to each other through properties according to an ontology, the exact meaning of each piece of information on a web page can be deduced by the computer, allowing data from many heterogeneous sources to be used together.

An example of the semantic web in action can be seen in Google Shopping<sup>4</sup>. Incentivised by higher exposure and search listing rankings, websites selling goods mark up their HTML pages with machine readable semantic web data according to the GoodRelations ontology [100], with information such as pricing, descriptions, and locations of items for sale. With knowledge of exactly what each ontological assertion means, Google Shopping is able to use and query across this data, such that users can obtain information about relevant products, suited to their interests, in their location, as shown in [Figure 2.12](#).

---

<sup>4</sup> <http://shopping.google.com/>

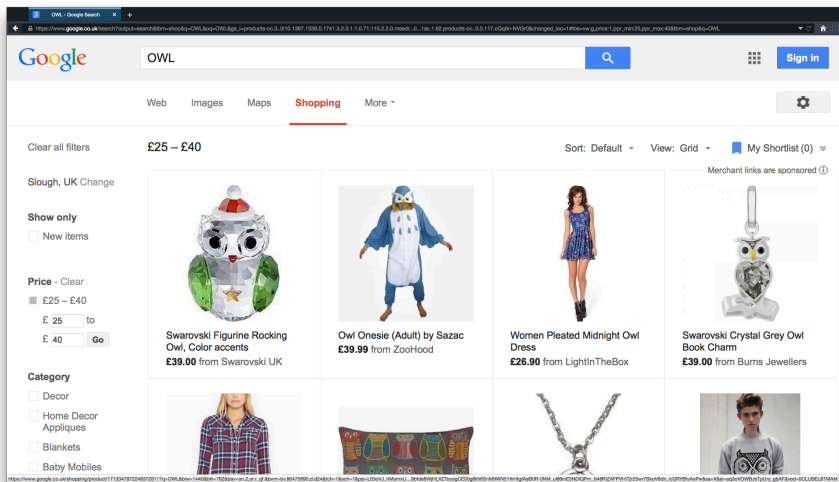


Figure 2.12: Screenshot of Google Shopping Results Page

Although the original semantic web vision enjoyed some success, its uptake did not grow rapidly [88]. It did, however, spur on the development of many technology platforms and tools that have uses outside of the web itself, such as in enterprise applications. The majority of the tools used to implement work discussed in this thesis are based on work driven by the semantic web movement.

### 2.5.2 *The Linking Open Data Movement*

The Linking Open Data movement has become a more recent continuation of the push for the Semantic Web adoption. Promoted in parallel with a wider social push towards data transparency in governments and public sector organisations, Linked Open Data (LOD) addresses some issues with the original semantic web offering by doing the following:

- **Lowering the technological barrier to entry**, by providing simpler implementation mechanisms and better documentation. Initially, few web developers coded semantic web data into their sites due to the vast amount of knowledge needed to do so. Creation of domain ontologies is not emphasised, and new data formats such as RDFa, and JavaScript Object Notation for Linked Data (JSON-LD) allow developers to use existing toolsets to implement linked data.
- Providing **simpler, unambiguous** vocabularies such as [schema.org](http://schema.org)<sup>5</sup> with a defined set of benefits to use.

<sup>5</sup> <http://schema.org>

- Approaching data publishers rather than web sites; and specifically those with a desire or **obligation to make data available** to the wider world (predominantly governments and public sector organisations)

As a result of these changes, and the success of the *open data* movement in campaigning for such organisations to publish their data, LOD has enjoyed significant uptake in recent years [98]. As part of this campaign, Tim Berners-Lee outlined a ‘5 star’ ranking system [17] for quality of open data resources, based on data structure, seen in [Table 2.3](#).

Table 2.3: The 5 Stars of Open Data

	Description	Format
★	Data is available, in some way, on the web	PDF
★★	Data available in computer-readable formats	XLS, DOC
★★★	Standardised computer-readable formats used	CSV
★★★★	Data provided in ‘linked data’ format	RDF
★★★★★	‘Linked data’ associated with existing data and ontologies	RDF/ OWL

This five star ranking system pairs loosely with the interoperability levels discussed in [Section 2.3.3](#): ‘two star’ data provides structural interoperability, ‘three star’ data brings syntactic interoperability, and ‘five star’ data provides good semantic interoperability.

### 2.5.3 *Disincentives to Uptake of Linked Open Data and Enterprise Ontology*

The recent enthusiasm for provision of linked open data on the web highlights technical and business motivations both for and against exposing organisational information. Many of these issues are analogous to those experienced across complex, multi-stakeholder systems such as the railway, and can be summarised as follows:

- High upfront investment to create ontologies and design patterns for publishing data can be discouraging and preventative [184], particularly when the missed opportunity cost of underutilising information assets within a business is not realised.

- Open licensing of data can be intimidating, especially where information assets make up a high proportion of a company's value [205]. In many cases this is a legitimate barrier to entry, and few models for monetising data access currently exist.

## 2.6 CORE TECHNICAL CONCEPTS AND NOTATION USED

The continued development of the semantic web and related concepts has, over the last fifteen years, enabled development and specification of a mature set of standards, tools, and technologies for working with ontologies and linked data. Drawing upon a number of prior efforts, the World Wide Web Consortium (W3C) recommendations for RDF [122], RDFS [26], OWL [221], and SPARQL [172] have become de facto technology standards in their areas, and the recent W3C Linked Data Platform [229] proposal is likely to further consolidate the semantic web technology ecosystem. These technologies are used extensively in realising the semantic data models created and implemented in this thesis, and are explained in the following section.

### 2.6.1 *The Resource Description Framework*

Resource Description Framework (RDF) provides the building blocks for creating and representing knowledge in semantic data models. It is a framework designed to allow exchange of data between applications without loss of meaning [122], and provides a way of encoding knowledge as a series of assertions about entities and concepts (henceforth called *resources*). RDF is built on the idea of a *triple*, which encodes a relationship between a *subject* and an *object* through a property, or *predicate*, in the form <subject> <predicate> <object>. Compiling many triples with the same subjects or objects allows a knowledge graph to be built up:

Subject	Predicate	Object
Miles	is a	Musician
Miles	plays	Trumpet
Miles	nationality	American

Each element in a triple is filled by either a *resource*—a unique identifier representing some concept—or a literal, which is a concrete value of some type<sup>6</sup>. Each resource is notated by a unique *URI*, either generated either by the designer of the knowledge base, or re-used from

<sup>6</sup> In RDF, only the object of a triple may take a literal value

some well-known or canonical identifier<sup>7</sup>. Web Uniform Resource Locators (URLs) are often chosen as Resource IRIs, allowing users to seek further information about the resource by retrieving its URI over the web<sup>8</sup>, although this is not mandatory. Examples of IRIs and literals are shown in [Table 2.4](#).

Table 2.4: Example RDF URIs and Literals

Resource	Description
<code>http://dbpedia.org/resource/MilesDavis</code>	The DBpedia canonical IRI for Miles Davis
<code>http://data.ordnancesurvey.co.uk/id/50kGazetteer/81356</code>	an IRI for Edgbaston, Birmingham, as described by Ordnance Survey Linked Data
<code>http://railwayontologies.org/Delay</code>	A concept of railway delay. Does not resolve to a web address, but is still a valid IRI
<code>"405500"^^xsd:int</code>	A literal, with integer datatype (as specified by XSD).
<code>"Edgbaston"@en</code>	A literal, with RFC 3066 defines compliant language tag

Provenance or ownership of a resource is often implied by its namespace (the part of the IRI before the final `'/ór #character`), and most semantic data models define one or several namespaces for this purpose. As such, IRIs are often abbreviated as ‘Compact URI (CURIE) names’ [20] of the form `namespace:prefix` for readability, a convention used throughout this thesis, and illustrated by the URIs and conversions in [Table 2.5](#). A full list of namespace mappings used can be found in [Section B.1](#).<sup>9</sup>

<sup>7</sup> Modellers may choose to create their own Internationalized Resource Identifiers (IRIs) for a concept, or to adopt a widely-used IRI for the same concept in another knowledge base. As an example, resources that are part of Wikipedia are often identified by their `http://dbpedia.org` IRIs, such that ambiguity is minimised when two disparate data sources are used.

<sup>8</sup> The ‘follow-your-nose’ design pattern [230] suggests that some information about a resource be available at the end of an Hypertext Transfer Protocol (HTTP) request to the resource’s URI. In this way, a user or computer system can gain some information about a resource

<sup>9</sup> CURIE mappings are not standardised, and prefixes used for the same IRIs can vary from document to document. Some very common namespaces are often abbreviated in the same manner; a repository of these is available at `http://prefix.cc/`

Table 2.5: Two Examples of Semantic Web URI abbreviated as CURIE identifiers

Full URI	CURIE identifier
<code>http://dbpedia.org/resource/MilesDavis</code>	<code>dbp:- MilesDavis</code>
<code>http://railwayontologies.org/Delay</code>	<code>ex:Delay</code>

RDF provides very little by way of formal semantics or assumptions about data structure, and there are few limits as to what can be represented as triples. The huge syntactic flexibility gained here is an advantage in some situations but hinders machine reasoning. In practice, many knowledge graphs use either an agreed vocabulary with some agreed semantics, or further semantic web standards such as Resource Description Framework Schema (RDFS) and OWL to convey explicit data semantics and structure.

#### 2.6.1.1 *RDF Serialization*

RDF is defined firstly as an *abstract syntax*, in that it only initially defines triples, resources, and literals as concepts, and not as any particular type of markup or data format on a computer. As such, RDF information can be represented in many forms, and several standard RDF serialisation formats are available.

- **RDF/XML** [12] was published with the initial RDF standard in 2004, and is widely used and supported. It represents RDF as a subset of XML, and is notoriously unreadable for human users. For this reason, its use is waning in comparison with other formats[13].
- **N-Triples** [11] is used predominantly by applications that require low processing overheads, and expresses RDF as line-delimited triples in plain text.
- **Turtle** [13] is an extension of N3 and adds support for abbreviation, URL prefixes, and several other readability-enhancing features. It has gained widespread use owing to its readability.
- **Header, Dictionary, Triples (HDT)** [62] is a newly-proposed binary serialization format, allowing compression of RDF whilst retaining search and browse capabilities.



- **Resource Description Framework in Attributes (RDFa)** [1] provides a method for embedding RDF information within HTML documents. It is used extensively in the field of Open Data, and allows human-readable web documents to be parsed by RDF-aware machine processors without the need for a separate RDF document.
- **JSON-LD** [198] is a subset of the widely-used JavaScript Object Notation (JSON) data serialization format, and was designed to enable linked data exchange over the web using existing JSON tools and expertise. JSON-LD is a syntactic superset of RDF; features such as lists that are part of the syntax of JSON-LD syntax must be expressed in RDF using an additional vocabulary.
- **Microdata** [101] allows RDF to be represented in HTML5 documents through the use of HTML key-value attribute pairs. Along with RDFa, it has enjoyed significant uptake on the web owing to its adoption by schema.org<sup>10</sup>[21].

This thesis uses Turtle notation to present RDF examples and ideas. A brief demonstration of how RDF content presented in Turtle is shown in [Listing 2.2](#).

#### 2.6.1.2 *RDF Rules and Semantics*

RDF, as well as the standards the build upon it, have several other key characteristics of note. These are outlined as follows:

- **The Open World Assumption (OWA)** entails that within an RDF model, anything not known to be true is unknown, rather than false. This is in contrast to relational databases, where a missing value implies falseness (Closed World Assumption). For example, given the knowledge in [Figure 2.13](#), we might ask ‘How many daughters does Homer Simpson have?’. A closed world system should answer ‘two’, whereas an open world system answers ‘at least two’. To tell the open world system that ‘two’ is the correct answer, an additional assertion is required - a ‘closure axiom’.
- **No Unique Name Assumption.** Similarly, in RDF, entities with different names or IRIs are not assumed to be unique. So in fact, unless the knowledge base is explicitly told that Maggie and Lisa are different people, the answer to the above question would in fact be ‘unknown’, as it is otherwise possible that ‘Maggie’ and ‘Lisa’ are the same person!

---

<sup>10</sup> <http://schema.org/>

```

# Anything following a hash symbol is a comment, and
  → dis-regarded by the Turtle parser.

# Firstly, define prefixes. Using a prefix to describe a
  → resource denotes that it has the namespace shown in
  → <brackets>

@prefix ex: <http://example.org/> .
# Multiple prefixes are used to ease the use of references from
  → different namespaces
@prefix dbr: <http://dbpedia.org/resource/> .
# RDF and XSD namespaces with defined semantics
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# The base URI of the document is defined using the @base
  → keyword
@base <http://example.org/> .

# Firstly, Assert the fact that a trumpet is a musical
  → instrument without prefixes (as an example). Triples are
  → terminated with a full stop:
<http://example.org/Trumpet>
  → <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  → <http://example.org/MusicalInstrument> .

# Now, the same thing asserted using prefixes:
ex:Trumpet rdf:type ex:MusicalInstrument .

# Predicate lists allow the assertion of multiple triples about
  → the same subject. Semi-colons indicate that the following
  → statement has the same subject as the previous:
dbr:Miles_Davis rdf:type :Musician;
^^Iex:nationality ex:American .

# Object lists allow the assertion of many triples with the
  → same subject and predicate. Commas indicate that that the
  → following statement has the same subject and predicate as
  → the last:
dbr:Miles_Davis ex:plays ex:Trumpet, ex:Piano, ex:Drums .

# RDF literals can be quoted or unquoted, and are followed by a
  → language tag or datatype URI. Quoted literals with no
  → datatype or language tag are defined to be <xsd:string>.
dbr:Miles_Davis ex:nickName "Miles", "Miles"^^xsd:string;
^^Iex:description "Miles Davis is a famous American jazz
  → musician"@en .

# Unlabelled blank nodes can be created using square brackets:
dbr:Miles_Davis ex:album [ ex:name "Kind Of Blue"; ex:year
  → "1959"^^xsd:date ] .

# The above statement is equivalent to:
dbr:Miles_Davis ex:album _:AlbumNode .
_:AlbumNode ex:name "Kind Of Blue" ;
^^Iex:year "1959"^^xsd:date .

```

Listing 2.2: Example of Facts Represented Using Turtle

- **Blank nodes** are an RDF construct that allows definition of unnamed intermediate nodes to cater for certain design requirements such as ternary relationships, as described below.

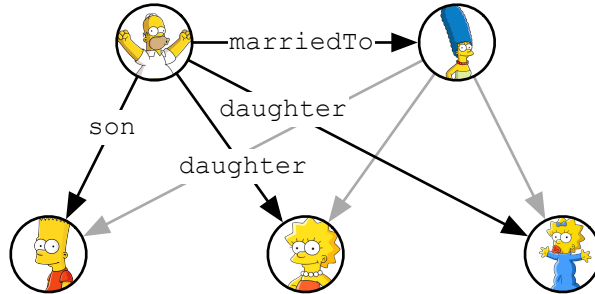


Figure 2.13: Example Relationships in The Simpsons Family

### 2.6.1.3 RDF Reification and Ternary Relations

The RDF model is limited to expressing binary predicates—relations that have one subject, one object and no more. It is desirable in many situations to model higher arity relations, such as when modelling time (‘Train X was located at Banbury at 10.30AM’). Two mechanisms for doing this within RDF are common:

#### N-ARY RELATIONS USING BLANK NODES

In this approach, ternary relations are expressed by introducing a third, intermediate, RDF resource between the subject and the object of the intended relation [147]. This resource then expresses two or more relationships of its own, linking the original subject to the eventual n-ary objects through itself. This pattern is demonstrated in [Figure 2.14](#), which expresses the statement ‘Signal 3191 was controlled from Oxford between 1949 and 2004’ using the N-ary relationships pattern.

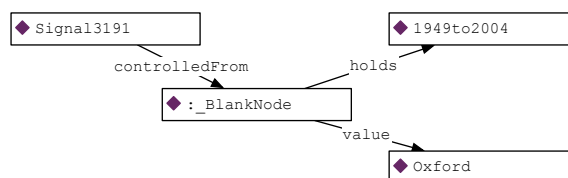


Figure 2.14: Example of N-ary Relationships Pattern

Signal13191 is linked to a blank node through the `:controlledFrom` property, which is the subject of two triples: `:holds :1949to2004`

asserts that the fact holds over a specified timespan, and `:value :-Oxford` asserts the actual object of the fact that holds over that period.

## RDF REIFICATION

**Reification** provides an alternate means for representing n-ary relations. Reification de-composes an RDF triple into an RDF entity in its own right, with its own corresponding properties. The addition of a time property using reification to `ex:Signal3191 :controlledFrom :Oxford` is shown in [Listing 2.3](#).

```
:_Triple1 rdf:subject ex:Signal3191
:_Triple1 rdf:predicate ex:controlledFrom
:_Triple1 rdf:object ex:Oxford
:_Triple1 rdf:atTime ex:TimeInterval1
```

Listing 2.3: Example of RDF Reification in Turtle

Reification in this way does allow effective representation of ternary relations, but is used infrequently in conjunction with ontologies. Reified assertions are no longer ‘first class’ triples, and so much of their semantics are lost. In OWL, described below, reasoning capability over reified relations is greatly decreased.

### 2.6.1.4 *RDF Conventions*

Several conventions have become best practice when working with RDF models, and are adhered to in the work presented by this thesis. These are as follows:

- RDF resources are usually assigned identifiers that correspond to URLs on the world wide web. This facilitates the ‘follow-your-nose’ approach taken by semantic web tools, by which additional information about a resource is provided by a document located on the web at its URL.
- RDF resources are described in terms of namespace and suffix. An RDF model usually uses one namespace, and resources defined in it are specified by their suffix.
  - The *namespace* is all of the IRI until the final / or #.
  - The *suffix* is the final part of the IRI, and is used as the local name of the resource by tools.

- These IRIs are often abbreviated using CURIE names in the form `namespace:prefix`. This notation is used in several RDF serialisation formats and will be used henceforth in this thesis.

### 2.6.2 *RDF Schema*

To make it possible for computers to use and infer knowledge from RDF data, some formal specification of how the model behaves is required. Whilst pure RDF provides the flexibility to assert a wide range of information, this flexibility can make reasoning over data sets unfeasible for machines.

RDF Schema is a syntactic subset<sup>11</sup> of RDF that provides a fixed vocabulary and enforces certain restrictions on the structure of a data model, in order to make automated inference of facts possible. Full details of RDFS are outlined in Brickley and Guha [26], and notable characteristics described below:

- The `rdf:type` property<sup>12</sup> entails that the subject is a member of the class of objects. This is equivalent to saying subject is an object, and is synonymous in most RDF serialisations with the predicate `a`. This allows models to infer membership of classes.
- `rdf:Property` describes resources that are used as properties that link two other resources. In RDFS, these must not be literals. RDFS semantics requires that anything used as the *predicate* of a triple is of type `rdf:Property` and infers this if it is not explicitly stated. `rdf:type` is of type `rdf:Property`.
- `rdfs:subClassOf` allows the definition of class hierarchies, such as `ex:ElectricTrain rdfs:subClassOf ex:Train`. The definition of such hierarchies is a powerful tool in reasoning, as it allows the entailment of class membership at any level of a hierarchy.
- `rdfs:domain` and `rdfs:range` allow the `rdf:type` of entities to be inferred based on the property that link them, a demonstration of which is shown in [Listing 2.4](#)

<sup>11</sup> RDFS models can only use a subset of RDF syntax

<sup>12</sup> Although `rdf:type` is defined in the RDF vocabulary, its semantics are specified only in the RDFS standard.

```

ex:plays rdfs:domain ex:Musician .
ex:plays rdfs:range ex:MusicalInstrument .
ex:MilesDavis ex:plays ex:Trumpet .

# infers

ex:MilesDavis a ex:Musician .
ex:Trumpet a ex:MusicalInstrument .

```

Listing 2.4: Demonstration of Inference based on `rdfs:domain:` and `rdfs:range:` Restrictions

### 2.6.3 *Ontology Languages, The Web Ontology Language and Description Logic (DL)*

In order to represent formal ontologies, some way of describing the world formally using well-specified formal semantics is needed. Notation systems that accomplish this are called ontology languages, and several prominent ontology languages are present in the literature, including Knowledge Interchange Format (KIF) [74], EXPRESS [111], and OWL [221]. By far the most widespread of these is OWL, which enjoys large scale public adoption and tooling support. OWL is built on Description Logic (DL) semantics, and both are described below.

#### 2.6.3.1 *Description Logics*

DLs are a set of knowledge representation languages that can be used to formally describe knowledge of an application domain [199, ch. 1]. They accomplish this by defining a set of concepts and roles, and allowing the nature of interactions and restrictions across such concepts to be described using a set of logic-based constructs, many of which are borrowed from First Order Logic (FOL), such as negation, conjunction, and restriction. For example, take the following information.

A train has at least one carriage and has powered or unpowered traction characteristics.

These facts could be represented using DL notation, as follows:

$$\begin{aligned}
 \text{Train} &\equiv \text{Thing} \sqcap (\geq 1 \text{hasCarriage}) \\
 &\quad \sqcap ((\forall \text{traction.Powered}) \\
 &\quad \sqcup (\forall \text{traction.Unpowered}))
 \end{aligned}$$

. Here, *Thing* and *Train* are unary predicates—‘atomic concepts’ [8], as are the two types of traction characteristic: *powered* and *unpowered*. *hasCarriage* is a binary predicate, or ‘atomic relation’. The *R.C* construct is a ‘value restriction’ symbolising the set of concepts related to the *C* class through the *R* relation, so the total description including union and intersection symbols can be summarised as:

A train is the set of concepts that:  
 ...belong to concept *Thing* *and*  
 ...belong to the set of concepts that are linked to more than one other concept via the *hasCarriage* relation, *and*  
 ...belong to the set of concepts that are related to the *Powered* concept through the *traction* relation, *or*  
 ...belong to the set of concepts that are related to the *unpowered* concept through the *traction* relation.

These DL descriptions of interactions and restrictions between things can be used to describe a huge variety of knowledge, as long as it can be conceptualised using binary relations. Further examples of DL notation constructs are given in Table 2.6, and a full detailed explanation of description logics (including the many different profiles that exist for different purposes) is provided by Baader et al. [8] and Harmelen et al. [95].

#### 2.6.3.2 *Web Ontology Language (OWL)*

OWL is a family of ontology languages designed to create formal, machine-interpretable ontologies. [221]. The OWL 2 standard defines several subsets, or ‘profiles’ for different purposes, and in this thesis we focus on the subset of OWL known as OWL DL which have semantics based on the  $\mathcal{SROIQ}(D)$  description logic. As such, OWL 2 provides a very expressive way of defining formal ontologies, whilst maintaining tractability—it is always possible for an automated reasoner to make sound and complete inferences over an ontology expressed in this way (although performance for large ontologies can be extremely poor)[8, ch. 1].

OWL was primarily designed to provide a way of formally describing ontologies for the semantic web. As such, they are usually notated in RDF, although several non-RDF formats exist [106, 143]. As an illustration, Listing 2.5 shows the DL fragment from above expressed in OWL using Turtle notation:

Table 2.6: Example OWL Constructs and Related DL Symbols

DL Symbol	OWL Term	Description/Example
	owl:Class	rdf:type of resources defined as Classes
	owl:Individual	rdf:type of resources defined as Individuals
$\top$	owl:Thing	The class to which all individuals belong
$\perp$	owl:Nothing	The class to which no individuals belong
$\sqcap$	owl:intersectionOf	Cat $\sqsubseteq$ Animal $\sqcap$ Pet
$\sqcup$	owl:unionOf	Human $\sqsubseteq$ Adult $\sqcup$ Child
$\exists$ R.C	owl:someValuesFrom	Existential 'has some' restriction: Cat $\sqsubseteq$ $\exists$ part Face
$\forall$ R.C	owl:allValuesFrom	Universal 'only has' restriction: Cat $\sqsubseteq$ $\forall$ eats CatFood
$\geq n$ U	owl:minCardinality	Minimum cardinality restriction: Cat $\sqsubseteq$ $\leq 4$ part Leg
$\leq n$ U	owl:maxCardinality	Max cardinality restriction: Cat $\sqsubseteq$ $\geq 5$ part BrainCell
	owl:TransitiveProperty	"Cat part Leg, Leg part Paw" infers "Cat part Paw"



```

:Train rdf:type owl:Class ;
      owl:equivalentClass [ rdf:type owl:Class ;
                            owl:intersectionOf ( [ rdf:type owl:Class ;
                                                    owl:unionOf ( [ rdf:type owl:Restriction ;
                                                                    owl:onProperty :traction ;
                                                                    owl:hasValue :Powered
                                                                    ]
                                                                    [ rdf:type owl:Restriction ;
                                                                    owl:onProperty :traction ;
                                                                    owl:hasValue :Unpowered
                                                                    ]
                                                                    )
                                                    )
                            ]
                        [ rdf:type owl:Restriction ;
                        owl:onProperty :hasCarriage ;
                        owl:onClass owl:Thing ;
                        owl:minQualifiedCardinality
↪ "1"^^xsd:nonNegativeInteger
                        ]
                        )
      ] .

```

Listing 2.5: OWL Markup Showing Restrictions on Train Class

### 2.6.3.3 OWL 2 Profiles

Several subsets of OWL DL are provided in the OWL 2 specification. These subsets are encoded in the same way as OWL 2 DL, but alter the ontology's expressivity to provide different trade-offs for computational efficiency:

- **OWL Full** places no restrictions on the OWL syntax, but can create models for which reasoning is intractable. OWL Full is often used as a notation language or in association with non-complete reasoners that are faster but do not necessarily compute 100% of inferences.
- **OWL DL** is the most expressive subset of OWL2 to provide sound and complete<sup>13</sup> reasoning using a suitable piece of software. Reasoning across large OWL DL ontologies can be intractable, and so DL reasoners are rarely used in applications with large volumes of data.

<sup>13</sup> Reasoning soundness implies that all inferred axioms are correct. Completeness measures the proportion of all possible inferences that are inferred. Reasoners which produce complete and valid results are called sound and complete.

- **OWL EL** is designed for applications with large ontologies but a small amount of instance data, and guarantees good<sup>14</sup> performance in this environment [47].
- **OWL Query Language (QL)** ontologies allow efficient reasoning over large numbers of individuals, as long as the ontology size (and expressiveness) is low. Queries across OWL QL ontologies can be re-written into SQL, allowing native relational databases to be used to store data.
- **OWL Rule Language (RL)** is a slightly restricted subset of OWL DL that guarantees better performance in most situations. OWL Rule Language (RL) reasoning can be implemented using rule-based reasoners, and can deliver excellent performance if sound but not complete reasoning is acceptable.

Figure 2.15 shows the expressivity of OWL subsets compared to other ontology languages.

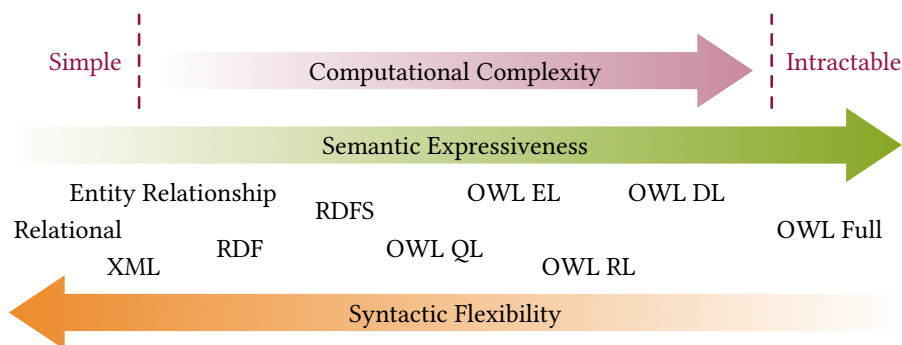


Figure 2.15: Expressivity Characteristics of OWL Profiles

#### 2.6.4 Reasoning and Inference in OWL and DL

Reasoning in OWL DL can provide the following features:

- **Subsumption:** checking which classes a concept is necessarily a member of (primarily through inheritance)
- **Satisfiability:** a class that necessarily has no members (through some conflicting definition) is unsatisfiable.

<sup>14</sup> OWL EL guarantees that a particular set of reasoning features can be inferred in less than polynomial time with respect to the number of assertions in the ontology [142].

- **Consistency checking:** checking whether an ontology conflicts with itself in some definition. For example, if classes ‘Human’ and ‘Cat’ are said to be disjoint, and ‘Fred’ is a member of both, the ontology is inconsistent.
- **Equivalence:** checking if two concepts are equivalent (the same) as each other.
- **Entailment:** the creation of new assertions following the logic provided by the ontology.

Under some circumstances, reasoning according to an OWL-compliant DL reasoner may be undesirable. These reasoners often calculate inferences using a ‘theorem-proving’ method, which calculates all logically permissible entailments and then discount invalid ones [23, p. 99]. Such circumstances include:

- Reasoning over medium or large knowledge bases that make DL complexity unfeasible or intractable
- Applications where complete inference is not required, and computational efficiency is preferred to full entailment
- Reasoning where axioms beyond the expressivity of OWL DL are needed, such as in ontology alignment or when asserting domain-specific knowledge.

Where this is the case, a *rule*-based approach may instead be taken. These reasoners work by iterating through a knowledge base attempting to match pre-defined rules, and then applying some logic when a match is found [141]. Several subsets of OWL can be reasoned over completely using rule-based approaches, such as RDFS-Plus [5] and OWL RL, and many off-the-shelf RDF data stores employ rule-based approaches in place of less efficient but more complete tableau-based algorithms [37, 121].

Several rule languages for RDF exist, including the Semantic Web Rule Language (SWRL), the Rule Interchange Format (RIF), and SPARQL Inferencing Notation (SPIN). Rule reasoning over RDF is also possible in Javascript, using SPINx [123] and in Datalog, using Java software library Jena<sup>15</sup>. Custom rules are used in Section 6.2 to encode application-specific logic into a demonstration ontology.

---

<sup>15</sup> see <https://jena.apache.org/documentation/inference/>

### 2.6.4.1 *Forward and Backward Chaining*

RDFS and OWL reasoners can be implemented in a number of ways. Software currently available broadly take two approaches to reasoning, in order to obtain best performance for particular applications and use cases: *forward-chaining* and *backward-chaining* [185, ch. 9]. These are described below:

**Forward-chaining** reasoners start with a set of assertions and a set of rules or axioms. They iteratively apply this set of axioms to the set of assertions to infer more and more knowledge until reasoning is deemed complete. Such reasoners can implement full DL inference as described above, and examples include Pellet [160], HermiT [190], and RacerPro [92]. Forward-chaining reasoners are useful for static ontologies and usually materialise all inferences in memory, so that they need not be run every time a query is made. They are often less appropriate for use in large or changing knowledge bases owing to their need to continually re-compute consequences whenever input data in a model changes. Full OWL DL reasoning itself has poor worst-case reasoning performance<sup>16</sup>, so frequently re-computing inferences using these across large models can be costly. Other forward-chaining reasoners such as OWLIM [121] base their implementation on rule-based algorithms, which reason over a practical subset of OWL DL in order to achieve increased performance.

**Backward-chaining** reasoners are those that work backwards from a query or fact in order to deduce associated inferences. Reasoners of this type tend to use rule-based implementations of ontology logics, and often accompany forward-chaining reasoners in implementations such as JESS—the rule reasoner used in Jena [102, ch. 3], Virtuoso [54], and Stardog [37], the RDF triplestore adopted in Chapter 6 of this thesis. Backward chaining reasoners suit applications where it is impractical to keep all inferences of a knowledge base in memory, such as if they are large or constantly changing. They need only to compute inferences related to each requested query, and can therefore perform better than a forward-chaining reasoner if materialisation of all inferences is not possible.

Most enterprise-level RDF triple stores such as those mentioned above now offer a hybrid approach, using both forward-chaining and backward-chaining. Non-changing parts of a knowledge base, for instance, can adopt forward-chaining and materialisation to optimise query perfor-

---

<sup>16</sup> The worst case reasoning performance of OWL DL rises double-exponentially with the number of assertions in the model [103]

mance, whilst constantly-changing or infrequently-queried parts can choose to use backward chaining instead.

#### 2.6.4.2 *Closed World Reasoning*

When using the OWA, only the assertions made in a particular knowledge base are known to be true. The non-existence of a fact does not imply that it is false; it is just unknown. This makes the OWA appropriate for semantic modelling and knowledge discovery, where only partial representations of a world view are created and the ability to reason over incomplete information is a desirable feature. In some situations, however, the closed world assumption taken by traditional database applications can be more appropriate. For example:

- *Counting instances.* In a closed world system that also makes the unique name assumption, the number of entities of a particular type can be counted. In an open world system, this is only the case if a closure axiom has been asserted: the knowledge that ‘this is the set of all of the entities’ is in the knowledge base.
- *Data validation and constraints checking.* In closed world systems, schema restrictions and constraints can easily be used to prevent users from entering data that does not fit the data model, and to validate user input. With the OWA this is more problematic, since data that would trigger a constraint violation in a closed world system may merely imply new knowledge in an open world. Consider the information in [Listing 2.5](#), which states among other things that a train must have at least one carriage. Should a user add a train to the knowledge base without any carriages, an open world system assumes simply that some carriages must exist that it does not know about, whereas a closed world system immediately assumes a validation error has occurred. These two contrasting behaviours may be useful for different applications.

The lack of closed world semantics in OWL leads to difficulties implementing certain practical features such as those listed. For this reason, several ways of undertaking closed world reasoning in OWL are highlighted here. Firstly, Grimm, Motik, and Preist [81] suggest an extension to OWL semantics to denote closed world axioms explicitly, allowing reasoners to understand both closed and open world semantics in the same knowledge base.

Alternatively, Motik, Horrocks, and Sattler [140] provide an alternative way of interpreting OWL existing OWL semantics under the closed world assumption, and suggest that applications themselves

choose which axioms should be reasoned over using each approach<sup>17</sup>. Tao et al. [207] show how this approach can be implemented through translating OWL into Sparql Protocol and RDF Query Language (SPARQL) queries; explicit closed world constraints could also be expressed directly in a rule language such as those described in [Section 2.6.4](#).

### 2.6.5 *Terminological and Assertional Knowledge*

In ontology engineering, a distinction is usually made between domain knowledge, which describes the *way* the world works—the *terminological* component, and assertional knowledge, which describes what is in it. By convention, the word ‘ontology’ usually refers to the *T-box* component, whereas ‘instance data’ is referred to as the *A-box* part of the knowledge base. Although in an RDF ecosystem the representation and semantics of the two do not vary, it is useful to separate them:

- T-box (ontology) data is re-usable across multiple sets of instance data. One domain ontology may be used independently in several organisations or applications; indeed, this is a fundamental motivation behind publishing ontologies on the Semantic Web, and ontologies are published as their T-boxes only for this reason.
- A-box data is re-usable across multiple ontologies. Given a set of assertional data about an application or domain, the level of reasoning (computational expense) may be dominated by a governing ontology; as such, switching this ontology for another may provide better performance for different applications.
- T-box and A-box data often (but not necessarily) have different reasoning characteristics, and thus may employ different reasoning strategies. For example, the axioms of a small, highly expressive ontology may be pre-computed by a DL reasoner, whilst large, dynamic A-boxes may be better suited to less expressive rule reasoning where reasoning completeness is less vital.

Henceforth in this thesis, the noun *ontology* is used to describe the *terminological* part of a model. Some models presented are shown populated with A-box vocabulary too; where this is the case, this part of the model is referred to separately.

---

<sup>17</sup> A commercial implementation of a reasoner that can interpret OWL axioms under closed world semantics is available as part of the Stardog triplestore used in [Chapter 5](#)

### 2.6.6 *RDF Storage and Presentation*

RDF models can be stored and queried in several ways. At the simplest level, they can be represented as triples in a physical file on a computer or hosted on a website. Many standard web ontologies are presented in this way: a document containing a number of triples, which together declare a vocabulary and set of axioms defining the ontology. Humans or computer agents then read these files, and process the information in them in some way.

More commonly for large datasets, RDF can be stored in a *triplestore*. A triplestore is a database built for storing large amounts of RDF content, and data access is provided through some form of query/update mechanism. Triplestores often implement methods for optimised storage and retrieval of content, and commercial offerings often advertise benchmarked performance for storage, querying, and inference in their marketing.

Finally, RDF can be presented by some form of mapping from other data formats. This is particularly useful in situations where interoperability is desired between systems but these systems themselves should not or cannot use semantic data models internally. Given the system's known semantics (for example, an XML schema or documentation), mapping software can translate queries for RDF data into queries for non-RDF data, fetch the results, map them into RDF, and then return them to the requester. This gives the functionality of a linked data store and interface, but makes sound/complete reasoning over assertional data unfeasible.

### 2.6.7 *Overview of Software Tools*

Owing to the emergence of the semantic web, a wide range of tools exist for editing and creating RDF and OWL data models. A brief overview of the main tools used in this thesis is provided below.

#### 2.6.7.1 *Ontology Development Environments (IDEs)*

Although ontologies can be fully described manually using DL notation formats and OWL abstract syntax, the rise of the semantic web and the desire to allow non-mathematicians to author ontologies has led to the creation of several mature ontology editing tools. These tools are typically used to create the terminology part of a data model—the domain knowledge itself—whilst other bespoke tools or mappers are used to import or acquire assertional knowledge depending on the application.

- **Protégé**<sup>18</sup> is an open source graphical ontology editing tool created by a team at the Stanford Centre for Biomedical Research. The current version of Protégé [148] builds on work carried out since 1987 on tools for biomedical knowledge representation [75]. Protégé allows manipulation of OWL ontologies, and provides tools for class/individual creation, axiom editing, DL reasoning, and basic visualisation. It is widely used, with recent surveys suggesting it is the most popular graphical editor available among ontology authors [120, 224].
- **Topbraid Composer**<sup>19</sup> is a commercial editor based on the Eclipse platform<sup>20</sup>. Sharing many features with Protégé, Topbraid Composer (TBC) also provides additional capabilities: mapping from non-ontology formats into models, better manipulation of A-box data (individuals), custom rule reasoning, and integration with several RDF triplestores. TBC was used extensively across the PhD project.

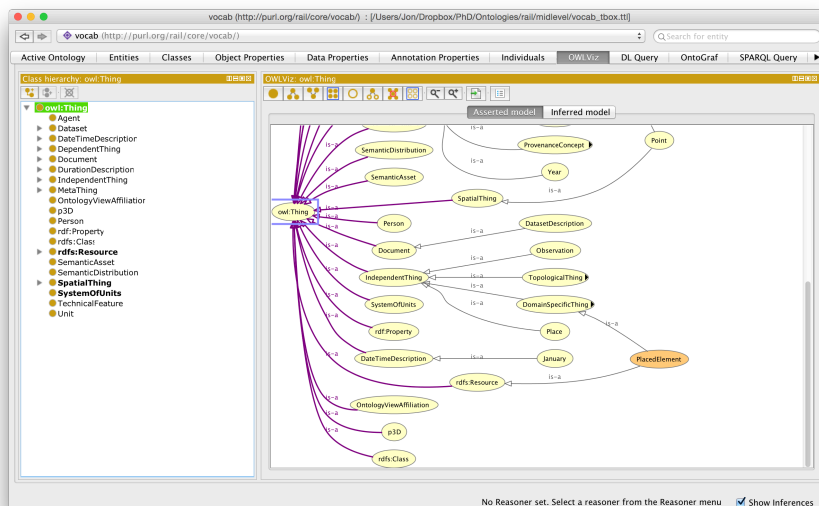


Figure 2.16: Protégé 5.0 Ontology Visualisation View

Many other ontology editing tools exist, and are not covered here. Meenachi and Baba [135] and Simperl and Luczak-Rösch [193] provide more detailed surveys and comparison.

<sup>18</sup> <http://protege.stanford.edu/>

<sup>19</sup> <http://www.topquadrant.com/topbraid/>

<sup>20</sup> <https://eclipse.org/>



### 2.6.7.2 *Software Development Libraries and Tools*

In development of applications based on semantic web technology, bespoke tools are often created to interface users or computer agents with data models. Several programmatic tools and libraries help this, and the following

- **Apache Jena**<sup>21</sup> is an open source Java library designed to aid development of semantic web applications. It provides a rich API for manipulating RDF models, a file or web-based RDF triplestore, and an inference API that supports OWL, RDFS, and rule reasoning.
- The **OWL API**, a Java library developed at the University of Manchester, provides similar functionality to Apache Jena but allows OWL concepts to be created and manipulated natively rather than through their underlying RDF model<sup>22</sup>. This makes working with ontologies easier, at the cost of limited flexibility when dealing with pure RDF graphs.
- **dotNETRDF** is another RDF library, written for the Microsoft .NET platform. Whilst it currently provides less functionality than the above Java libraries, it allows easy RDF integration in .NET applications, and is used heavily in implementing applications in [Section 6.2](#).

Several other tools were used in developing models and software throughout this thesis, and are described in other chapters. These include ontology mapping tools, modelling software, and visualisation applications. A well-maintained list of other semantic web development tools is provided on the former W3C Semantic Web Working Group Wiki<sup>23</sup>.

### 2.6.8 *Querying RDF Data*

In order to interact with RDF models, some method for representing or modifying data is required. The two approaches so far described have allowed for this to be done:

- Visually, using a graphical ontology editor
- Programatically, using pattern matching on triples—e.g. ‘show me all triples with `ex:MilesDavis` as subject’

<sup>21</sup> <https://jena.apache.org/>

<sup>22</sup> Jena also supports this to a limited extent, through the `OntModel` class

<sup>23</sup> <http://www.w3.org/2001/sw/wiki/Tools>

SPARQL (Sparql Protocol and RDF Query Language) standardises a language for executing more complex queries over graphs. SPARQL provides operations for finding and updating RDF data, and searches for data using patterns expressed in a format similar to Turtle (described above). For example, [Listing 2.6](#) searches for all known train stations in a model, and returns their URI and label.

```
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX vocab:<http://purl.org/ub/rail/vocab/>

SELECT ?station ?label WHERE {
    ?station a vocab:Station ;
    rdfs:label ?label
}
```












Listing 2.6: SPARQL Query for Railway Stations in Model

This query returns matches (a station and label) for every part of the model that matches the pattern inside the WHERE clause - that is, wherever there is an RDF triple stating [something] `rdf:type vocab:Station` and that same [something] also has an `rdfs:label [something else]`. All permutations of this pattern that exist within a graph are returned; if a station is marked up with more than one `rdfs:label`, as is common, two results are created. The standard allows for far more powerful querying than shown here; [Chapter 6](#) includes several more complex examples.

### 2.6.9 *Presentation of OWL Examples and Patterns*

To illustrate OWL constructs in the rest of this thesis, diagrams will be used in addition to the DL notation described above. These diagrams follow the style used by several OWL software tools and textbooks, including Protégé and Topbraid Composer, and specifically follow how patterns and examples are set out in Allemang and Hendler [5]. Sets of concepts (classes, individuals, properties, literals, and restrictions) are illustrated by rectangular boxes containing a symbol denoting type and followed by the concept's name or description using CURIE notation, and relations between these boxes are shown by a line with a directional arrow from one to another overlaid with the linking property's name. Elements drawn with solid black lines denote asserted knowledge, whilst dashed lines denote knowledge obtained through

Table 2.7: Icon Symbols Used to Denote OWL Entities

Symbol	Description
	Class
	Individual
	Literal (integer)
	Object property
	Datatype property
	Equivalent class restriction
	Existential restriction
	Universal restriction
	Union
	Max cardinality restriction
	Min cardinality restriction

inference.

Many of the examples used later in this thesis require that sets of individuals with an associated class be shown. Rather than explicitly diagramming these using two separate boxes related with an `rdf:type` relation between them, they are often shown directly atop one another, with the class entity in the upper box, and the individual in the lower box, and the `rdf:type` relation omitted. This representation makes many of the examples shown clearer to read. A full list of symbols to denote different types of concepts and restrictions is shown in [Table 2.7](#), and an example of this graphical representation of OWL examples is shown in [Figure 2.17](#).

This example shows the information from [Listing 2.4](#), with two extra facts to demonstrate the diagramming system: that `ex:MilesDavis` `rdf:type` `ex:Person`, and that `ex:MilesDavis` `ex:name` “Miles Davis”. The first of these facts is illustrated on the bottom left, and the second by the arrow and string literal shown bottom right. The `rdf:type` inferences caused by the domain and range restrictions on `ex:plays` are shown using dashed arrows.

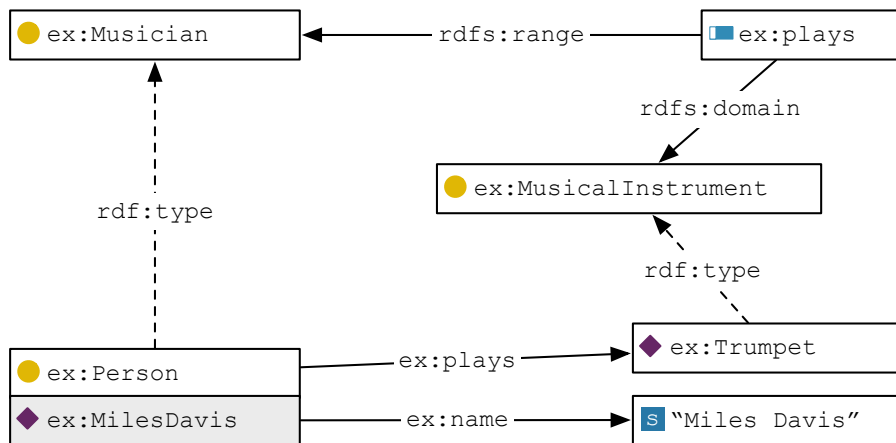


Figure 2.17: Demonstration Graphical Representation of OWL Example

## 2.7 PRACTICAL PROBLEMS AND ASSUMPTIONS IN OWL

Whilst there is a tendency to advocate OWL as a ‘silver bullet’ solution to data integration challenges, some characteristics and traits must be considered:

- The lack of *Unique Name Assumption (UNA)*, and the *Open World Assumption (OWA)* make operations such as counting assets impossible in OWL, and fundamentally restricts any form of constraint checking natively. In many enterprise systems it is desirable to assume a closed world, as the number of negative facts about a system usually vastly outweigh the number of positive facts over the same domain [56, p. 9]—in OWA systems, closure axioms must be asserted to assert these negative facts. [Section 2.6.4.2](#) discusses a number of existing approaches which allow closed world reasoning using OWL.
- The restriction of RDF to using *binary relationships* allows for efficient reasoning but does not suit certain real world concepts. This makes representation of such concepts (for example time and measurement) difficult[225], as a marked deviation from perceived real world semantics is required.
- Toolset maturity is not yet established. Although now supported by large enterprise software providers such as International Business Machines (IBM) and Oracle, widespread support for RDF/OWL technologies is still limited. In traditionally cautious environments such as the railway, the risk of investment in new technologies is often avoided, although newer initiatives within Network Rail may indicate that attitudes towards innovation are changing [195].

These traits can make for incorrect assumptions among new OWL users. A selection of common mistakes and anti-patterns partially caused by such traits is discussed by Rector et al. [182].

## 2.8 SUMMARY

Over the course of this chapter, an overview of the key data modelling techniques and technologies surrounding the central work of this thesis has been given. Data modelling as a discipline has been used as a tool to facilitate data sharing and in systems engineering for many years, but the semantics encompassed in these models are usually not formal enough to allow a machine to interpret and act on them. Ontologies provide a way of writing semantic data models using formal logic, and thus allow automated reasoning over knowledge bases that would have traditionally required bespoke interfaces or applications to accomplish.

Recent advances made in light of the development of the semantic web have provided practical tools that can be used to easily build both the models themselves and the software systems around them, and these tools are used extensively in chapters four, five, and six to build and implement ontology-based information systems for a variety of railway applications.

Having introduced the technologies used in this thesis, the next chapter will introduce prior art in the area of railway and industrial data modelling, as well as overviewing commonly-used OWL patterns and models.

## RAILWAY DATA MANAGEMENT, INDUSTRIAL MODELS, AND NOTABLE ONTOLOGIES

---

### 3.1 INTRODUCTION

This chapter highlights existing approaches to data exchange in the railway industry, and examines current state-of-the-art industrial data models and common ontologies. Initially, [Section 3.2](#) describes current railway data exchange practice and UK-based data sharing initiatives currently being undertaken. [Section 3.3](#) then describes standard transportation data models, and [Section 3.5](#) documents other models and common design patterns for semantic modelling, as considered in [Chapter 4](#) and [Chapter 5](#).

### 3.2 STATE OF UK RAIL DATA MANAGEMENT

#### 3.2.1 *Current Wheel Maintenance Workflow*

Firstly, to illustrate the inefficiencies caused by information silos in the current UK rail industry, an overview of one particular maintenance workflow is given, gathered from interviews with the head of maintenance at a large UK train depot. The use of Remote Condition Monitoring to diagnose asset faults across the rail industry is growing, but implementation of ‘siloes’ systems can lead to inefficient and potentially erroneous exchange of data, as described below.

#### WHEEL IMPACT LOAD DETECTION AND MAINTENANCE

Train wheels must be completely round for proper operation across the network. Wheels which are ‘out of round’ cause discomfort to passengers damage to railway track, and for this reason infrastructure maintainers such as Network Rail continuously monitor trains for wheel ‘flats’ using devices called Wheel Impact Load Detector (WILD). Wheel Impact Load Detectors (WILDs) monitor one point on the railway network and record the force with which wheel pass over track, and raise alarms if this impact load exceeds a threshold. Identified wheels are later re-turned on a lathe, preventing further damage. The current workflow for this system is shown in [Figure 3.1](#). Currently, several interfaces between machine and human operator

exist, as wheel impact data is taken from its silo, manually compared to train running information in another silo, and finally input into a maintenance system silo.

A future workflow involving shared data could save significant effort. Information shared between the WheelChex system and an existing realtime Train Describer system would allow inference of the identity of a train, whilst consignment information from a timetabling system would allow the rolling stock identity and direction to be gathered. This information could then be used by a maintenance planner, or scheduled automatically. Maintenance engineers re-profiling a wheel could have access not only to the identity of the train, but its history and other maintenance information.

### 3.2.2 *Network Rail Intelligent Infrastructure*

In contrast to the current siloed Wheel Impact Load Detectors (WILDs) workflow, Network Rail's Intelligent Infrastructure Project (IIP) is a recently implemented remote condition monitoring platform that facilitates 'predictive' maintenance, from data acquisition to data analysis. By instrumenting 22 000 railway assets and monitoring them for failures, the system has so far saved an estimated 300 000 delay minutes through predictive maintenance, and saved over £9m for Network Rail [222] in reduced 'delay minutes'<sup>1</sup>. IIP currently monitors five types of railway assets, with data marked up using the International Organization for Standardization (ISO) 13374 exchange format (discussed in Section 3.5.1). Further project progress aims to integrate trend analysis for prognostic assessment and automatic advisory generation for maintenance of degraded assets [197]. The components of the IIP systems are illustrated in Figure 3.2.

The IIP project does not consider integration of existing data on the railways, and uses an off-the-shelf systems to store and analyse data. Interfaces to other systems are not considered in the available literature, although Invensys Wonderware<sup>2</sup> is identified as the Supervisory Control And Data Acquisition (SCADA) data store and analysis solution. The extent to which data to and from the IIP system is or will be shared is unknown [222].

---

<sup>1</sup> Financial attribution of faults on the UK railway is measured by the total amount of time affected services are delayed by.

<sup>2</sup> <http://software.invensys.com/wonderware/>

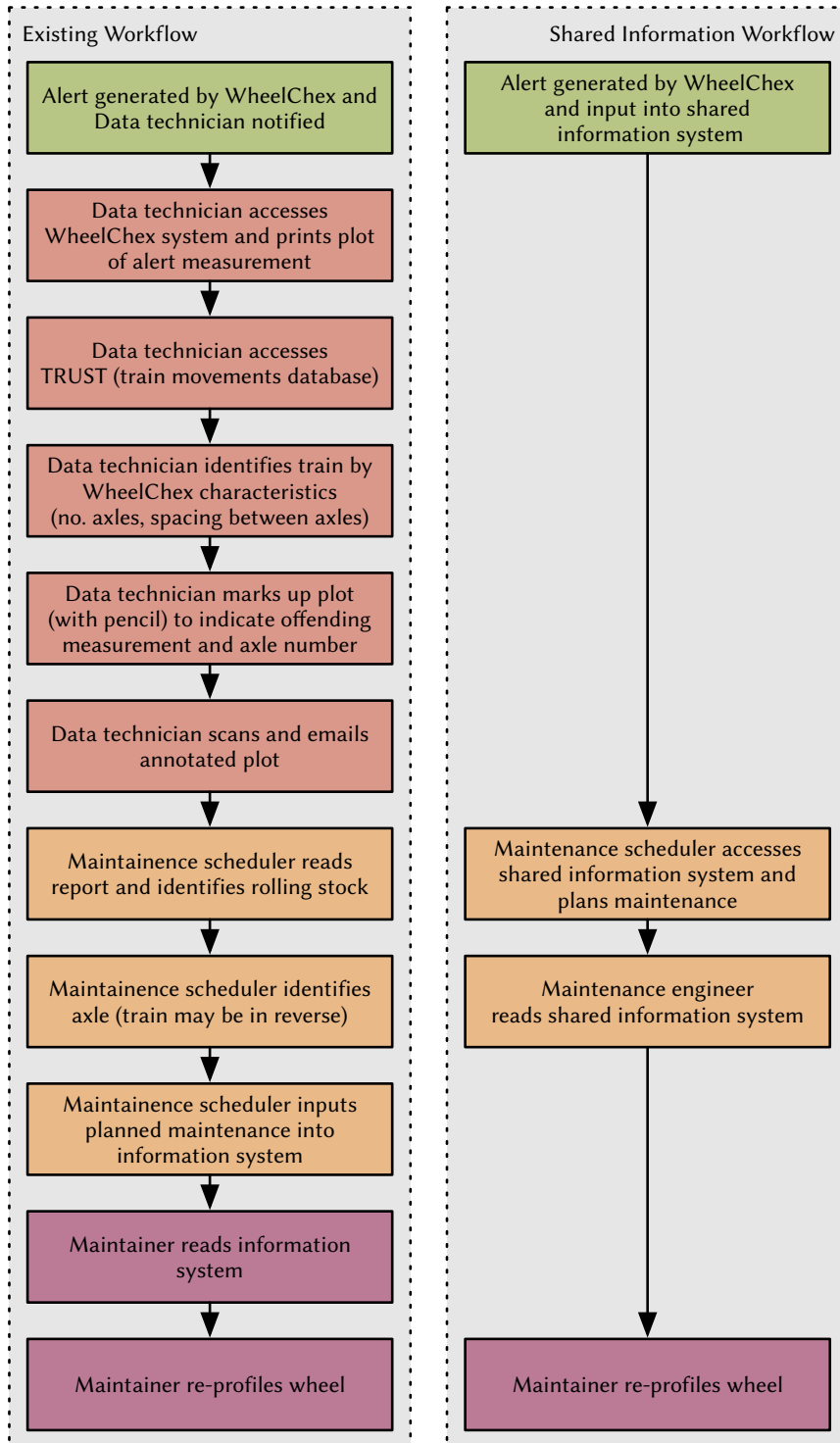


Figure 3.1: Flow Chart of Current and Future Train Wheel Maintenance Workflows (data taken from Groom [82])



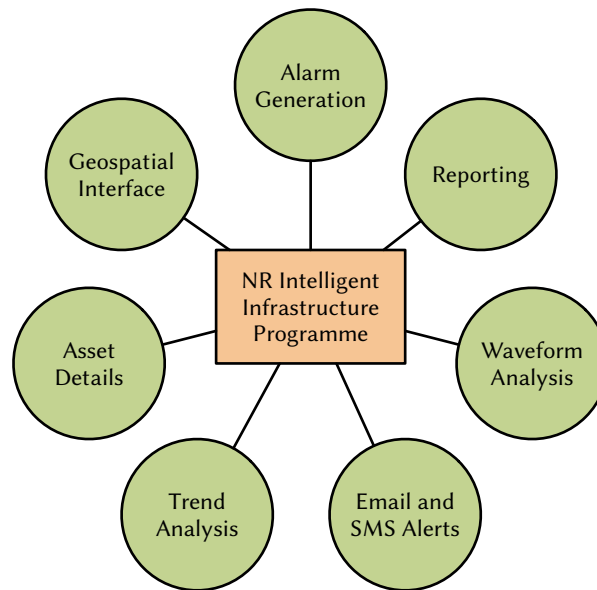


Figure 3.2: Network Rail Intelligent Infrastructure User Interface Components

### 3.2.3 *DARWIN and Network Rail ORBIS*

Two other data sharing initiatives in the rail industry are also relevant to work undertaken in this thesis. DARWIN is a Association of Train Operating Companies (ATOC)-owned system aimed at providing consistent passenger information across the country, whilst Offering Rail Better Information Services (ORBIS) is a Network Rail initiative for providing railway maintainers with accurate asset information.

#### DARWIN

Initially implemented as an ATOC-only system, DARWIN is a nationwide system for providing Real Time Passenger Information (RTPI) driven from train running data sources such as Train Running System TOPS (TRUST). In the past ten years, significant investment has been provided to build interfaces between several bespoke Train Operating Company (TOC)-owned passenger information data sources and the ATOC-owned system, in order to provide more unified information.

In April 2014, DARWIN began integrating data from Real Time Passenger Information systems owned by all UK TOCs. Owing to the heterogeneity of systems, DARWIN currently only accepts a minimal set of data from each, rather than being able to take full advantage of specific data available in each TOC 'silo'. This manifests itself in discrepancies between publicly available passenger information data,

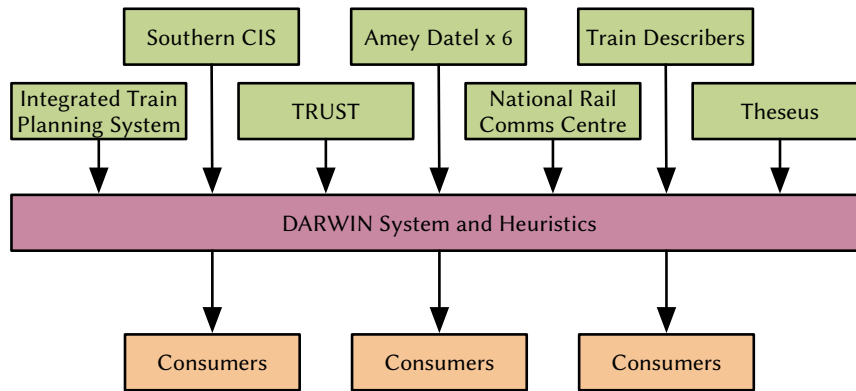


Figure 3.3: Principal DARWIN Input Data Sources (re-drawn from National Rail Enquiries [144])

and that displayed at railway stations. As more and more Real Time Passenger Information (RTPI) systems are implemented, more interfaces to DARWIN are required, each of which need individual maintenance and upkeep.

#### ORBIS

Network Rail's Offering Rail Better Information Services (ORBIS) system is a series of projects centred around providing staff with better access to existing asset information data across diverse sources. It is a £330m project running from 2012 to 2019, and has been credited with having saved the company £27m since its introduction [144]. ORBIS to date provides applications for the following purposes [144]:

- Workforce safety support
- Controlled document distribution
- Asset condition monitoring and reporting
- Infrastructure modelling
- Wheel impact analysis

Key applications focussed on for 2014 to 2019 include [145]:

- Mobile data and works management, to allow maintenance staff to collect and view asset-related data<sup>3</sup>.

<sup>3</sup> The foundations of a system for mobile asset management and analysis have already been established by Network Rail by commissioning the 'Linear Asset Decision Support' system, as described in Bentley [16]

- Geography and location data improvement, by acquisition of new track data
- Improved management of information handover between railway stakeholders
- Development of a Rail Infrastructure Network Model (RINM) for central data exchange

Several of these objective align significantly with the work undertaken in this thesis, and coordinate with efforts across the European Union to develop standardised railway infrastructure models. Whilst the data acquisition and design of many of these systems is already underway, the company recognises that semantic data models provide a longer term solution to ensuring that information is available across the entire organisation [146].

### 3.3 TRANSPORTATION DATA MODELS AND FRAMEWORKS

This thesis build upon several prior efforts to create standard railway data models for a number of applications. A review of the state-of-the-art in railway data modelling is described below in order to contextualise the work carried out in this thesis.

#### 3.3.1 *RailML*

RailML<sup>4</sup> is a cross-industry data model developed by a consortium funded by a group of European railway infrastructure maintainers, rolling stock firms, and software providers. Developed as a set of XML schemas, RailML covers a set of data exchange use cases driven by its stakeholders, and has a significant bias towards railway simulation and planning data. Although studies of its uptake in production systems are elusive, many railway software tools now support RailML import and export functionality, and recent legislative developments across Europe [59] have led to further buy-in by stakeholders.

RailML comprises the following schemas:

- **Infrastructure (IS):** Terms, relationships, and restrictions concerning infrastructure representation, track layout, and static infrastructure assets such as signals, platforms, and train stations.

---

<sup>4</sup> <http://www.railml.org/>

- **Rolling Stock (RS):** Maintenance and operations data for rolling stock concepts, such as vehicles, train consists, composition, and type.
- **Timetable (TT):** Schema for timetable planning, used predominantly for storing markup for train simulators. The current (version 2.2) timetable schema is not well documented, but semantics can be deduced from example files.
- **Common (CO):** Metadata concepts common to all subschemas
- **Interlocking (IS):** A new draft schema recently contributed to by the European Union (EU) Optimal Networks for Train Integration Management across Europe (ONTIME) project. The draft IS subschema stores interlocking logic and routing data, with the aim that this data can be used for simulation and design work.

Whilst figures for its uptake are not published, RailML has notably been used in large EU research projects such as ONTIME [3, 19], and advertises applications by several high profile organisations on its website<sup>5</sup>. Work in this thesis draws upon the RailML vocabulary extensively, as it provides an authoritative, direct and unambiguous set of terms for reuse.

### 3.3.2 TAF/TAP TSI

Telematics Application for Passengers (TAP) and Telematics Applications for Passenger Services (TAF) are two railway interoperability standards recently published by the International Union of Railways (UIC). They are aimed at defining Europe-wide procedures and interfaces ‘between all types of railway actors’ [57] in order to better facilitate transport links and changeovers between countries. The two standards are as follows:

- **Telematics Application for Passengers (TAP)** dictate a set of requirements for infrastructure managers to fulfil to facilitate interoperability across EU passenger train services.
- **Telematics Applications for Passenger Services (TAF)** are interoperability standards for freight, and are oriented towards freight services rather than passengers.

---

<sup>5</sup> <http://www.railml.org/index.php/applications.html>

## TAP TSI

The TAP requirements centre around the provision of passenger-centric data across Europe, and is based around eleven key activities as described in [Table 3.1](#).

Table 3.1: TAP Key Activities and Descriptions

	Task	Description
1	Common Requirements	Key vocab from TAF
2	Making Own Reference Data Available	Network data
3	Train Preparation	Train ready /not ready
4	Train Running	Forecasts/disruptions
5	Passenger Information	Station /vehicle-based information
6	Timetable Data	Passenger timetable exchange
7	Tarriff Data	Fare information
8	Reservation Data	Seat and bicycle bookings
9	Ticketing	'Print-at-home' cross-EU ticketing
10	PRM Assistance	Assistance requests and contact details
11	Retail Architecture	Contact details and retail reference data

Motives for TAP come from the desire (and mandate) to better facilitate passenger travel across Europe. EU passenger rights legislation requires that travellers are presented with options to plan the shortest/cheapest trip, as well as being shown accessibility information, seat availability, and procedures for filing complaints.

## TAF TSI

The specification for TAF was introduced by the European Commission in December 2014 and requires all railway stakeholders involved in the transport of freight to provide data corresponding to the following use cases [57]:

- Applications for freight services, including information systems (real-time monitoring of freight and trains).

- Marshalling and allocation systems, including information systems (real-time monitoring of freight and trains).
- Reservation systems, whereby here is understood the train path reservation.
- Management of connections with other modes of transport and production of electronic accompanying documents.

Under the regulations, participating organisations are required to supply at least the following types of information:

- Train paths
- Running information (departure, interchange, and arrival points and times of contracted transport)
- Estimated time of arrival of freight trains
- Service disruption information

#### TIMELINES AND IMPLEMENTATION STRATEGY

Within the UK, Network Rail have published a TAP conformance plan that predicts a full implementation by 2018 [14]. European-wide progress on both TAF and TAP projects are shown in [Figure 3.4](#).

#### 3.3.3 *RailTopoModel and National Topology Models*

The UIC RailTopoModel [114] is a recently developed initiative to provide a data standard for railway topology mapping across Europe. Developed by the desire to allow network data to be shared between infrastructure managers effectively, RailTopoModel defines a data exchange format for the interchange of railway data, drawing from several existing data models [113]:

- **Register of Infrastructure Model (RINF)** [59], a format corresponding to the EU regulation for infrastructure managers to submit topology information to the EU on a quarterly basis
- **Infrastructure for Spatial Information in the European Community (INSPIRE)**<sup>6</sup>, a European Union Geographical Information System interoperability standard.
- **InfraNet**, the infrastructure model of InfraBel, Belgium

---

<sup>6</sup> <http://inspire.ec.europa.eu/>

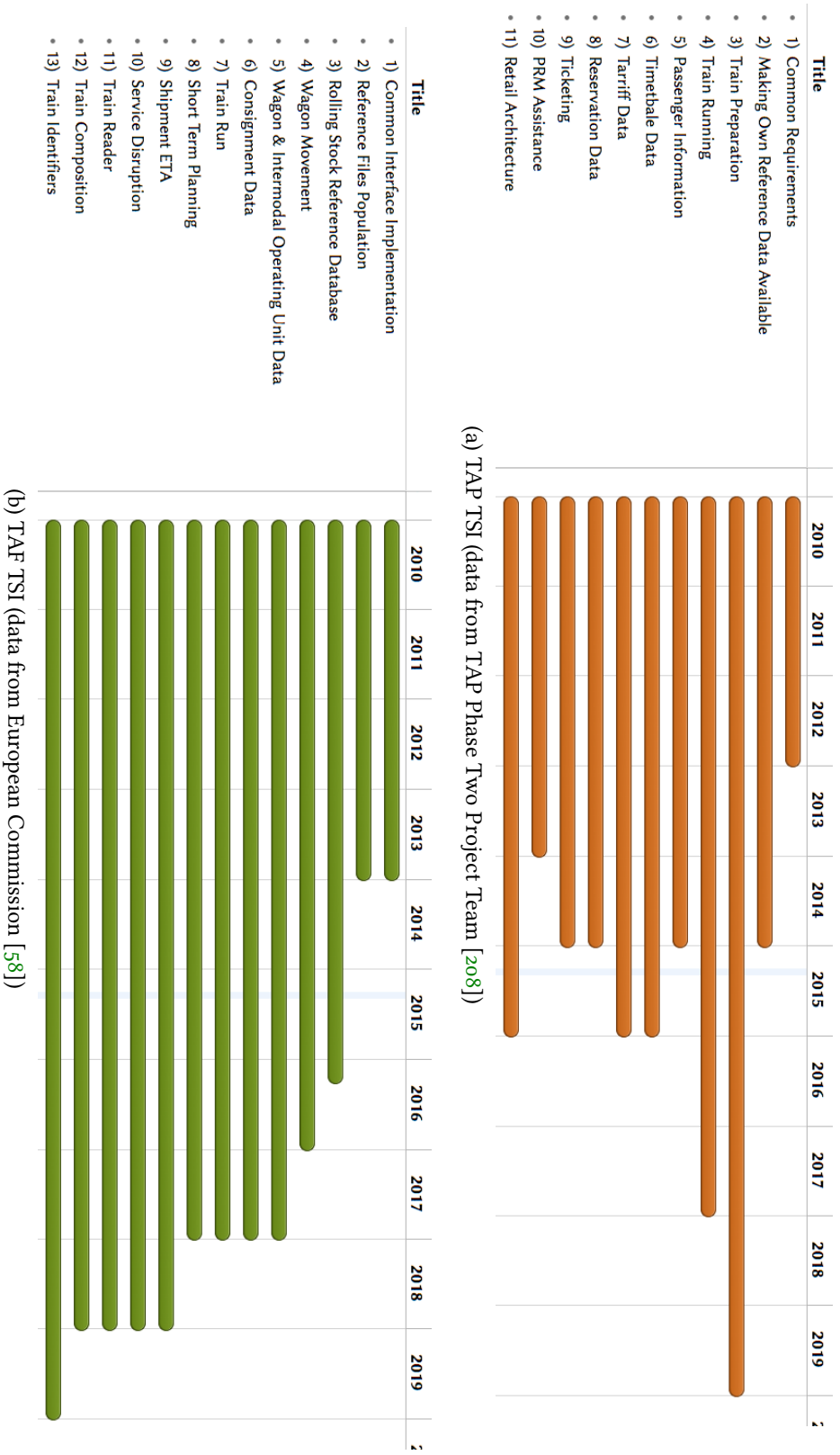


Figure 3.4: TAF/TAP TSI Implementation Tasks and Timescales for Completion

- **ARIANE**, the infrastructure model of Réseau Ferré de France (RFF), France.
- **Banedata**<sup>7</sup>, the infrastructure model of Jernbaneverket, Norway
- **RINM**, the infrastructure model of Network Rail.

Its stated use cases are predominantly around minimising the effort required to interchange topology data in Europe, by unifying existing standards from member countries. As an XML standard, Rail-TopoModel also requires buy-in from every state, and will form part of the next major version of RailML.

#### 3.3.4 *Proprietary Systems and Models*

Two proprietary railway data models were also encountered over the course of this project, as used by Invensys Rail Group and Network Rail respectively.

- **Invensys Layout Description Language** is a geographical format described in a proprietary syntax based on Backus Naur-Form notation. It was used internally at Invensys Rail Group (IRG) for exchange of signalling layout diagrams in proprietary design tools. Developed initially as part of a collaborative project with the University of Manchester, it encapsulates track topology, control systems, and signalling information, but is not in wider use within the rail industry [32]
- **Network Rail Signalling and Data Exchange Format** is a similar effort undertaken by Network Rail, and originally intended as a standardised infrastructure and signalling model for use by Network Rail (NR) suppliers and customers. Its uptake outside of the organisation is unknown, although vocabulary and concepts from Signalling and Data Exchange Format (SDEF) are considered in the design of ontologies shown in [Chapter 5](#) and [Chapter 6](#).

#### 3.3.5 *InteGRail*

InteGRail [108] was a collaborated project funded by the European Commission from 2001–2006. It aimed to:

---

<sup>7</sup> <http://www.njk.no/banedata>



“[...] create a holistic, coherent information system, integrating the major railway sub-systems, in order to achieve higher levels of performance of the railway system in terms of capacity, average speed and punctuality, safety and the optimised usage of resources.” [108].

Much of the work in InteGRail focussed on the use of ontologies for rail data exchange. Four such ontologies were built as part of the project:

- **Core Ontology** [109]: High level concepts, including rail-specific entities that are common to most use cases.
- **Hot Axlebox Detector Ontology**: An application-specific ontology to capture knowledge about overheating railway axle boxes
- **Wheel Impact load Measurement Ontology** [128]: A further application-specific ontology to support detection and diagnosis of high railway vehicle wheel impacts.
- **Network Statement Checker Ontology** [217]: An extension to support a cross-european transport planning application for conformance of a route to a selected set of capabilities.

The project also implemented a Service-oriented Architecture for exchange and querying of data across Europe, and presented this proof-of-concept by allowing applications to run at geographically dispersed locations. Resolution of queries from across disparate sources was undertaken by consulting a central service repository, sending SPARQL queries to individual datasets, and performing local queries over the union of the results [68].

#### 3.3.5.1 *InteGRail Core Ontology*

The InteGRail core ontology was a model inherited by all applications used in the project. It sets out fundamental railway terms, and ways of representing key concepts. Its contents can be generalised into four main components:

- A set of railway vocabulary, elicited by InteGRail team members. This centres around railway asset vocabulary, but provides very sparse semantics for terms. Most vocabulary terms are placed in an ‘isa’ hierarchy for categorisation, but have no further relationships to each other.

- A design pattern to describe railway topology, allowing representation of railway networks at two levels of generalisation, and provides constraints for capturing relationships between low level ('road') and high level ('route') concepts.
- An 'observation' design pattern to represent measurements in asset monitoring systems, including their values and diagnoses.
- Time and measurement concepts reused from existing models.

### 3.3.5.2 *InteGRail Applications*

The InteGRail project produced two demonstration data integration applications that are relevant to [Chapter 5](#) and [Chapter 6](#). The first focussed on the integration of condition monitoring data from multiple sources to diagnose faults on vehicles that could not be found in constituent systems. By asserting the relationships between faults and railway components, rule reasoning could be used to deduce the likely severity of cross-data-source faults [128]. Ontological models of railway infrastructure such as [Figure 3.5](#) provide the reasoner with enough knowledge to establish which components may be faulty in a given scenario. Additionally, probabilistic reasoning was demonstrated to provide a 'best guess' estimate of fault status, rather than relying on the usual monotonic nature of OWL [127].

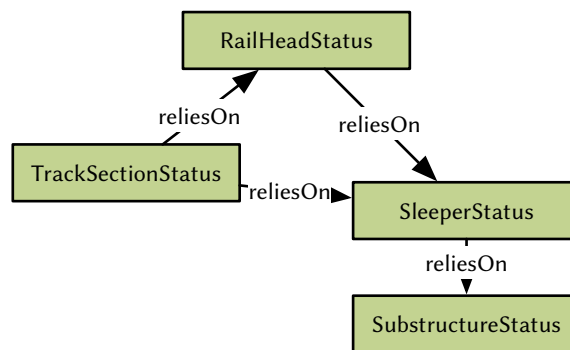


Figure 3.5: Railway Infrastructure Status Dependence Pattern (from Lewis et al. [128])

The second application showed how a graph of infrastructure and capabilities could be exploited to allow for network compatibility checking of a particular train for a particular route across Europe. By extending the rail core ontology to encompass knowledge of track capabilities, the application used federated SPARQL queries and OWL reasoning to check if a proposed route through the railway network was compatible with a particular train [217]. Topology data was taken

from several locations, and either stored as RDF at each location or mapped from other formats to form ‘virtual RDF’ stores using D2RQ<sup>8</sup>. The architecture and design of these InteGRail demonstrators is of particular relevance to the new applications presented in [Chapter 6](#).

### 3.3.6 Rail Functional Architecture

The Rail Functional Architecture (RFA) model is the result of a project undertaken by RSSB very recently that aims to identify the technological functions that are performed in order to operate a modern railway [176]. The result is a TRAK [166] diagrammatic model that identifies a vast number of railway functions, from enterprise level to operations level. From its perspective as a functional model, it does not identify any particular technologies or entities explicitly, but provides specific details of railway operations roles and tasks.

The model itself focuses on three main levels of function, as required by TRAK:

- The ‘Enterprise Perspective’ describes enterprise /capability goals, such as ‘provide value for money’ and ‘deliver UK transport policy’. It references more specific functions on the concept perspective as dependencies, so ‘provide value for money’ requires ‘control cost’ which requires ‘provide passenger services’.
- The ‘Concept Perspective’ describes lower level railway functions required to achieve the enterprise capabilities, such as providing freight and passenger transport, safety management, and service planning. Each of these concepts is related to others where appropriate, and national/EU railway standards are referenced appropriately.
- The ‘Solution Perspective’ identifies how concept activities are achieved operationally. The RFA currently contains a few example solution perspective diagrams, but this part of the model is not complete for all concept entities.

[Figure 2.2](#) shows a TRAK representation of the ‘train movement’ activity present in the concept perspective in the Rail Functional Architecture (RFA). Two other perspectives are also specified as part of the model: the ‘management perspective’ and the ‘procurement perspective’. The management aspect is intended to convey the scope of the model as it is, and the procurement perspective, not yet implemented, focuses on the methods in which different functions are physically

<sup>8</sup> <http://d2rq.org/>

achieved. The scope of the model covers all aspects of railway operations, but is designed to describe exclusively the UK railway industry as it currently exists.

### 3.4 ISO 15926

In comparison to the numerous railway-oriented data exchange standards, work in this thesis also draws upon two comprehensive generic models for cross-domain data integration, and particular in their approaches to modelling certain concepts. The first of these is ISO 15926, “Integration of life-cycle data for process plants including oil and gas production facilities”—a set of data integration standards originally created for asset management across the oil and gas process industries. It currently comprises six parts, and is not only relevant owing to its applications of enterprise ontology, but also to its approach to representing temporally changing information, as discussed below. The first parts to be published consider the conceptual modelling of the process industry, with parts 1, 2, and 4 being of interest:

- **Part 1** is a general summary of the ISO 15926 project
- **Part 2** is a generic domain ontology, written in EXPRESS-G, for process management. Although designed for the oil and gas process industry, the data model is vocabulary-agnostic.
- **Part 4** is a set of reference data (vocabulary) for the oil and gas domain, which populates the ontology outlined in part 2.

More recent parts are concerned with uptake and tooling:

- **Part 7** defines a set of *templates* for implementing ontologies. These are similar to ontology design patterns [70] or Data Shapes as defined by W3C Data Shapes Working Group [220], and concern the ways in which data is presented within the ontology.
- **Part 8** is an implementation of parts 2, 6, and 7 in RDF and OWL, although this implementation is not openly licensed.
- **Part 9** is unpublished, and considers a federated implementation using SPARQL called ‘facades’.

Work in ISO 15926 has been used by Shell and Bentley, and is gaining traction in other areas. The generic conceptual model used to drive it has several key benefits that are of interest in the context of this work:

- **“4D” Architecture.** ISO 15926:2 takes a *perdurantist* approach to time, where every concept is described not by its traits, but by its spacial or temporal characteristics. As such, the ontology makes it very easy to specify the composition of entities over time, by the definition of ‘timeslices’ that characterise an individual over time. This approach is described in more detail below in [Section 3.6.2](#).
- **Rich compositional modelling.** The ontology takes a thorough mereological view on how entities are composed, and makes a distinction between physical ‘things you can kick’ and the roles which they inhabit. This allows the ontology to represent the lifecycle of a plant and all of the components in it as both a set of roles, and a set of physical things. It is possible to ask ‘how many physical pumps have fulfilled the role of PumpXYZ’, or ‘What role does motor C carry out in the system?’.
- **Rich definition of characteristics and physical quantities.** Following the ontological ideals that physical properties should not be defined as attributes of an object (as physical properties are not inherent to the object itself), ISO 15926:2 proposes an architecture that allows objects to be assigned properties using classes that describe the nature of the assignment. For example, a ‘temperature\_setpoint’ class may allow measurement units of ‘temperature\_quantity’, which in turn allows Kelvin, Celsius, or Fahrenheit measurement scales. This is a complex way of defining quantities in ontology, but preserves data meaning such that reasoning on these quantities can be performed later.

ISO 15926:2 is a thorough generic conceptualisation of a domain model, which is likely to be effective in facilitating data sharing across many domains. However, its high expressivity comes at a price, and its formal semantics are not fully representable in OWL Fiatch [66].

### 3.5 GENERIC ASSET INFORMATION INTEGRATION STANDARDS

In addition to ISO15926, a review of two other relevant approaches to asset information integration are provided: the MIMOSA ISO standards, and a Siemens ontology-based method for ontology-based information exchange.

### 3.5.1 MIMOSA OSA

ISO 13374 ‘Condition monitoring and diagnostics of machines’, is a standard that provides a conceptual framework for enterprise information exchange and condition monitoring systems. Its development was assisted predominantly by MIMOSA, a cross-industry non-profit organisation, whose ‘Open Systems Architecture’ forms the foundation of both parts of the standard. MIMOSA OSA is divided into two parts:

- ISO13374:1 provides a general overview of the structure and data requirements of a standardised condition monitoring system, from data acquisition to prognosis generation. It suggests a number of discrete stages in this process, and overviews the requirements for each.
- ISO13374:2 ‘Data Processing’ defines a comprehensive specification for a data processing system to conform the general architecture set out in part one. It defines the requirements for each stage as shown in [Figure 3.6](#), as well as the interfaces between components and the levels of abstraction for which a processing system should be specified.
- MIMOSA OSA-CBM is an implementation of ISO13374:2, which provides a conformant conceptual model, XML implementation, and toolset to facilitate exchange of data across condition monitoring systems. The promise of OSA-CBM is to facilitate a standardised, modular system for communication between applications in order to reduce costly bespoke interfaces.
- MIMOSA Open System Architecture for Enterprise Application Integration (OSA-EAI) complements the CBM model by defining a model for representation of the assets and infrastructure within a system itself. EAI provides another conceptual model, as well as a reference relational database implementation and data library for storing and describing assets, and a set of query and exchange standards for communication of this data. This database provides coverage of many subject, including asset reliability, vibration analysis/condition monitoring data storage, and work management. The components of the OSA-EAI standard are shown in [Figure 3.7](#).

The ISO13374 standards and associated MIMOSA implementations are intended as a comprehensive model for data exchange of asset condition data and diagnoses. As such, they prescribe a very good

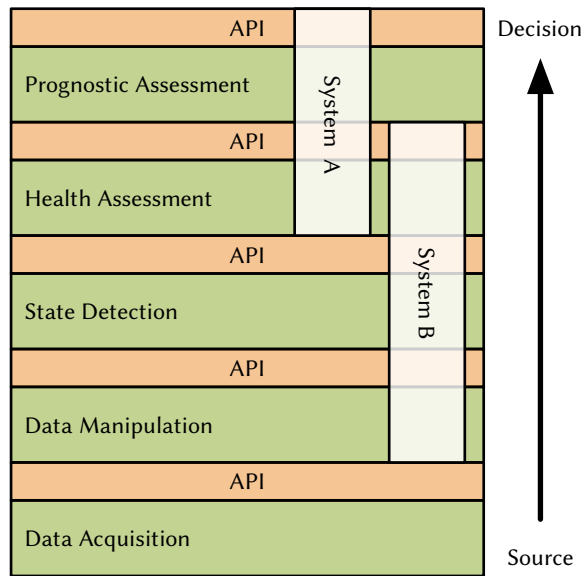


Figure 3.6: MIMOSA OSA-CBM Architecture Levels And Example Applications (adapted from International Standards Organisation [110])

		Application HTTP/SOAP Spec
DB Producer/Consumer Spec	Data Exchange App Spec	Client/Server Application Spec
DB Data Exchange Format Epc	Data Exchange Query Spec	Client/Server Transaction Schema
Reference Data Library (XML)		
Logical Model (XSD & SQL)		
Conceptual Object Model (UML/Visio)		
Terminology (PDF)		

Figure 3.7: Key Components in MIMOSA OSA-EAI Standard

way of integrating and exchanging data between components of a CBM system, and are used extensively in this field by projects such as Network Rail's Intelligent Infrastructure Programme [197], as well as in prior project by Boeing [126].

It does not, however, address data integration standards outside of this domain. Whilst its conceptual models could be extended, their current implementations in relational databases and XML schemas still require any non-conformant system information to be integrated using bespoke interfaces, or require an extension or modification to the standard.

### 3.5.2 Siemens Ontology-based Data Access System

Kharlamov et al. [119] present an ontological model and demonstration system for facilitating data interoperability and integration in the energy domain, and particularly in order to unify diagnostics data for a range of Siemens-manufactured turbines and process equipment. The motivations for this work are similar to those highlighted by the InteGRail, MIMOSA, and ISO13374 initiatives already described, and describe difficulties and expense accessing data analysis services within Siemens, and the need to facilitate preventative maintenance. The Siemens application ontology presented includes axioms to allow inference of condition on faulty machine components, and utilises reasoning and RDF streams to provide a demonstration system with temporally-aware information on the condition of a set of assets. The authors cite a lack of available production-ready systems for acting on temporal data, and use a novel proprietary technique and query language to provide data access.

## 3.6 RELEVANT ONTOLOGIES AND COMMON MODELLING PARADIGMS

This section overviews a selection of ontologies and design approaches that are of relevance to the work undertaken in [Chapter 4](#) and [Chapter 5](#). The first section outlines notable *upper ontologies*, which attempt to provide a high level standardised platform for representing knowledge across multiple domains, whilst the following sections discuss approaches to two common modelling issues: representing temporal information, and representing quantities. These approaches are discussed further where they are utilised in [Chapter 5](#).



### 3.6.1 *Upper Ontologies*

Upper ontologies are models of high level concepts that are common across many domains and disciplines. They seek to provide a framework for lower level ontologies to adopt and use, such that information from across multiple domains can be shared. Some such ontologies have enjoyed widespread adoption in some disciplines, such as the Basic Formal Ontology (BFO) within biosciences [97]. Whilst a comprehensive review of these is given by Mascardi, Cordi, and Rosso [134], two are of relevance to the approaches described in this thesis:

#### THE BASIC FORMAL ONTOLOGY

The Basic Formal Ontology (BFO)[79] was created at the University of Leipzig, and is intended for use across a number of different domains including medicine , geography, and disaster relief[80]. Its relevance to this thesis lies in its ability to represent temporal constructs using two different paradigms, which loosely fit with the 3D and 4D approaches taken by ISO 15926, and allows entities to be represented using one or the other depending on circumstance. This is attractive for modelling temporal characteristics of railway data models, as it allows intuitive expression of both relatively static data (such as infrastructure maps) and dynamic data (such as timetabling).

#### DOLCE ULTRA-LITE

The DOLCE & DnS Ultralite (DUL) ontology<sup>9</sup> is a simplified subset of Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE), and provides a very understandable and semantically sound framework to work from. It possesses a wider scope than that required of the RaCoOn ontologies, and provides a large number of concepts in areas that are not relevant to RaCoOn, owing to DOLCE's roots as a linguistic and cognitive engineering upper ontology.

### 3.6.2 *Approaches to Time Representation*

The majority of data models considered so far are *synchronic*—that is, they have no mechanism for representing data that varies over time, and represent a view of the world at one instant. In the case of many applications, this works well: models are kept updated to reflect the view of the world *now*, and versioning systems are used to store snap-

<sup>9</sup> <http://www.loa-cnr.it/ontologies/DUL.owl>

shots of past system states. In some environments, however, there is a need for explicit representation of temporally-varying data; the need to represent *diachronic* information. In order to do this, two theories of time can be considered Loux [130]:

- **Endurantism** asserts that objects, called *endurants*, always exist but have changing properties. The ‘Jon Tatcher’ that exists today is the same ‘Jon Tatcher’ that existed last year, but with fewer hairs on his head. Changes in time are represented by properties with known temporal extents, as demonstrated by Figure 3.8.
- **Perdurantism**, or *four-dimensionalism*, regard different perceptions of an entity over time as different objects, which have some spatial extent. The identity of a concept is considered the aggregate of all its temporal parts. This is the approach modelled in ISO 15926, and is well suited to process engineering and domains in which roles that change over time must be commonly represented. A perdurant representation of the example shown in Figure 3.8 can be seen in Figure 3.9.

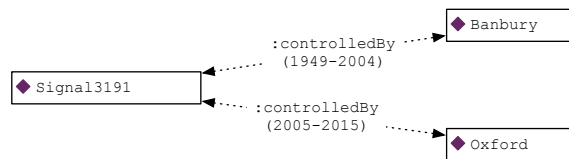


Figure 3.8: Example of Temporally-Varying Data Represented Using Endurantist Approach

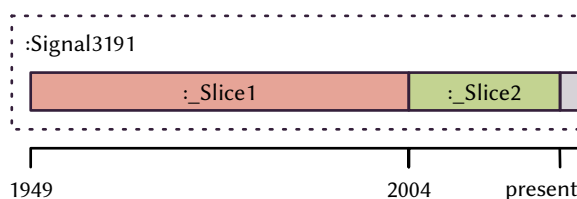


Figure 3.9: Example of Temporally-Varying Data Represented Using Perdurantist Approach

#### REPRESENTING ENDURANTS IN OWL

To represent temporally-changing data using endurants in OWL, a mechanism for defining the times over which relations hold is needed.

This can be done either by re-defining concepts (classes or properties) as new entities for every time extent, or by representing these relations using reification as described in Section 2.6.1.3. The first approach is shown in Figure 3.10, while the second approach is shown in Figure 3.11.

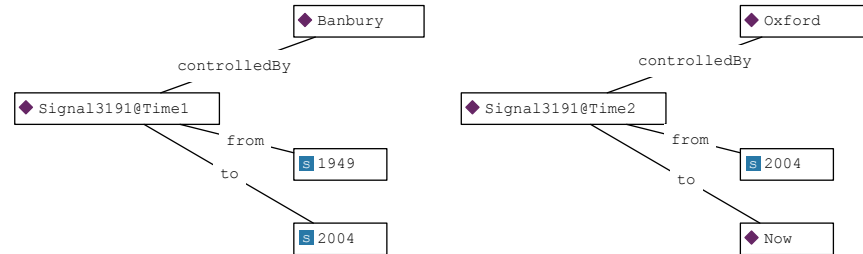


Figure 3.10: Representation of Time Extents Using Endurant Entities

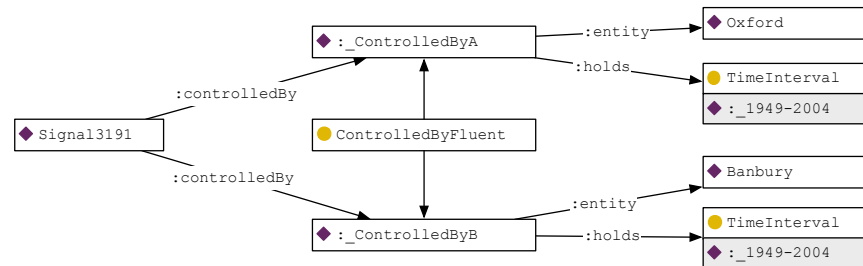


Figure 3.11: Reified Endurantist Time Extents Representation

The latter method imparts all of the required knowledge, and can be extended to a number of other scenarios involving time variant characteristics. Both of these forms of reification, however, have several drawbacks:

- **Proliferation of objects.** Reification by necessity creates redundant information, and in the example given creates at least six triples, where only two assertions in an appropriate First Order Logic (FOL) knowledge base would be required.
- **Reduction in OWL reasoning capability.** Employing this pattern across a whole ontological knowledge base dramatically reduces the usefulness of tractable OWL reasoning. There is no longer a single binary relation from the subject to intended object, so axioms such as inverse properties, transitivity, functionality, and symmetry cannot easily be asserted. The `rdfs:domain` and `rdfs:range` of each relationship also change, and this must be taken into account.

## IMPLEMENTING 4D ONTOLOGY IN OWL

The ISO15926:2 ontology uses ternary relations to implement its perdurantist view of the world in EXPRESS, a highly expressive but computationally intractable formal ontology language. In OWL, ternary relations are not straightforward to model, but modelling 4D concepts can still be achieved by using a reification technique proposed by Welty, Fikes, and Makarios [225], as shown in Figure 3.12.

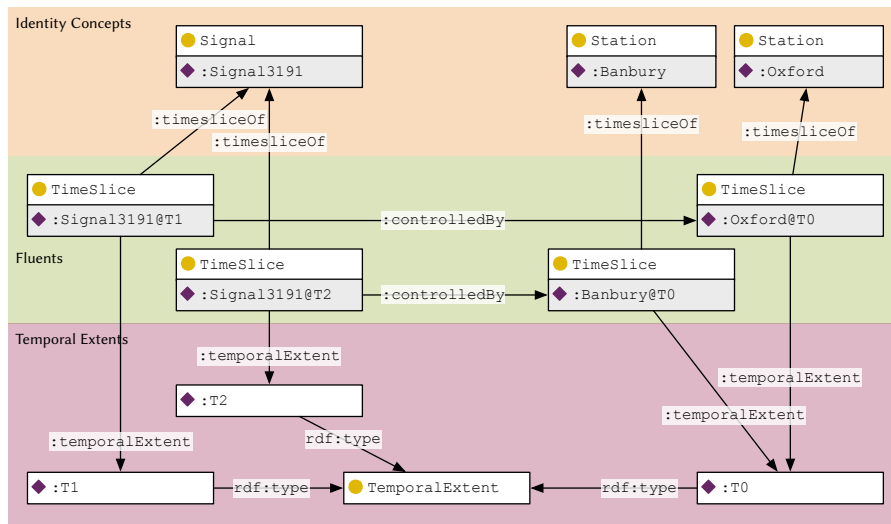


Figure 3.12: Example of 4D Fluents [225] Approach in OWL

## ADVANTAGES AND APPLICATIONS OF 4D ONTOLOGY

The main advantage in using a four-dimensional representation of fluents is its conceptual consistency when dealing with time-varying data. All characteristics are represented as spacio-temporal parts of some identity, so no change in modelling technique or semantics is necessary to begin asserting some new characteristics that vary with time. Implementation in OWL, although verbose, also has some advantages. Relationships between entities can be asserted using binary properties, so the semantics of these properties (such as transitivity or symmetry) are maintained. Domain and range assertions are also possible, by subclassing the ‘Timeslice’ class.

## BARRIERS TO IMPLEMENTATION

The wide-scale uptake of a four-dimensional approach is prevented by several significant disadvantages to representation in OWL:

- Proliferation of triples. The translation of temporal characteristics into binary predicates leads to a high number of triples asserted over each concept. This is not necessarily a problem, but results in readability and slow reasoning in some scenarios.
- Complete loss of semantics for reasoning over entity types. By containing the characteristics of every individual in one or more fluents, OWL inference based on entity properties is impossible. In contrast, a 3D approach allows this in cases other than where temporal assertions are required.
- Unnecessary complexity for non-time-variant data. Many models do not need to consider time in detail, or can sufficiently be described by a snapshot in time. 4D ontology requires that even these entities must be represented using time slices, significantly adding modelling complexity.

#### OTHER APPROACHES TO TIME REPRESENTATION

Several other approaches to modelling time can also be taken, but are not discussed in detail here. These include:

- **Streaming RDF.** Another approach to representing time in dynamic systems is simply to ensure that the information system used continuously reflects the state of information in the real world. Such *real time* systems do not require explicit representation of temporal variance, because a system state when read is taken to reflect the state of the world at that time. Although ontologies have traditionally been viewed as static documents, recent research has shown the possibility for *streaming RDF*, which could facilitate real time semantic knowledge representation. Streaming RDF raises complications in some areas (particularly reasoning); a recent survey of use cases and implementations is undertaken by Margara et al. [133], and a W3C working group on the subject was formed in August 2013<sup>10</sup>.
- **Graph-based Encoding of Time:** It is also possible to encode temporal information using RDF named graphs. Named graphs append RDF triples with a fourth attribute, which can be used to track the context of triples within a knowledge base. Extensions to SPARQL [10, 91] then allow the filtering of query results based on time, providing a way of accessing time-variant information.

<sup>10</sup> <https://www.w3.org/community/rsp/>

### 3.6.3 *Approaches to Representing Quantities, Units, and Dimensions*

Representing quantities, units, and dimensions correctly is important when designing technical ontologies. Taking an approach that is too simplistic impedes the ability of the model to represent knowledge correctly; modelling them in too much detail can affect the performance and usability of the model through the additional assertions required. Three approaches are presented here, and implementation of these approaches is considered further when describing design of the RaCoOn ontologies in [Section 5.3.2](#).

#### DIRECT MODELLING USING DATATYPE PROPERTIES

In the first instance, `owl:DatatypeProperty`s or RDF properties and literals can be used to encode quantities and rely on XSD datatypes to provide restrictions and semantics. Properties can be defined for each different attribute to be modelled, and documented accordingly. Examples include `foaf:age`, which is defined as ‘age in years’ of an agent, `geo:lat`, the latitude in degrees of a point, and `po:duration`, the duration in seconds of a media program. [Listing 3.1](#) shows an example of this approach.

```
ex:Jon ex:ageInYears "29"^^xsd:Double;  
ex:weightInKG "65"^^xsd:Double ;  
ex:weightInPounds "143.3"^^xsd:Double .
```

Listing 3.1: Using Datatype Properties to Represent Attributes

#### THE QUDT ONTOLOGY

Where more semantic certainty is required, the National Aeronautics and Space Administration (NASA) Quantities, Units, Dimensions, and Types (QUDT) ontology provides a rigorous ontology for describing measurements using more descriptive means, demonstrated in [Listing 3.2](#). In this case, the semantics of the units and values themselves are defined formally, allowing for machine-interpretability.

```

ex:Class167Train ex:length :_SomeLength .
:_SomeLength a qudt:Quantity ;
  qudt:quantityKind qudt:Length ;
  qudt:quantityValue :_SomeValue .
:_SomeValue a qudt:QuantityValue ;
  qudt:numericValue "40"^^xsd:double ;
  qudt:unit qudt:Metres

# supporting assertions in the QUDT ontology:
qudt:Metres a qudt:Unit ;
  qudt:quantityKind qudt:Length .
qudt:Length a qudt:BaseUnit

```

Listing 3.2: Turtle Listing Showing Time Represented using the QUDT ontology

QUDT provides a comprehensive vocabulary of *units* such as Metres, Volts, and Radians, which are each linked to a particular *quantity kind*, such as Length, Potential, and Angle. Uncertainty of measurements is also supported through the `qudt:Quantity` class.

#### THE ISO 15926 APPROACH

The ISO 15926:2 ontology takes advantage of the additional expressivity afforded by the EXPRESS-G notation format, and defines dimensional quantities by using ‘classes of properties’ that allow the approximation ternary relationships. In addition to providing `property_quantification` and `scale` attributes, the model also facilitates representation of multi-dimensional properties, indirect properties such as ‘temperature drop’ and property ranges using similar patterns. This approach is semantically very rich, but requires a higher number of assertions when translated into OWL than the other two solutions.

### 3.7 SUMMARY

Standardisation of data across the railway domain is a relatively new pursuit, with the majority of efforts in cross-stakeholder data modelling being undertaken in the last ten years. The current state-of-the-art shows a number of candidate data models in the asset information domain, many of which duplicate the same knowledge in efforts to create systems for diverse and changing stakeholders. Uptake of these models has been slow except in the case of national or international mandates for their uptake, as in the case of Register of Infrastructure

(RINF) and TAF/TAP TSI.

As discussed in [Chapter 2](#), semantic models provide flexibility in creating data exchange standards, such that models can be built upon and extended rather than replaced. This may prevent the proliferation of conflicting data standards seen so far across Europe, and instead provide a single, evolving data framework for development and use by all. For this reason, their development is explicitly cited as a key component of railway information system strategy in several recent industry reports [[146](#), [175](#), [184](#)].

Work in [Chapter 5](#) draws upon elements from several of the models reviewed in this chapter: RailML is drawn upon to provide a data source for railway vocabulary and concepts, and RailTopoModel used as a basis for the way in which geographic and topological railway network concepts are mapped in the novel data models described. The demonstration projects described later in [Chapter 6](#) share common requirements with the condition monitoring elements of the InteGRail project described, although the novel work in this thesis demonstrates reasoning across infrastructure elements rather than on-board train vehicle systems.

Following the overview of state-of-the-art research and techniques in ontology engineering given in [Chapter 2](#) and the summary of existing railway data models and approaches given here, the chapters that follow contribute novel methods and ontologies for facilitating data exchange using semantic models in the railway and other industries. The next chapter, [Chapter 4](#), describes a new ontology engineering methodology for creating industrial semantic data models.





## DESIGNING EXTENSIBLE MODELS FOR LARGE COMPLEX SYSTEMS

---

### 4.1 INTRODUCTION

In [Chapter 1](#) and [Chapter 3](#), the current limitations of knowledge management techniques within the railway industry were set out, as well as the challenges and problems currently faced in maintaining and implementing new systems. In [Chapter 5](#), a set of novel ontologies known as Rail Core Ontologies (RaCoOn) are presented that aim to address some of these limitations, and in particular to provide easier ways of sharing data between heterogeneous data sources and to encourage doing so.

This chapter presents the methodology used in designing and implementing these ontologies. Existing ontology engineering techniques and workflows are drawn upon, and a new set of extensions and adjustments that better facilitate the needs of domain models for data integration in large complex systems are introduced. A set of best practice design patterns for implementation of such ontologies is also presented, drawing upon stated requirements and existing literature. The content of the RaCoOn ontologies is discussed in the next chapter, and [Chapter 6](#) presents two novel applications based on extensions to the new ontologies.

#### 4.1.1 *Introduction to the RaCoOn Ontologies*

Rail Core Ontologies (RaCoOn) are a set of domain ontologies that model areas of the rail domain commonly used in railway data exchange, with the aim of allowing the uptake of data exchange based on semantic data models in a number of existing railway use cases and in future applications. They are implemented in OWL, and are designed to be extended by specific applications when needed. This has two immediate consequences for the design and methodology adopted in creating the ontologies:

- An initial set of models need not consider all possible use cases, but rather should focus on objective representation of a common framework for high level concepts. By providing this frame-

work, information from applications that extend it are more easily reusable.

- The models can be resilient to changing needs and circumstances. Whereas XML implementations of models often require changes that render previous versions incompatible in order to accommodate new requirements, OWL models can be extended based on existing semantics without the need for breaking changes.

As such, the RaCoOn ontologies do not attempt to provide a conceptualisation of the entire railway, but provide terms and patterns for representing data at a high level, along with a set of key railway terminology. Its two principle components are a cross-domain ‘upper’ ontology of key high level concepts such as space and time, and a set of domain and subdomain ontologies which express knowledge and relationships about railway-specific concepts, centred around infrastructure.

#### 4.1.2 *Methodological Requirements*

The approach taken in design of the RaCoOn methodology draws upon techniques in the existing ontology engineering methodologies described in [Chapter 2](#), but provides specific approaches in order to fit with a set of defined requirements for industrial domain models. The methodology addresses the following set of non-functional requirements, which are suggested to allow the uptake of such models across multiple existing and future applications:

1. **Provide ways of semantically integrating data from many heterogeneous sources.** The key requirement of the domain model is an ability to allow data currently trapped in multiple information silos and restrictive document structures to be easily mapped and integrated to an extension of the domain ontologies described here.
2. **Support a multi-stakeholder workflow.** Given the current organisational state of the railways in the UK and abroad, it is unrealistic to expect a central authority to take ownership of railway data. Thus, the domain model should be highly extensible, provide its own incentives for use, and should allow stakeholders to make as much use of extended data models as possible.
3. **Provide flexibility as information system requirements change over time.** Semantic data models inherently provide this flexibility, and domain models should be designed to allow further

modifications over time by ensuring they do not provide overly restrictive views on the domain

4. **Present itself as intuitive and understandable as much as possible.** One barrier to entry of similar systems such as ISO 15926:2 has been the extremely steep learning curve necessary for adoption. Any domain model created should be as intuitive as possible, to lower the barrier to entry for new users.
5. **Provide scalability.** The use of an ontology-based system in production is likely to encompass far more data than an academic prototype. Commercial RDF data stores can cluster and handle vast data volumes, but OWL reasoning is notoriously tricky to scale. Therefore, an ontology should consider the price and practicality of using DL axioms where necessary.

#### 4.1.3 *Proposed Approach*

The design approach adopted in the RaCoOn methodology draws heavily upon some of the modular ontology engineering tasks described in the NeON methodology [203], many of which consolidate best practice from the ontology engineering literature, and most of which are already tailored towards the creation of ontology ‘networks’, a key design feature in our methodology.

Based on the high level requirements given above, a multi-stage approach tailored to creating pragmatic OWL DL models is taken, and a set of workflows and patterns that are best suited to these requirements. These stages are presented here in five sections: specification and scoping(Section 4.2), architecture definition and modularisation(Section 4.3), knowledge acquisition and conceptualisation(Section 4.4), implementation and formalisation(Section 4.5), and evaluation(Section 4.6). The methodology recommends an iterative development process, as illustrated by Figure 4.1.

This iterative method provides a way of validating and reinforcing the model throughout, as demonstrated by Presutti et al. [170]. Knowledge which is modelled through the ‘top down’ approach is reinforced by terms that appear in the ‘re-engineering’ stage, and terms created during the re-engineering stage are validated and categorised by the top down approach. The stages in this process are discussed in the following sections.

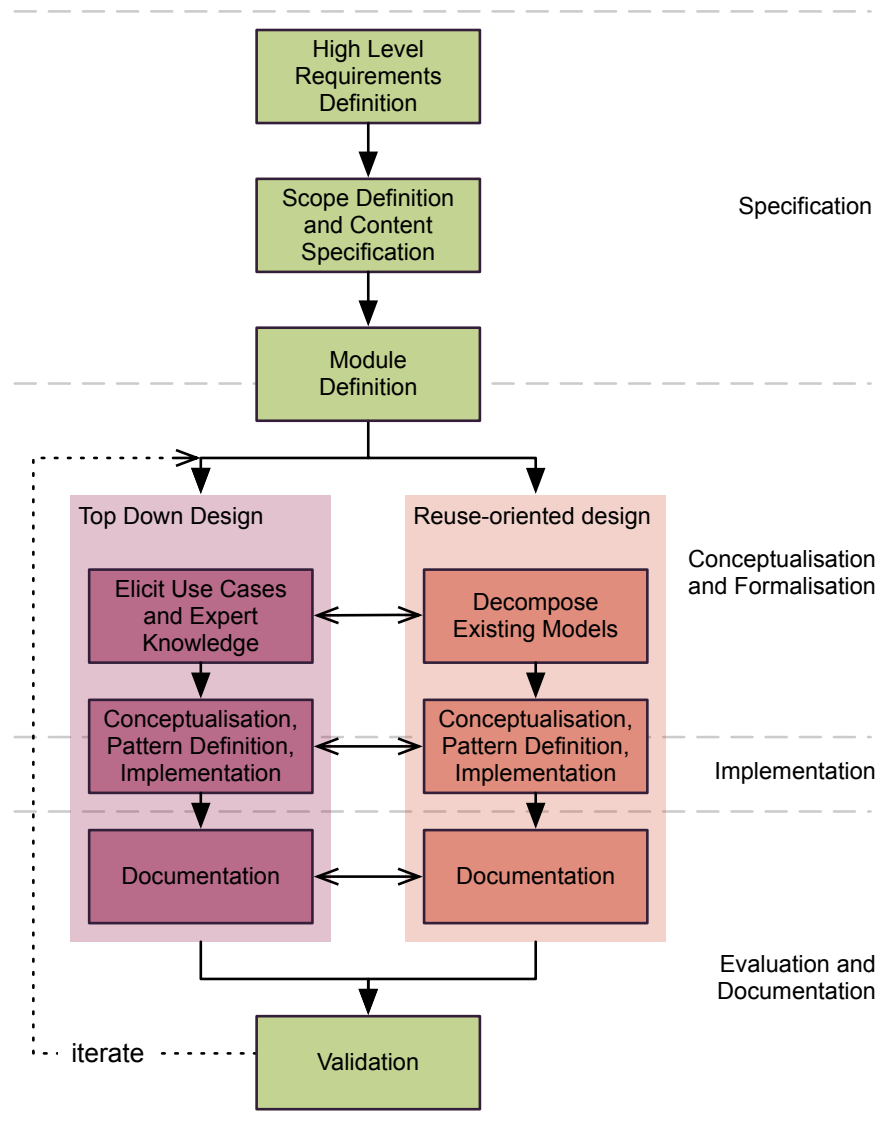


Figure 4.1: RaCoOn Ontology Engineering Methodology

#### 4.1.3.1 *Observations on Ontology Verbosity*

Domain modelling in OWL necessarily involves some trade-off between accuracy and semantic truthfulness. Gruber [84] stated five design criteria that are still widely adhered to in the field of ontology engineering. All five are listed here, with points four and five being of particular relevance:

1. **Clarity:** An ontology should effectively communicate the intended meaning of defined terms.
2. **Coherence:** Inferences made across an ontology should be consistent with their definitions.
3. **Extendibility:** An ontology should be designed to anticipate the uses of the shared vocabulary.
4. **Minimal Ontological Commitment:** ontologies should assert as little as possible in order to support knowledge sharing activities
5. **Minimal Encoding Bias:** models should minimise assumptions made by implementation-specific issues.

When encoding domain models, the limits of ontological commitment are not necessarily defined. Thus, a formal definition of scope must be created, and the ontology engineer must define the model to a reasonable level of detail across this scope. In OWL, this also influences encoding bias: less ontological commitment can lead to more semantic assumptions, and these can potentially restrict extensibility. A good example of this is in representing measurements (discussed in [Section 5.3.2.2](#)). At first glance, it is intuitive to represent a UK train vehicle's maximum speed as shown in [Listing 4.1](#). Immediately, however, this is restrictive. Even if we annotate the `ex:maxSpeed` property with the knowledge that speeds are represented in miles per hour, there is no semantic annotation of this property, and crucially speeds in other representations cannot be made. Instead, the approach taken in [Listing 4.2](#) can be taken.

```
ex:Train ex:maxSpeed "125.0"^^xsd:float''
```

Listing 4.1: Train Speed Represented in RDF Using Direct Data Property Approach

```
ex:Train ex:maxSpeed [ex:unit ex:MilesPerHour, ex:value
↪ "125"^^xsd:float]
```

Listing 4.2: Train Speed Represented in RDF Using Ternary Relations Design Pattern

This now provides more expressiveness, but with a significant overhead—it now takes three triples to assert the information rather than one. However, it still makes an assumption on the meaning of `ex:maxSpeed`. If it is defined to mean ‘maximum speed on a flat gradient’, then how are speed profiles represented? How about different maximum speeds depending on the traction package? Asserting these in the same way could lead to a proliferation of `ex:xxxSpeed` properties and make the ontology unreadable.

The methodology described here is aimed at the definition of ontologies that are pragmatic enough for their intended uses, but flexible enough to be extended by applications in these scenarios. As such, emphasis is placed on usability and brevity rather than on ontological completeness. This attitude mirrors that taken when designing ‘linked data’ application ontologies, but applies it to the creation of domain ontologies instead.

#### 4.2 STAGE 1: SPECIFICATION AND SCOPE DEFINITION

To create a domain model, some idea of its scope must initially be known. Whilst an all-encompassing model of every interaction across the industry is very desirable, such models are expensive to implement and may have little utility in the real world.

In this methodology, we consider that one or more *sections* of a wider domain should be modelled. This approach leads us to create a set of models that have characteristics somewhere between the philosophically sound ‘formal’ upper ontologies that attempt to completely model a domain to a high degree of abstraction, and semantic web ‘application’ ontologies that are designed to represent knowledge for one application and give little regard to wider domain context.

For example, considering simply ‘the railway industry’ may encompass a number of different disciplines and worldviews, and in order to create a pragmatic model, only a section of this should be modelled at the concept level. [Figure 4.2](#) shows an illustration of this. The rail domain is shown as a pyramid, with one high level concept encompassing a number of progressively more specific subdomains, and with

applications at the bottom. The ontology we aim to build here is an intentional subset of this diagram, highlighted in red.

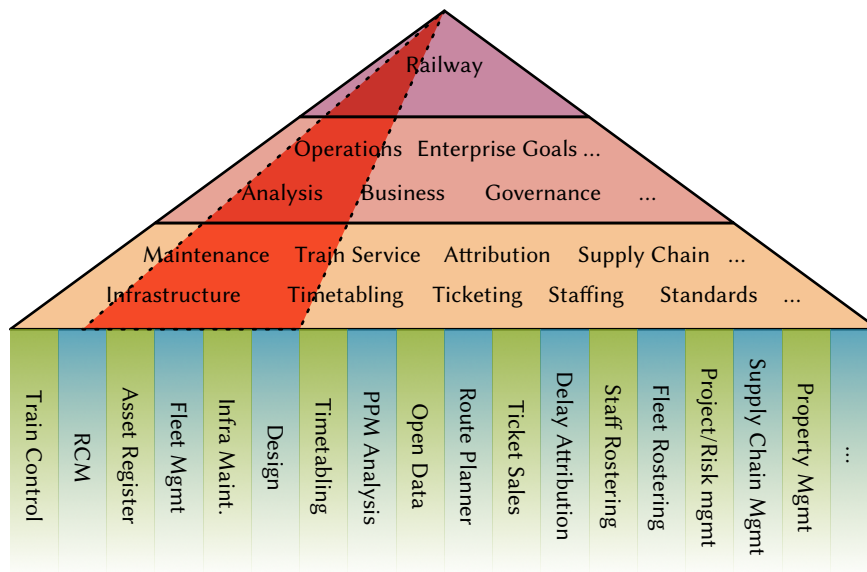


Figure 4.2: “Torchlight” Diagram Showing an Overview of the Rail Domain, and Proposed Scope of Domain Ontology

The diagram illustrates two dimensions in which the scope and requirements of a model must be defined:

- The **breadth** of the domain (left-right) is the decision of which and how many high level concepts should be modelled in a domain model to provide the most use.
- The **depth** of the model is the level of detail in which it is modelled to. In our methodology, we consider domain models that attempt to model concepts that will be widely reused, but do not consider application or subdomain-specific details.

#### 4.2.1 *Scope Definition Methodology*

In software engineering, the requirements that a system or data model must fulfil can be elicited using one of a known set of requirements analysis techniques based on the system’s objectives. These requirements drive development and validation of the system. The process is demonstrated by the cross-discipline “Vee” model shown in [Figure 4.3](#), which illustrates a generic system development lifecycle, and forms part of several systems, software, and project management methodologies [49, p. 31].



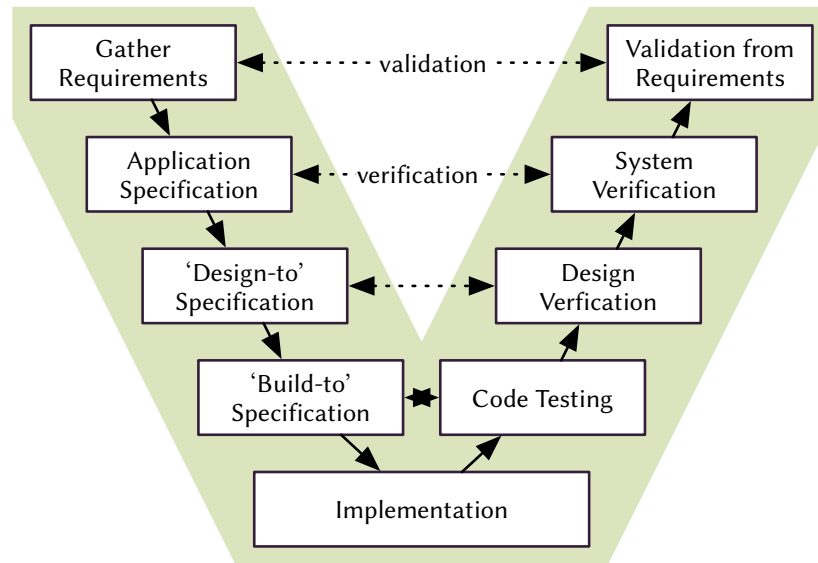


Figure 4.3: Simplified Diagram of ‘Vee’ Systems Engineering Methodology (adapted from Forsberg and Mooz [67])

Existing ontology engineering methodologies also take this approach: they elicit requirements from known use cases, and use these to construct a framework for design and validation of a model. In METHONTOLOGY and the NeON methodology, these are expressed as a set of Competency Questions (CQs) that determine exactly what a model should be able to represent, and against which a resulting ontology can be validated.

For the scenario proposed here, this approach is not as effective. We aim to build a high level domain model which should be useful to a number of unknown future applications, rather than to a set of current, specific applications. Additionally, there is no need for us to model the domain in detail, as subdomain-specific ontology extensions should provide this as demand for them increases. Thus, our ontology is limited to answering Competency Questions (CQs) that can be posed against domain-level concepts, and validation against these does not guarantee that the model’s coverage matches that expected by a suite of applications.

Instead, we base the scope of our methodology on an initial set of objectives elicited by project stakeholders, and on an iterative design process in which appropriate modelling depth is decided based on the perceived usefulness of concepts and definitions:

1. High level objectives and requirements are elicited from principal project stakeholders. This provides the primary motivation for building the model, and gives an indication of the areas to be modelled.
2. Perceived initial scope is evaluated against existing and potential *use cases* and the nature of the data they may use, to establish the benefit of a proposed model with the suggested domain.
3. Ontology *modules* are designated and created, to categorise model theme and scope into different sub-ontologies
4. Refinement and development of scope is undertaken, and model scope determined based on iterative ontology development. A set of Competency Questions (CQs) for model scope are created based on a set of design questions posed as the model is further developed.

This approach is demonstrated in the design of the RaCoOn ontologies.

#### 4.2.2 *RaCoOn Stakeholder Requirements and Applications*

At the start of the ontology design process, several meetings with industrial representatives at the CASE sponsor company were undertaken to establish the high level aims and likely use cases for the core domain model and associated application scenarios. These aims contextualised the requirements gathering and scoping of the rest of the project, and were as follows:

- Development of an industry-wide semantic data model to aid interoperability between stakeholders.
- Emphasis on ‘Infrastructure-centric’ world view, to match the current organisational and inter-organisation conceptualisation within Invensys Rail Group
- Demonstration of extensibility to widely-used IRG and Network Rail data models, such as Layout Description Language (LDL) and SDEF

These original meetings also suggested three demonstration systems to be developed using the developed ontology and custom extensions, but organisational changes instead provided the opportunity for a different set of applications to be built as part of the FuTRO project detailed in [Chapter 6](#). Nevertheless, these original potential use cases helped to influence initial scope for the ontologies:

1. Infrastructure visualisation tool, based on combined infrastructure data from Network Rail and IRG.
2. Remote condition monitoring application, to better integrate IRG monitoring hardware with infrastructure data and systems
3. Signalling scheme design process tool, demonstrating levels of detail by discipline

A broad conceptualisation of railway domain subject areas was created and agreed upon by IRG, as shown in [Figure 4.4](#). The expected subdomains to be conceptualised in detail by the rail domain ontology are shown in **bold**.

#### 4.2.2.1 *Context of RaCoOn Scope Within Wider Domain*

To further understand how the model scope proposed by IRG fits with demand, we draw upon data from two independently organised workshops, both of which consider use cases for railway data integration. The first, carried out by a team from the University of Birmingham and the University of Nottingham on behalf of Rail Research UK Association (RRUKA), was part of a study into creating a modal shift towards rail transport, and focussed on the specification and benefits of a ‘System-wide Data Framework for the Railway Industry’ [184]. It presents results from a set of cross-industry stakeholder workshops, including a large set of potential use cases for data sharing, as well as a set of requirements necessary to realise the use cases that were discussed.

The second report [205] was published during the development of the core ontology, and was the result of another cross-industry workshop organised at the launch of the UK Transport Systems Catapult, a government organisation set up to encourage innovation and collaboration across the transport industry. The Transport Systems Catapult (TSC) session was an informal one day long event attended by 62 representatives from multiple organisations (including the author), and discussed data integration and challenges in the wider transport sector rather than just rail. Its outputs captured cross-mode use cases, current challenges and opportunities for data integration, and business incentives and restrictions around the sharing of information.

#### 4.2.2.2 *Overview of RRUKA Workshop Use Cases*

The RRUKA workshop report initially reported a total of 153 use cases for data exchange across the rail industry, categorised by the area in which they were likely to provide the most impact. From these

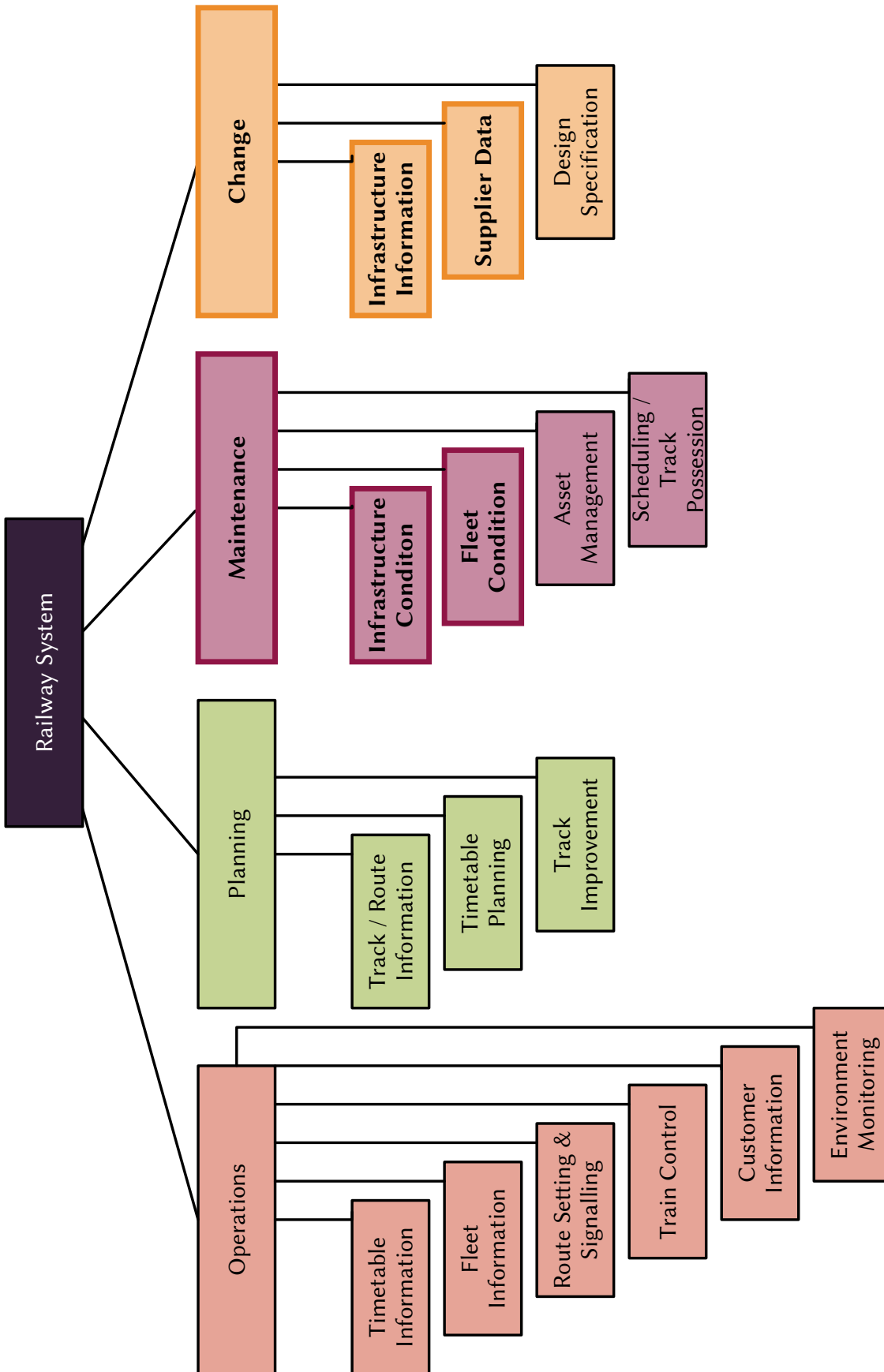


Figure 4.4: Initial Rail Domain Functional Conceptualisation with Invensys Rail Group Stakeholders (2011)

use cases, commonalities were identified to produce a list of nine key application areas, which were studied by the workshop in more detail. These areas are reproduced in [Table 4.1](#).

Table 4.1: Use Cases for a Standard Rail Data Model, Categorised by Area

Categorisation	Numbers
Asset management	40
Operations	30
Business and strategy	26
Interoperability	15
Customer	12
Data	12
Vehicle monitoring and management	8
Standardisation	4
Testing /modelling	4

The workshop also created a graph of interconnected application areas, reproduced in [Figure 4.5](#). Application areas given in this graph were summarised during the workshop from given use cases, and links between them provided by a consensus amongst workshop participants. Arrows in the diagram depict an identified link from the source idea to the destination idea, and does not necessarily imply transitivity between the ideas.

This graph shows a representative view of possible data integration applications across the railway, and provides a way of contextualising the utility of the proposed RaCoOn model scope.

#### 4.2.2.3 Overview of TSC Workshop Use Cases

The aims of the day-long TSC workshop were much broader than simply examining data sources and use cases for integration. The morning ‘Use of Data’ session divided attendees into several teams, who were then asked to complete several tasks concerning their views on data use and availability. The aggregated answers from two of these questions are presented here:

1. **List examples of the types of data you currently use (or would like to use in the future).** The results of this question are visualised in [Figure 4.6](#).

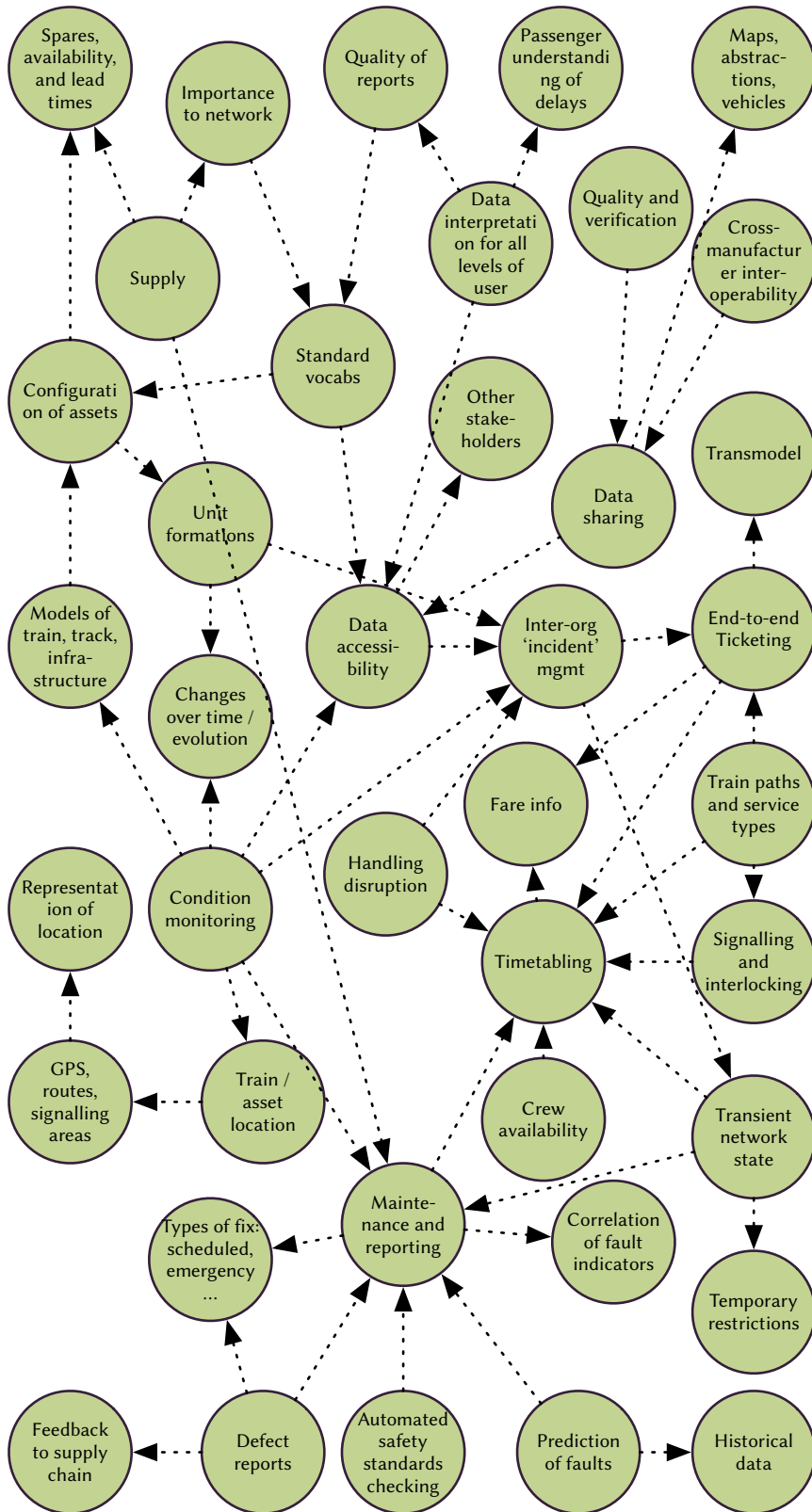


Figure 4.5: Interconnected Railway Data Management Application Areas (re-drawn from [184, p. 30])

2. **How are you currently using or hoping to use this data?.** Workshop results here were categorised by attendees into six main areas, which are shown in [Figure 4.7](#).

The results of the TSC session show a bias towards multi-modal, and public data, which reflects the nature of the participants at the session. Whilst some of the data uses exhibited here were not directly relevant to the railway, many of the cross-mode use cases could benefit directly from the use of a railway-wide domain model.

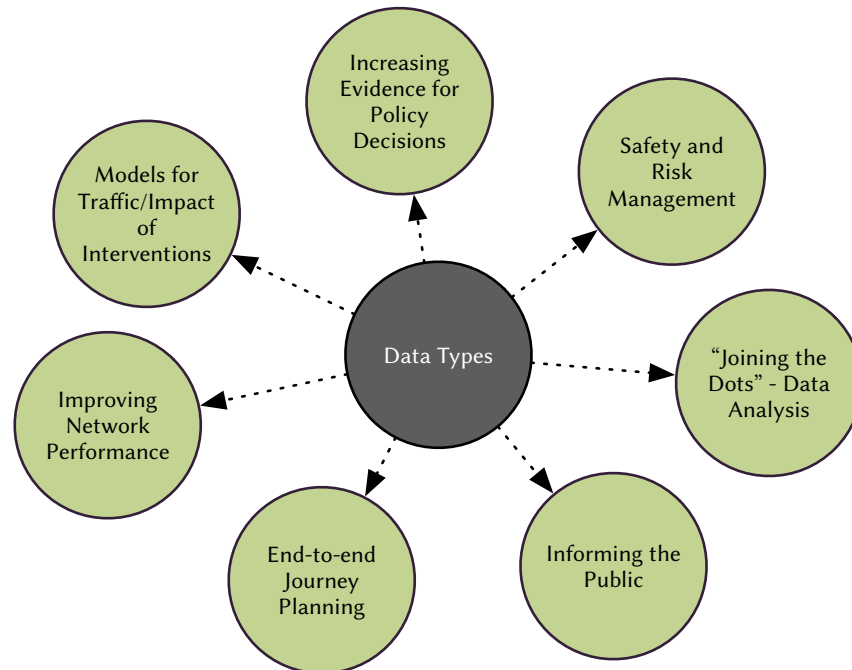


Figure 4.6: Transport Data Applications Mentioned by Participants at TSC ‘Data Challenge’ Workshop (data from Szluinska et al. [205])

#### 4.2.2.4 *The Case for an Infrastructure-centric Model*

The scope of RaCoOn was decided based on proposed implementation areas, perceived cross-application usefulness, and implicit domain knowledge. For this reason, the domain models presented here were authored around railway infrastructure and asset concepts, and incorporate high level concepts for rolling stock, timetabling, and asset management to act as a base ‘domain’ model.

Considering again [Figure 4.5](#), 23 out of 40 scenarios can be intuitively seen to have a strong link to infrastructure information, and would benefit directly from a domain model representing these concepts, as shown in [Figure 4.8](#). 70 out of 153 individual use cases were

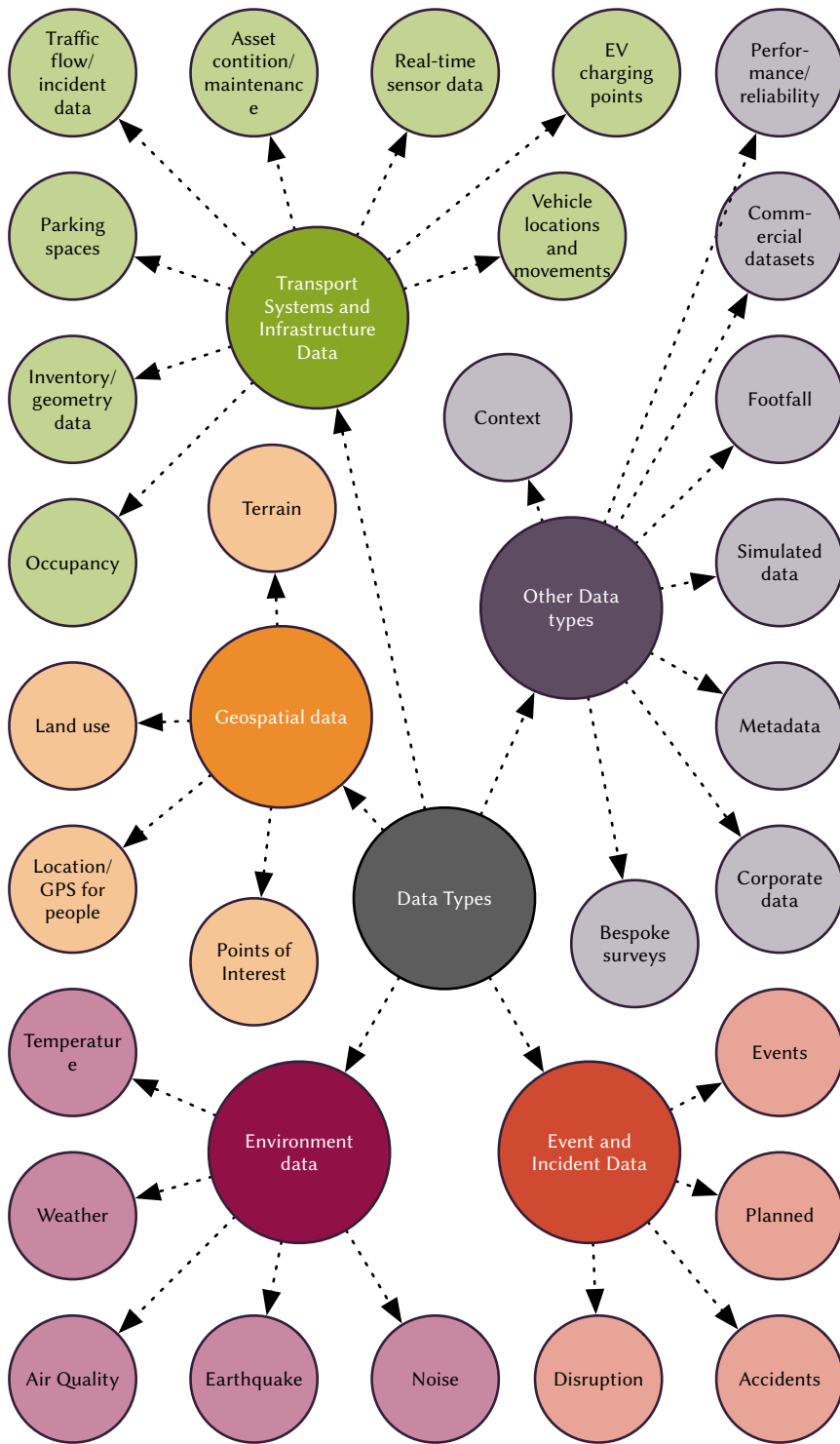


Figure 4.7: Data Types Used by Participants at TSC ‘Data Challenge’ Workshop (data from Szluinska et al. [205])



categorised as either in the realm of asset or operations data, both of which are areas in which infrastructure data is typically used heavily. The TSC workshops, although oriented towards multi-modal data, also show a number of use cases that could benefit from a model of this scope. Five out of seven of the stated use cases could be directly assisted by the definition of an infrastructure-centric model, and two out of five of the themes identified for existing datasets directly align with the themes mentioned (*transport systems and infrastructure data* and *Geospatial data* respectively).

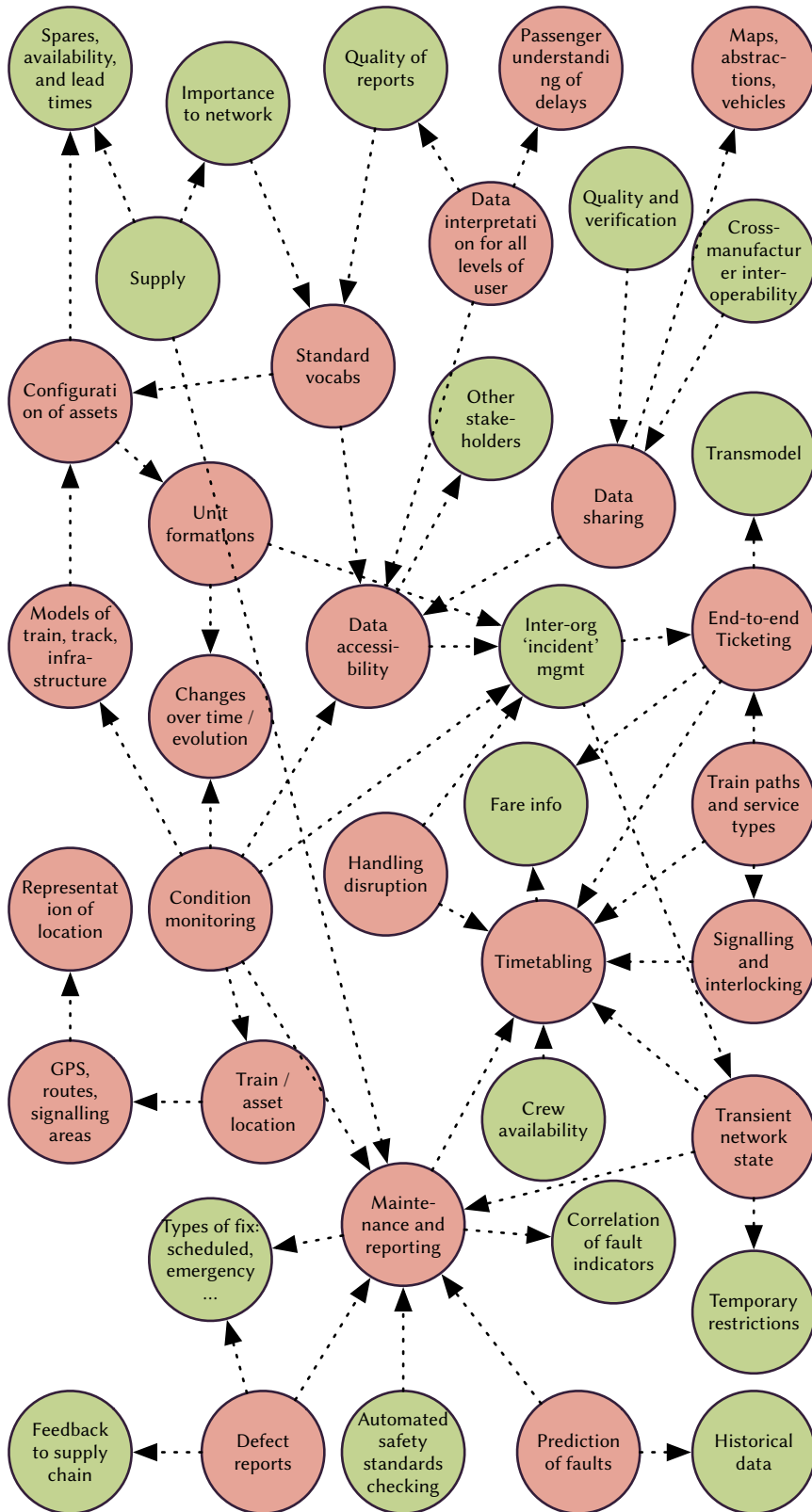


Figure 4.8: TSC Workshop Infrastructure-centric Data Integration Themes, Highlighted From Figure 4.5

### 4.3 STAGE 2: ARCHITECTURE AND ONTOLOGY MODULARITY

Modularity, as discussed in [Section 2.4.7](#), is employed in the RaCoOn methodology to encourage creation of a model made up of several small modules rather than a single, monolithic one. Modularity is encouraged here for the following reasons:

- **Re-use.** Creating a number of small, disjoint modules allows future users to include only the modules required for their application and aids performance and maintainability
- **Understandability.** Self-contained ontology modules with a defined scope are intuitive for non-experts who wish to explore concepts in a particular subdomain.
- **Extension and business case.** Modularising ontologies makes access control, versioning, and provenance easier across subsets of models. In a multi-stakeholder system, individual actors can be made responsible for individual models, and organisations wishing to build on an ontology and extend it for their own use case have a well-defined set of extension points from which to do so.

The approach taken in the RaCoOn methodology is to modularise ontology creation in two ways: firstly into a semantics-driven hierarchy of increasingly specific subdomain ontologies, and secondly by splitting models into modules of different expressivities. The combination of these two approaches results in a set of modules which can be combined and used to cover many different industrial use cases, and can be customised by domain coverage and expressiveness.

#### SEMANTIC MODULARISATION AND HIERARCHY

The creation of domain models is a difficult business decision to justify, as by their nature they provide very long term gains rather than a short term return on investment. Thus, modularisation within industrial ontologies plays another role; that of splitting the effort involved in creating domain models into several smaller parts, making a business proposition to develop each module less expensive or risky. Effort required to build initial parts of a core ontology to facilitate some level of interoperability is lessened, without commitment to building an expensive, comprehensive domain model being needed.

The RaCoOn methodology suggests a hierarchical framework in which to build ontology modules, with five distinct layers corresponding to different semantic levels of specification and abstraction, sim-

ilar to the architecture described by Guarino [86] and described in [Chapter 2](#). At each level, modules inherit and specialise concepts from the level above, and are segregated by subdomain to allow applications to pick and choose which parts of the model to utilise.

- **High level ontologies** describe a cross-domain conceptualisation of foundational concepts and relationships.
- **Domain ontologies** build on an upper ontology to provide vocabulary and interactions for common concepts across a particular domain
- **Subdomain ontologies** provide more detailed knowledge about specific domain areas, for use in a subset of domain use cases.
- **Application ontologies** adopt one or more domain and subdomain ontologies and build on them to represent application-specific knowledge.
- **Task-based ontologies** are independent modules which address a self-contained problem, and may be imported by other models.

The RaCoOn ontologies described in [Chapter 5](#) fully implement modules in the first two of these layers. [Chapter 6](#) implements two application ontologies based on known use cases, and utilises skeleton subdomain ontologies in the areas of rolling stock and timetabling. It is anticipated that further subdomain ontology development will be undertaken when the demand for it arises.

#### EXPRESSIVITY-DRIVEN MODULARITY

To produce a set of ontologies that are easily usable in industrial application, the RaCoOn methodology also suggests segregation of ontologies into different modules depending on expressivity and verbosity. This allows applications to choose what concepts to re-use based on a trade-off between semantic correctness and computational efficiency. In RaCoON, ontologies are split into ‘vocabulary’ and ‘constraints’ modules, allowing a distinction between applications using OWL RL concepts and those using fuller DL axioms. Further structure-driven modularisation is also used to allow identical concept vocabularies to be viewed from 3D and 4D perspectives, as detailed in [Chapter 5](#). [Figure 4.9](#) shows how a set of expressivity-driven modules fit within the semantics-driven modularity structure described above.

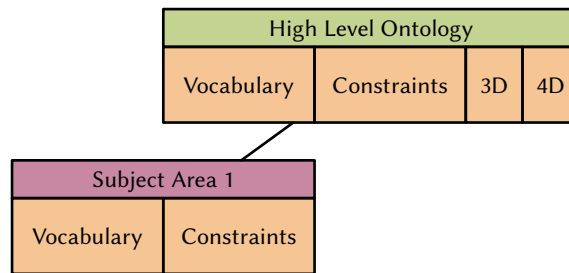


Figure 4.9: Modularity Based on Expressivity in a Set of Ontologies

#### 4.3.1 Module Interdependence

Considering ontology interdependence is essential to re-usability and testability of modules, as discussed by Parent and Spaccapietra [159]. Modules that depend on others are harder to test and re-use, as they in turn require that their dependent modules are loaded. In the semantic modularisation approach, interdependence should be such that:

- Low level modules depend on high level modules. Vocabulary modules may extend the structure of more general ontologies by providing ‘glue’ between newly defined concepts and higher level modules.
- Application modules should depend on several domain and sub-domain vocabularies
- Task-based ontologies should minimise their dependence on other modules, such that they can be re-used outside of the scope of just one domain ontology. Examples of this include the RaCoOn documentation ontology discussed in [Chapter 5](#).

In this way, high level ontologies can be tested and validated first, and then lower level ontologies validated knowing that upper level concepts they depend on are valid. This approach is outlined further on in [Section 4.6](#).

#### 4.4 STAGE 3: KNOWLEDGE ACQUISITION AND CONCEPTUALISATION

The creation of any conceptual model first requires that an accurate set of knowledge about the problem domain be obtained. In railway infrastructure and in similar systems, obtaining this information from subject experts can be difficult or costly, and no ‘gold standard’ conceptual domain model exists to transform. In contrast, the data models

of existing systems from across the industry provide clues to vocabulary terms and relationships, but often do not provide sufficient semantic clarity to produce an ontology from.

As such, we propose an iterative approach to knowledge elicitation and design comprising two parallel workflows. Firstly, key terminology is elicited using a ‘top down’ approach from domain experts and from domain literature, based on high level scope. A high level formalisation of entities within scope is defined, and a pragmatic set of high level concepts defined. Secondly, and in parallel, transformation of existing resources is undertaken, drawing upon and extending the high level concepts already defined by experts. The apparent semantics of these terms are taken into account, and are verified and expanded upon by domain experts to provide even coverage of a domain at the high level stage.

This process is repeated as more knowledge becomes available, until an ontology that fits the high level requirements is reached. It does not guarantee any level of detail or coverage, but provides a way of building an effective and extensible high level domain ontology.

#### 4.4.1 *Top-down Knowledge Acquisition*

The aim of a top-down approach to conceptualisation is to elicit knowledge and structure of a domain independently of any bias provided by existing models or resources. It should usually be undertaken in collaboration between domain experts and ontology engineers, and its aim is twofold:

- Initially, it should establish a skeleton upper level conceptualisation of a domain, to be used as a basis for further modelling and for vocabulary terms.
- Iteratively, it should provide a well-grounded conceptual structure for vocabulary terms and relationships. Terms that are elicited from existing vocabularies should prompt additional analysis, maintaining a consistent ontology.

In this process, the knowledge elicitation, conceptualisation, and formalisation stages can be combined. For creation of semi-formal models, software tools such as Topbraid Composer and Protégé now provide enough assistance and modelling cues that knowledge can be conceptualised by a knowledge engineer and directly constructed using these tools, rather than having to use separate specialised tools to

formalise and build the set of OWL axioms later. This simplifies the design process significantly.

#### 4.4.2 *Initial Conceptualisation and Iteration of RaCoOn ontologies*

In the RaCoOn ontologies, knowledge used in top down design was elicited by initial meetings between the ontology engineers, signalling experts at Invensys Rail, and other railway academics at the University of Birmingham. Following these workshops, the domain conceptualisation was extended by drawing on personal domain knowledge and specific resources, as follows:

- Visits to Invensys Rail Group in Chippenham and Birmingham, to gain familiarity with signalling, control, and design concepts within the organisation
- Attending Railway Systems Engineering and Integration MSc. modules at the University of Birmingham
- Discussions with visiting academics and engineers
- Manual knowledge extraction from the rail domain wiki discussed in Roberts et al. [184].

Starting with the scope and structure defined in [Section 4.2](#), the domain was decomposed into subdomains corresponding to the main areas of interest, providing a meta-model on which to base concepts. This meta-model was then filled out with an initial set of concepts, along with typical traits, that led to the definition of a set of design questions for modelling other domain and upper level concepts. For example:

- “How do we deal with the distinction between physical things and the roles that they provide?”
- “Trains have wheels. How do we model the whole-part relationship here?”
- “A track may be electrified, or may not be. It may have train protection, it may not. How do we represent track characteristics?”

By attempting to model the answers to these questions using the implementation process described in [Section 4.5](#), further questions are asked and further concepts discovered. As model design continues, its scope is refined further by domain experts, and reuse of existing data

models extract further vocabulary terms to be considered. Once a set of questions is specific enough to be formalised in OWL, a set of CQs are defined from expert knowledge and existing models to formalise the exact requirements of a part of the model, and modelling is undertaken based on these.

The repetition of this process allows a complete view of the domain to be created independently of specific application requirements, an example of which is shown in Figure 4.10.

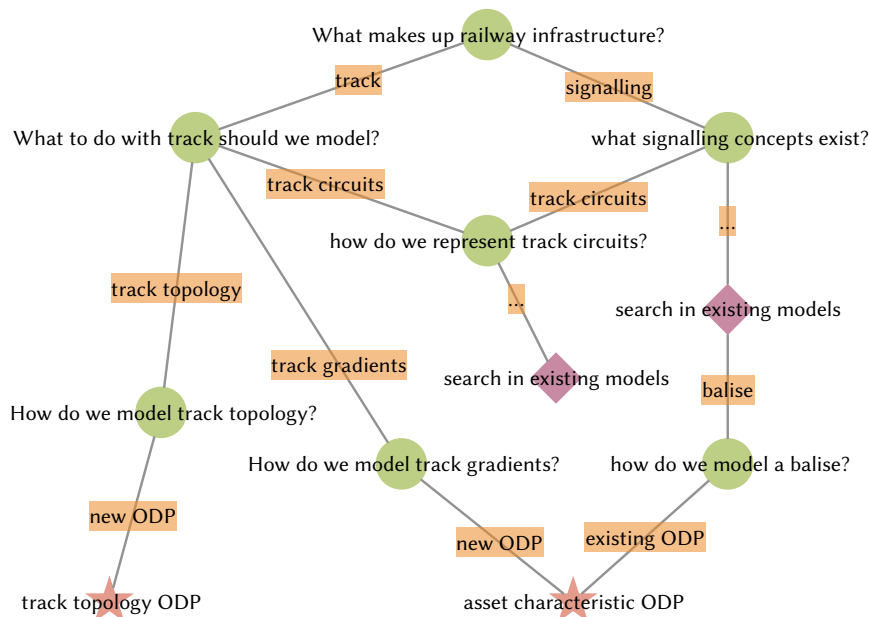


Figure 4.10: Diagram Showing Design Question Paths to Domain Model Construction

Once a specific design question is reached, CQs can be developed. In the case of ‘how can we model a balise’, these questions may be:

- “What type of balise is Balise X?”
- “What position along the track is Balise X located?”
- “What operational state is Balise X in?”
- “Is Balise X part of a group of balises?”

#### 4.4.3 Knowledge Extraction from Non-Ontological Resources

The second approach to knowledge elicitation involves extraction of terms from known non-ontological models. In this methodology, the



process of re-engineering non-ontological resources such as XML data models are focussed on, although the same manual approach can be applied using ontology-based resources too. Re-engineering existing resources in parallel with a top-down approach has two distinct advantages:

1. Gaining domain vocabulary and semantics ‘for free’. Whilst these resources are usually designed for specific applications and exchange situations, they also provide a mechanism for integrating concepts that may otherwise be missed.
2. Extracting vocabulary and semantics forces validation of top-down approaches. If an initial model makes assumptions that do not hold when re-engineering an existing resource, this may indicate a problem with the conceptualisation.

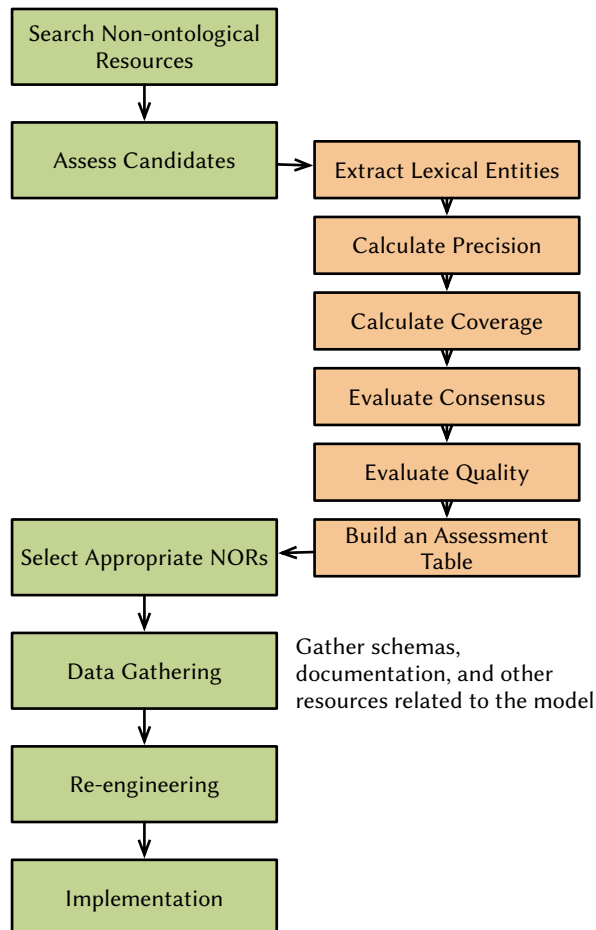


Figure 4.11: Stages in NeON Methodology Non-ontological Resource Reuse Process

The NeON methodology describes a comprehensive technique for evaluating resources for reuse and then re-engineering them using a six stage process, as shown in [Figure 4.11](#). Here, we additionally use NORs as a prompt for further ontology development: concepts found feed back into the iterative design process and prompt further manual knowledge elicitation.

#### 4.4.3.1 *Finding and Assessing Candidate Models for Reuse*

When designing the RaCoOn models, a number of factors were taken into account when considering models for reuse. These factors were as follows:

- **Perceived Scope Alignment:** the amount to which the model's stated aims and requirements align with those of the candidate ontology or required data to be modelled.
- **Authority and Provenance:** the perceived credibility of the author with regard to their data modelling capabilities.
- **Industrial support:** the level of uptake the model has across the industry, to be regarded as endorsement for the model's usefulness or quality
- **Semantic expressiveness:** the level to which the model describes its concepts.
- **Documentation and support:** the amount of human-readable additional documentation that accompanies the model. In in-expressive models, this documentation can be the only way of inferring the definite semantics of terms.
- **Perceived Modelling Quality:** from inspection, the quality and consistency of the model's schema design.

An evaluation of candidate models for the RaCoOn ontologies using these metrics (ranked between 0 and 5) is shown in [Table 4.2](#)

#### 4.4.3.2 *Data Gathering, Re-engineering and Implementation*

The NeON methodology provides a complete overview of methods for transforming non-ontological resources into OWL, and the exact techniques for doing so are likely to depend on the nature of the non-ontological resources to be transformed. In the RaCoOn ontologies, RailML was used extensively for knowledge extraction, and [Section 5.4.7](#) describes the motivations for and methods of manual transformation that was undertaken for this purpose.

Table 4.2: Table Showing Subjective Metrics for Non-ontological Resource Reuse in RaCoOn Ontologies

Model	Authority	Uptake	Scope	Expressivity	Quality	Documentation	Total
RailML	3	3	5	3 (XML)	3	4	21
SDEF	4	-	3	3	3	3	16
LDL	4	2	3	2	3	3	17
RFA	5	-	2	3	4	5	19
MIMOSA	4	4	1	3	5	5	22
ISO15926	3	3	1	5	5	4	22
RailTopoModel	2	4	4	4	3	3	20

#### 4.5 STAGE 4: IMPLEMENTATION AND ONTOLOGY REUSE

Implementation of ontologies in the RaCoOn methodology relies upon ontology engineers to formalise knowledge in OWL. In order to encourage reusability and adoption, emphasis the use of well-documented and designed Ontology Design Patterns (ODPs) [69], as well as adopting current semantic web best practice, which focusses on reusability and pragmatism.

##### 4.5.1 *Ontology Design and Implementation Best Practice*

To encourage re-use and extensibility, ontologies should follow best practice in the disciplines of computational ontology design as well as semantic web development. Fortunately, these two disciplines are becoming increasingly well-aligned and standardised, and a set of de facto standard design procedures is now apparent from the literature.

Firstly, the need for minimal ontological commitment has already been stated in [Section 4.1.3.1](#). Models should not include axioms and concepts that are not likely to be required by users for the sake of efficiency and manageability, and over-constraining models should also be avoided. Therefore, in domain modelling efforts should be taken to ensure that out-of-scope concepts are not included, and that axioms constraining relationships between concepts are only included

if they add value to the model. In the RaCoOn methodology, this is undertaken by reviewing concepts with domain experts prior to inclusion, and by considering likely use cases for each addition prior to including it in a model.

Another tenet of contemporary ontology design is to ‘re-use as much as possible’ [65], in order to ease the reuse of data that has been represented. When considering ontologies for circumstances where automated reasoning is to be used, this issue of reuse is particularly important: popular ontologies for representing some concepts are often too inexpressive or too expressive for an intended use case, and so compromises have to be made. In the RaCoOn methodology, 4.5.5 describes methods for reusing ontologies.

Finally, in order to encourage adoption and reuse of an ontology, good documentation is required. A number of strategies for documenting ontologies are discussed in Section 4.7.1, and the RaCoOn methodology includes several novel documentation methods, shown in Section 4.7.2.

The following sections further address how OWL is used in the RaCoOn methodology in order to create ontologies that are pragmatic and fit the aims described in Section 4.1.

#### 4.5.1.1 *Constraints in Domain Models*

A key principle of ontology modelling in general, and particularly domain modelling, is:

“Keep domain models and usage models separate” (from Uschold and Grüninger [213])

This alludes to the tendency by novice ontology engineers to treat domain models as they would models for syntax validation—that is, to over-commit the domain model such that errors in data entry could be detected as logical inconsistencies and rejected. Instead, it is suggested that instance data entered as RDF triples should be pre-validated by applications, such that reasoners assuming correct and valid knowledge can function efficiently. OWL axioms can still be exploited to provide search a service by interpreting them under the closed world assumption, as discussed in Section 2.6.4.2, and the RaCoOn models keep such axioms in separate modules to easily facilitate reasoning in this manner.

#### 4.5.2 *Use of Ontology Design Patterns To Encourage Re-use*

A significant element of the RaCoOn methodology is in the definition of useful ODPs to assist in creating ontologies. The definition and documentation of new patterns in the ontology design process itself is a key component in ensuring reusability and extension of models that are created, and the methodology itself suggests several patterns to aid creation of pragmatic industrial ontologies. Work in this thesis draws upon a number of existing, accepted design patterns [51, 70, 71], and proposes some novel patterns arising from the need to represent new concepts.

Ontology Design Patterns (ODPs) presented in this thesis are described in this chapter and in [Chapter 5](#). Those described here are generic patterns to aid in the construction and documentation of ontologies, whilst ontology-specific patterns used in the creation of the RaCoOn models are described in the next chapter. Additionally, patterns to encompass specific reasoning solutions are discussed in [Section 4.5.6](#).

#### 4.5.3 *Pattern Design vs. Reuse*

In order to satisfy the requirements for reusability and understandability shown in [Section 4.1](#), the RaCoOn methodology encourages the use and creation of ontology design patterns to describe reusable elements of new ontologies and to document ways of representing terminological concepts. Initially, the adoption of other commonly-used existing design patterns from other domains is encouraged in order to ease the use of knowledge from these domains in future. Should other patterns be absent or unsuitable for adoption, the creation and documentation of new patterns makes it far easier for new users to build upon concepts and themes present in constructed ontologies, in a similar manner to how design styles and patterns are documented in software engineering.

By encouraging the reuse of patterns and ontologies, de facto standards for the representation of certain terms appear, reducing the amount of effort required when mapping knowledge between ontologies in applications. Examples of extensive reuse can be found in many semantic web ontologies, an overview of which can be found in Allemang and Hendler [5].

A summary of the approach taken by the RaCoOn methodology for concept formalisation using design patterns is shown in Figure 4.12. Section 4.5.4 and Section 4.5.5 consider in detail the design and implementation of these patterns.

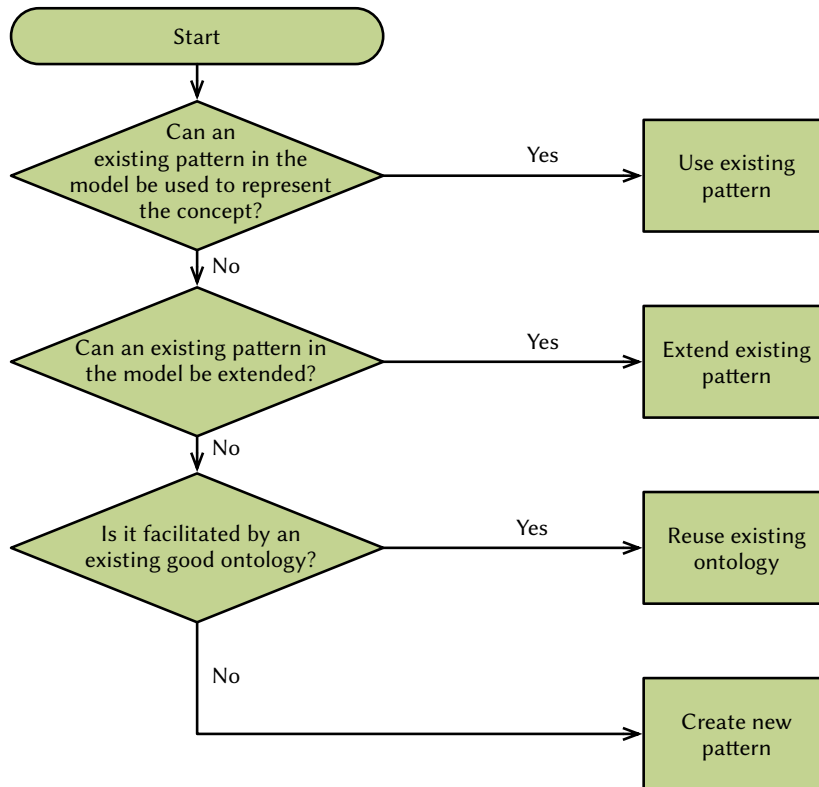


Figure 4.12: Steps Towards Concept Formalisation in RaCoOn Methodology

The first step in the process is to search out existing similar approaches to representing the knowledge to be modelled within the subject ontology itself. Some concepts share design patterns, and employing consistency in how concepts are modelled across a module or ontology may be worth more in understandability and reusability than the advantages gained by taking a new approach. Should this not be the case, engineers are asked to seek out patterns or mechanisms present in other third party ontologies that may be of use—such patterns may be found in ‘gold standard’ models from other domains, or from dedicated libraries of ontology design patterns such as [ontologydesignpatterns.org](http://ontologydesignpatterns.org) and Dodds and Davis [51].

Finally, ontology engineers are asked to engineer the new concept themselves, and to document it as a new design pattern if appropriate (i.e. if it is likely to be of any other use modelling other concepts). This approach ensures the development of ontologies that maximise

reuse of existing resources and are consistent with themselves, and is similar to existing methods [65]. It should be used both when designing domain ontologies and when extending them to model new knowledge.

Often, the decision between ontology reuse and original pattern creation is dictated by the quality of candidate ontologies rather than their existence. The subject of formalisation and reuse itself has been discussed at length by a number of authors and in the methodologies already examined (see Falbo et al. [60], Fernández-López, Suárez-Figueroa, and Gómez-Pérez [65], and Pinto and Martins [163] for examples).

#### 4.5.4 *Developing Ontology Design Patterns*

When an original contribution to the ontology is made that is likely to be reusable, it should be as part of an content pattern. A concept's association with an ODP provides traceability and an explanation for its existence, as well as a mechanism for future users who may wish to model similar concepts to do so.

The actual conceptualisation and implementation of a particular design pattern are specific to the modelling requirements, and are for the ontology engineer to create. They should, however, be documented in the following two ways to aid usability and extension in the future:

- By using the ODP documentation pattern described in [Section 4.7.2](#), which provides a mechanism for representing design patterns within the ontology itself.
- By filling out ontology design pattern 'filling cards', as described in Suárez-Figueroa, Gómez-Pérez, and Fernández-López [203, p. 32]. Filling cards provide a template for documenting a new ontology design pattern, encouraging ontology engineers to describe the definitions, goals, input, output, and other characteristics of the pattern they have created.

In development of the RaCoOn ontology, filing cards were kept for each ontology design pattern using Evernote<sup>1</sup> and are present in the OWL ontology files themselves. Further detail on RaCoOn design patterns as well as those used in [Chapter 6](#) are given in Tutchter, Easton, and Roberts [211].

---

<sup>1</sup> <http://evernote.com/>

## 4.5.5 Reusing Best Practice Ontologies and Patterns

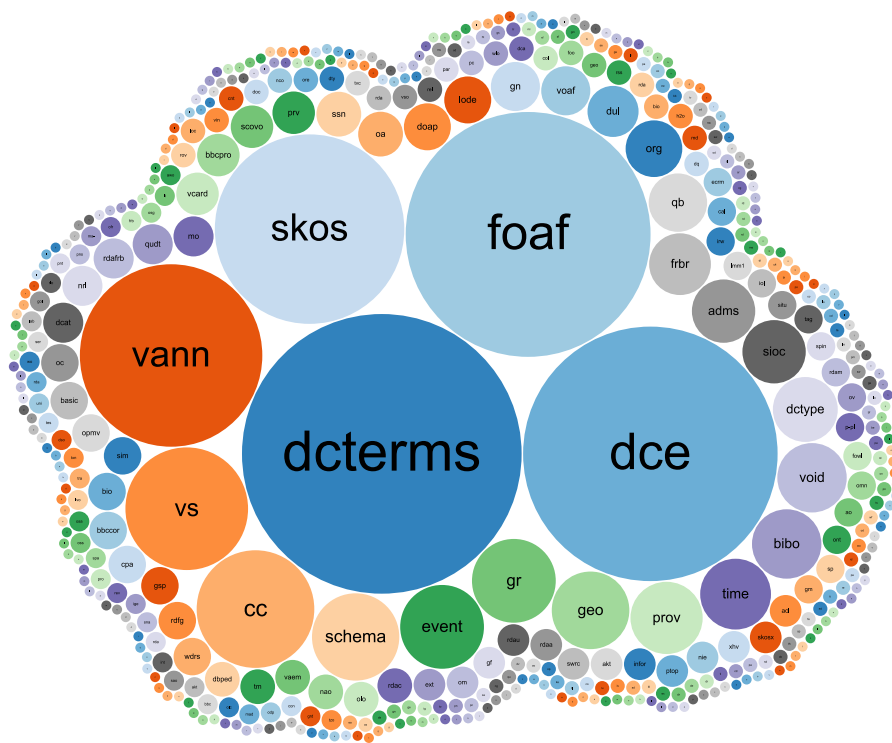


Figure 4.13: Visualisation of ‘Linked Open Vocabularies’ Datasets by Size, Taken from <http://lov.okfn.org/dataset/lov/>

A significant driver for the re-use of best practice vocabularies is the extensibility afforded by subscribing to a representation used by other parties, in order to maximise the potential for easy data integration. Recently, catalogues of such ontologies and design patterns (such as those shown in Figure 4.13) have become available online<sup>2345</sup>, and are easily reused in new ontologies. However, many linked data vocabularies cannot be directly reused:

- A large proportion of web datasets are intentionally written in pure RDF or using RDFS semantics, which aids usability in linked data applications but can lead to inconsistency in OWL DL.
- Some ontologies provide are conformant to OWL DL, but require richer or looser semantics within a domain model (for example the W<sub>3</sub>C Geo vocabulary [25])

<sup>2</sup> <http://lov.okfn.org/>

<sup>3</sup> <http://www.gong.manchester.ac.uk/odp/>

<sup>4</sup> <http://ontologydesignpatterns.org/>

<sup>5</sup> <http://www.linkedmodel.org/>



- Many web ontologies written in OWL are technically outside the scope of OWL DL, and fall into the OWL Full profile [223].

A method for reconciling such ontologies to the needs of the domain model under construction is required. This method should allow linked ontologies to conform, but also encourage re-use according to the original semantics of the vocabularies. In contrast to other ontology re-engineering approaches, we aim here to maintain the original structure and terminology of the original vocabularies, such that mapping between terms is trivial. Figure 4.14 shows the stages in re-using best practice ontologies, and further explanation of each stage is provided below.

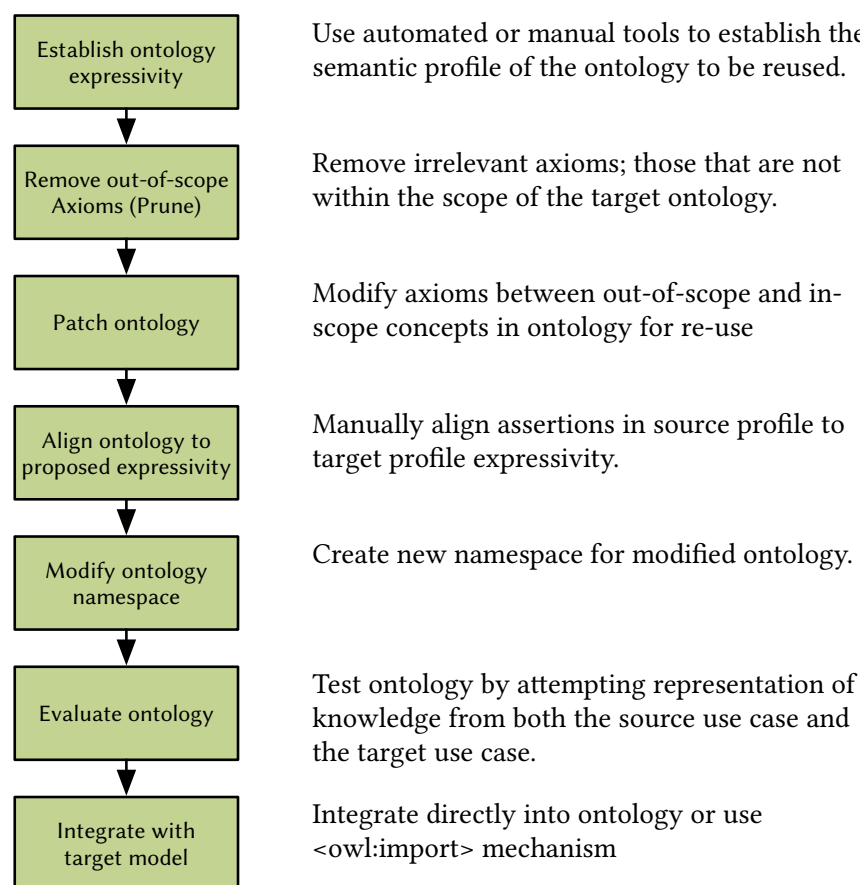


Figure 4.14: RaCoOn Methodology Ontology Integration Process

#### 4.5.5.1 *Establishing Ontology Expressivity*

The first step in re-using best practice ontologies is to identify the profile and axioms within the ontology. This is established either by inspection or using online validators [105, 171]. Ontologies that are

valid in the target profile require no modification to their semantics for re-use, significantly reducing the amount of reuse effort required.

#### 4.5.5.2 *Ontology Pruning*

Ontology pruning [44] is the act of removing axioms from a candidate ontology for reuse, and is often considered as part of an ontology modularisation and extraction process, in techniques such as those described by Courtot et al. [41]. Irrelevant axioms may have a negative effect on the target ontology by unnecessarily increasing the T-box size and therefore increasing reasoning complexity. In some cases, incorrect or incompatible axioms may also be removed. An example of this is given in Section 5.3.2.2 where the NASA QUDT ontology is partly re-used. A formal methodology for the pruning of OWL ontologies is given by Conesa and Olivé [39].

Conesa et al's technique prioritises ontology soundness; it does not allow the removal of irrelevant concepts that are depended on by concepts in the desired part of the candidate ontology, and it directly removes all constraints which connect relevant with irrelevant concepts. Here, we propose a more pragmatic manual approach, and 'patch' ontologies to approximate their original semantics. This patching process is undertaken by:

1. Identification of constraints between relevant and irrelevant concepts.
2. Manual curation and re-construction of constraints and class hierarchies based on relevant concepts.
3. Removal of irrelevant concepts.

#### ONTOLOGY ALIGNMENT

This process seeks to convert the pruned ontology to the correct semantic profile and structure of the target ontology. Terms that are either too expressive or conflict with the target ontology profile may be either deleted or reconstructed. For reuse in the RaCoOn ontologies, this was undertaken manually, intuitively mapping RDFS ontologies into OWL<sup>6</sup>, or reducing the expressivity of OWL Full ontologies to suit our requirements, as in the following examples:

- Refinement or modification of property domains and ranges

<sup>6</sup> Some semantic conflicts exist between RDFS and OWL, as discussed in Pan and Horrocks [157]

- Conversion of `rdfs:Class` concepts to `owl:Class`
- Assertion of `owl:DatatypeProperty` or `owl:ObjectProperty` membership over property resources.
- Restriction of object property assertions to within the OWL DL profile (from OWL full).

Additionally, consideration of how concepts fit with the target ontology meta-model should be made. In the RaCoOn ontologies, re-used concepts were subclassed underneath existing categories and classes, so as to fit with the existing ontology structure and to allow conformance with existing constraints and axioms. Efforts were also made to ensure that imported ontologies met the documentation standards set out in [Section 4.7.2](#), which require that all concepts have a minimal set of RDF annotations describing their purpose.

This method of ontology alignment pruning works well for extracting vocabulary and patterns from large ontologies for reuse elsewhere. It does, however, affect compatibility with the original models themselves, particularly where axioms have been altered to fit the expressivity of the target ontology. This is a necessary trade-off where the profiles of two models conflict, but limits compatibility with these existing models in the future.

#### 4.5.5.3 *Proposed Ontology Evaluation and Namespace Assignment*

Once the alignment stage is undertaken, the candidate ontology should be validated and tested, by:

- Checking its expressivity conforms to that expected and required by the target ontology
- Testing that it can represent information in the target ontology as expected, by evaluation against test data from the original application and for the target domain ontology (see D’Aquin [44])

The namespace of the modified ontology should be modified to reflect that it no longer reflects the exact same semantics as its original model, and provenance information added to state this. In RaCoOn, the original CURIE prefixes continue to be used, to visually indicate that the re-used ontology is similar to its source and should be used in the same way.

#### 4.5.5.4 *Ontology Integration*

Re-used ontologies can be integrated either through `owl:imports` declarations in the target ontology, or through adding axioms to the document directly. In RaCoOn, ontologies representing small design patterns were integrated directly (such as the W<sub>3</sub>C Geo vocabulary) whilst larger ontologies such as PROV and QUDT were imported from local files.

#### 4.5.6 *Expressivity and Reasoning*

OWL allows ontologies to be built with a high level of expressivity; indeed, this is one of the motives for its choice as an ontology language for the models described in this thesis. However, highly expressive ontologies that take full advantage of the semantic constructs OWL DL provides are difficult to reason over: the worst case N<sub>2</sub>EXP-time performance characteristic means that inference over large knowledge bases can be impossible using standard tools. Although optimised reasoners show better performance, full DL reasoning over large knowledge bases is still unfeasible [104], and other approaches are often required.

To better understand the implications of ontology axioms and inference, it is worth highlighting three applications for an ontology written using OWL, considered in [Chapter 2](#):

1. **Open World reasoning:** using formal semantics to deduce additional information that is useful to a domain or set of systems.
2. **Closed World reasoning and constraint checking,** to provide data validation. In this way, a new set of assertional data about a system can be checked to see if it conforms to an ontology's model of the domain.
3. **Detailed conceptual modelling:** using OWL as a language with which to formally describe a domain or system, with little emphasis on reasoning. Such models are typically created in OWL Full, and ignore OWL DL semantic restrictions where convenient.

In the case of open world reasoning, three general use cases for an industrial domain ontology are envisaged:

1. **Materialisation and validation of the ontology (T-box) itself** using reasoning.

2. **Enrichment of small assertional models** (such as static railway network data) by inferring axioms from a domain ontology.
3. **Web-scale reasoning** over large or federated knowledge bases or mapped systems, in order to infer extra semantics based on domain logic. Examples are shown in [Chapter 6](#).

The requirements of these use cases lead to different characteristics of ontologies to be designed. Axioms required by detailed conceptual models lead to performance decreases in reasoning, and design patterns designed using expressive OWL DL characteristics may be better re-written using different constructs if web-scale reasoning is required.

#### 4.5.6.1 *Model Expressivity*

It is likely that a domain ontology will be used in all three of the described situations. For small models, expressive ontologies provide a way of inferring data about a set of knowledge, and for large models reasoning provides a way of encapsulating domain knowledge and business rules, as long as it is reasonably efficient to compute these rules. The logical modularisation approach shown in [Section 4.3](#) allows combinations of different modules to be used in different scenarios.

In the RaCoOn ontologies, each semantic module is split into two logical modules: a ‘core’ module containing terminology, T-box relations, and other minimal semantics<sup>7</sup>, and a ‘constraints’ module, containing restrictions on classes and more highly expressive constructs. This allows construction of expressive and descriptive ontologies, whilst allowing only the core part of the model and to be used in applications where scale and speed are more essential. [Table 4.3](#) shows the constructs which are placed in each ontology.

This approach provides an alternative to simply choosing to interpret a subset of a DL ontology’s semantics when scalability and performance is required. It has several advantages:

- Easier predictability of inferred axioms. By having an RL-compliant core subset and using an OWL RL reasoner over this subset, sound and complete reasoning over it is achieved. If using an OWL RL reasoner on an OWL DL ontology, knowing what inference behaviour to expect requires in-depth knowledge of the OWL RL profile.

<sup>7</sup> both the upper and rail core ontologies conform to OWL 2 RL

Table 4.3: Division of OWL constructs between core and constraints ontologies

Core	Constraints
Classes	Property chain restrictions
Object & Datatype Properties	<code>owl:EquivalentClass</code> assertions
Core Individuals	<code>owl:subClassOf</code> restrictions
Class and property hierarchies	Cardinality restraints
Annotation properties	Closure axioms
Simple domain and range restrictions <sup>9</sup>	Complex domain/range restrictions

- **Finer control over expensive constraint axioms:** some assertions in the constraints module are valid in OWL RL, but still expensive and have limited use in large scale applications.

#### 4.6 STAGE 5: VALIDATION, EVALUATION, AND ITERATION

In light of existing state-of-the-art ontology validation approaches, as described in [Section 2.4.8](#), several techniques are suggested for use in the RaCoOn methodology. Owing to the nature of the industrial domain ontologies the methodology is intended for, many conventional validation techniques cannot be used. The need for additional modules to satisfy any one application's requirements, for example, makes corpus-based evaluation difficult. Four goals are focussed on, which are applicable to domain ontologies with a high degree of knowledge re-use, as follows:

- **Goal: Structural, syntactic, and logical correctness.** These factors can be assessed through inspection using automated means. Ontologies should be checked to see that they conform to relevant RDF and OWL profiles, and that the structure of the model conforms with expected semantics.
- **Goal: Domain Coverage and Scope.** The scope of the model should be assessed through consultation with industry experts and stakeholders, and through comparison with similar modularisation approaches shown in literature. This is achieved through comparison with existing models and expert knowledge.
- **Goal: Assess Suitability.** The usefulness of the models should be assessed through in-use validation. Legacy systems and applica-

tions should be re-implemented using the model framework in order to assess system functionality and suitability.

#### 4.6.1 *Logical Validation*

Structural, syntactic, and logical validation is an initial ontology validation step that guarantees at least structural, syntactic, and logical interoperability with other models. Here, we suggest three methods for validating models that check conformance with OWL and analyse ontology quality using automated means. Owing to its automated nature, this form of validation can be carried out at the end of every design iteration. Analysis results can then be used in the following iteration of the ontology design process to address any inconsistencies or issues in model design.

#### RDF AND OWL CONFORMANCE

RDF syntax compliance is a requirement of all semantic models and vocabularies suggested in this methodology. To ensure RDF validity, two approaches are suggested:

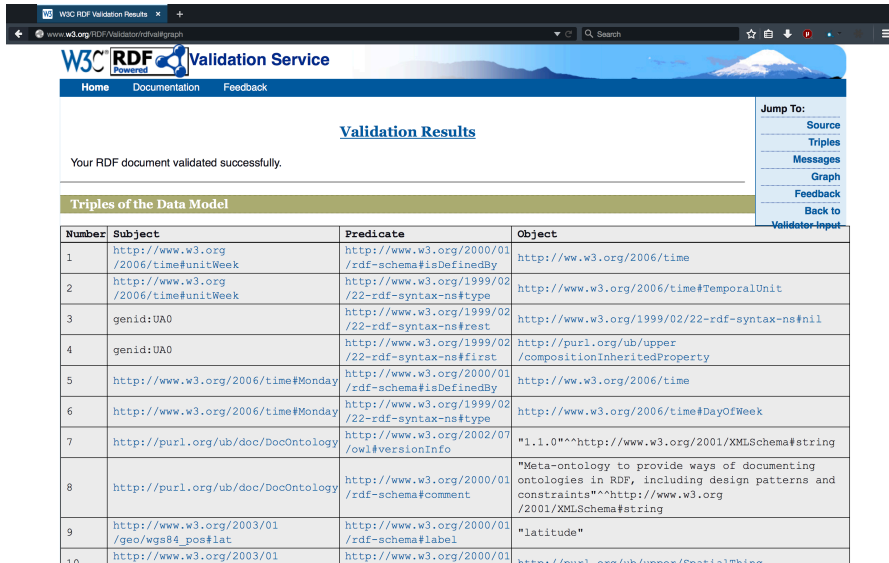
- **Development of ontologies in RDF-centric development tools** such as Topbraid Composer and Protégé, as well as RDF APIs, restricts ontology development to valid RDF documents. Where other tools were used in the development of the RaCoOn models (such as text editors), ontologies were validated by inspection in Topbraid Composer.
- **Validation and debugging using the W<sub>3</sub>C RDF Validator<sup>10</sup>** at milestones. This relies on a different parser implementation to those used in Jena and the OWL API, and can be used to confirm the validity of document syntax. A screenshot is shown in [Figure 4.15](#).

For ontologies (T-boxes), all models should conform to the OWL DL profile. [Listing 4.3](#) shows a method that calls upon the OWL API to check both syntax and conformance to all OWL profiles.

An alternative to this custom validator is the Manchester OWL 2 Validator<sup>11</sup>, a web application built on the OWL API that provides a similar service, but presents its results in an easier to read format and provides additional information on complex axioms.

<sup>10</sup> <http://www.w3.org/RDF/Validator/>

<sup>11</sup> <http://owl.cs.manchester.ac.uk/validator/>



W3C RDF Validation Results

www.w3.org/RDF/Validator/validgraph

W3C RDF Validation Service

Home Documentation Feedback

Validation Results

Your RDF document validated successfully.

Jump To: Source Triples Messages Graph Feedback Back to Validator-Input

Triples of the Data Model

Number	Subject	Predicate	Object
1	http://www.w3.org/2006/time#unitWeek	http://www.w3.org/2000/01/rdf-schema#isDefinedBy	http://www.w3.org/2006/time
2	http://www.w3.org/2006/time#unitWeek	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2006/time#TemporalUnit
3	genid:UA0	http://www.w3.org/1999/02/22-rdf-syntax-ns#rest	http://www.w3.org/1999/02/22-rdf-syntax-ns#nil
4	genid:UA0	http://www.w3.org/1999/02/22-rdf-syntax-ns#first	http://purl.org/ub/upper/compositionInheritedProperty
5	http://www.w3.org/2006/time#Monday	http://www.w3.org/2000/01/rdf-schema#isDefinedBy	http://www.w3.org/2006/time
6	http://www.w3.org/2006/time#Monday	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2006/time#DayOfWeek
7	http://purl.org/ub/doc/DocOntology	https://www.w3.org/2002/07/owl#versionInfo	"1.1.0"^^http://www.w3.org/2001/XMLSchema#string
8	http://purl.org/ub/doc/DocOntology	http://www.w3.org/2000/01/rdf-schema#comment	"Meta-ontology to provide ways of documenting ontologies in RDF, including design patterns and constraints"^^http://www.w3.org/2001/XMLSchema#string
9	http://www.w3.org/2003/01/geo/wgs84_pos#lat	http://www.w3.org/2000/01/rdf-schema#label	"latitude"
10	http://www.w3.org/2003/01	http://www.w3.org/2000/01	http://www.w3.org/2000/01

Figure 4.15: W3C RDF Validator Results for Upper Ontology

```

public void profileOntology() throws OWLException {
    // Get hold of an ontology manager
    OWLOntology o = m.loadOntologyFromOntologyDocument(new
    → File("ontology.rdf"));
    ArrayList<String> conformantProfiles = new
    → ArrayList<String>();
    for (OWLProfile profile : PROFILES)
    {
        OWLProfileReport report = profile.checkOntology(o);
        for (OWLProfileViolation v :
    → report.getViolations())
            System.err.println(v.toString());
        if (report.isInProfile())
            conformantProfiles.add(profile.getName());
    }
    System.out.println(String.format("Ontology is in
    → profiles %s", conformantProfiles.toString()));
    return;
}

```

Listing 4.3: Ontology Validator Method Listing



## LOGICAL ANALYSIS USING OOPS!

The Ontology Pitfall Scanner (OOPS!) [168] is a validation tool that uses an automated tool to check several OWL ontology characteristics, and its use is suggested in the RaCoOn methodology to check for common pitfalls in ontology design to evaluate work throughout the development process. It categorises these characteristics into six groups, including human understanding, consistency and compliance, modelling issues, and import resolution. Results from OOPS! should be considered with some caveats:

- Some OOPS! ‘pitfalls’ are the result of intentional design decisions and misdiagnosed, and discretion should be used when interpreting results. For example, OOPS! misdiagnoses ‘missing’ inverse properties even if they are intentionally omitted.
- OOPS! does not consider ontology re-use or modularisation: tested models must be complete and compliant. This can create validation warnings for several reasons, even if the ontology is logically consistent. Errors occur, for example, when re-using non-DL web ontologies from other sources.
- Warnings about missing axioms in imported ontologies. OOPS! analyses the union of an ontology and its imports: errors present in third party ontologies are presented in the same way as warnings about the authors’ models.

Thus, the results of the OOPS! scanner are an indication of the likely quality of an ontology, but some ‘minor’ pitfalls are expected from large ontologies. The RaCoOn ontologies are validated in this way in [Section 5.5](#).

#### 4.6.2 *Ontology Coverage through Application Data Mapping*

When designing models for domains that currently employ a large number of heterogeneous information systems, it is possible to draw upon some of these systems to validate candidate new models. Authoritative data models and sources are utilised for initial ontology design, scope, and construction, and smaller applications and models can be used to verify and validate that a model’s scope and implementation are appropriate.

One metric for validation of the *usefulness* of an ontology proposed here is its ability to represent data from such applications; especially

those not initially drawn upon in the knowledge engineering stage. In the absence of a gold standard model or a large corpus of domain-centric text, application data models can be used to elicit domain terminology and relationships manually. By creating mappings from known real-world data sources to the ontologies, a measure of coverage and modelling quality can be established. An approach that applies this is suggested as follows:

1. **Discovery and choice of legacy applications** and corresponding datasets for integration. Choice should be based on the scope of applications themselves rather than underlying data models to avoid bias against those known to include concepts foreign to the testing ontologies.
2. **Conceptualisation of legacy data.** The semantics of data present in a model should be re-established, either through consultation with an appropriate schema, through use of the application itself, or through consultation with application users.
3. **Mapping of legacy data to ontology under test.** This could be undertaken using manual or automated schema mapping tools, OWL mapping languages, or bespoke software. Such approaches are not discussed in detail here; overviews of ontology mapping techniques are provided by Kalfoglou and Schorlemmer [118] and Rahm [174].
4. **Analysis of initial coverage:**
  - How much of the information represented in the legacy model can be represented in the ontology appropriately?
  - How much of the information represented in the legacy model can be represented, but with unintended semantic repercussions?
  - Are axioms present in the ontology invalidated by legacy data? Is this caused by false assumptions in the legacy model or false assumptions in the ontology?
5. **Ontology extension.** In the case where legacy data is not fully represented, can extensions be used to encompass all data?
6. **Analysis of extension**
  - Was extension impeded by the ontology at any point (for instance through overly restrictive constraints)?
  - Should any of the knowledge in the extension ontology be present in the domain ontology?

7. **Evaluation and Iteration:** Metrics from the above questions should be compiled to give an indication of validity based on integrating this system. The ontology may be extended in light of gaps or errors found, and the process repeated.

This approach provides an indication of the quality and scope of the part of a domain model corresponding to a given application's use case. The techniques used to identify necessary ontology concepts vary between mappings; in some cases, manual schema mappings of a small data model may validate some part of an ontology, whilst in others a formal project to map all elements of a more comprehensive legacy data model may be necessary to more accurately establish coverage.

The application of this approach to the RaCoOn models is shown in [Section 5.5](#).

#### 4.6.3 *In-use Validation*

As an extension to the ontology mapping approach outlined above, domain model usability in the RaCoOn model was further evaluated through its use in several in-depth application implementation projects. In [Chapter 6](#), the development of several real-world use cases around railway data is described, all of which extend RaCoOn models to suit their applications. These projects warranted extensions to the domain ontology to represent missing concepts, relationships, and axioms, and created application-specific ontology modules for this purpose.

A measure of coverage for the base (domain) ontology can thus be achieved through analysis of the concepts present in the extension ontologies. In [Section 5.5](#), concepts created in the application ontologies are categorised according to their purpose (domain or application) and detail, so that concepts missing from the core ontology can be identified. OWL concepts and relationships are rated with a measure of certainty into one of the following categories:

1. **Application-specific.** This concept has been created only for the use of the application itself, and does not have any wider use in a domain ontology.
2. **Subdomain-specific.** This concept is common to the railway domain, but does not fit into the remit of the core ontology.
3. **Domain-wide.** This concept should be in an existing domain model.

By evaluating concepts in this way and inspecting results, a further measure of domain ontology coverage and quality can be obtained. Entities that are categorised as ‘domain-wide’ are likely to indicate concepts that are either missing from the core ontology, mis-labelled, or have incorrect semantics.

#### 4.6.4 *Similarity Measurement Through Expert Knowledge Elicitation*

To further assess domain coverage, and accuracy of the obtained model, a method for evaluation through workshops with domain experts is described in [Section 5.5](#). Rather than trying to validate every term in the vocabulary of a candidate domain model, this approach aims to elicit a wide range of concepts and relationships such that a set of ontologies can be evaluated against another conceptualisation of the domain. The workshop aims were as follows:

1. Elicit a list of top level domain concepts from attendees, to further evaluate the scope of the model against perceptions of the wider domain.
2. Generate a set of key subdomain terms, to evaluate ontology coverage of chosen subdomains.
3. Gather sets of interrelationships between terms, to ‘spot check’ relationships gathered in the domain ontology.

The approach was designed to gain the best possible conceptual view from a group of experts without pre-biasing them towards the ontology that had already been created. An exhaustive approach to this task would have been prohibitively expensive or time-consuming (in effect, it asks attendees to fully conceptualise a domain themselves, so that an ontology can be compared to the ‘correct’ answer), but the idea of asking attendees to assert random facts onto their constructed models allowed us to take advantage of the depth of their knowledge. A partial workshop conceptualisation that agrees with the domain ontology constructed implies that the quality of the modelling in the domain ontology is good; missed axioms that are considered ‘in scope’ implies that the model is not yet thorough enough.

#### 4.6.5 *Iteration and Version Control*

To track the iterative design process, it is useful to record and recall different versions of ontologies over time. Version Control Systems

(VCSs) are used widely in software engineering to manage collaborative software development, and allow tracking and backup of progress through time. In the development of RaCoOn ontologies, the Git<sup>12</sup> VCS was used to facilitate iterative design in the following ways:

- **Commits** to the repository are snapshot backups of the set of ontologies at the time they are made. Commits were made often, marking development progress of the ontology on a day-to-day basis. *Commit messages* allow the progress undertaken to be documented in one or several lines of documentation.
- **Branches** were used to allow the development of a pattern or feature, and provide an easy way of documenting progress or design of new functionality.
- **Merges** were undertaken to combine features back into the master set of ontologies.

In software development, VCS systems such as Git allow automatic merging of different feature branches, based on line-by-line differences between versions. In OWL, care must be taken when merging files, as an ontology's consistency can be affected. For this reason, manual merges were performed in the RaCoOn development process. [Figure 4.16](#) shows an extract of the RaCoOn ontology's development history.

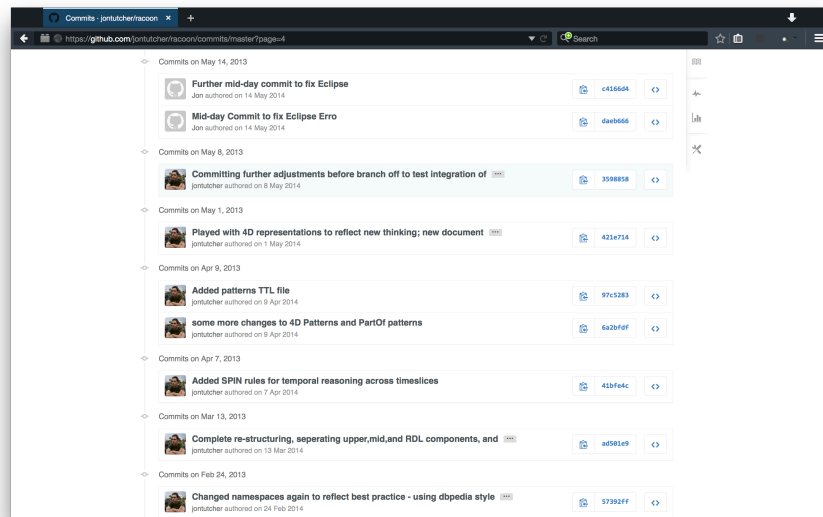


Figure 4.16: RaCoOn Ontology Git Commit History

<sup>12</sup> <http://git-scm.com>

## 4.7 BEST PRACTICE IMPLEMENTATION DESIGN PATTERNS

As part of the RaCoOn ontology engineering methodology, we also describe several additional ontology design patterns in addition to those already mentioned that encourage model extensibility and ease of reuse. These patterns are implemented in the ontologies described in [Chapter 5](#), and examples provided here.

### 4.7.1 *Annotation Best Practice and Naming Conventions*

‘Self-documenting’ ontologies help users intuitively understand models, and our ontologies should be annotated accordingly. Annotation patterns used follow linked data best practice and are summarised in this section. Firstly, ontological entities are annotated in natural language using additional `owl:AnnotationProperty`s, which have no impact on its formal semantics. Each entity should provide at least the following attributes:

- `rdf:label` provides a human-readable label for each concept
- `rdfs:comment` provides a design comment for each entity

Additionally, individuals should be marked up with a `dc:description` to provide a brief description of what they represent. Extension modules should also follow this pattern, other motives for which are summarised in Allemang and Hendler [5]. Secondly, consistent naming conventions are used throughout all ontologies as follows:

- Ontology namespaces should convey provenance. Ontology modules built by the same author, in a hierarchy, should follow a URI pattern that reflects this hierarchy. If appropriate, URIs should correspond to an organisation’s web presence.
- All newly defined entities should be assigned URIs corresponding to the namespace of the ontology module they belong to, as described in [Section 5.2.1](#)
- Classes and individuals should be named according to their identity criteria in English, in UpperCamelCase.
- Properties should be named by noun (without a preposition), and are in lowerCamelCase. Inverse property names match original property names but are suffixed in some way: for example `u:measurement owl:inverseOf u:measurementOf`.

Ontology prefixes used across the RaCoOn project are shown in [Section B.1](#).

#### 4.7.2 *Ontology Self-documentation*

Documentation of ontology design patterns is typically carried out by annotating constructs and axioms within the ontology, and providing accompanying textual documentation separately<sup>13</sup>. In the RaCoOn methodology, we present a new pattern to allow concepts associated with ODPs to be documented through further OWL meta-links to individuals representing the ODPs themselves. This allows ontologies to be somewhat self-documenting, and aids new users. The DUL ‘Content Pattern Annotation Schema’ ontology also provides a set of properties for representing a design pattern in OWL, but no mechanism for defining them as entities<sup>14</sup>).

#### IMPLEMENTATION

An implementation of this pattern is provided in the RaCoON documentation ontology, described in [Chapter 5](#). An annotation property and class, `doc:partOfODP` and `doc:ODP` are provided. A `doc:ODP` instance represents one design pattern, and all entities that are part of this pattern are linked to via `doc:partOfODP`. The design pattern entity can then be annotated using standard vocabularies to suit the documentation format.

#### EXAMPLE

[Figure 4.17](#) shows how this pattern is used to describe the measurement pattern documented in [Section 5.3.2.2](#), providing a name, comment, and link to HTML documentation.

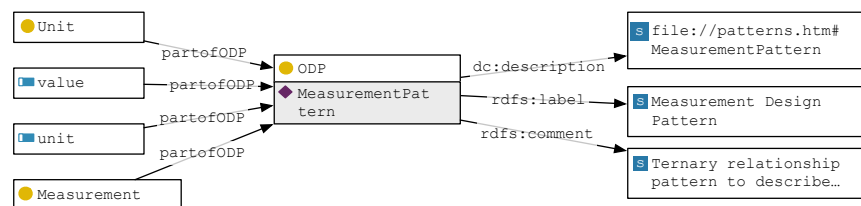


Figure 4.17: Documentation Pattern for Representing Content Pattern Association in OWL

<sup>13</sup> Well-regarded design patterns from the literature are almost always presented in this way, for example: Dodds and Davis [51], Gangemi and Presutti [71] and [5].

<sup>14</sup> <http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl>

### 4.7.3 *Provenance, Trust, and Metadata*

Provenance is information about the origin of a set of data, such as the organisations responsible for creating or modifying it, the activities used to generate it, and when or where it originated. Such information is of great importance when considering data interoperability, and can drive choices and considerations on how a dataset is interpreted, either at a system design level or during evaluation by a machine. Even if the explicit semantics of a dataset are fully considered, its origin and the context under which it was created can have a significant bearing on its use. Provenance in the RaCoOn ontologies is represented by considering two scenarios:

1. The provenance of an individual represented within the model, such as an instance of a class `ex:Document`.
2. The provenance of assertions made in the semantic data models themselves, i.e. model *metadata*.

Here, three levels of granularity for encoding provenance in RDF models are considered:

1. **Graph provenance.** Metadata can be added to RDF graphs in order to represent the provenance of a physical or logical dataset itself.
2. **Concept provenance:** encoding the provenance of classes and individuals within an RDF model using annotation properties.
3. **Triple level provenance:** where finer description of data provenance and trust is required, triples can use *context identifiers* to encode an additional URI to represent additional data.

For domain ontologies, we suggest that provenance of data sets should be asserted at both the graph level and at the concept level. Two patterns using existing ontologies facilitate this, and are recommended for use when asserting provenance of assertional data.

#### 4.7.3.1 *Graph Level Provenance and Dataset Descriptors*

When considering interoperability between RDF datasets, provenance of the document itself becomes valuable. For example, an application displaying rail infrastructure by combining several data sources may wish to prioritise information from one publisher over that of another.



The RDF data model does not provide any explicit mechanisms for representing such provenance information, but linked data standards for doing so have emerged.

Our methodology recommends that ontologies and assertional datasets follow include RDF metadata expressed using the Vocabulary of Interlinked Datasets (VoID) vocabulary [4], which provides data provenance such as versioning and ownership, as well as other annotational information. Whilst VoID is primarily a tool to aid linked data vocabulary discovery, and is designed to describe large datasets of instance data rather than ontologies themselves, the vocabulary it provides is effective for describing ontologies too. Dedicated ontology metadata models also exist [96], but are far less widely used than VoID. A minimal example is provided in Listing 4.4.

```
<http://purl.org/ub/upper/> rdf:type void:Dataset, owl:Ontology
↪ ;
foaf:homepage <http://purl.org/ub/upper/> ;
dc:title "BCRRE Upper Ontology" ;
dc:description "A simple upper ontology intended for use with
↪ pragmatic industrial data models" ;
dcterms:publisher <http://jtutcher.co.uk/jon> ;
void:dataDump <http://purl.org/ub/upper/> ;
```

Listing 4.4: Example VoID Description of Upper Ontology

This data can be thought of as ‘global’ provenance and metadata of a data set, and is used to represent unchanging, model-wide information. In an application that draws upon multiple datasets, this information could be used to express descriptions, or data origin, to allow consuming applications to decide whether to trust and incorporate a data source.

#### 4.7.3.2 *OWL Concept Provenance Using Meta-modelling*

Several ontologies for representing information provenance exist, with the W3C PROV-O ontology recently having become a de facto standard amongst the linked data community. It is a widely reused method for representing the provenance of information resources, and is thus a desirable way of representing such data in domain models. The methods and constructs for representing data provenance using PROV-O are not discussed here, but are overviewed succinctly in the W3C PROV Model Primer [76].

PROV-O is designed to model the provenance of real world concepts such as documents (physical or digital) and artifacts, and repre-

sents these real world entities using OWL individuals. In other words, it is designed to model the provenance of the thing expressed by its OWL representation, and not the provenance of the OWL concept itself. To document an ontology, a form of *meta-modelling* must be used to allow the assertion of provenance about OWL concepts themselves. This is easily accomplished using `owl:AnnotationProperty` relations, which maintain a logical distance between the concept and its meta-modelled provenance.

#### IMPLEMENTATION AND EXAMPLE

To illustrate this approach to meta-modelling the provenance of concepts in PROV, an example is provided below in three stages:

1. Firstly, a `prov:Entity` individual is created to represent the provenance of a particular class, property, or collection of terms.
2. Appropriate provenance information is asserted on this individual according to the PROV-O ontology. Information may include details such as author, modification date, and other tools or PROV entities that contributed to the creation of the term(s) that provenance is being expressed on.
3. Classes within the candidate ontology are linked to this `prov:Entity` individual by means of an `owl:AnnotationProperty`. As such, rich provenance information can be expressed about any OWL or RDF terminology expressed in the ontology without affecting the logical characteristics of the terms themselves.

Whilst this is not an intuitive approach, it allows the assertion of provenance on all concepts in the ontology under development very succinctly, using a known vocabulary and toolset. Provenance can be asserted over the design of the ontology with the full power of the PROV-O model, allowing changes to its structure and ownership of different parts of models to be tracked and controlled. It does, however, have several disadvantages. Firstly, the use of OWL annotation properties with no formal semantics mean that provenance information cannot be easily used to reason over the ontology itself. Secondly, storage of provenance metadata alongside domain knowledge in an ontology increases its total number of concepts, and could affect reasoning performance as the provenance entities are still first-class OWL constructs. One solution to this may be to store detailed provenance information in separate ontology modules as required, although this possibility is not explored further here.

An example of this technique is shown in [Figure 4.18](#).

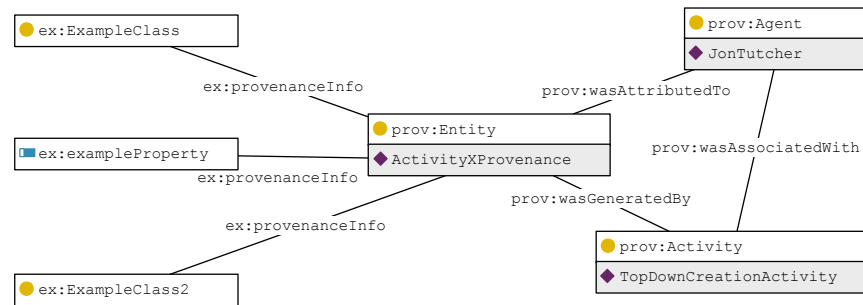


Figure 4.18: Example Use of Meta-modelling to Assert Provenance Information on Ontology Concepts

#### 4.7.3.3 *RDF Triple Provenance Using Named Graphs*

Occasionally, it may be necessary to represent the provenance of one particular (or one group of) RDF triples. This can be undertaken by taking advantage of RDF quads, or ‘named graphs’, which append a fourth ‘context’ item to each triple. Named graphs are used (and abused) for many purposes, but can only be used in one particular way in each application. Our methodology recommends that the use of named graphs is reserved for representing provenance information:

- Assigning named graphs to sets of triples allows an RDF triplestore to treat some sets of data differently to others, whilst maintaining the ability to query across all datasets. Stardog [37] for example allows reasoning capabilities to be specified per graph.
- Named graphs are part of the SPARQL 1.1 specification, and allow users to query across any logical combination of graphs. If they are used to represent data provenance, the capabilities of SPARQL queries for extracting data are extended. Assuming that each data supplier is assigned a named graph, it is for example possible to easily answer the query ‘which organisations hold information about asset X’ using SPARQL.
- Named graphs are not part of the OWL specification, and most OWL tools do not support their use to imply concept semantics. By restricting their use to data provenance, the use of OWL toolsets for ontology creation is simplified.

This technique is used extensively in [Chapter 6](#) to integrate asset monitoring and passenger information data from multiple sources.

## 4.8 SUMMARY

This chapter has summarised a new ontology engineering methodology for creating semantic domain models using ontology engineers, expert knowledge and existing enterprise information resources. It outlined pragmatic methods for scoping, building, and validating such ontologies in the absence of a defined set of final use cases—a property which allows such models to be built to assist data sharing across a domain, as is the requirement in this thesis. Importantly, the methodology was developed during work with several railway companies, and addresses some of the practical barriers encountered during this process.

The next two chapters use the methodology described here to build a set of domain ontologies for the railway ([Chapter 5](#)) and two applications with accompanying application ontologies ([Chapter 6](#)) to demonstrate real-life system implementations. These applications additionally allow the domain ontologies to be assessed and validated according to the techniques proposed in [Section 4.6.3](#), and for the other design practices outlined in this chapter to be demonstrated.



## RACoon: PRAGMATIC ONTOLOGIES FOR THE RAIL INDUSTRY

---

### 5.1 INTRODUCTION

As described in [Chapter 4](#), the Rail Core Ontologies (RaCoOn) are a set of novel ontology modules that together form a conceptualisation of part of the railway domain. The ontologies were designed according to the methodology set out in the previous chapter, and are focussed on infrastructure-centric concepts and provide a set of practical models for use in data exchange applications.

This chapter presents the implementation and content of this set of ontologies, and shows the results from validation of these ontologies according to the methodology set out above. It overviews concepts included in the cross domain and rail domain modules, and explains design decisions made in each case. To start, a description of the modular structure adopted by the RaCoOn ontologies is given.

### 5.2 MODULAR ONTOLOGY DESIGN

#### 5.2.1 *Ontology Module Structure*

To enable maximum extensibility, the RaCoOn ontologies are divided into several subject-based modules based on the principles outlined in [Section 4.3](#). The two principle modules are described in detail in this chapter. The first, a ‘cross domain’ ontology, provides a conceptualisation of cross-domain concepts, including space, time, concept type, and documentation. The ‘rail core’ module extends this upper ontology with generic railway domain concepts and relationships, to act as a base domain ontology and vocabulary for exchange of data across applications. Three small subdomain placeholder ontologies are suggested for rolling stock, infrastructure, and timetabling, although these were not the primary focus of this thesis. A diagram of RaCoOn ontology modules is provided in [Figure 5.1](#).

The granularity of these modules is based upon the use cases discussed in [Chapter 4](#). The scope of each was determined by the modelling process, using knowledge from existing resources and experts. Each module imports relevant modules from higher levels but none

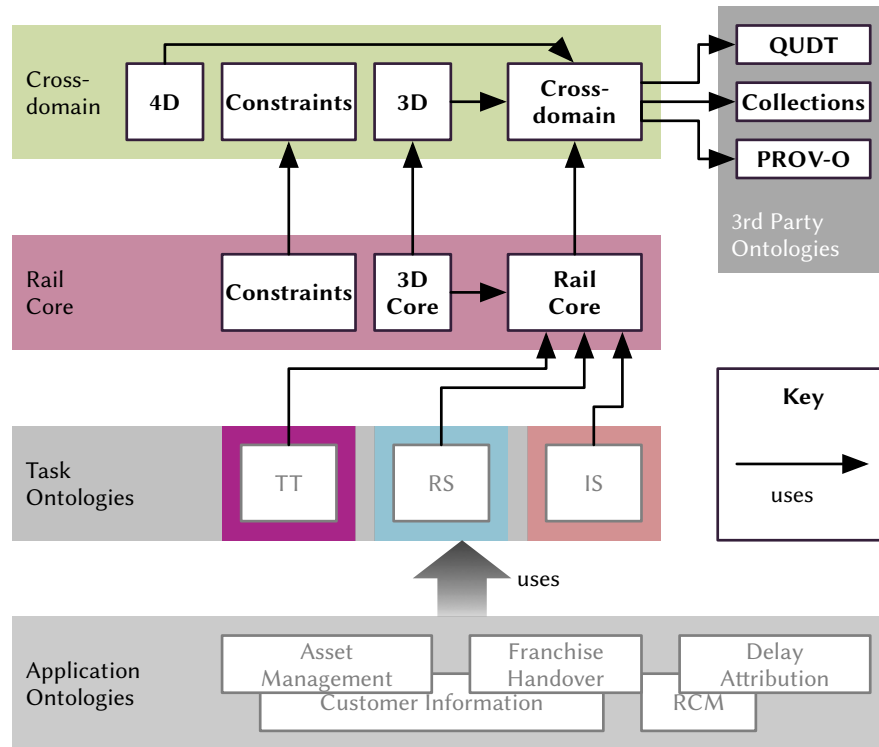


Figure 5.1: RaCoOn Ontology Modules Structure

from lower levels. The purpose, namespace, and dependencies of each module are shown in [Table 5.1](#).

Thus, applications which require a particular subset of features can import only the modules they require. In OWL, dependencies are asserted by the `owl:imports` property on an ontology entity, and so are transitive—using the ‘3D Core’ module imports the core vocabulary, cross-domain ontology base, and cross-domain 3D modules.

### 5.2.2 Key Concepts and Semantic Trade-offs

To facilitate the requirements for an industrial data exchange ontology given in [Section 4.2](#), certain high level design decisions were made in the design and implementation of the RaCoOn ontologies in OWL. These were as follows:

- **Minimal constraints.** RaCoOn does not assert the types of content constraints that could be associated with railway application models, as these rarely hold across an entire domain, and the model is not intended to provide closed world data validation. Typical restrictions avoided include: the confinement of railway components to pre-enumerated types; existential rela-

Table 5.1: RaCoOn Modules and Dependencies

Module Name	Namespace prefix	Dependencies	Description
Cross domain—Base	u:	None	Base concepts
Cross domain—4D	u4d:	u:	Base 4D ontology
Cross domain—3D	u3d:	u:	Base 3D ontology
Cross domain—Constraints	ucv:	u:	(3D) Constraint Annotations
Rail core—Base	rcn:	u:	Railway vocabulary
Rail core—3D	core3d:	u3d: , rcn:	3D core axioms
Rail core—Constraints	corecv:	core3d: , ucv:	(3D) Railway constraints
Timetable	tt:	core3d:	(3D) Timetable module
Infrastructure	is:	core3d:	(3D) Infrastructure module
Rollingstock	rs:	core3d:	(3D) Rolling stock module

tions for expected characteristics (for example ‘stations must have platforms’). Constraints *are* asserted where it makes ontological sense to do so, such as restricting spacial properties to linking only spacial things, and where there is an immediate reasoning need to do so, such as the assertion of equivalent classes to identify characteristics.

- **Pragmatism over correctness.** Where possible, simple and extensible concepts have been used in place of more verbose but more expressive alternatives. The set of ontologies is intended as a practical tool and follows ‘linked data’ best practise, rather than a philosophically perfect mirror of reality. Examples of this include the approach taken to representation of quantities, units, and scales, and how time is encoded.
- **Self-documentation over semantic perfection in annotation.** Labels and names in ‘plain English’ are used to identify terms, rather than equivalent ontology engineering terms. Terms like ‘endurant’ and ‘fluent’ are avoided in favour of more easily understandable words.



### 5.3 THE CROSS-DOMAIN ONTOLOGY

The RaCoOn cross-domain (‘upper’) ontology comprises a simple set of extensible high level concepts that provide a foundation for domain-level industrial ontologies. It is intended as a minimal set of axioms to support the lower level domain ontologies rather than as a true upper level model, and seeks to address the following design questions:

- How can we represent the upper level concepts required for realisation of a railway domain ontology?
- How are these concepts conceptualised in a domain-independent way, such that the potential for re-use and extensibility across other industries is maximised?
- What are the basic distinctions between *things* in the real world?
- What high level concepts are subclasses of other high level concepts?
- What properties are required to encompass high level and common relationships between entities?

In pursuing answers to these questions, several more specific questions are raised, which are themselves answered by content patterns introduced later. The cross-domain ontology was created in collaboration with the railway domain modules described in [Section 5.4](#), and as a result many of the concepts it includes are in direct response to needs of the domain model itself.

#### REUSE OF CONCEPTS FROM STANDARD UPPER ONTOLOGIES

The cross-domain ontology does not directly reuse any existing upper ontologies (as reviewed in [Section 3.6.1](#)), but reuses certain concepts from other models. Existing upper ontologies evaluated either did not fit the domains addressed by the RaCoOn models, or introduced a large amount of abstraction and verbosity in order to properly represent concepts across all domains. This abstraction conflicts with the the requirement for understandability set out in [Section 5.1](#).

Ideas taken from the Basic Formal Ontology (BFO) are used to distinguish between *dependent* and *independent* entities and to consequently provide a way of using RaCoOn vocabularies in both 3D and 4D, as described in [Section 5.3.2.1](#). The cross-domain ontology subclasses dependent and independent entities separately, and

allows both temporal paradigms to be used to describe information. Where these are used, they are noted in subsequent sections.

### 5.3.1 *Conceptualisation, Structure and Patterns*

This section will outline the structure and concepts present in the cross-domain ontology, and provide details of their implementation as a set of reusable design patterns.

#### 5.3.1.1 *Top-level Conceptualisation*

The RaCoOn cross-domain ontology defines a top-level class hierarchy from which to extend domain and task models. The first two levels of this hierarchy are shown in [Figure 5.2](#).

The main classes in this top-level conceptualisation were arrived at firstly through top-down consideration of what concepts should exist in an upper ontology (driven by the design questions approach documented in the previous chapter) and also iteratively in conjunction with the rail domain ontology described further in this chapter. The semantics of these key entities are presented in the following sections.

#### DISCRIMINATION BASED ON DEPENDENCE

`u:IndependentThing` and `u:DependentThing` represent the distinction between entities that can exist in their own right, and concepts which are dependent upon some other entity. `u:IndependentThing` represents ‘real world’ physical and abstract entities such as railway objects and information assets, and anything whose identity criteria can exist in the real world its own right. `u:DependentThing` is the class of dependent entities such as measurements, attributes, and functions. Dependent objects can be broadly viewed as objects that describe independent objects, and are thus distinguished from independent objects. `u:IndependentThing` and `u:DependentThing` are similar to `dul:PhysicalObject` and `dul:SocialObject` respectively.

#### EVENTS AND TIME

`u:Event` is the class of events: things that cause change in other entities. Examples are a train departure, the activity of building a railway, or the act of observing a measurement in a condition monitoring system.

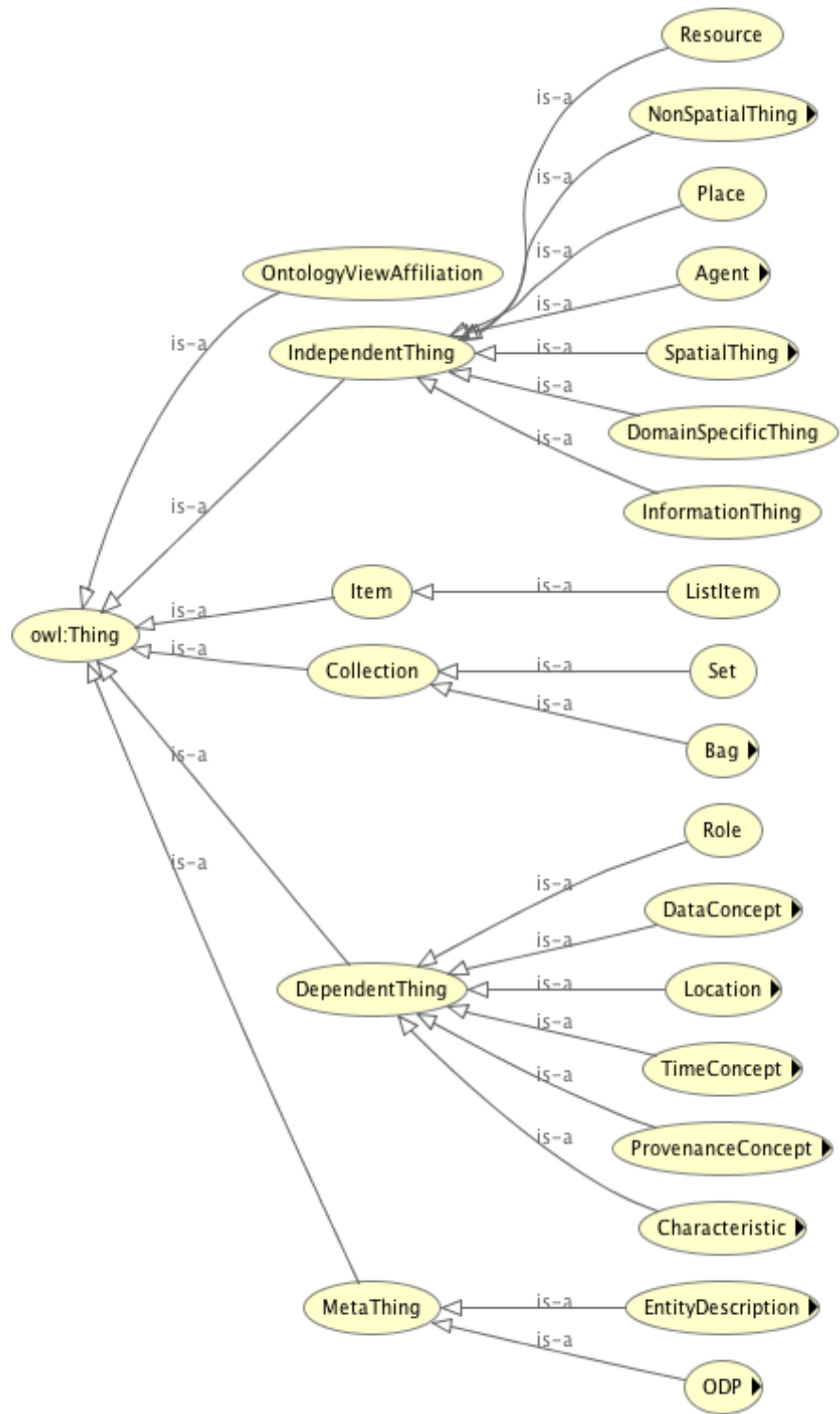


Figure 5.2: OWLViz Diagram of RaCoOn Cross-domain Ontology

## METADATA

`doc:MetaThing` represents meta-modelling and annotation concepts, such as specific methods for documentation of data provenance and design. The concepts discussed in [Section 4.7.2](#) are subclassed as part of `doc:MetaThing`, as are representations of dialect and presentation.

5.3.1.2 *Key Design Patterns*

## CHARACTERISTICS AND CONVENIENCE PROPERTIES

In OWL and DL models, the semantics of certain axioms allow a level of assertional redundancy. For example, defining the `rdfs:range` of an `owl:ObjectProperty` allows a reasoner to infer the class membership of an object (its type); conversely, linking to an individual of a particular `rdf:type` can portray additional semantics about the link without using a specialised property. For example, given the set of facts shown in [Listing 5.1](#) and the DL axiom below, a new set of facts can be inferred, as illustrated in [Listing 5.2](#):

$$\top \sqsubseteq \forall \text{ speed } \text{SpeedCharacteristic} \quad (\text{Range axiom})$$

$$\top \sqsubseteq \forall \text{ traction } \text{TractionCharacteristic} \quad (\text{Range axiom})$$

```
:Train :speed :_CharacteristicX .
:Train :traction :_CharacteristicY .
```

Listing 5.1: Example Assertions to Demonstrate `rdfs:range` Restrictions

```
:_CharacteristicX rdfs:type :SpeedCharacteristic .
:_CharacteristicY rdfs:type :TractionCharacteristic .
```

Listing 5.2: Example Inferences Made Through `rdfs:range` Restrictions

In the ontologies described here, it is often useful to define characteristics by creating specific `owl:Classes` to convey their semantics: this avoids the proliferation of many confusing object properties, and

allows greater extensibility. It is recognised, however, that more specific sub-properties may be easier to understand and use, and may facilitate use of the knowledge base without inference at all. The following design recommendations are therefore made:

1. Characteristics adhering to some design pattern that may require future extensibility are specialised by their `rdf:type` and related through a generic `owl:ObjectProperty`.
2. Commonly used relations to one particular subclass should use a more specific and descriptive property, which should be named as such and should be a sub property of the generic property.
3. These sub-properties should *not* be relied upon to infer the `rdf:type` of their object; this should be asserted explicitly.

These properties will be referred to throughout this thesis as *convenience properties*, as they do not provide new expressivity to the model.

#### OPTIONAL PROPERTY ASSERTIONS

OWL is a *monotonic* language. If a restriction is asserted over a class, this restriction must hold true at all times, and no exceptions to it can be made. Consequently, care must be taken to only assert class and property restrictions where necessary, so that infrequent exceptions to expected norms can still be expressed.

For ontology users, an indication of the optional properties expected of an individual could be useful, in order to provide further documentation of interactions within the domain modelled. Two solution patterns are used in RaCoOn to convey this information.

- **Solution 1:** Several `owl:AnnotationProperties` are defined to allow the assertion of optional properties. Users can either express optional property assertions using string literals (plain text), or by linking to `owl:Restriction` entities through these properties. The latter approach is more intuitive when reading documentation in ontology editors and in some serialization languages, but is not OWL DL compliant.
- **Solution 2:** The more widely used solution in documentation of RaCoOn is the use of zero cardinality restrictions to signify optional properties. Rather than enforcing a mandatory property (in open or closed world) by using an existential restriction, a cardinality restriction with minimum value 0 is asserted instead. This has no semantic consequence, but allows the ontology user to see that a relationship of this type may be expected.

### 5.3.2 Representation of Common Concepts

The following sections outline descriptions and patterns for representing core concepts in the RaCoOn cross-domain ontology. methods for describing temporally-changing data, quantities and units, and mereological concepts (composition and aggregation) are overviewed.

#### 5.3.2.1 Representing Temporal Data

The RaCoOn ontologies focus on representing data in 3D, and provides a set of patterns and restrictions to allow static and dynamic data to be represented using this paradigm. The majority of use cases considered in [Chapter 4](#) centre around information which is either static, or can be temporally expressed using versioning systems or metadata.

It is also, however, recognised that some applications may be better suited to a 4D paradigm. Rather than creating a complex single model to encompass both points of view, the RaCoOn module provides a skeleton 4D framework to re-use vocabulary in a 4D context.

Within the 3D paradigm, the *n-ary relations* pattern is used to reify temporal extents as described in [Section 2.6.1.3](#), and as shown again in [Figure 5.3](#). This pattern was chosen so as to reduce complexity and increase understandability of the application, and is described fully with regard to train timetabling in [Section 6.2.6.3](#). The relationships and constraints discussed in the rest of this chapter are implemented in the 3D model, whilst terminology is in another ontology to facilitate the use of the four-dimensional module.

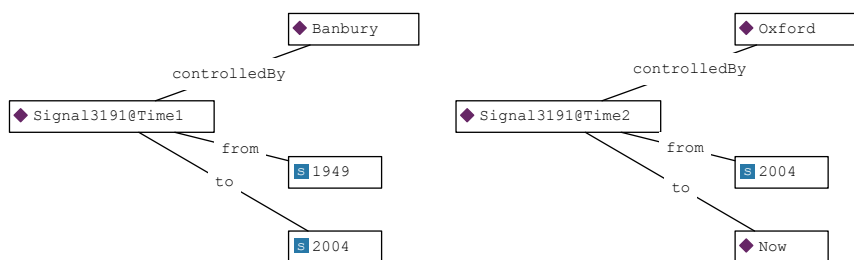


Figure 5.3: Representation of Temporally Changing Data in RaCoOn Cross-Domain Ontology

To represent data using the 4D paradigm, a module implementing the pattern described by Welty, Fikes, and Makarios [225] is also pro-

vided, which defines a new set of classes and properties to represent fluents and relationships. Although this vocabulary is missing the semantics and constraints of the 3D implementation of the ontology, it is useful for capturing knowledge using the same classes and properties as the 3D model.

### 5.3.2.2 *Quantities, Units and Dimensions*

Representing quantities, units, and dimensions in a data model is a recurring cross-domain problem, and one for which a number of solutions with different merits exist. Although the thought of a measurement such as ‘twenty-seven degrees’ is intuitive to humans, such a measurement actually makes a number of assumptions physically and meta-physically, and representing the exact semantics of such a measurement can be a challenge. The representation of these concepts is a fundamental part of the RaCoOn upper ontology, and builds on current state-of-the-art as discussed in [Section 3.6.3](#). The following CQs set out examples of knowledge that should be encoded using the patterns in this section.

- What is the length is Entity X?
- What units is the length of Entity X expressed in?
- What is the length of Entity X in metres?

#### SOLUTION

In RaCoOn, an approach based on QUDT is taken for modelling direct properties, and multidimensional properties are modelled in a similar fashion to in ISO 15926. RaCoOn defines a ‘Measurement’ class, which is the class of dimensional properties, and defines constraints such that a measurement is linked to a unit and a value datatype property. The definition of a dedicated class for measurements allows the definition of axioms and constraints, as well as measurement subclasses for specific purposes. An example of this pattern is shown in [Figure 5.4](#).

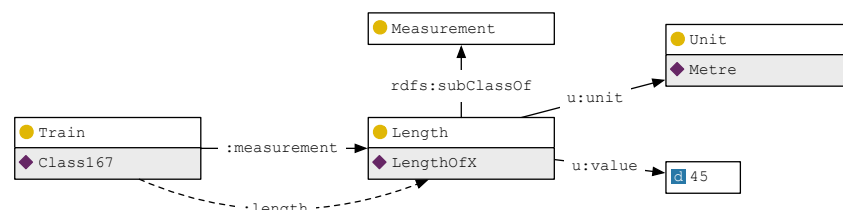


Figure 5.4: Example of Measurement Design Pattern in RaCoOn Ontologies

This is a specialisation of the ‘Ternary Relations’ pattern shown in [Section 2.6.1.3](#), and is rich enough to sufficiently represent single dimensional units. It requires fewer triples than the QUDT approach, but sacrifices some expressiveness. Using blank nodes and convenience properties, it allows measurements in an ontology to be represented in one line of Turtle, as shown in [Listing 5.3](#). Vocabulary and properties from the QUDT ontology are used under a separate namespace within the RaCoOn ontology, allowing restrictions on unit types and classes to be utilised.

```
:Class167Train :length [:unit :Metre, :value "167"^^xsd:Double]
```

Listing 5.3: Demonstration of Ternary Relations Measurement Pattern in Turtle

Multi-dimensional properties are modelled through the use of other `u:Measurement` subclasses, each of which links to one dimension through an appropriate object property. The geographic locations pattern shown in [Section 5.4.4.5](#) provides a specialism of this pattern, where it is explained in more detail.

### 5.3.2.3 *Representing Composition and Aggregation*

Mereology is the study of parts and wholes, and is a long established area of philosophical debate. Various mereologies, that is, ways of conceptualising part-whole relationships, have been formalised into OWL, and a comparison of these techniques is provided by Fernández-López, Gómez-Pérez, and Suárez-Figueroa [64].

Although it is foreseeable that complex part-whole relationships may need to be represented in applications based on RaCoOn, this need did not arise in the requirements elicitation process, and does not manifest itself in any of the existing data models studied. As such, we use the pattern from DUL [70], and model only very general mereological axioms. Those that are provided can be specialised and extended by applications and subdomains that require a more expressive way of representing such knowledge, and the simple representation included serves to infer component membership allow property inheritance. The simple competency questions required to model this basic mereology are:

- What parts does Entity  $X$  contain?



- What other entities is Entity X a part of?
- What logical components does Entity X have?
- What is Entity X a logical component of?

## IMPLEMENTATION

Two sets of object properties are provided to model entities as components of other entities:

- Reflexive properties `u:physicalpart` and `u:partOf` are analogous to `dul:hasPart` and `dul:partOf` respectively, and represent the physical composition of an entity. These properties are transitive: if `[A] u:part [B]`, and `[B] u:part [C]`, it is inferred that `[A] u:part [C]`.
- `u:logicalPart` and `u:logicalPartOf` represent logical composition, and are not transitive. They are used to mitigate the transitivity of `u:part` inferring that logically composed entities share each others' physical parts. These properties are similar `dul:hasConstituent` and `dul:isConstituentOf`, and are necessary because RaCoOn does not distinguish between entities identified by function and those identified by spatial extent<sup>1</sup>.

## EXAMPLE

Figure 5.5 shows the composition of a points machine from the perspective of its functional and physical components.

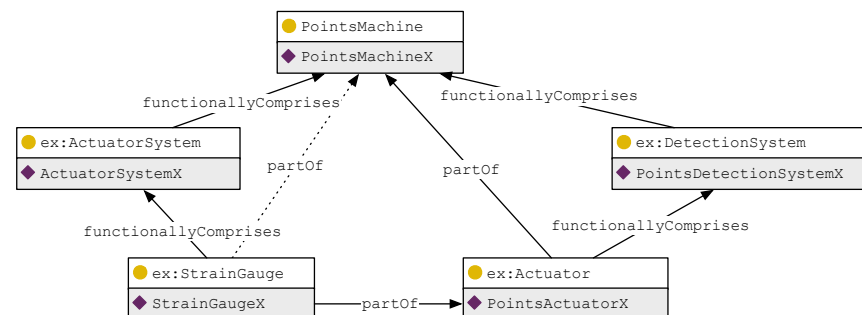


Figure 5.5: Example of Composition Design Pattern Showing High Level Points Machine Components

The patterns shown above are the key ontology design characteristics of the RaCoOn upper ontology. The patterns provided by this

<sup>1</sup> or, as described by West [226], “things you can kick”

ontology are designed to be simple, pragmatic, and lightweight, such that adoption of the ontology is easy and concepts can be extended where necessary. In the following section, a railway domain model that builds on the high level concepts presented here will be described.

#### 5.4 THE RAIL CORE ONTOLOGY

The RaCoOn core ontology represents key railway concepts elicited from subject experts and existing data models. It seeks to represent knowledge that is commonly used across the railway domain, summarised by the following design questions:

- How can I represent the fundamental concepts of any set of railway industry data examined in [Chapter 4](#)?
- What common vocabulary should be represented in order to ease data exchange in infrastructure-based railway applications and what interactions exist between these concepts?

From these high level questions and the use cases discussed in [Chapter 4](#), several more specific aims can be stated to shape the key parts of the vocabulary:

- How is track topology and geography represented? What network information in existing railway data models should be included in a high level domain model?
- How can the capabilities and characteristics of railway assets be modelled such that applications can infer class membership based on known information?
- What extension points can be provided to allow easier reuse and development of subdomain ontologies?

This first parts of this section describes a number of design patterns implemented in the RaCoOn core ontology to address these questions and provide ways of modelling common railway concepts. The final section ([Section 5.4.7](#)) also describes and documents a process of curating and utilising a set of existing vocabulary for use within the ontology from RailML [177], as recommended in [Chapter 4](#).

##### 5.4.1 *Subdomains and Terminology*

In line with consensus from stakeholder workshops and from existing data models, the RaCoOn rail domain ontology provides a set of

meta-classes based on railway subdomains. This improves readability and documentation of the ontology, provides semantics for property restrictions, and creates explicit extension points for subdomain ontologies. In cases where concepts span multiple subdomains, multiple inheritance is used to assert their membership of both superclasses. The subdomains provided in the rail domain ontology are as follows:

- The `rcn:InfrastructureConcept` class contains the set of all possible railway infrastructure-related terms. The scope of this class is consistent with the scope of the RailML infrastructure subschema, and can be thought of as the class of all things that are, or are intended to be statically positioned assets. Examples include railway stations, tracks, and signals.
- `rcn:RollingstockConcept` represents the class of railway vehicle concepts. Whilst not extensively defined in the core ontology, concepts in this class include different representations of trains and vehicles based on formation, physical or functional role, or owner.
- `rcn:InformationConcept` refers to the class of railway-related information assets such as timetables, tickets, and standards documents.

All of these concepts inherit from `rcn:RailwayDomainConcept`, and it is anticipated that future applications will extend this conceptualisation into other subdomains, with each new module specialising one particular class. For example, the core ontology makes no explicit mention of timetabling or signalling concepts, although these would make valid subdomain additions to the core ontology.

#### 5.4.2 *Local Naming Pattern*

It is recognised that many organisations and groups use specific dialects to refer to railway entities, and represent certain concepts with different terms to those formalised by the ontology. For example, the term ‘train’ has one meaning to a maintenance engineer, and another to a signaller. In order to build applications around this domain ontology, a method for representing different dialects to different parties is required, and may be applied for different companies or disciplines. The set of competency questions set out that must be met by this approach is as follows:

- What descriptions are available for Concept X?

- What is the description of Concept  $X$  in Dialect  $Y$ ?
- What dialects are present in the knowledge base?
- What items are described in Dialect  $Y$ ?
- What is the generic label for Concept  $X$ ?

## SOLUTION

The ternary relations pattern is used to create a `doc:EntityDescription` class, which relates a term to a `doc:dialectLabel` and a `doc:dialect`, which are the preferred term literal and `u:Dialect` individual respectively. This pattern is intended for use for both A-box and T-box terms.

## EXAMPLE

In [Figure 5.6](#), the `rcn:TrainConsist` entity is assigned two different labels for maintenance and signalling applications.

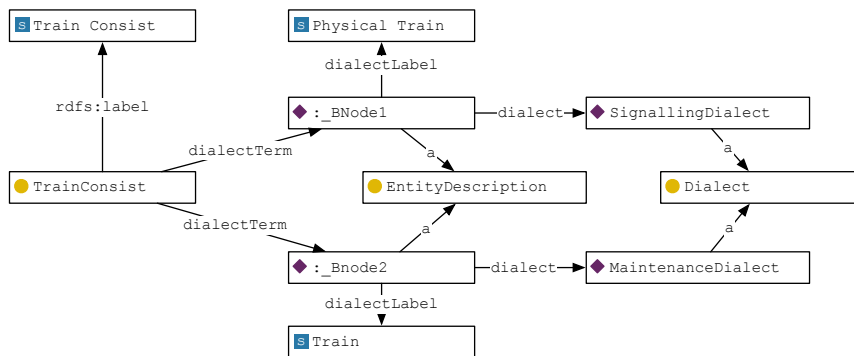


Figure 5.6: Example Showing Local Naming Design Pattern

## DISCUSSION

This method of representing different terms and dialects is verbose when compared with other methods. One alternative, for example, is to exploit RDF literal language tags to encode different dialogs. Since it is expected that dialect terms will usually be applied to (relatively small) T-box concepts only, the increase in verbosity is seen to be worth the additional semantic information and provenance it is capable of representing.

### 5.4.3 *Representing Asset Capabilities and Characteristics*

Railway assets have a set of domain-specific traits and capabilities that often require representation in a data model. These include concepts such as traction characteristics, conformance to standards, and supported communications systems. In many OWL models, object properties are used for this purpose: new properties are created for each new type of characteristic and used to assert information about entities. In RaCoOn, it is expected that a plethora of characteristics will be present in instance data, and an approach that provides finer control over the relationships between characteristics is required. The design question this section seeks to answer is:

“How can we use OWL inference to represent the capabilities of railway concepts and assets?”

To answer it, the following competency questions outline the requirements to be addressed:

- What characteristics does track Section *X* possess?
- What type(s) of electrification does track Section *X* possess?
- Is track Section *X* ERTMS compatible?
- What characteristic changes are present on the track along Section *X*, Section *Y*, and Section *Z*?
- What type(s) of electrification are there?

#### SOLUTION

A class, `rcn:RailwayCharacteristic` was defined, and used to link concepts with their characteristics. In this way, defining conformance to a standard or particular capability is asserted through either the `u-characteristic` property or a more specific subclass of it. Additionally, classes whose identity criteria depend upon a set capability can define `owl:EquivalentClass` axioms through this relation to create the necessary and sufficient conditions for class membership needed, as shown in the example below.

## EXAMPLE

The following DL snippet shows how the capability pattern could be used to infer the set of ERTMS-compatible entities, and the set of radio-protected track:

$$\begin{aligned} \text{ERTMSCompliantThing} &\equiv \top \sqcap \exists \text{characteristic} . \text{ERTMSCapability} \\ \text{ProtectedTrack} &\equiv \text{TrackSection} \sqcap \exists \text{characteristic} . \text{TrainProtectCapability} \end{aligned}$$

In [Figure 5.7](#), a piece of track with an asserted `rcn:ERTMSL2Capability` is shown. As a consequence of this, OWL axioms can infer membership of the `ex:ERTMSCompliantThing` class as well as the `ex:-ProtectedTrack` class.

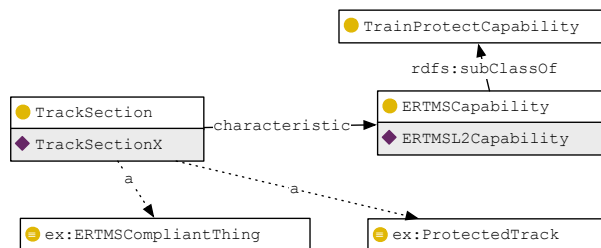


Figure 5.7: Design Pattern Showing Assertion of ERTMS Capability on Track Section and Class Inference

## DISCUSSION

The modelling of characteristics as OWL classes rather than object properties allows more detailed knowledge to be expressed over the characteristics themselves. For example, it is possible to define an individual named `DieselAndElectricTractionCharacteristic`, which belongs to both the electric and diesel traction classes. More specific properties can be created when required in applications, through sub properties of `u:capability`. Relationships of this kind asserted through `rcn:inheritedCapability` will be inherited by components of the original high level concept in OWL DL.

5.4.3.1 *Changing Capabilities*

The capabilities of railway track vary depending on position, and the majority of data models reviewed in [Chapter 3](#) provide some way of expressing how these capabilities change over the length of a railway network or line. All take the same approach scale: assuming linear positioning along a track and a characteristic  $X$ , a list of points along

a track is enumerated with values for  $X$  at every point at which the value of the characteristic changes—for example electrification, gradient, or signalling system. The characteristic is maintained until the next point is reached assuming direction of travel, and applications using these models implement sufficient logic to infer the capabilities of a piece of track by interpolating.

Whilst this approach is structurally simple, it is not consistent with the view of the world in the RaCoOn model. Instead, it is more appropriate to define an area of track and represent the capabilities of that particular section, as described in [Section 5.4.4.2](#). Where capabilities change, a new network node should be defined, such that the track is modelled in enough detail to assert capabilities on.

It is, however, recognised that this method is not always optimal. Assertion of the same characteristics many times over a dense graph may result in unnecessarily large models, and so a `rcn:CharacteristicChange` concept is also provided. This concept links an initial capability with a changed capability, and is associated with a route node through the `rcn:characteristicChange` property. In this way, only changing characteristics are represented, which could then be reconstructed by a tool that parses the network. `rcn:CharacteristicChange` entities can additionally add information about the change itself (such as insulated areas in the case of electrification changes), and may be used in addition to the standard characteristic representation pattern, as shown in [Figure 5.8](#).

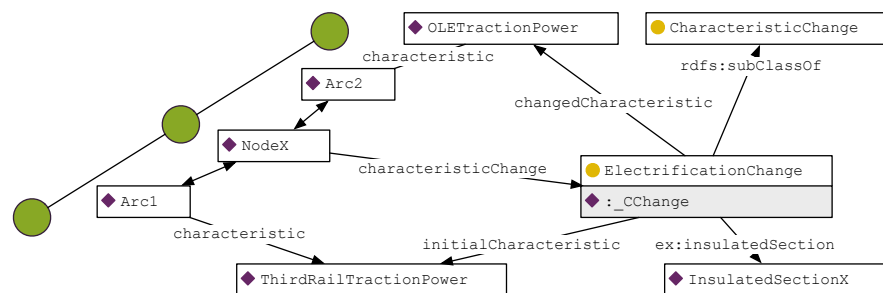


Figure 5.8: Example Showing Track Characteristic Change Design Pattern

#### 5.4.4 Geographical Positioning and Location

Much of the data within the established scope for the RaCoOn ontology is strongly associated with physical assets and infrastructure across the railway. RaCoOn takes cues from other railway and transportation models, and provides a geography-based perspective with

which to represent assets. Existing approaches include those taken by models described in [Chapter 3](#):

- RailML, SDEF, Track Maps, and other signalling, simulation and maintenance tools [77, 173] normalise the position railway tracks and locate assets relative to them. This approach has historically been used by the railways, with signals and track layouts in the UK represented by *chainage*, or ‘miles from London’.
- Geographical Information System (GIS) systems represent information using three-dimensional cartesian co-ordinates based on a known reference system such as WGS84<sup>2</sup> or OSGB36<sup>3</sup>. This method is commonly used for a variety of use cases, including route planning/evaluation [27], asset management [90] and for interoperability with other sources. OpenStreetMap<sup>4</sup> and Google Maps<sup>5</sup> both present railway information in this way.

#### 5.4.4.1 *Modelling Railway Networks*

One of the most important components of the RaCoOn core ontology is its conceptualisation of railway infrastructure, and in particular, the railway network itself. The current and legacy models reviewed in this thesis are all (with the exception of the Rail Functional Architecture) modelled around this infrastructure, and as such it is of great importance to model it in a way that aids data exchange. From the literature discussed in [Section 3.3](#), commonalities can be found between approaches to representing railway network topologies. These commonalities are as follows:

- Railway networks are modelled as graphs, with a set of nodes and edges.
- All but one of the models provided several views on network data by means of aggregation—encoding railway networks as a series of layers at different levels of detail, with detailed components subsumed by more generalised routes or areas. Most considered two or three discrete levels: a *track* level, a *network* level.

The infrastructure topology model in RaCoOn takes a similar approach, based on design questions derived from the top level requirement to provide integration with existing models:

<sup>2</sup> WGS84 is a positioning standard used by Global Positioning Systems

<sup>3</sup> OSGB36 is a reference grid used by Ordnance Survey in the United Kingdom

<sup>4</sup> <http://wiki.openstreetmap.org/wiki/Railways>

<sup>5</sup> <http://www.google.co.uk/transit>



1. How can the network topology of the railway be represented, along with its navigability?
2. How can the track-centric position of railway assets be described unambiguously?
3. How can absolute (geographic) positioning of railway assets be described?

These questions are addressed with a series of ontology design patterns that follow in subsequent sections.

#### 5.4.4.2 Representing Basic Track Topology

In order to answer design question #1, methods for representing railway topology must be examined. Topology can be naturally represented as a graph at many levels of abstraction, owing to its physical manifestation as nodes (stations and junctions) and edges (lines or tracks). Figure 5.9 shows such a fictional railway network, with four destinations *A* to *D* represented as nodes or vertices, and three routes connecting them shown as lines. A valid train travelling through the network may traverse the graph via [*C*, *B*, *A*, *D*] to provide a service to those four stations.

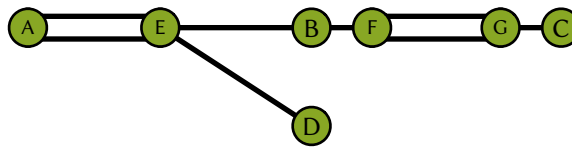


Figure 5.9: Example of 'Network Level' Railway Route Graph

For representing timetabling or journey planner information, this level of detail may be sufficient: a timetable can specify the times at which a train should arrive and depart a station, and a journey planning application can suggest routes based on this data. For other purposes, a more detailed view of the network may be required, as in Figure 5.10.

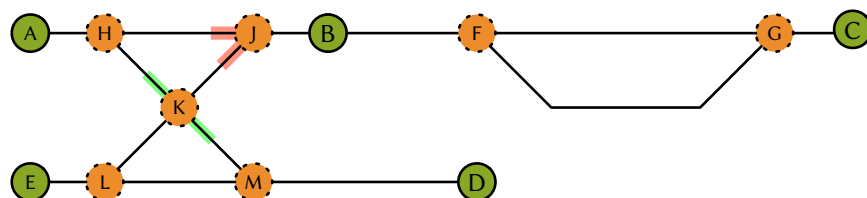


Figure 5.10: Example of 'Route Level' Railway Route Graph

The information represented here includes not only the paths from one station to another, but the choice of railway tracks and junctions that are used to do so. Elements shown in orange with dashed outlines are junctions: switches and crossings where multiple physical railway lines intersect. At this level, signallers can assign routes between trains, and occupied routes can be associated with signals and interlockings. Route  $[H, J, K]$  (highlighted in red) is an example of an *invalid route*—a route that cannot physically be taken by a train due to the constraints of the switch and vehicle. Route  $[H, K, M]$ , shown highlighted in green, is an example of a *valid route*—one that can be traversed by a train.

Thus, the basic requirements for representation of track topology as required by design question #1 are provided by a graph representation of the railway network, and are illustrated in these CQs:

- Which other topology nodes is Node  $X$  connected to?
- What are the track arcs downstream of Node  $X$ ?
- What are the track arcs upstream of Node  $X$ ?
- What is the start node for track Arc  $Y$ ?
- What is the end node for track Arc  $Y$ ?

These basic CQs are met by descriptions in the following section.

#### GRAPHS OF GRAPHS

To represent a basic, directed graph in OWL is simple, as assertions form a graph themselves. The following DL expression shows three destination nodes linked by an arc relation:

```
arc(Destination_A, Destination_B),
arc(Destination_B, Destination_C)
```

Owing to the semantics of the OWL language, this approach only allows assertion of properties on track nodes such as stations, and does not allow the representation of characteristics relating to the arcs (tracks). Modelling tracks as nodes instead avoids this problem but prevents representation of information about stations and junctions. To combat this, and to aid understandability and self-documentation,

a form of reification<sup>6</sup> is employed, with both nodes and arcs represented as `owl:Individuals` of differing types. This approach is similar to the ‘Qualified Relation’ pattern described in Dodds and Davis [51] and concepts used to this end are shown in Table 5.2.

Although railway networks are inherently *undirected* in that there is no limitation on the direction of travel, it is desirable to represent railway networks with an arbitrary directivity in RaCoOn to allow greater flexibility in positioning of assets and the expression of navigability. IN RaCoOn, this is represented by introducing several sub-properties which allow the assertion of direction between along graph nodes and arcs:

```
startingArc ⊆ arc,
endingArc ⊆ arc,
startNode ⊆ node,
endNode ⊆ node
```

It is suggested that this directivity corresponds to the track location positioning system used, as shown in Figure 5.11. The knowledge represented in the original graph above now becomes:

```
Destination_A : Node
Destination_B : Node
Destination_C : Node
Arc_X : Arc
Arc_Y : Arc
startNode(Arc_X, Destination_A)
endNode(Arc_X, Destination_B)
startNode(Arc_Y, Destination_B)
endNode(Arc_Y, Destination_C)
```

#### RESTRICTIONS ON ROUTE GRAPH CONCEPTS

The RaCoOn ontology places few restrictions on network layouts, but does assert some axioms about the route graph to enable inferences

<sup>6</sup> for an explanation of reification, see Section 2.6.1.3

Table 5.2: Basic Track Topology Design Pattern OWL Constructs

Concept	Description
● <code>rcn:-RouteGraph</code>	The class of all individuals that create a railway network graph
● <code>rcn:RouteNode</code>	The class of nodes (stations, junctions, termini)
● <code>rcn:RouteArc</code>	The class of arcs (paths, tracks, lines of way)
■ <code>rcn:node</code>	Connects an arc to a node
■ <code>rcn:arc</code>	The inverse property of <code>rcn:node</code> , and connects a node with an arc.

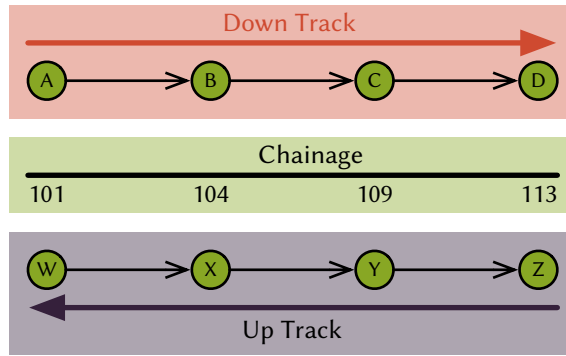


Figure 5.11: Example of Arbitrary Graph Directivity in Track Topology

about nodes and arcs. In addition to domain and range assertions, these are as follows:

$$\begin{aligned}
 \text{arc} &\equiv \text{node}^-, \\
 \text{startingArc} &\equiv \text{startNode}^-, \\
 \text{endingArc} &\equiv \text{endNode}^-, \\
 \text{RouteArc} &\sqsubseteq 1 \text{ startNode} \sqcap 1 \text{ endNode} \sqcap \\
 &\quad (\forall \text{ startNode}(\text{RouteNode})) \\
 &\quad \sqcap (\forall \text{ endNode}(\text{RouteNode}))
 \end{aligned}$$

- $\text{rcn}:\text{RouteArc}$  classes have exactly one  $\text{rcn}:\text{startNode}$  and exactly one  $\text{rcn}:\text{endNode}$  property.
- $\text{rcn}:\text{node}$  and  $\text{rcn}:\text{arc}$  are inverse properties, as are their children. Thus, an arc  $V$  having a  $\text{rcn}:\text{startNode}$   $X$  infers that  $X$  has  $\text{rcn}:\text{startingArc}$   $V$ . These properties have domain and range  $\text{rcn}:\text{RouteArc}$  and  $\text{rcn}:\text{RouteNode}$  accordingly.
- $\text{rcn}:\text{RouteNode}$  has no cardinality restraints—a node may be joined to as many vertices as necessary.

By asserting inverse properties, this pattern then also allows the inference of:

$$\begin{aligned}
 &\text{arc}(\text{Destination\_A}, \text{Arc\_X}), \\
 &\text{arc}(\text{Destination\_A}, \text{Arc\_Y}) \\
 &\dots
 \end{aligned}$$

#### 5.4.4.3 *Representing Different Levels of Abstraction*

Having established how basic railway topology graphs are represented, the following design questions can now be addressed:

“How can we provide views on a railway network at differing levels of abstraction/detail? How can we make data available at one level useful to applications that view the network at another?”

As stated at the start of this section, many infrastructure design use cases require the representation of a railway network at varying levels of detail. This is explicitly encoded in existing models, where multiple interconnected layers provide views ranging from broad country-level route networks to detailed representations of track sections and joints. Existing models require that characteristics are represented in their entirety at every level of abstraction, but this results in data duplication and inhibits compatibility between tools. In RaCoOn, we provide a way to infer track characteristics at a particular level of detail, based on what other information is available in the ontology. The first set of competency questions required to model these capabilities are as follows:

- What levels of track detail are presented in this data set?
- What level(s) of detail does Abstraction level  $X$  generalise?
- What level(s) of detail does Abstraction level  $X$  specify further?
- At what level of detail does Node  $X$  present topological information?
- What other infrastructure does Node  $X$  encompass and generalise?
- Is Node  $X$  abstracted by any other objects?
- What characteristics does Node  $X$  possess?
- What characteristics does Node  $X$  inherit?

#### SOLUTION

To allow extensibility, track and arcs are defined according to subclasses of `rcn:RouteArc` and `rcn:RouteNode`, and bound to a defined level of abstraction through the `rcn:trackAbstraction` property. Subclasses instantiated in the vocabulary model include ‘LineLevel’

‘LineDetail’ ‘NetworkLevel’ and ‘ELRLevel’<sup>7</sup>. Each is defined with the following restriction:

$$\text{SomeDetailLevelNode} \sqsubseteq \text{RouteNodeSomeDetailLevelNode} \equiv \exists \text{trackAbstraction. SomeDetailLevel}$$

As such, a set of classes for routes and nodes are created representing different levels of track modelling detailed, which creates a set of distinct modeling levels. Two new object properties, `rcn:comprisesRouteElement` and its inverse, link more generalised concepts to more specific ones. These properties are defined as transitive and irreflexive, such that links between adjacent levels of detail also hold transitively up the generalisation hierarchy. Finally, a `rcn:inheritedCharacteristic` property is defined with a property chain axiom that associates low level concepts with inheritable characteristics asserted on higher level elements.

#### EXAMPLE

An example of this pattern is shown in [Figure 5.12](#).

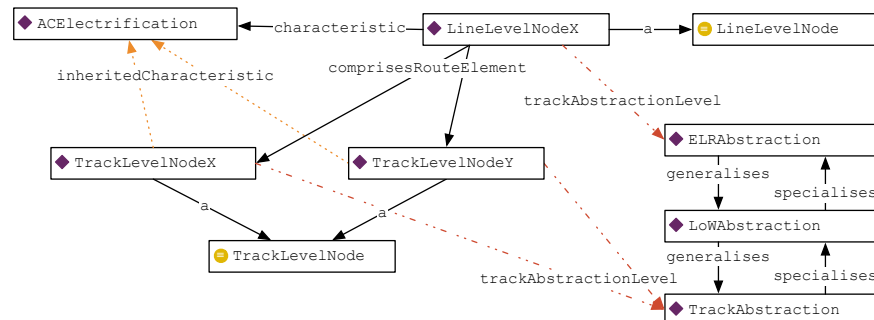


Figure 5.12: Example Showing Track Inheritance Design Pattern

#### DISCUSSION

This pattern relies heavily on OWL DL reasoning to work. As such, its applications may be limited to small scenarios or to relatively unchanging situations where inferences can be materialised in advance. It is also recognised that the assertion of `rcn:inheritableProperty` is not intuitive, but this is necessary to distinguish between properties that can be inherited at lower levels (such as electrification type)

<sup>7</sup> Across the UK Rail Network, an Engineers’ Line Reference is a unique Identifier (ID) assigned to a particular route or part of route

and those that cannot (such as weight). Given a class hierarchy with this knowledge, the assertion of `rcn:inheritableProperty` could be automated using rule-based reasoning instead.

#### 5.4.4.4 *Representing Linear Positions*

So far, only the track topology has been represented. To represent knowledge according to design question #2, the following additional CQs must be answerable:

- Which part(s) of a network is Entity X located on?
- What is the position of Entity X on track Arc Y relative to Node Z?
- What is the linear position (chainage) of Entity X

To encode this knowledge, an approach similar to that taken in the UIC RailTopoModel [114] is employed, allowing two different representations:

- Track elements or assets assert their linear position relative to a global/line reference. Given a track arc, elements use the `rcn:relativePosition` property to associate with a relative position individual—a measurement with an associated element reference. This relation simply infers the same semantics as are present in UK TrackMaps: that an entity has a position some length away from a reference point.
- Railway assets may position themselves relative to an arc itself. In some cases it may be suitable to assert the position of an asset relative to the arc that it is placed on. In this case, a proportion or absolute position is represented relative to the start node of an arc, as shown in Figure 5.13. Related components in the ontology are given in Table 5.3.

For both methods, RaCoOn provides the content pattern illustrated in Figure 5.13.

Elements represented by a linear position (such as a chainage) specify a `rcn:LinearPosition` individual to record their position from a start point. They may optionally assert `rcn:locatedOn` to convey that these components are located on a particular track section. The semantics of `u:Measurement` provide a way of expressing the correct units and scale. This approach is also applicable to track nodes themselves, who can assert the same `rcn:LinearPosition` to establish



Table 5.3: Linear Track Positioning OWL Constructs

Concept	Description
● <code>rcn:LinearPosition</code>	Subclass of <code>u:Measurement</code> providing a length and position reference
● <code>rcn:-RelativeTrackPosition</code>	Subclass of <code>rcn:Measurement</code> equivalent to locating component on track
● <code>rcn:PositionReference</code>	The class of all things that act as position references
● <code>rcn:position</code>	The position of the entity relative to
■ <code>rcn:locatedOn</code>	The spatial relationship between an entity and another on which it is located
■ <code>rcn:positionReference</code>	The (optional) 'shortcut' element that the entity is positioned on

their chainage.

As `rcn:position` is transitive, track arcs which assert their `rcn:position` as another entity (such as an `rcn:RouteArc`) inherit their position.

To place elements relative to a track arc, they may assert `rcn:RelativeTrackPosition` relations instead, which foregoes the need to explicitly state that they are `rcn:locatedOn` a component.

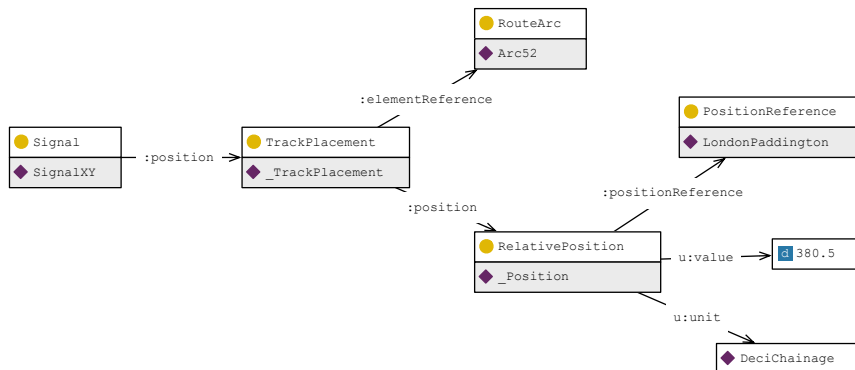


Figure 5.13: Track Element Positioning Design Pattern

This method has some caveats, and was designed to provide the most intuitive encoding of track position in most basic circumstances. It does not allow the assertion of two different positions for an asset (as may be the case on signal bridge shared between railway lines), as it correlates the asset with its position, which may not be the case. Patterns to express this differently are discussed in [Section 5.4.3](#).

#### 5.4.4.5 Location Design and Spatial Relations

Design question #3 concerns representing the absolute geographic locations of assets, as may be required by a Geographical Information System (GIS). The set of competency questions that should be answered by the patterns presented in this section is as follows:

- What is the absolute location `Location Y` of Entity `X`, in latitude, longitude, and altitude?
- What units are the attributes of `Location Y` expressed in?
- What measurement system is used to represent `Location Y`?

Geographic representations of network entities are somewhat easier to represent in OWL than the linear positioning systems shown

above, as several well-defined conceptualisations of 3D space already exist. In the UK, the commonly used reference formats for these are WGS84, a geodesic reference mapping used in Global Positioning System (GPS) systems, and OSGB36, which is used by Ordnance Survey in the United Kingdom [153].

In RaCoOn, the locations of entities are represented through the `u:location` property, and the semantics of the location imparted by the `rdf:type` of its object. This pattern allows any geographic reference system to be used with preserved semantics, such that conversions between them can be carried out by external tools if information about these co-ordinate systems is known.

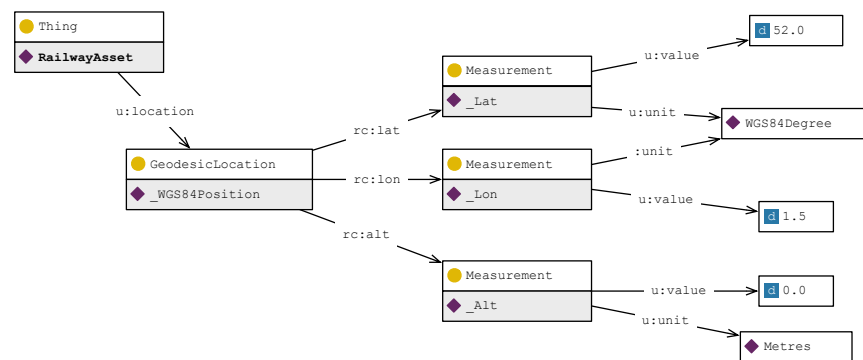


Figure 5.14: Example WGS84 Position Represented Using Location Pattern

This pattern draws upon the W3C Geo RDF vocabulary [25], but provides a more expressive representation in OWL that accounts for differing measurement units and datum. Core concepts are shown in Table 5.4.

Table 5.4: Geodesic Location Representation OWL Constructs

Concept	Description
● <code>rcn:-</code> GeodesicLocation	Similar to <code>geo:Point</code> : defines a point location
⊕ <code>rcn:-</code> WGS84Location	Class of locations that match WGS84 characteristics
▣ <code>u:location</code>	Upper level property to link to location representation
▣ <code>rcn:lat</code>	Measurement convenience property to relate a point to its latitude
▣ <code>rcn:lon</code>	Convenience property for longitude
▣ <code>rcn:alt</code>	Property to define height from sea level in metres

These measurement properties are subproperties of `u:measurement`, and link a location point with values for each attribute. The following DL shows these subproperties, and provides a definition for a valid WGS84 measurement:

$$\begin{aligned} \text{lat} &\sqsubseteq \text{measurement}, \\ \text{lon} &\sqsubseteq \text{measurement}, \\ \text{alt} &\sqsubseteq \text{measurement}, \\ \text{WGS84Measurement} &\equiv \text{Measurement} \\ &\sqcap (\forall \text{lat.Measurement} (\forall \text{unit.WGS84Degree})) \\ &\sqcap (\forall \text{lon.Measurement} (\forall \text{unit.WGS84Degree})) \\ &\sqcap (\forall \text{alt.Measurement} (\forall \text{unit.Metre})) \\ &\sqcap (\exists \text{lat.Measurement}) \sqcap (\exists \text{lon.Measurement}) \end{aligned}$$

The unit `rcn:WGS84Degree` captures the knowledge that measurements are referenced from the WGS84 datum, without modelling the intricacies of the geodesic system explicitly. Note that there is no restriction on altitude measurements, so measurements with no altitude are still considered to be valid WGS84 measurements.

#### COMPATIBILITY WITH W3C GEO ONTOLOGY

Interoperability with the widely-used W3C Geo ontology [25] is possible through a simple mapping, expressed in Listing 5.4.

```
CONSTRUCT {?point a geo:Point ;
  geo:lat ?lat .
  geo:lon ?lon .
  geo:alt ?alt .
} WHERE {
  ?point a rc:WGS84Location ;
  rc:lat [u:value ?lat ] ;
  rc:lon [u:value ?lon ] ;
  rc:alt [u:value ?alt ] ;
}
```

Listing 5.4: Example SPARQL Location Mapping Between RaCoOn and the W3C Geo Ontology

The mappings required to *import* data in W3C Geo representation to a knowledge base using the RaCoOn ontologies additionally require the assertion of new `u:Measurement` entities with appropriate

u:units and values. This could be achieved using another similar SPARQL CONSTRUCT query to that shown in [Listing 5.4](#).

#### 5.4.5 Representing Diagrammatic Network Layouts

In addition to the representation of ‘real world’ positioning systems, Railway network models also often encode one or several non-geographic ‘presentation’ layouts, to aid cross-organisation working, which it is important for a domain model to represent. Thus, a new design question is posed:

“How can we support diagrammatic views of railway network topologies?”

In [Figure 5.15](#), an example signalling layout application is shown, using data from the Invensys Layout Description Language (LDL) data model. Although this information can be considered to be more specialist in scope than other areas of the RaCoOn model, it was included owing to its presence across many other existing rail data models, and its perceived importance in cross-application working.

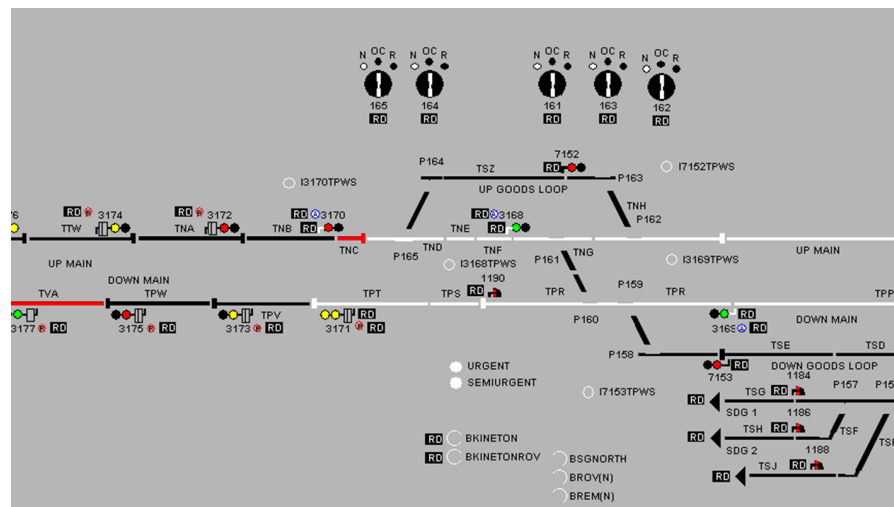


Figure 5.15: Screenshot From Invensys Westlock Interlocking Simulator

The competency questions arising from the need to provide this representation are as follows:

- What is the presentation position of Entity X?
- What type of units is the presentation position of Entity X given in?
- Does the presentation position of Entity X have a reference point to draw from?

## SOLUTION

In SDEF, RailML, Invensys LDL and Railsys, diagrammatic locations are notated as a set of co-ordinates with no explicit reference system. RaCoOn therefore provides a `doc:position` property that relates elements to abstract positioning characteristics as part of the rail core module. A similar pattern to as used with geographic co-ordinates is employed: A class of `doc:ViewPosition` is defined as are properties `doc:xPos`, `doc:yPos`, and `doc:zPos`, each of which relates a `doc:ViewPosition` individual to a `u:Measurement` class. Optionally, a `rcn:positionReference` relationship can provide an origin.

Whilst this pattern does not explicitly cover the representation of the same entities from multiple viewpoints (such as signals that are shown in many different application displays), this could easily be accommodated by subclassing `doc:ViewPosition` in an extension ontology.

## EXAMPLE

A `rcn:RouteNode` positioned according to a unit-less cartesian co-ordinate system with no reference point, shown in [Figure 5.16](#)

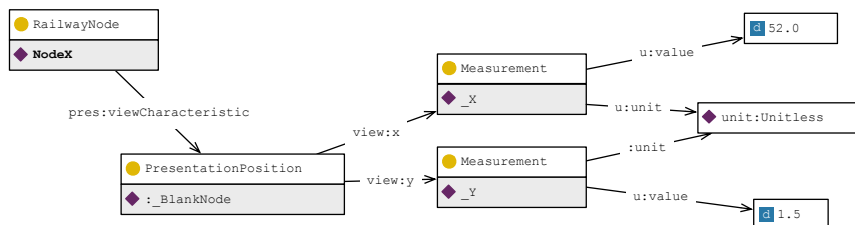


Figure 5.16: Example XY Presentation Position Represented Using Location Pattern

## DISCUSSION

Arbitrary dimensions are either created as `u:Measurement` with `u:unit unit:Unitless`, or created as instances of `u:UnitlessMeasurement`, which makes the same inference:

$$\text{UnitlessMeasurement} \equiv \text{Measurement} \sqcap (\forall \text{unit.} \textit{Unitless})$$

Unitless dimensions by their nature assume some implicit positioning/presentation system, but this is assumed by the task-oriented nature of this pattern.

#### 5.4.6 *Navigability and Routing Across Networks*

The next design question considered relates to the transportation capabilities of a physical railway network:

“How can the physical navigability of a railway graph be represented in OWL”

The route navigability of a railway network is vital to many railway applications such as maintenance, signalling design, and passenger information. Trains can only move physically across junctions in pre-defined ways, and so the ways in which they can traverse the network are restricted. For example, [Figure 5.10](#) highlights in red an invalid route across a railway switch. In order to properly represent the ways in which railway infrastructure can be physically traversed, additional knowledge must be appended to the network representations mentioned so far. Thus, several additional competency questions are presented:

- Is the route from Node X to Node Z through Node Y navigable?
- Is the route from Node X to Node Z through Node Y impossible to navigate?
- Is the route from Arc X to Arc Z through Node Y navigable?
- Is the route from Arc X to Arc Z through Node Y impossible to navigate?

Note that a negative answer to the question ‘is Node X navigable...’ does not imply that a route is impossible to navigate owing to the OWA, so additional competency questions are required as shown.

#### SOLUTION

Navigability is provided to route arcs through the `rcn:navigable` and `rcn:notNavigable` object properties. Navigable routes across a node can be found using the transitive `rcn:nodeOf` property and simple queries.

## EXAMPLE

In Figure 5.17, Node G is a switch. Arcs X, Y and Z represent tracks connecting with the switch. Routes [XY] and [YX] are ‘normal’ routes, [YZ] and [ZY] are diverging routes, whilst routes [XZ] and [ZX] are not navigable. The navigabilities of these routes are asserted by `ex:-ArcY rcn:navigable ex:ArcZ, ex:ArcX` (and their inverses), and by `ex:ArcZ rcn:notNavigable ex:ArcX` (and its inverse).

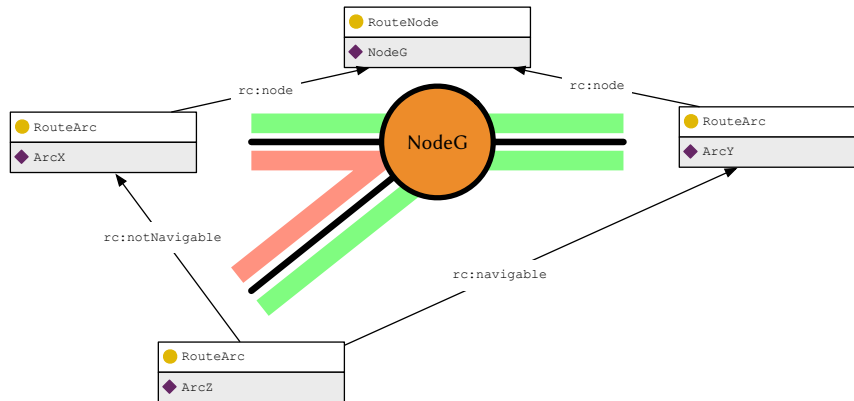


Figure 5.17: Example Representation of Network Navigability Across Routes

## DISCUSSION

Whilst this approach appears odd compared to asserting the navigability of a route on the nodes themselves (a points machine will always have a characteristic navigability), it is intuitively simpler than other approaches, and the chain of `rcn:navigable` relations across a network creates a directed graph onto which routing algorithms can be directly applied. The two properties are not asserted to be symmetric, as it is feasible in some circumstances that a route cannot be passed in both directions<sup>8</sup>.

## DISADVANTAGES

This approach requires assertion of route navigation onto each route arc (including on straight lines), and in both directions. This can be mitigated to some extent by the inclusion of inference rules to assume that tracks passing through nodes with only two edges are navigable in both directions.

<sup>8</sup> Such an example can be found in ‘sprung’ railway switches, which are designed to allow the passage of a vehicle in the ‘normal’ direction even when the points are set against it



#### 5.4.7 *Re-engineering Knowledge from RailML*

A large part of the vocabulary and semantics present in the RaCoOn core ontology are re-engineered from existing information resources, rather than being explicitly given by human domain experts. This section examines how knowledge from one such resource was mapped into the ontology using the technique shown in [Chapter 4](#).

After undertaking a survey of existing knowledge for re-use in the RaCoOn ontologies, RailML [177] was identified as an existing resource that provided an extensive and accurate model of railway infrastructure and operations. Consequently, parts of this model were re-engineered into OWL, principally to grow the RaCoOn core ontology's vocabulary and reinforce the semantics of existing concepts. RailML comprises five components: common (CO), infrastructure (IS), timetable (TT), rolling stock (RS), and interlocking (IL). In the RaCoOn ontologies, knowledge was elicited from the common, infrastructure, and rolling stock subschemas, corresponding to the areas of interest identified in [Chapter 4](#). [Figure 5.18](#) shows an overview of RailML subschemas and shows some example concepts from each.

##### 5.4.7.1 *RailML Structure and Automated Mappings*

When considering knowledge extraction from existing non-ontological resources, it is sometimes possible to use automated systems to extract terms and semantics from existing models and transform them into ontologies, with or without assistance. Such automatic systems were considered in mapping RailML to RaCoON but not ultimately chosen for several reasons. These reasons include:

1. Model expressivity is confined to the original semantics of XML. More intuitive representations of elements may be possible by human interpretation of XML schemas, but automated tools have no mechanism for detecting and modelling such interpretations.
2. Semantic work-arounds encoded as XML are mis-mapped into OWL. For instance, RailML utilises high level group elements to structurally contain different parts of the model, which is not necessary in OWL. These elements are mis-mapped as OWL classes.
3. Partial mappings are difficult. The scope of RailML exceeds the scope of RaCoOn, and automated tools do not provide a method for mapping only concepts that are in scope.

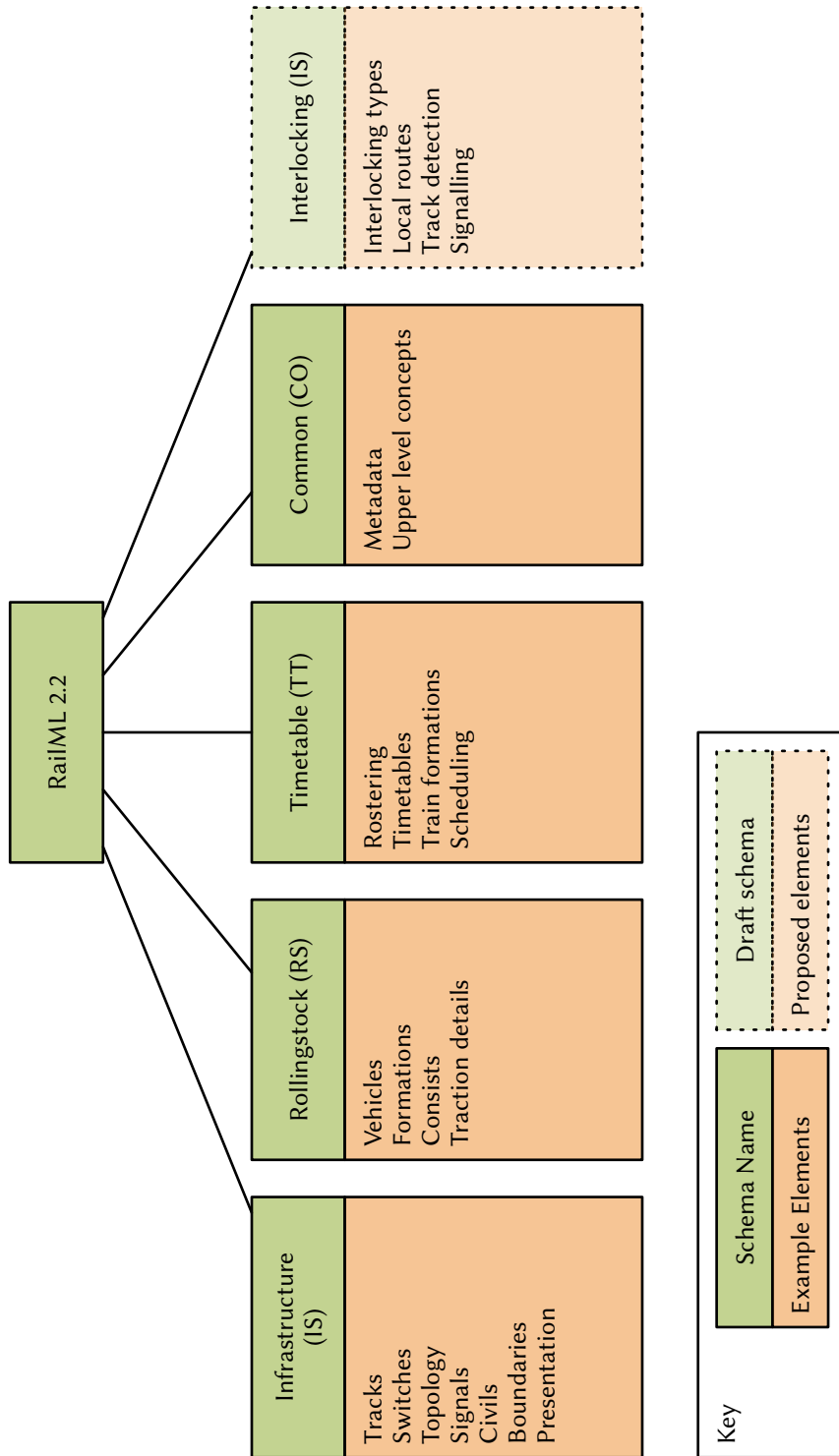


Figure 5.18: Overview of RailML Subschemas and example elements

Whilst such techniques work well for simple schemas with semantics that align directly with a target conceptualisation [22, 61], it was found that better results could be obtained by using a manual schema transformation process. For reference, an automated mapping of the entire RailML schema using the XSD2OWL tool García [73] is available online<sup>9</sup>, and shows how RailML XSD semantics mapped directly to OWL appear.

#### 5.4.7.2 *Manual Mapping Process*

A manual re-engineering process was undertaken to transform RailML concepts into the RaCoOn ontology. This process allows the XML schema's explicit semantics (found in its structure and attributes) to be combined with its implicit semantics (found both in the model's documentation and in external domain knowledge) to form a better representation of domain knowledge in OWL. The process undertaken is shown in Figure 5.19, and adapts elements of the NeON methodology for non-ontological resource re-use [44].

Key elements of this process are described as follows:

- **Filter terms by scope.** Using RailML documentation, we determined whether the term was inside the core scope of the rail core ontologies. Task or application-specific vocabulary was avoided, and terms thought to be more appropriate in subdomain ontologies were omitted.
- **Analyse term.** The exact semantics implied by RailML and its documentation were determined, and evaluated against existing terms in the ontology, other data models, and domain knowledge.
- **Re-engineering methods:** One of three methods was used to integrate the term with the RaCoON ontology:
  - **Re-use existing pattern.** This method was undertaken where a design pattern already existed for a conceptually similar term. The new term was added according to this pattern
  - **Extend existing pattern.** Where a term could not be directly re-engineered using an existing pattern but could be through extending it, this approach was preferred to establishing new patterns.
  - **Create new pattern.** For concepts that had little or no coverage in the ontology or could not be represented using current patterns, a new pattern was created.

<sup>9</sup> <http://phd.jtutcher.co.uk/examples/railml2owl>

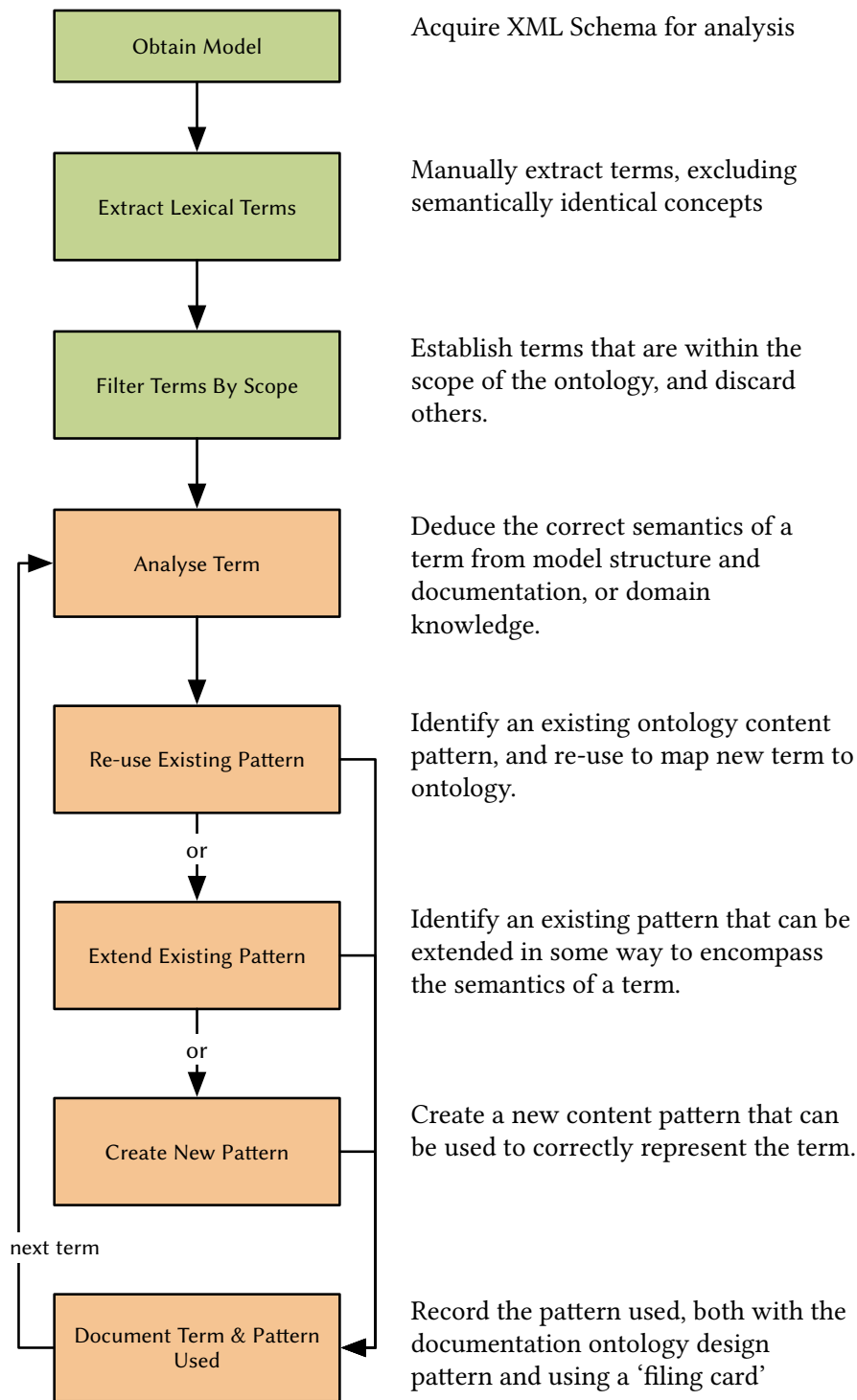


Figure 5.19: Flowchart Showing RailML Re-engineering Workflow

- Following the re-engineering of each term, its method of transformation was documented both in the ontology itself and independently.

#### DIS-AMBIGUATING RAILML CONCEPTS

RailML specifically includes many concepts that are interrelated, in order that intended semantics are contained within the XML schema structure. An example of this is the frequently used pattern of defining both an XSD element and XSD complex type for each item: the element definition states where and how an item should be defined within a document, and the complex type contains its attributes and restrictions:

```

<xs:complexType name="eMileageChanges">
  \rdfe{xs}{sequence}
    <xs:element name="mileageChange"
  ↪ type="rail:tMileageChange" minOccurs="0"
  ↪ maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="tMileageChange">
  \rdfe{xs}{complexContent}
    <!-- content -->
  </xs:complexContent>
</xs:complexType>

```

Listing 5.5: Extract from RailML Infrastructure Model Showing Complementary Elements and Types

The full list of terms drawn from RailML was compiled by aggregating information in these connected elements into one ‘semantic’ element, which was then mapped according to the schema’s content across all similar elements.

#### 5.4.7.3 RailML Transformation Patterns

In addition to a set of content patterns, other ODPs and considerations were made when manually transforming RailML concepts into RaCoOn. These were made in order to better align the transformed knowledge to the other needs of the ontology and to improve understandability, and are shown as follows.

## MAINTAINING MAPPINGS TO RAILML

To encourage re-use, mappings from RailML to RaCoOn were documented within the ontology using an annotation pattern, showing the origin of the mapping in RailML within the RaCoOn core ontology. To create this mapping pattern, some competency questions were considered:

- What semantic match for Term  $X$  exists in RailML?
- What similar RailML term exists for Term  $X$ ?

When mapping another data model into OWL, concepts and relationships are represented differently, and owing to the difference in expressivity between RailML and RaCoOn, no formal mappings between the models are specified. The annotation of terms in the ontology aims to guide users into modelling RailML data correctly using the ontology, and to consider new ways of conceptualising this data in the cases where direct alignment does not exist.

## SOLUTION

Annotation properties `rcn:RailMLExactMatch` and `rcn:RailMLCloseMatch` are provided to take string literal representations of CURIE-formatted RailML terms. As they are annotation properties (with no OWL semantics), they can be asserted on properties as well as classes.

## EXAMPLE

Figure 5.20 shows two entities, `rcn:Signal` and `rcn:Platform`, and their exact or close RailML entity matches.

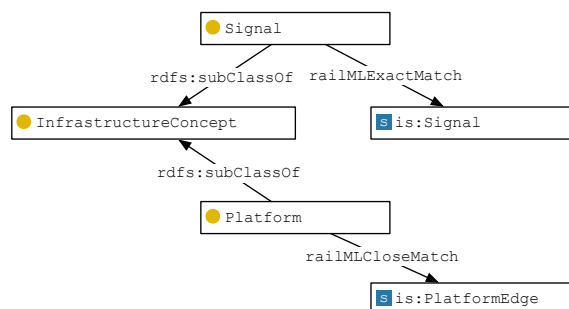


Figure 5.20: RailML Entity Provenance Annotation Pattern

#### 5.4.7.4 *Dealing with Constraints*

RailML is designed as a task-specific data exchange model, and defines set restrictions on data which assures validation for users and applications, but which may not necessarily hold in a wider domain view of the world. For example, RailML requires that a railway junction be defined as one of a particular type of track switch or crossing, which allows applications to make assumptions based on a valid choice for this attribute.

The RaCoOn ontologies are designed to be extensible, and should thus avoid such constraints unless there is certainty of their applicability in all possible circumstances. As such, constraints from RailML are avoided and included as either annotations or minimum cardinality o restrictions (as discussed in [item 5.3.1.2](#)).

Examples of this include the RailML definition of a railway signal, which is asserted to have ‘min o’ cardinality restrictions for `rcn:-sightDistance` and `rcn:interlocking`.

#### OTHER TRANSFORMATION PATTERNS

Several other patterns employed do not warrant full explanations within this thesis, but are briefly overviewed below.

#### RAILML NAMING PATTERN

Where possible and semantic meanings are identical, terms mapped from RailML are given identical names in RaCoOn, excluding any RailML prefix. One example is that `eSignal` and `tSignal` transforming to `rcn:Signal`.

#### RAILML ELEMENT ID PATTERN

RailML issues each element within its model with a unique ID. In RaCoOn these elements are not transformed; instead, individual URIs are considered to be RailML IDs.

#### 5.4.7.5 *RailML-specific Vocabulary and Content Patterns*

The majority of RailML-derived content patterns used in RaCoOn are not overviewed here for brevity; they are documented in the ontology

itself<sup>10</sup> and shown in Section C.1, where a list of RailML vocabulary concepts mapped is also shown. Two key patterns are described here.

#### REPRESENTING GROUPS

Groups of elements are represented readily in RailML through the `xdd:sequence` feature of XML schema. OWL does not represent groups natively, but several RailML concepts, for example `BaliseGroup` and `SignalGroup`, rely upon the grouping of entities for a particular purpose or function. To implement groups, we reuse the Collections Ontology [35] in the following ways:

- The collections ontology (`co:`) is imported into the RaCoOn vocabulary ontology, but with many of its restrictions extracted and placed into the RaCoOn ‘constraints’ ontology to allow it to adhere to OWL RL.
- Grouped concepts such as `rcn:SignalGroup` are defined as equivalent to `co:Collection` containing elements only of type `rcn:Signal`, allowing OWL representations of collections according to that ontology’s definitions. An example is shown in Figure 5.21.

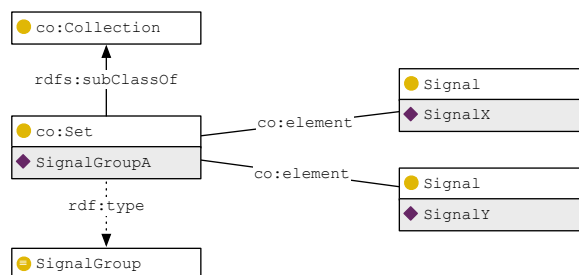


Figure 5.21: Example of a RailML Signal Group, Represented Using the Collections Ontology

#### ROUTE PROFILES

Another recurring theme in RailML was the dependence of characteristics and concepts on ‘routes’—preset paths through a network that are used for signalling. A number of track characteristics depend on these routes, such as speed profiles for trains that vary depending on the path taken through a network. The Collections Ontology is also used to represent these:

<sup>10</sup> <http://purl.org/rail/core/vocab>



- A `rcn:Route` class is created as a subclass of `co:List`, with a constraint that only `rcn:RouteGraph` elements may be associated through the `co:element` predicate. This allows an ordered list of route elements to be represented.
- The characteristic representation design pattern is then used to link speed profiles to routes, through the `rcn:characteristic` property. An example is shown in Figure 5.22.

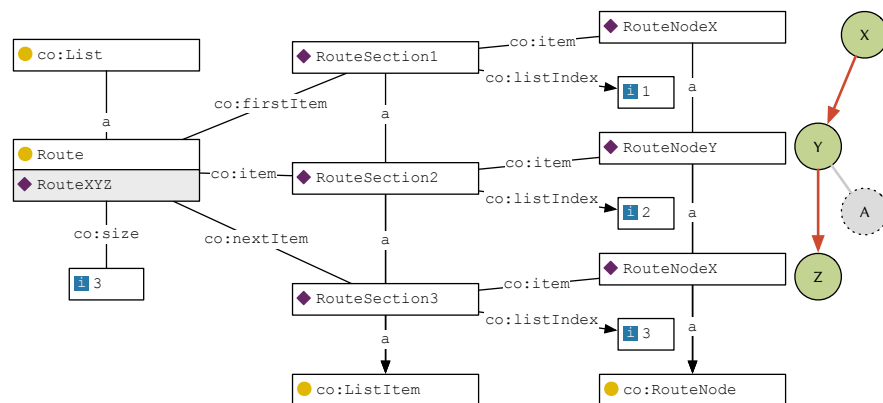


Figure 5.22: Example Showing Route Profile Design Pattern Across Three Route Nodes

## 5.5 RACOON ONTOLOGY EVALUATION

Validation of the RaCoOn core and upper ontologies was performed using the techniques identified in Section 4.6. Firstly, automated techniques were used to establish the structural, syntactic, and logical validity of the ontology as an OWL document. Then, the models were evaluated against a shared conceptualisation elicited from a set of expert workshops. Finally, validation of task suitability and accuracy is established through establishing mappings from and to several existing railway data exchange formats.

### 5.5.1 Structural and Syntactic Validation

For the RaCoOn upper and domain ontologies, automated validation was used to establish their structural, syntactic, and logical correctness. Three tools were used: an RDF validator to detect problems in the underlying syntax of each model, an OWL validator to check the models' conformance to syntactic and semantic restrictions, and a multi-purpose 'ontology pitfall scanner' to detect further logical and semantic issues, as well as certain content anti-patterns.

### 5.5.1.1 OWL 2 Validator Results

All ontology modules were checked for conformance against the set of OWL 2 expressivity profiles using a Java application based on [Section 4.6.1](#). The results can be seen in [Table 5.5](#), where tick marks show conformance to a particular profile, and dashes indicate non-conformance.

Table 5.5: OWL Expressivity Profiles of RaCoOn Modules

Ontology Module	Full	DL	RL	EL	QL
Upper (3D)	✓	✓	✓	-	-
Upper (4D)	✓	✓	✓	-	-
Upper (Constraints)	✓	✓	-	-	-
Core (3D)	✓	✓	✓	-	-
Core (Constraints)	✓	✓	-	-	-

This result is as expected: the vocabulary and simple axioms modules conform to OWL DL and OWL RL, but not to any less expressive OWL fragments. The ‘constraints’ modules contain more expressive axioms, including constraints and restrictions on classes, and so only conforms to OWL DL.

### 5.5.1.2 OOPS! Ontology Pitfall Scanner Results

Due to the dependence of ontology modules on each other, ontologies submitted to the OOPS! scanner were grouped into ‘cross-domain’ and ‘core’ ontology groups and validated in this way. The results of the validation were as follows:

#### CROSS-DOMAIN (3D) ONTOLOGY

The OOPS! verdict for the full cross-domain ontology, encompassing the 3D vocabulary and constraints, is shown in [Figure 5.23](#). These pitfalls were:

- **P04: Unconnected Ontology Elements.** This pitfall was detected due to the documentation meta-model being unconnected with the rest of the ontology. This is by design.
- **P11: Missing Domain or Range.** This pitfall highlights three elements of the `co:` ontology as having no domain or range assertion. This is due to the complex constraints used in this on-

tology and its inverse relationships; the domain and range of these properties are left to be inferred based on their inverses.

- **Suggestions: Transitive or symmetric properties.** The properties highlighted here have been designed as not transitive, as discussed in [Section 5.3.2.3](#).

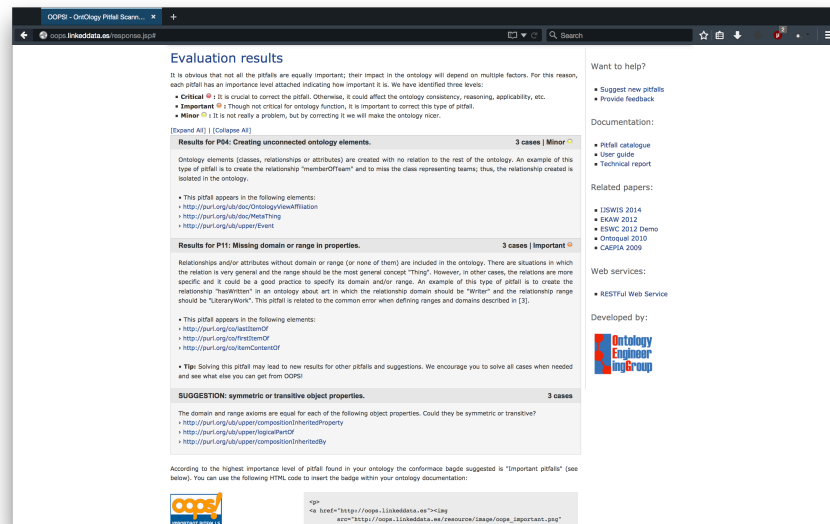


Figure 5.23: Screenshot of Oops! Result for 3D Cross-domain Ontology

Several more evaluation pitfalls occur when validating the 3D core ontology. This module depends on and so includes the upper ontology, and the pitfalls listed below are those unique to the lower level module.

### CORE (3D) ONTOLOGY

The cross domain ontology suffered several more pitfalls in validation by OOPS, mostly owing to the large number of vocabulary terms and imported concepts. In addition to those shown above, explanations for several others are provided:

- **P07: Merging different concepts into the same class** is caused by over-generalised QUDT vocabulary, for example `EnergyAndWorkUnit`. OOPS recommends against this in favour of creating individual classes for each.
- **P13: Missing inverse relationships** highlights the fact that not every property in the RaCoOn ontology is defined with an inverse. This is mostly for readability and efficiency reasons: In-

verse properties to those linking to reified ternary relations are never likely to be needed.

- **P30: Missing equivalent classes** is shown because several ontology classes share the same domain and range, and are distinguished at a high level by their use only.

### 5.5.2 Workshop Evaluation

Evaluation of the context of the rail ontologies was provided by workshops undertaken at the University of Birmingham and Invensys Rail Group headquarters during 2012. These workshops elicited a high level view of the railway domain, with attendees having no prior knowledge of the RaCoOn ontology project. [Figure 5.24](#) and [Figure 5.25](#) show photos of each session.



Figure 5.24: Photograph of RaCoOn Validation Session at the University of Birmingham

Although the half-day workshops only allowed very high level elicitation of railway domain knowledge, results from the workshops did provide an opportunity to evaluate the scope and context of the RaCoOn models in the context of the wider railway industry. For the duration of each session, attendees were split into groups of 5–6 people, with efforts made to distribute the expertise of participants between groups according to discussions prior to the start of the workshop. The following tasks were then undertaken:

1. A high level conceptualisation of the railway domain



Figure 5.25: Photograph of RaCoOn Validation Session at the University of Birmingham

2. Conceptual modelling of identified subdomains
3. Detailed modelling and elicitation of relationships between identified concepts.

Attendance at each workshop was as follows:

- **Edgbaston workshop:** Eighteen attendees divided into three teams, comprising sixteen University of Birmingham staff and students and two attendees from Network Rail. Knowledge possessed by attendees varied, and included experts in rolling stock, signalling, strategy, and infrastructure.
- **Chippenham workshop:** Ten attendees divided into two teams, comprising eight Invensys Rail Group engineers and two University of Birmingham staff. Most attendees possessed detailed knowledge of railway signalling and control systems. This workshop was intentionally held at IRG to emphasise concepts relating to the use cases described in [Section 5.1](#).

The difference in expertise and outlook at each session is reflected in the outputs presented below, and it should be noted that

#### 5.5.2.1 *Task 1: Top-down Domain Coverage*

Teams were first asked to break railway systems down into constituent high level subsystems. Most teams found between six and ten railway industry sub-categories, and it was possible to establish a 'room

consensus’ on these categories in both workshops without interference from the workshop organisers. Categories given are shown in [Table 5.6](#), and a diagram of the original ideas elicited by each individual group at the University of Birmingham in [Figure 5.26](#). The full list of domains is given in [Section C.2](#).

Table 5.6: Rail Domain Subsystems Validation Workshop Consensus

Edgbaston Consensus	Chippenham Consensus
Operations	Service
Infrastructure	Assets, Infrastructure, Maintenance
Rolling Stock	Customers
People	External Stakeholders
Regulation	Organisations & Governance
Commercial/social Environment	Resources (suppliers, land, energy)

The core concepts elicited by each group in [Table 5.6](#) align well with each other, with some exceptions. The Edgbaston cohort treated ‘people’ as a general theme where the Chippenham workshop distinguished between ‘customers’ and ‘external stakeholders’, whilst they generalised ‘assets, infrastructure and maintenance’ into a single theme. The Chippenham workshop also separately identified ‘Resources’ as a theme, where ‘Commercial and social environment’ was separately chosen in Edgbaston.

#### 5.5.2.2 Task 2: Subdomain Decomposition

In the second task, each teams were was assigned one or two of the high level areas already elicited, and asked to decompose them into a number of subdomains. The aim of this task was to build a more detailed view of subdomains within the industry as perceived by experts, and no further direction was given to groups.

In both workshops, the resulting decomposition consisted of concepts, use cases, and motivations in each subdomain, although modelling in the Edgbaston workshop produced a far greater corpus of results. Diagrams of ideas from both are available in [Appendix Section C.2](#). Results from the Edgbaston workshop for the ‘Infrastructure’ category are presented in [Figure 5.27](#).

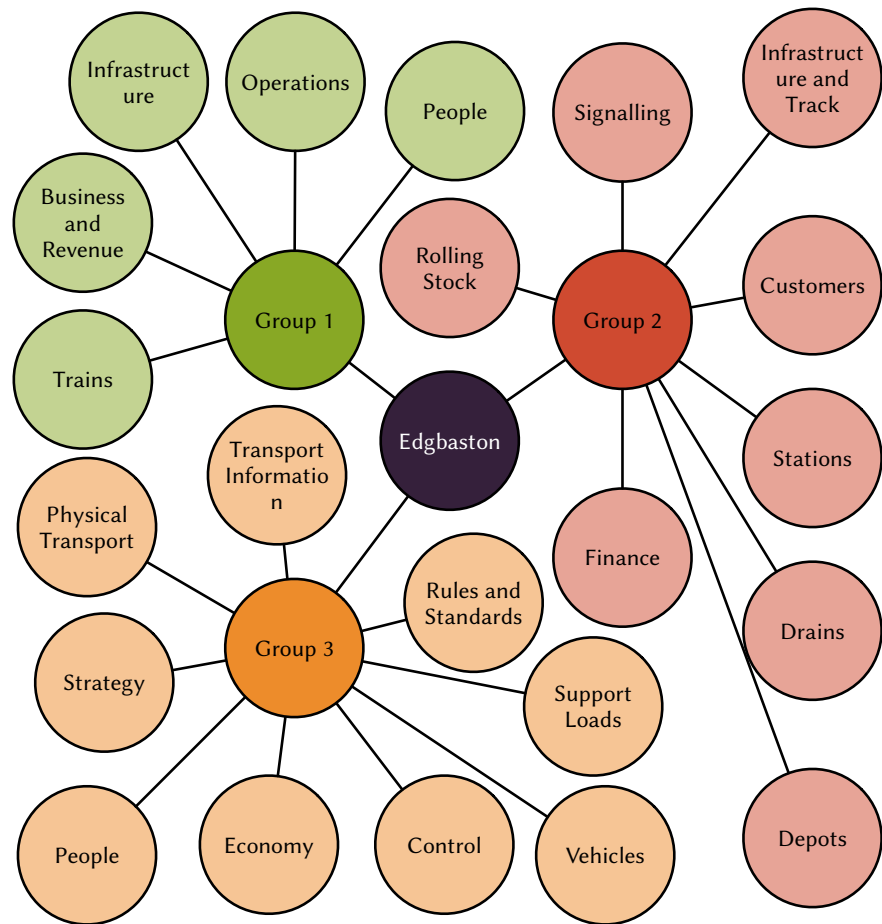


Figure 5.26: Railway High Level Systems As Conceptualised by Groups at Edgbaston Validation Workshop

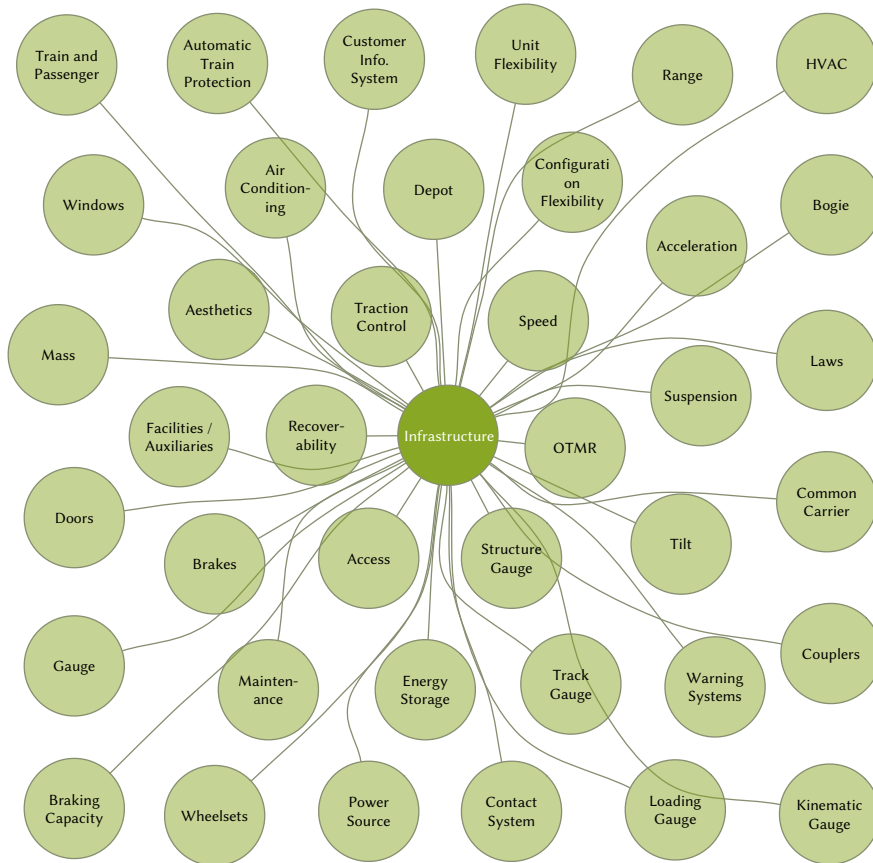


Figure 5.27: Edgbaston Validation Workshop: Results of 'Infrastructure' Category Decomposition



### 5.5.2.3 Task 3: Interaction Identification

In the third task, individuals were taken out of their groups and invited to identify low level interactions between any concepts identified by workshop participants. An example interaction provided to the groups is shown in [Figure 5.28](#). This task was designed to elicit some dependencies and relationships that may be reflected in a domain ontology, such that relationships that were in scope but missing from RaCoOn ontologies could be explored further.

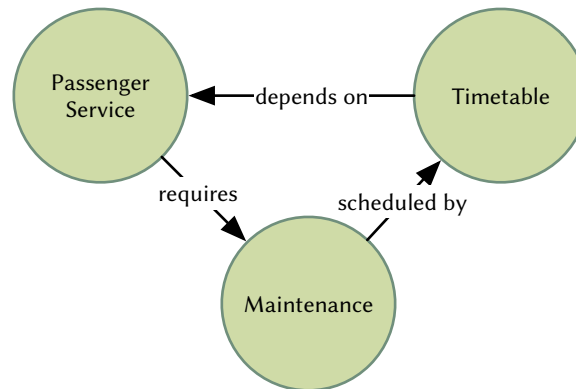


Figure 5.28: Example Interaction for Validation Workshops

The resulting interactions identified by participants are listed in [Section C.2](#). Responses from this task varied in style to those expected, with most participants identifying high level themes between subdomains rather than low level relationships between themes. Examples of relationships elicited include those shown in [Table 5.7](#).

Table 5.7: Table Showing Example Interactions Elicited from Ontology Evaluation Workshops

Subject	interaction	Object
Infrastructure	track/train gauge	Rolling stock
Operations	cargo	Rolling stock
People	maintenance staff	Operations
Assets	require investment by	Resources

### 5.5.2.4 Analysis of Ideas Generated By Workshops

Analysis of the results obtained from these workshops allowed a number of key concepts to be extracted and used. Firstly, around half of

the the high level categories identified by the workshop overlapped with the defined RaCoOn ontology project scope. These concepts and their alignment with the ontology are shown in [Figure 5.29](#).

However, several themes discussed had been given no consideration in the initial scoping of the ontologies. These areas were:

- **Stakeholders.** The relationships between actors in the railway system and assets/processes. A meta-class and concept for `u-:Actor` and `rcn:Stakeholder` were added to the ontology to provide a standard definition for these concepts, but no further modelling was undertaken.
- **Governance and regulation /Commercial and social environment.** Although signalling standards were considered, wider adherence to regulation and commercial environment was not considered as part of the initial ontology scope. The model's bias towards infrastructure terms somewhat explains this, although the groups' emphasis on these themes show that topics in these areas may require representation in a wider domain model. The representation of these concepts is left for extension into sub-domain ontologies by further work.

### 5.5.3 *Measuring Ontology Fit Using Railsys*

Following the workflow set out in [Section 4.6.3](#), a brief evaluation of the core ontology was undertaken using data from Railsys<sup>11</sup>, a railway simulation package used extensively by Network Rail and across the industry to test and optimise timetables, energy profiles, and other scenarios [173]. The railsys application was selected for validation for three reasons:

- A corpus of Railsys model data was available for analysis and use
- The Railsys data model has not previously been considered as a knowledge source for ontology design
- The scope and use case of the application fit with the scope and use case of the model as outlined in [Chapter 4](#).

<sup>11</sup> <http://www.rmcon.de/index.php?page=railsys-classic-planning>

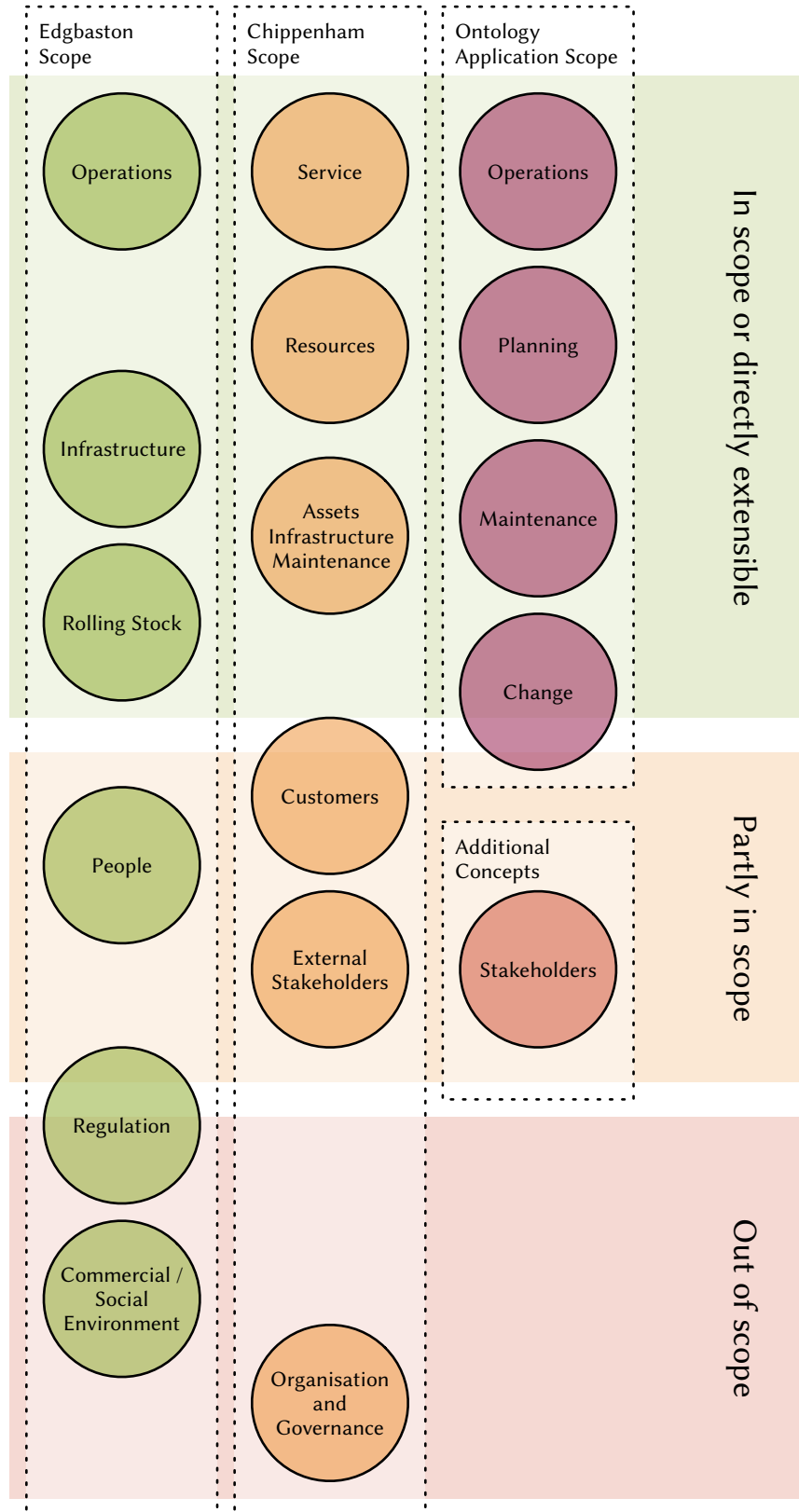


Figure 5.29: Alignment Between High Level Railway Themes Elicited in Workshops and RaCoOn Scope

### 5.5.3.1 Railsys Evaluation: Technique and Aims

A set of 119 Railsys XML infrastructure files for the East Coast Main Line was obtained for analysis. These files do not conform to a published XML Schema, but were found all to conform to the same implicit structure when analysed. Oxygen XML Editor<sup>12</sup>, a piece of XML authorship software, was used to generate an explicit XSD schema document from this set of files, and this schema used as a basis for data transformation. Figure 5.30 shows an extract of the schema's definition for line.

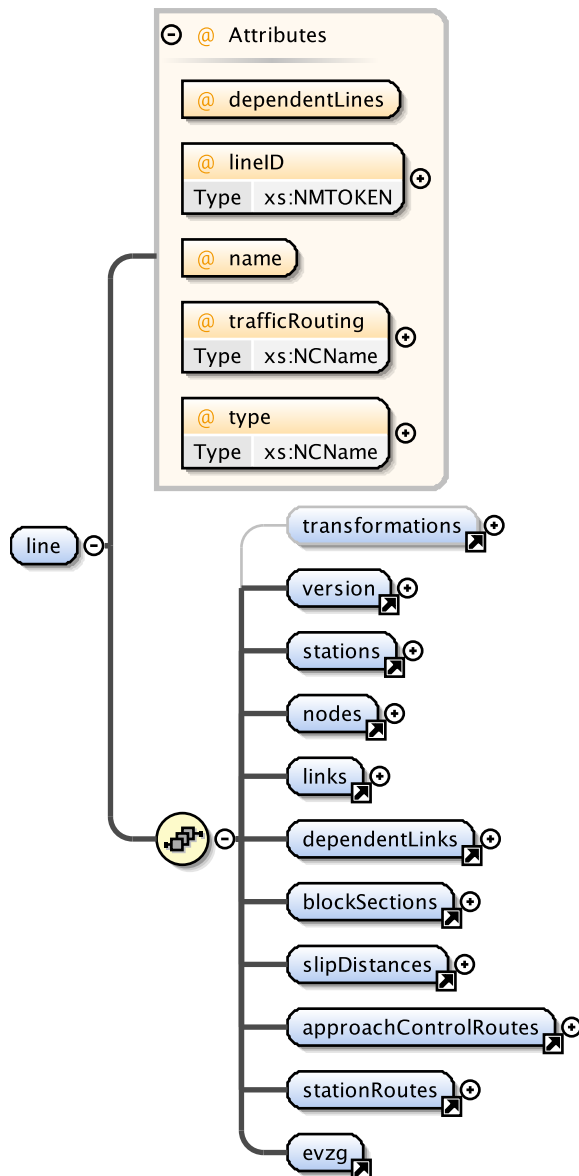


Figure 5.30: XML Schema Definition of line from Railsys Example Documents

<sup>12</sup> <http://oxygenxml.com/>

From this document, elements and their attributes were extracted and listed in a spreadsheet for review. No documentation was available alongside Railsys, and semantics for all terms were established through manual inspection and analysis of the schema's structure, term names, and data types and ranges across the set of Railsys files. Examples of the semantics extracted are shown in [Table 5.8](#).

Table 5.8: Table Showing Disambiguation of Railsys platform element

Attribute	Type	Disambiguation
<b>platform</b>	<b>element</b>	Used as a sub-attribute of 'track'
endPos	xs:integer	Platform position relative to absolute reference
left	xs:boolean	The alignment of a platform (L/R)
open	xs:boolean	The status of the platform (open/closed)
startPos	xs:integer	
width	xs:integer	

In total, 91 unique concepts or attributes were extracted, of 201 total entries in the spreadsheet. Attributes with the same name, type, and value ranges in inspected models were assumed to be identical, and marked as duplicates.

### 5.5.3.2 *Railsys Mapping and Conceptualisation*

Having compiled all of the terms in the Railsys model into a spreadsheet, each term and its potential transformation into OWL was considered. Firstly, a 'best guess' pattern for mapping each term into the existing ontology was considered and categorised, based on the term's meaning and example data given in the East Coast Main Line (ECML) files. For terms that could not be represented using existing patterns or extensions to them was then further discussed, with likely candidate patterns presented as 'extension mappings' in the spreadsheet. The resulting categories created were as follows:

- Concepts considered expressible using only existing domain ontology, including its vocabulary (marked 'mappable')
- Concepts that could likely be expressed by a small extension of an existing pattern (marked 'extension')
- Concepts that were not mappable using current ontology, and would require a more significant extension to the ontology, or new design patterns (marked 'new')

- Concepts or attributes that are prohibited from being expressed using the current domain ontology (marked ‘difficult’).

Following this categorisation based on ontology coverage, terms were also subjectively marked as either ‘relevant’ or ‘irrelevant’. This distinction was made in order to discount terms that are intentionally out of scope of the domain ontology. Mappings for concepts marked as ‘new’ but ‘relevant’ were then considered and added to the table, providing a set of entries that could be compared for coverage against the RaCoOn models.

An example spreadsheet entry after this process is shown in [Table 5.9](#).

### 5.5.3.3 *Analysis of Coverage and Quality*

The completed spreadsheet found the following:

- 67 out of 91 unique Railsys concepts or attributes could be mapped directly into the RaCoOn ontology using existing patterns
- Six concepts were found to be relevant but not currently representable in the RaCoOn ontology
  - One of these six were deemed ‘difficult’ to model by extending the current ontology. The railsys ‘track’ element defines a relationship between a train station’s platforms and their track, and uses a perspective that is not directly representable in RaCoOn.
  - The other five were found to be easily representable through the extension of design patterns, but not currently included. Three concepts were versioning attributes, for representing specific version numbers (which the RaCoOn ontology and OWL itself lacks). Of the other two, the first was a `trackRouting` attribute to represent how vehicles are routed on a set of tracks (on the left or on the right), and the second was `interlockingMachine`, an attribute to represent the type of signalling interlocking used by an entity.
- Thirteen concepts were deemed irrelevant (out of scope) but easily representable. These concepts should be implemented in an application ontology rather than a domain ontology. Concepts in this category included specific signalling characteristics, control area borders, and preset routes into railway stations.

Table 5.9: Extract from Railsys Transformation Table Showing ‘Station’ Pattern

Status 1	Status 2	Element	Usage	Type	Disambiguation	Core Mapping	Ext. Map	Notes
Mappable	Relevant	station				<b>vocab:Station</b>	-	
Mappable	Relevant	kilometre		xs:decimal		vocab:RelativePosition	-	
Mappable	Relevant	name	req.			rdfs:label	-	
Mappable	Relevant	stationID	req.	xs:Ncname		u:id	-	
Mappable	Relevant	xNet	req.	xs:double	Local positioning	[vocab:GeodesicPosition vocab:lat]	-	‘Required’ because Railsys is a simulator
Mappable	Relevant	xWGS84		xs:double		[vocab:WGS84Pos vocab:lat]	-	
Mappable	Relevant	yNet	req.	xs:double	Local positioning	[vocab:GeodesicPosition vocab:lon]	-	‘Required’ because Railsys is a simulator
Mappable	Relevant	yWGS84		xs:double		[vocab:WGS84Pod vocab:lon]	-	

‘Ext. Map’ indicates an external mapping pattern (not used in this example)

‘req.’ indicates a required attribute in Railsys

- Five attributes were found to be irrelevant (out of scope) but difficult to implement with current patterns. Application ontologies using these features would be required to create new patterns. Four of these attributes related to the spatial representation of railway switches, which are represented by distances down main and converging railway lines rather than in 3D space. The other was representation of the ‘number’ of a track passing through a railway station for signalling purposes.

From the evaluation carried out, 67 of 73 concepts deemed within scope of the intended rail ontology use case were found to be directly representable using existing patterns, indicating a high degree of coverage of the domain. No Railsys concepts were found to be inexpressible.

#### 5.5.4 *In-use Validation*

The utility of the RaCoOn ontologies is further validated in [Chapter 6](#), where they are extended and used in the storage and integration of data for two railway applications. [Section 6.2](#) builds upon the RaCoOn ontologies to create a flexible asset monitoring system, whilst [Section 6.3](#) implements a real time train tracking ontology based on network topology and track characteristics.

## 5.6 SUMMARY

This chapter has shown in detail how a set of railway domain models were created using the RaCoOn methodology shown in [Chapter 4](#). The resulting ontologies encode knowledge from existing railway data models and systems, as well as concepts elicited from industry and academic experts over the course of this project.

The domain ontologies presented focus predominantly on modelling concepts around railway infrastructure and operations. Although this is evidently a subset of the full breadth of concepts used across the entire railway industry and amongst passengers, the emphasis provided fits well with the majority of use cases and existing models examined in this thesis, and provides a relatively rich set of concepts and patterns in areas that are commonly used by railway information systems, such as geography, network layout, infrastructure characteristics, and timing. The modular ontology design approach taken provides a way of extending the models at a later date; the cross-domain ontology described provides a more generalised extension point from



which to further specify concepts for other domains and applications.

The RaCoOn models are intentionally inexpressive in some areas to allow for practical uptake of both the ontologies and reasoning; complex OWL axioms that encode knowledge are often omitted for the sake of computation simplicity. For the intended uses of the domain model, it is more important to provide a set of models that are simple enough to allow useful reasoning than to formally encode all knowledge and end up with an almost intractable ontology. The use of annotation properties, as discussed in [item 5.3.1.2](#) allows this knowledge to be maintained within the model rather than omitting it for the sake of reasoning simplicity.

In [Chapter 6](#), the ontologies shown here are built upon and utilised to present two demonstrators based on semantic web technologies and ontology reasoning.

## INTEGRATION OF RAILWAY REMOTE CONDITION MONITORING DATA

---

### 6.1 INTRODUCTION

Over the summer of 2014, the author and collaborators at Siemens Rail Automation participated in a feasibility study project funded by the Future Railway, a UK body seeking to encourage innovation across the industry. The ‘FuTRO Universal Data Challenge’ sought novel approaches for handling dynamic data in the railway industry, and the team’s response to the challenge was a set of demonstrators and documentation based on the author’s work described in [Chapter 5](#) illustrating how semantic data modelling techniques could be used to assist data sharing for two different use cases. This chapter documents the author’s contribution to the two demonstrators, which are summarised as follows:

- *AMaaS* is an asset monitoring and management framework for use by railway maintainers and managers based on cloud computing technologies and graph data storage. Using the RaCoOn model as a base, the demonstrator provides a flexible platform for processing and viewing asset monitoring data from diverse wayside data sources, and uses the domain ontology to provide enriched contextual information for this information .
- The *Train Locator* is a small passenger information system simulator that shows how simple reasoning and inference can be used to aid knowledge management across system upgrades. Using a small set of custom OWL axioms and rules, the web application shows how ontology-based systems can easily transform data to allow legacy systems to continue operating without changes.

### 6.2 THE AMAAS APPLICATION

Asset Monitoring As A Service (AMaaS)<sup>1</sup> is a system designed by Siemens Rail Automation and the author that utilises semantic web technology to provide a scalable software framework for railway asset monitoring and management. Built upon a prototype application

---

<sup>1</sup> <http://purl.org/rail/futro>

developed by Siemens for their rail asset monitoring customer base, AMaaS uses commercial off-the-shelf software with the RaCoOn ontology, and demonstrates how semantic data modelling can be used to add flexibility and value to existing asset monitoring data. Here, details of the author's contributions to the whole system are presented, including implementation details and ontology design decisions.

As discussed in [Section 1.1.2](#), asset management systems used by railway maintainers and operators in the UK are usually closed systems that operate independently of other railway systems and rely upon human operators to input information about the state of the railway. Remote Condition Monitoring systems such as the recent instrumentation of London's Victoria Line [55] are implemented in similar ways, with off-the-shelf applications relying on maintenance staff to transfer information between monitoring systems where appropriate. By working with the Siemens team during the implementation of AMaaS, the author was given the opportunity to demonstrate how ontology-based systems improve data representation and management in such a system, and the resulting benefits that can be achieved.

### 6.2.1 *Overview and Use Case*

#### 6.2.1.1 *Project Motivations and Requirements*

Siemens' initial motivation for designing a new asset monitoring system was to provide a generic, scalable platform that could be used and customised for each client, but share the same technology and infrastructure, following the Software-as-a-Service (SaaS) paradigm.

The cloud platform was to form part of a whole condition monitoring solution, with all components provided by the company: sensors and data acquisition hardware, back-end data processing software, and front end applications for a variety of different users. A key requirement of the AMaaS system was that it could scale over time and respond to customers' changing requirements, as they incorporate more and varied data sources into the platform, as shown in [Figure 6.1](#). The use of a semantic data model had been previously considered by the Siemens team to allow flexibility in provisioning and managing data sources, but had not yet been implemented—design of the AMaaS system in its entirety provided an opportunity to fully develop this part of their prototype system.

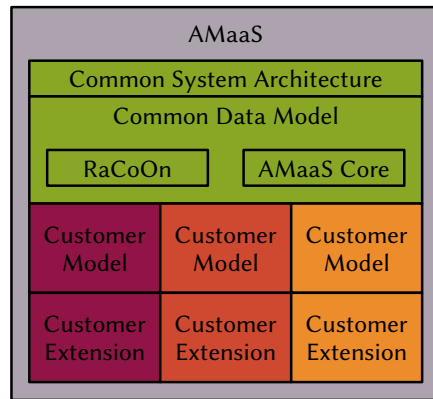


Figure 6.1: AMAAS Modularity

Alongside Siemens' goals for AMAAS itself, the FuTRO Data Challenge brief required that candidate technologies should be capable of integrating and interacting with data from inside and outside of the system. This led to the production of several more high level requirements by the University of Birmingham for the project:

- Demonstration of data integration between heterogeneous datasets, and proof of specific benefits gained from an asset monitoring and maintenance perspective.
- Design of an open platform to allow data to be harnessed from other, external, information sources and to allow publication of system data for use by other platforms.
- Development of a system in RDF/OWL to show proof of concept for semantic interoperability using linked data technology and web ontologies.
- Demonstration of semantic enrichment of system information through RDFS and OWL ontology reasoning.
- Proof of feasibility of implementation using off-the-shelf, scalable technology.

The creation of such a system was also intended to publicise the advantages of data interoperability across the rail industry, through exposure generated in response to the FuTRO Universal Data Challenge project.

#### 6.2.1.2 Selected Applications

The demonstration scenario created for AMAAS was constructed from two known application use cases, as detailed below. The value gained

through better management of maintenance data is demonstrated for both groups of users, and this added value provided a strong incentive for development of the project in these areas.

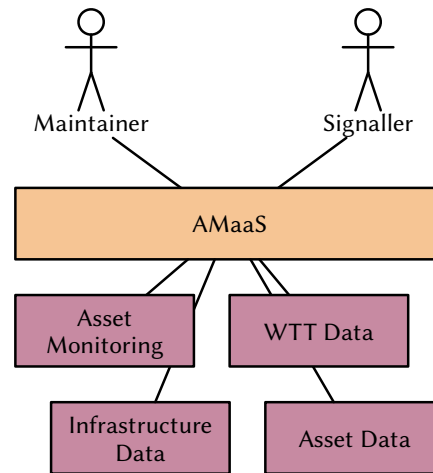


Figure 6.2: Data Sources And Actors In AMaaS

- **Infrastructure Maintenance and Asset Management** was chosen as the primary application area for the AMaaS project. Infrastructure managers have a need to track the health of assets on the railway, and condition monitoring systems allow them to better assess and react to deterioration of assets. Combining Remote Condition Monitoring (RCM) and historic asset data provides a fuller picture of the state of an asset, allows clearer decisions to be made on maintenance and management, and facilitates easier day-to-day operations through immediate access to location and asset register information.
- **Signalling and Control** systems do not usually include any extra contextual information on the railway network, instead relying on signallers to gather this information from other sources. Although newer technology allows for basic monitoring and alarm inputs to be used [115], the provision of rich asset information directly to such systems may allow signallers and train operators to make better informed or more timely decisions in the case of degraded railway operation.

### 6.2.1.3 Demonstrator Application Storyboard

Taking into account these two use cases, an application storyboard was created for the demonstrator, showing a number of steps through the application, each of which add a new feature and exemplify its advantages. These steps are shown in [Figure 6.3](#).

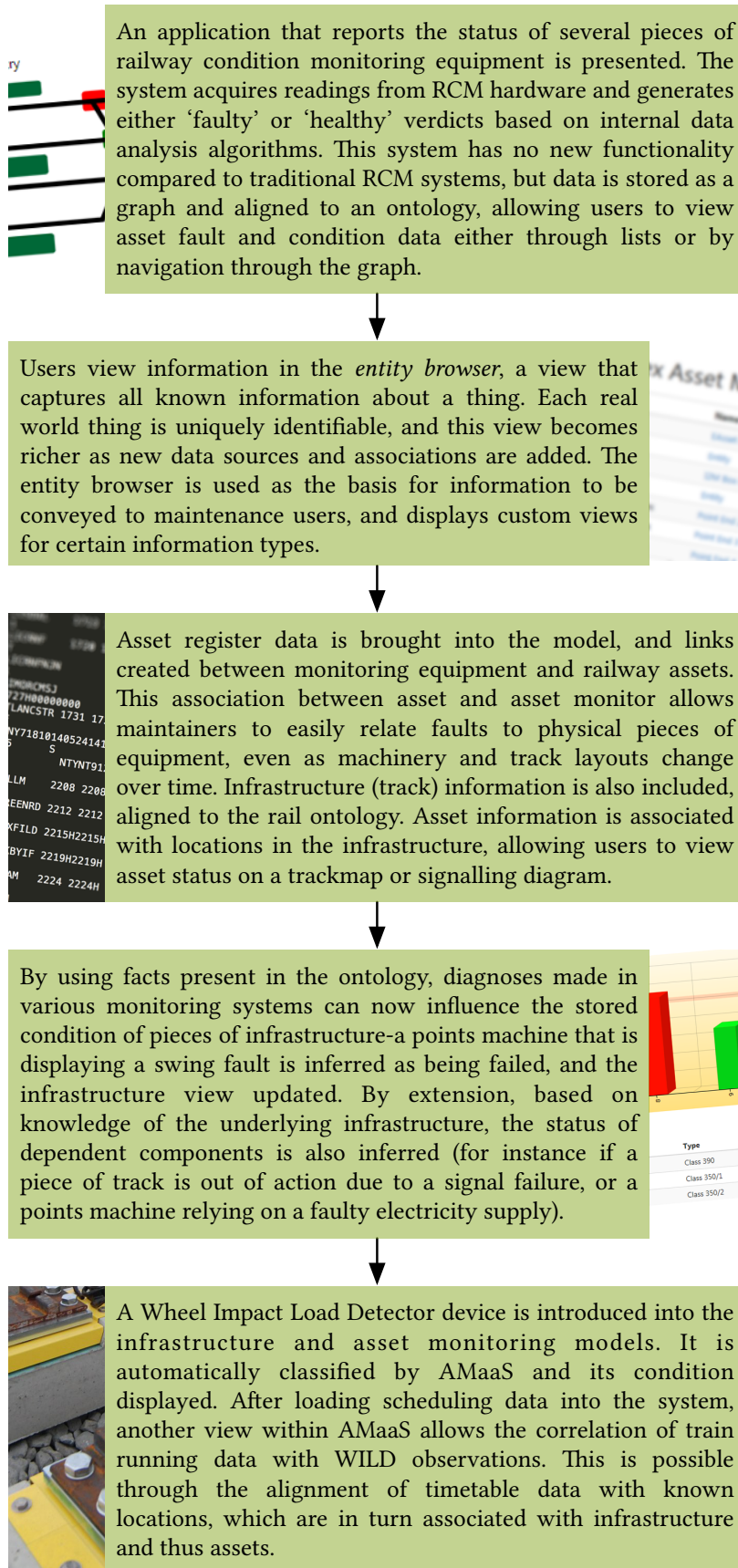


Figure 6.3: System Design Storyboard for AMAAS System

### 6.2.2 Existing Prototype Architecture

The Architecture of the AMaaS platform is based on a prototype system conceived by Siemens Rail Automation before the start of the project<sup>2</sup>. Whilst this chapter focusses largely on the novel data model used in AMaaS, the following section describes elements of the Siemens prototype system used or adapted for the implementation of AMaaS.

#### 6.2.2.1 Siemens System Architecture

Prior to the development of the AMaaS system, the Siemens Rail Automation prototype used cloud computing technologies to provide volume scalability and resilience. A high level diagram of the initial architecture is provided in Figure 6.4. The prototype includes a number of components that were built upon and extended over the course of the development of AMaaS, and these are explained briefly below.

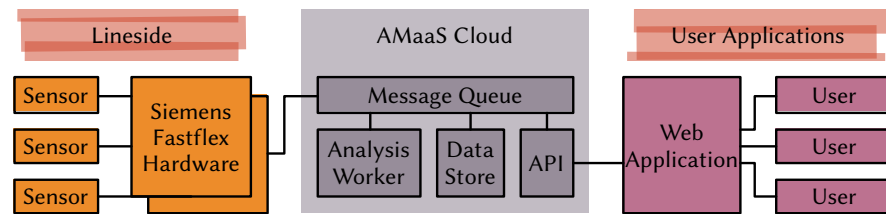


Figure 6.4: Siemens Cloud-based Asset Monitoring System Architecture

#### DATA ACQUISITION

The Siemens Fastflex condition monitoring system [210] was chosen as the data acquisition stage for AMaaS. The system provides data acquisition and limited analysis functionality, and comprises the following components:

- **Sensors** instrument railway assets, and provide an initial input to the system. In a typical Points Condition Monitoring (PCM) scenario, current sensors detect motor torque characteristics, whilst microswitches and temperature sensors establish movement state and environmental condition.
- **Data acquisition input/output and conversion hardware** provides an interface between the sensors and the Fastflex software.

<sup>2</sup> This prototype is unrelated to the Siemens Energy system described in Chapter 3

- **Aggregators** acquire signals from sensors or other input devices, and process them according to application. In the prototype system, Siemens Fastflex control units act as aggregators, and process data for up to four points machine monitoring devices. An industrial Personal Computer (PC) inside the Fastflex unit is used to encode data and either store it onboard, raise alarms via digital (hardware) outputs and relays, or transmit it to a maintenance control centre. The flexibility of the PC-based software approach meant that it could be modified as part of the AMaaS project to transmit data using standardised linked data protocols.

#### PROCESSING

The ‘cloud’ component of the prototype system was built using a service-oriented architecture, and used a message queue system to process data arriving from wayside RCM hardware and service requests from applications. Monitoring and state data was stored in a document database (MongoDB) and key-value store (Redis), and presented over a web API to interface with the AMaaS web application. The modular nature of the system allowed it to respond to changing demand by provisioning new worker nodes to provide more computing power as necessary.

#### FRONT END

The prototype system’s front end was developed as a web application, presenting asset information to users in tabular form. The web application submitted requests to a web API residing inside the AMaaS cloud for each view, and information was displayed through an HTML interface. Application logic was encoded into the web API, such that it could fulfil the tasks necessary to display asset information to users.

#### 6.2.3 *Proposed System Architecture*

Given the existing system architecture, aims and requirements discussed earlier, a modified system design was proposed for the AMaaS demonstrator. This system architecture builds on the architecture already in place, but uses a distributable RDF triplestore to store both the domain ontologies (T-box) and instance data in the asset monitoring system (A-box). An overview of this system architecture is shown in [Figure 6.5](#). The initial architectural design of the system required the following tasks to be undertaken:



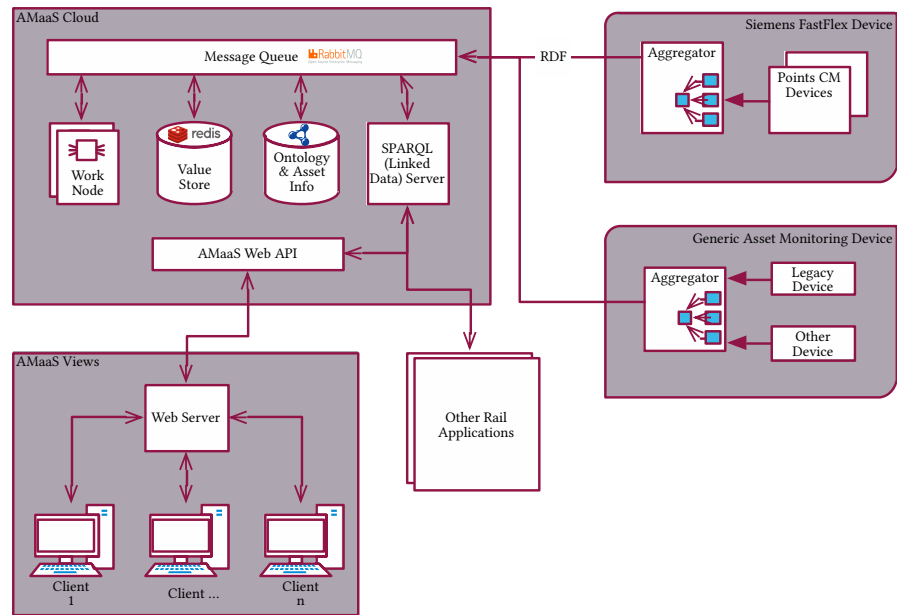


Figure 6.5: AMaaS Demonstrator System Architecture

- Extension of RaCoOn ontology to facilitate asset self-description, condition monitoring observations, and associated metadata.
- Selection and configuration of off-the-shelf RDF triplestore to replace relational database in prototype system.
- Development of Siemens Fastflex software to provide asset self-description and alignment to ontological models.
- Alignment of prototype system data model with new AMaaS ontology.
- Provision of Linked Data Platform compliant public endpoint for integration with other data sources.
- Capture and conversion of logic from prototype system into AMaaS ontology; re-implementation of prototype front end as SPARQL compliant application.
- Extension of front end applications to cover demonstration use cases.

These tasks are discussed in the next section.

### 6.2.3.1 T-box vs A-box Knowledge

In traditional schema-based data storage systems, the terminology, or domain knowledge (T-box), is usually inherently separated from the

instance data (real world facts, or A-box). XML models make this distinction by encoding T-box (domain) knowledge into the structure, or syntax, of an XML schema document, and storing the A-box (instance) data according to these definitions. Relational databases and spreadsheets do the same, by encoding domain knowledge into table structures and labels. In RDF, both T-box and A-box assertions are stored as triples in the same data store, although it is useful for reasoning to keep the two sets of facts separate from each other.

### 6.2.3.2 *System Requirements*

The functional requirements for each of the storyboard stages given in [Section 6.2.1.3](#) are summarised in [Table 6.1](#). Progression through each stage of the demonstrator brings new requirements, and the features implemented along the way are discussed in the following section. Modelling tasks (shown in bold) are discussed in particular detail.

### 6.2.3.3 *Implementation Technology Choices*

In creating the demonstrator application, it was important that the technology and implementation used could be easily replicated and used by developers in the rail industry. The technology choices made during the implementation of AMaaS reflect this, where only very well supported or standardised technologies have been used. The selection of these tools also aided rapid development, and allowed developers at Siemens to draw upon existing work.

#### RDF DATA STORE

The requirements for a well-supported, standardised system led to a commercial, off-the-shelf RDF triplestore being used for AMaaS. A number of such triple stores exist, with several performance reviews examining the effectiveness of each in different situations [138, 139]. Five were selected for comparison for the AMaaS project, based on factors including reasoning performance, scalability, and ease of development use. An overview of their feature sets are shown in [Table 6.2](#).

Given the rapid development lifecycles in triplestore software (Stardog advanced three minor versions over the course of the project), it is anticipated that a production implementation of AMaaS would conduct a more detailed evaluation of available options and potentially reconsider choice of a triplestore at a later date.

- **Cluster Ready** indicates whether or not a triplestore is scalable within the AMaaS cloud framework; in practical terms,

Table 6.1: Functional Requirements Defined From AMaaS Storyboard

Storyboard Stage	Data Model and Interface Requirements
1. Basic Asset Monitoring	<b>Taxonomy of asset monitoring system</b> (for navigation) <b>Base model of asset monitoring equipment</b> Current and historic observations for each device Current diagnoses and analysis of assets PCM-specific application: fine-grained data display
2. Entity Browser	User interface to allow navigation through system
3. Infrastructure-linked Asset Data	<b>Asset register and infrastructure model</b> and information Signalling (track map) view Links between assets and asset monitors
4. Status Inference and Dependent Devices	<b>Fault diagnoses and correlated asset health</b> <b>'Dependence' relations and reasoning semantics</b> <b>Links between dependent assets on trackmap</b> Indication of asset health on track view Alarm panel for unhealthy assets
5. Wheel Impact Load Detector (WILD) extension	WILD device information WILD display (trackmap) information WILD condition (WILD observations displayed using generic model)
6. WILD Rolling Stock Correlation	<b>Working timetable (WTT) model</b> and data <b>Infrastructure-linked route data</b> <b>Inference of rolling stock traffic on infrastructure</b> WILD-specific application extension (Train finder view)

Table 6.2: Comparison of Features Across Popular RDF Triplestores

Triplestore	Standards Compliant?	Cluster Ready?	Reasoning	Cost
Stardog Community Edition <sup>4</sup>	X	X	DL (T-box) /RDFS/Rule (A-box)	Free
Virtuoso Enterprise	X	X	Rule-based	Paid
Virtuoso Open Source	X	-	Rule-based	Free
OWLIM Enterprise	X	X	OWL RL/Rule	Paid
Allegrograph	X	X	RDFS/Rule/Temporal	Paid

whether it offers a clustered/distributed configuration option. Many triple stores scale to billions of triples, but these often rely upon centralised/single server solutions.

- **Standards Compliant** indicates whether each triplestore complies with semantic web standards. Minimally, these are RDF 1.0 and SPARQL 1.1 Query and Graph Store, although many support RDF 1.1 and a larger subset of the SPARQL 1.1 specification.
- **Reasoning** gives an indication of the level of reasoning support provided by the triple store. Reasoning poses a huge problem for web-scale data, as discussed in [Section 4.5.6](#), and many triple stores provide intelligent mechanisms for realising some level of reasoning. In AMaaS, both OWL RL and reasoning are used with a view towards ultimate scalability of the solution.
- **Cost** indicates whether a license for the triple-store was free of charge, or paid for. Triplestores available for free all impose functionality or licensing restrictions to prevent enterprise use; the requirements of AMaaS were well within these limits.

Clark & Parsia’s Stardog triplestore was chosen for use with AMaaS as it satisfied key requirements for scalability, reasoning, standards-compliance, and usability. Stardog’s reasoning capability provides OWL 2 DL reasoning across the ontological part of the triplestore, and OWL 2 RL across the A-box (instance) data, allowing the the semantics of

the RaCoOn ontology to be reasoned over whilst maintaining sufficient speed and scalability for the demonstrator. The reasoner uses *query rewriting*, or backward-chaining, so reasoning is only triggered at query time, rather than storing and maintaining a set of inferred axioms persistently.

#### BACK-END AND FRONT-END TECHNOLOGIES

Influenced by the initial prototype, both the aggregator (Fastflex) software and front end web applications for AMaaS were constructed using Microsoft's .NET technology stack. The Aggregator software was extended from a C# .NET application running on an embedded Microsoft Windows PC, and the AMaaS web application developed in C# on Microsoft Internet Information Services. In both cases a well-written and compatible codebase was already available, and third party libraries were used to allow easy compliance with semantic web interface standards.

For the demonstrator's implementation, a Microsoft Azure virtual machine was used to run the web (front end) API, message broker, analysis workers, and the Stardog triplestore.

#### MESSAGING MIDDLEWARE AND ANALYSIS

The AMaaS cloud used an off-the-shelf Service-oriented Architecture framework to queue and handle incoming and outgoing requests. Data producers or consumers, such as RCM devices or web APIs, create and send messages to a central message broker, which then queues the messages for processing. Worker agents subscribed to the message queue carry out fixed processing tasks on the message content (such as analysis or data storage), and any number of workers can be provisioned according to purpose and demand. Pivotal RabbitMQ<sup>5</sup> was chosen as the message-brokering software to be used in AMaaS, due to its compliance with the web-standard Advanced Message Queuing Protocol (AMQP) messaging protocol.

To allow appropriate workers to be assigned the correct tasks in the AMaaS cloud, a service registry is maintained within RabbitMQ. To further encourage interoperability it was desirable that a method for semantic description of services (to facilitate flexibility as the system grows) should be used, but current methods for achieving this too immature or overly complex for implementation within the project's time constraints. The Semantic Automated Discovery and Integration

---

<sup>5</sup> <http://www.rabbitmq.com>

design pattern [228] provides one appropriate methodology for implementing this as the project is developed further in the future.

#### DATA MODELING AND TRANSMISSION STANDARDS

In addition to using OWL and RDF to model data in the RaCoOn ontology and the AMaaS application, RDF data serialised using Turtle was employed in the transmission of messages from RCM equipment to the AMaaS cloud. The front end application, hosted separately from the main platform, used SPARQL 1.1 queries to request RDF information from the triplestore. These choices mirror those made in the Linked Data Platform 1.0 recommendation [229]. In a production system, binary formats such as HDT [62] may be considered in order to achieve higher compression of transmitted messages.

#### FINE-GRAINED DATA

One weakness of RDF is that it is inefficient for storing very fine-grained data such as waveforms with correct semantics. This was found to be an issue in the AMaaS PCM application—current, temperature, and switching waveforms are generated for every PCM observation, and this data must be stored and displayed to users, as well as used for analysis. Storing such data as RDF with exactly preserved semantics would require assertion of huge number of triples (>100 000 per observation), compared to only a few kilobytes of binary data. As such, a clustered key-value store, Redis, was used to store measurement data for the PCM system, and mappings created in RDF to allow applications to access and query this data in appropriate ways (as described in Section 6.2.4.4).

Binary data in Redis was serialised into a C# object storage format, and queried directly by the front end application. Representation of fine-grained data in a semantic model requires trade-offs between modelling perfection and pragmatism; an alternative approach considered was to represent data as encapsulated Comma Separated Values (CSV) within RDF triples. This use case is not limited to the AMaaS project, and a vocabulary for describing CSV semantics in RDF is listed as a current deliverable for the W3C ‘CSV On The Web’ working group<sup>6</sup>.

---

<sup>6</sup> <http://www.w3.org/2013/csvw/>

## SPARQL ENDPOINT

The demonstrator includes a SPARQL 1.1 endpoint to enable other systems to take advantage of data captured within AMaaS. The system exposes a single federated endpoint, which provides access to both the Stardog data store and a custom Redis interface to fine-grained data designed by Siemens. Data flow in the system is shown in [Figure 6.6](#).

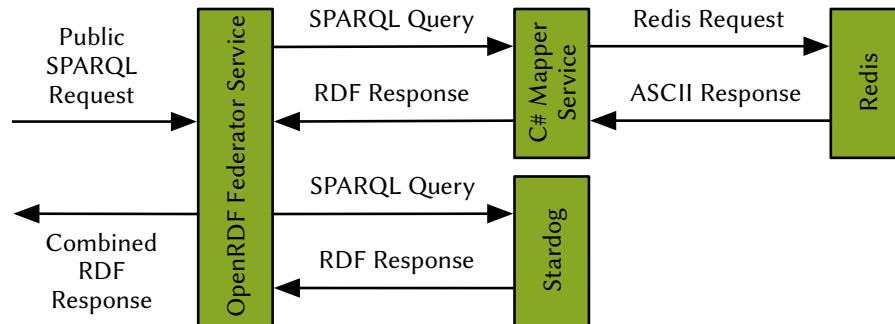


Figure 6.6: Block Diagram Showing Query Activity across Federated Data Stores in AMaaS

#### 6.2.4 Stages 1 & 2: Asset Monitoring System Implementation

The following sections highlight specific technical details from the implementation of the AMaaS system. This first section describes implementation of asset monitoring fundamentals, and specifically the conversion of the Siemens prototype asset monitoring system to use an OWL model and RDF data storage. The model itself is presented, as well as how the initial application works and is presented to users.

Having established the overall system architecture, the first stage in development of the AMaaS prototype was to ensure that asset monitoring concepts involved with the system could be correctly represented in an ontological model. Identification of ontology concepts for both generic asset monitoring devices and then for more application-specific PCM devices was undertaken; implementation of these new concepts took the form of both an OWL extension of the RaCoOn ontology described in [Chapter 5](#), and a set of natural language design patterns for documentation.

##### 6.2.4.1 Fundamental Asset Monitoring Concepts and Patterns

To model the layout of asset monitoring equipment in the railway infrastructure, the AMaaS ontology extends concepts in the core on-

tology. Subclassing existing classes allow other applications to infer traits of asset monitoring equipment<sup>7</sup>, whilst retaining expressivity necessary for the AMaaS application. An example model of the proposed pattern, showing both core domain level concepts and AMaaS asset monitoring concepts, is given in Figure 6.7.

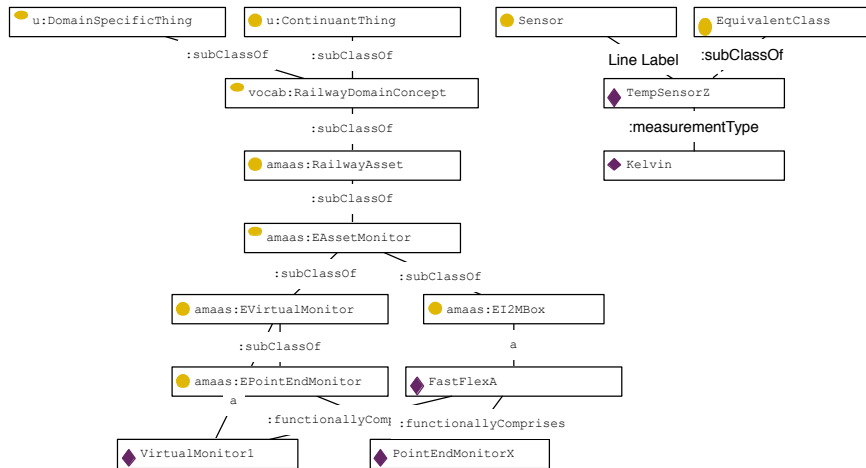


Figure 6.7: AMaaS Component Topology Design Pattern

The key new subclasses in this pattern are `amaas:RailwayDomainConcept` and `amaas:EAssetMonitor`, whose members are all railway domain objects and spacio-temporal entities. `amaas:EAsset` is the class of all railway assets, and `amaas:Sensor` and `amaas:EVirtualMonitor` allow the representation of ‘virtual’ asset monitors within physical hardware. Properties to allow modelling of compositional elements are provided, and link to quantity/type definitions provided by the core ontology. The combination of these elements allow for basic modelling of asset monitoring concepts, without the need for additional constraints. Other useful concepts include:

- ‘Virtual’ device monitors are related to physical devices through the `u:functionallyComprises` relation. The same object property can also be used to build a hierarchy through asset monitoring devices, such as in the case of distributed mesh networked sensors.
- `amaas:sensor` defines the mapping between a hardware sensor and the acquisition device it is connected to.
- `amaas:I2MBox` is the Siemens-specific subclass used for its ‘Fast-flex’ branded asset monitoring devices.

<sup>7</sup> For example, subclassing physical concepts as `rcn:InfrastructureThing` infers that they are physical, located objects on the railway.



## CREATION VS REUSE

Ontology design methodologies heavily favour re-use of existing patterns and concepts over re-invention. Before creating key patterns for the AMaaS ontology, a review of existing patterns was undertaken, and the W<sub>3</sub>C's Semantic Sensor Network ontology [38] was identified as a possible candidate for re-use. Conclusions drawn from a review of the Semantic Sensor Network (SSN) ontology were as follows:

- The Semantic Sensor Observation pattern [117] represents observations and measurements correctly, but is unnecessarily complex for the AMaaS application. The AMaaS observation pattern mostly aligns with the Semantic Sensor Observation pattern, but implements more granular temporal representation of measurements. Extensions were also made to allow for representation and diagnoses of measurements stored outside of the RDF model—allowing the inclusion of fine-grained measurement data.
- Sensor characteristics are modelled effectively in SSN, and this pattern could be used in the AMaaS ontology to capture details of asset monitoring sensors, although its use was felt to be beyond the scope of the AMaaS demonstrator.
- Groups of observations can be represented using the Semantic Sensor Network ontology, but not in a way that is optimised for use cases such as PCM, where one “observation” will always involve a set of sensors. As this model is frequently required in AMaaS, a different pattern requiring less complexity was created to cater for it.

Owing to the need for variations as outlined above, the Semantic Sensor Network ontology itself was not used within AMaaS. However, care was taken to keep concepts semantically similar such that integration and extension with SSN concepts is possible through simple mappings or OWL axioms.

#### 6.2.4.2 *The Observation Pattern*

Central to the model of asset monitoring is the notion of an observation or event; some observed change by a device at or over a specified time. In the case of PCM, this is usually a points swing, but the exact nature of these observations varies in other cases. In AMaaS, an observation is defined as *a distinct set of measurements to which a diagnosis can be attributed*, providing a sensible way to aggregate data

from multiple sensors<sup>8</sup>. Additional property chain axioms have been added in the AMaaS ontology to allow the inference of extra properties for measurements themselves:

$$\begin{aligned} \text{startTime} &\sqsubseteq \text{measurement} \circ \text{startTime} \\ \text{unit} &\sqsubseteq \text{observationOf} \circ \text{unit} \end{aligned}$$

Figure 6.8 demonstrates an example of the observation design pattern, including inferences made based on these axioms.

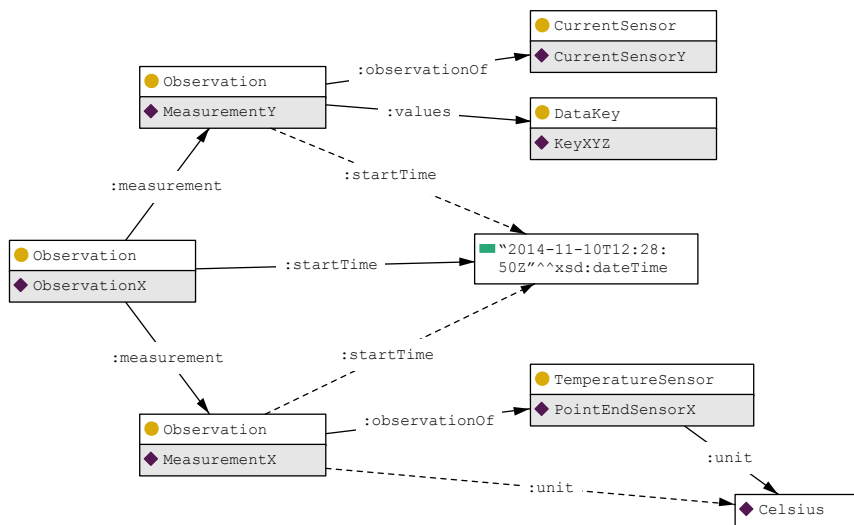


Figure 6.8: AMaaS Observation Pattern

A Siemens FastFlex PCM device utilises this design pattern for marking up data in the following ways.

- Following a points swing, an `amaas:Observation` entity is created. This entity is marked up with a start time (as well as optionally other temporal attributes) and a number of `amaas:-measurement` relations asserted.
- Each `amaas:measurement` assertion links an observation with either a particular measurement or a sub-observation, representing a particular sensor's observation for that time. The observation's metadata is then inferred through the axioms shown

<sup>8</sup> When considering systems (such as weather monitoring) that continuously observe conditions, the division of measurements into discrete observations seems counter-intuitive. It does, however, make asset monitoring data more manageable, and since none of the data semantics are lost, was considered a good approach in AMaaS.

above. This compositional approach allows diagnoses or meta-data to be attached to either the parent observation (the points swing) or an individual sensor's output.

- Metadata relating the original sample data to the points movement is added in the form of URI pointers, as described in [Section 6.2.4.4](#).

### 6.2.4.3 AMaaS Front End

The User Interface for the AMaaS demonstrator user interface is an extension of that provided by Siemens in their original asset monitoring system. It is written as a web application, and communicates with the AMaaS web API. In the initial stage of the demonstrator, where only asset monitoring concepts are represented, its key feature is the *Entity Browser*, a view which allows railway maintainers to view asset monitoring information in text form.

Navigation in the entity browser is done by hyperlinks between asset monitoring devices based on relations in the AMaaS ontology. Each class or individual has a page, and most relations from each entity are displayed in a table<sup>9</sup>. As a result of this, the structure of the user interface mirrors precisely the structure of the data available in the ontology, and allows it to adapt immediately to new resources, classes, or other extensions that are loaded into AMaaS—a feature which is used extensively as new functionality is introduced. At this stage, navigation between asset monitoring devices is provided through the hierarchy of the AMaaS system, as described in [Figure 6.9](#).

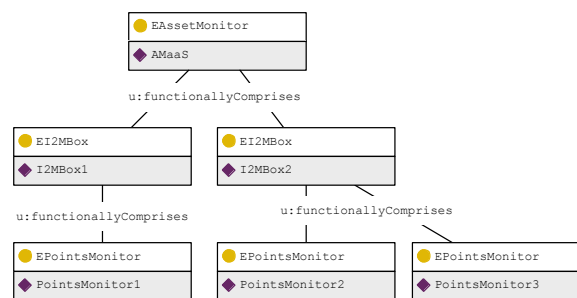
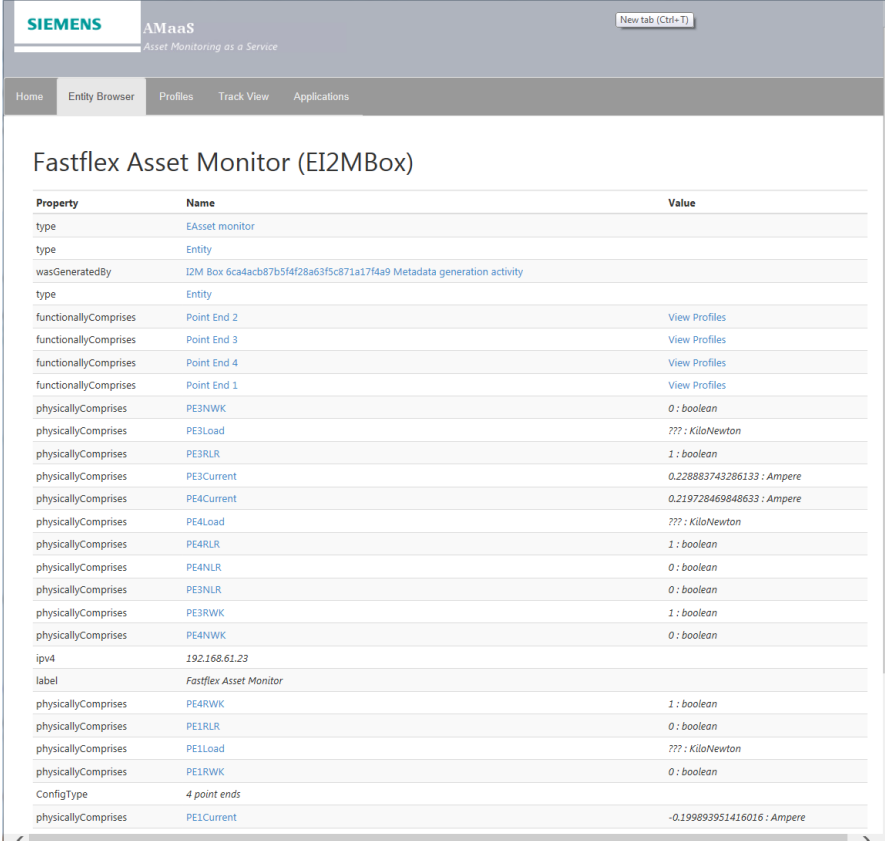


Figure 6.9: Hierarchy of PCM Devices in AMaaS Demonstrator System

<sup>9</sup> Some RDFS and OWL built-ins are intentionally excluded from the user interface for usability reasons; these include OWL restriction axioms and documentation properties.

The entity browser presents information on an asset based on its URI. Information about each entity is gained by submitting a request to the web API over SPARQL, and then formatting the response to return a web page containing the content. Rather than the web application resolving each entity's URI directly, they are passed into the entity browser as a request parameter, ensuring that the application only presents a useful subset of asset data to users. Figure 6.10 shows a screenshot of the Entity Browser.



Property	Name	Value
type	EAsset monitor	
type	Entity	
wasGeneratedBy	I2M Box 6ca4acb87b5f4f28a63f5c871a17f4a9 Metadata generation activity	
type	Entity	
functionallyComprises	Point End 2	<a href="#">View Profiles</a>
functionallyComprises	Point End 3	<a href="#">View Profiles</a>
functionallyComprises	Point End 4	<a href="#">View Profiles</a>
functionallyComprises	Point End 1	<a href="#">View Profiles</a>
physicallyComprises	PE3NWK	0 : boolean
physicallyComprises	PE3Load	??? : KiloNewton
physicallyComprises	PE3RLR	1 : boolean
physicallyComprises	PE3Current	0.228883743286133 : Ampere
physicallyComprises	PE4Current	0.219728469848633 : Ampere
physicallyComprises	PE4Load	??? : KiloNewton
physicallyComprises	PE4RLR	1 : boolean
physicallyComprises	PE4NLR	0 : boolean
physicallyComprises	PE3NLR	0 : boolean
physicallyComprises	PE3RWK	1 : boolean
physicallyComprises	PE4NWK	0 : boolean
ipv4	192.168.61.23	
label	Fastflex Asset Monitor	
physicallyComprises	PE4RWK	1 : boolean
physicallyComprises	PE1RLR	0 : boolean
physicallyComprises	PE1Load	??? : KiloNewton
physicallyComprises	PE1RWK	0 : boolean
ConfigType	4 point ends	
physicallyComprises	PE1Current	-0.199893951416016 : Ampere

Figure 6.10: Screenshot of the AMaaS Entity Browser

#### 6.2.4.4 Referencing Sample Data

The decision to allow fine-grained sample data from RCM systems to be stored separately from the RDF store led to the need for a way of representing links to external systems within the AMaaS main platform. This was modelled in RDF in the following way:

- `amaas:ExternalStoreObject` was created to represent external data sources. Whilst full semantic description of access mechanisms to such external objects is complicated, the creation of

this entity allows for some description of how agents may interface with them.

- `Amaas:MovementRecordObject` entities are created for each new external data store object, and record an identifier for use in the external system, as well as a relationship to the correct `amaas:ExternalStoreObject` from which the information can be retrieved.
- `AMaaS` observations are linked to these objects through the `amaas:movementRecordObject` relation. Enough information is now present for an application to correctly retrieve sample data relating to an observation<sup>10</sup>.

In `AMaaS`, every PCM measurement is described in RDF and written to the triplestore. Sample data is independently uploaded to a Redis key-value store as a serialised JSON object, and a pointer to its location attached to each observation object in the RDF store. Figure 6.11 shows the activity flow across `AMaaS` when a PCM measurement is taken.

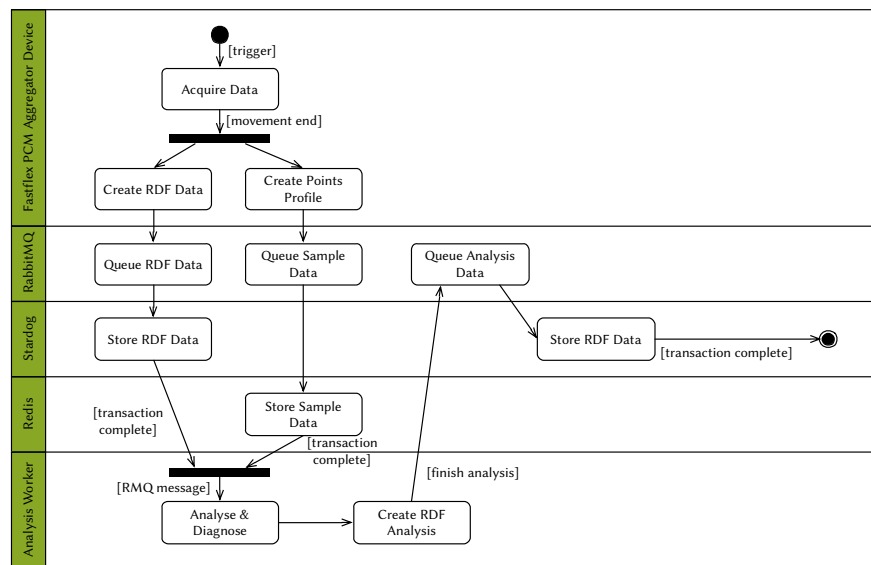


Figure 6.11: AMaaS Observation Pattern

At this stage, diagnostic data is asserted for each points movement profile, using either pre-defined or new instances of the `amaas:Condition`

<sup>10</sup> More specific metadata for samples, such as sample rates and error tolerance, were not recorded in the prototype `AMaaS` system, and instead rely upon application-level logic for this information. It is likely that as the system is further developed and extended, proper mechanisms for handling different external data stores (perhaps assuming a standardised web architecture) will be created.

class. Each points movement is viewable in the AMaaS front end, which is invoked when a user clicks on a point end monitor asset in the Entity Browser (see Figure 6.12).

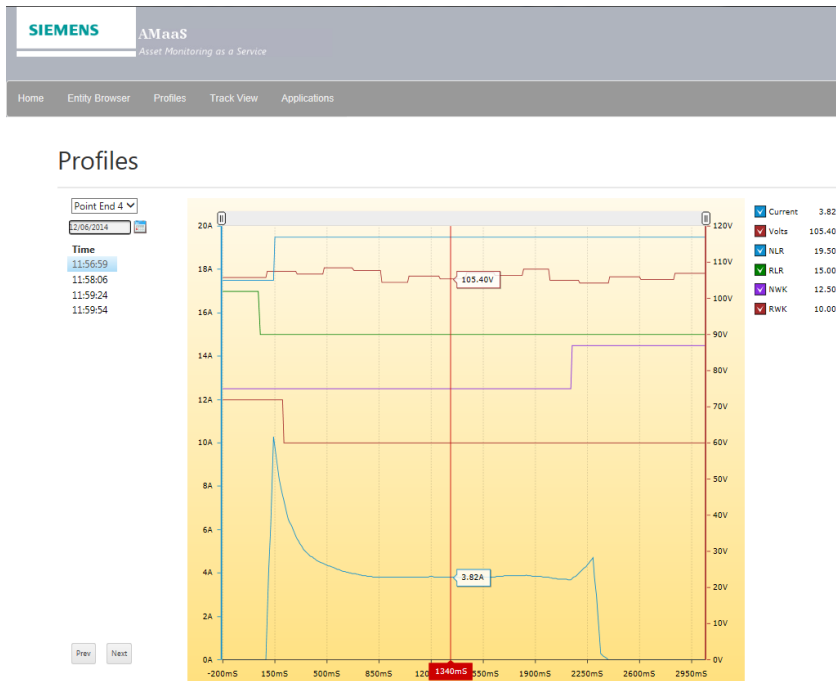


Figure 6.12: AMaaS Web Application Profiles View

### 6.2.5 Stages 3 & 4: Infrastructure Integration and Reasoning

Stages three and four of the AMaaS demonstrator integrate infrastructure data with monitoring equipment, such that inferred faults and conditions can be assigned to assets across the railway system. Most information monitoring systems accomplish this by contextualising through labelling of condition monitoring equipment in databases or ‘hard-coding’ in user interfaces—for instance Siemens’ WestCAD control system [115], which uses graphical ‘signalling schemes’ to contextualise train describer information. Linked data instead allows linking two datasets together dynamically, such that information from both sources is enriched, and changes in either data set are seen across systems.

- Asset monitoring information remains as in the first stage of the demonstrator—metadata is represented as RDF and sample data stored separately. AMaaS analysis worker nodes diagnose the condition observed for each measurement.

- Infrastructure and asset register information is mapped from proprietary formats into RDF, following design patterns defined in RaCoOn and extensions to AMaaS.

### 6.2.5.1 Mapping Infrastructure Data into RDF

A subset of infrastructure data was mapped into RDF from Network Rail Sectional Appendices. Track layout information from the vicinity of Coventry railway station (Figure 6.13) was selected and mapped according to RaCoOn design patterns, and the initial mapping structure defined in Table 6.3. The resulting RDF data was used for the AMaaS demonstrator.

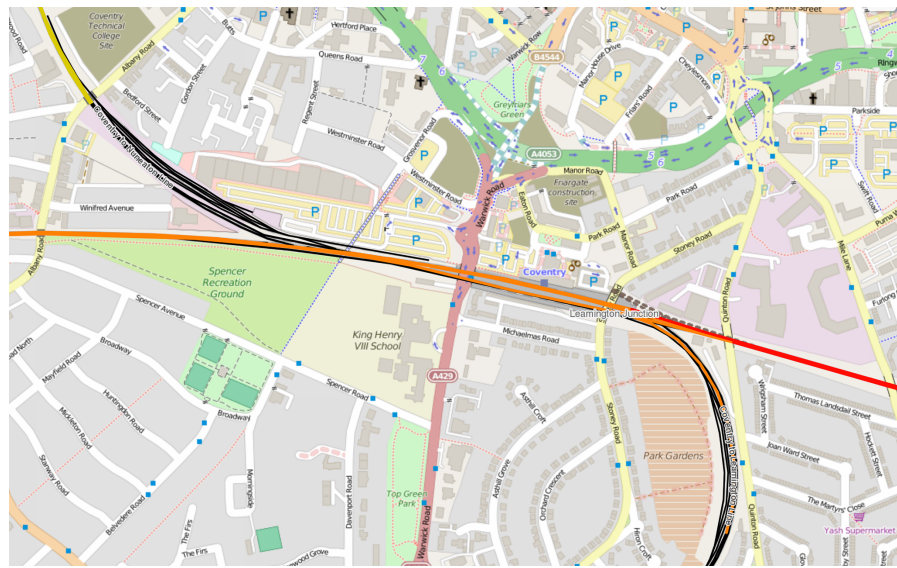


Figure 6.13: Railway Layout Around Coventry Station According to OpenStreetMap

Several `amaas:Points` entities were created at assumed junctions, some of which were later associated with demonstration Siemens Fast-flex PCM equipment, as described in the following sections.

### 6.2.5.2 Presenting Infrastructure Data Graphically

To convey inferred asset health information on rail infrastructure, a graphical depiction of the track layout was created for use in the AMaaS web application. Intended to show the use of the AMaaS system for signallers, the diagram is similar in layout to British Rail standard signalling and control interfaces<sup>11</sup>, and shows all elements that

<sup>11</sup> Care was taken to ensure that the diagram looked different from British Rail standard user interfaces for signalling systems, such that users did not expect similar functionality.

Table 6.3: RaCoOn Elements Used for AMaaS Infrastructure Mappings

Element	Description
<code>rcn:LineDetailNode</code>	Created for every junction on the trackmap
<code>rcn:LineDetailArc</code>	Created for every piece of track on the trackmap (between each node)
<code>rcn:LineLevelNode,</code>	
<code>rcn:LineLevelArcs</code>	Defined for major junctions
<code>rcn:startNode,rcn:-</code> <code>endNode</code>	Defined between all arcs and nodes
<code>rcn:Station</code>	Added for every railway station on map
<code>rcn:Points</code>	Asserted for (fictional) points machine entities



were mapped into RDF. A Scalable Vector Graphics diagram (shown in Section D.1) with embedded RDFa was written, containing entity URIs for each element. Site note: The Scalable Vector Graphics (SVG) diagram with embedded RDFa data is also available online<sup>12</sup>.

The *track view*, shown in Figure 6.14, provides a visual indication of the state of assets within the AMaaS system. After querying the AMaaS RDF store for asset status, each unavailable element is shown in red on the diagram. By interacting with elements, users can access further details of assets, including detailed fault information. Asset condition information is combined with infrastructure layout using OWL inference, allowing users to easily geographically locate faults.

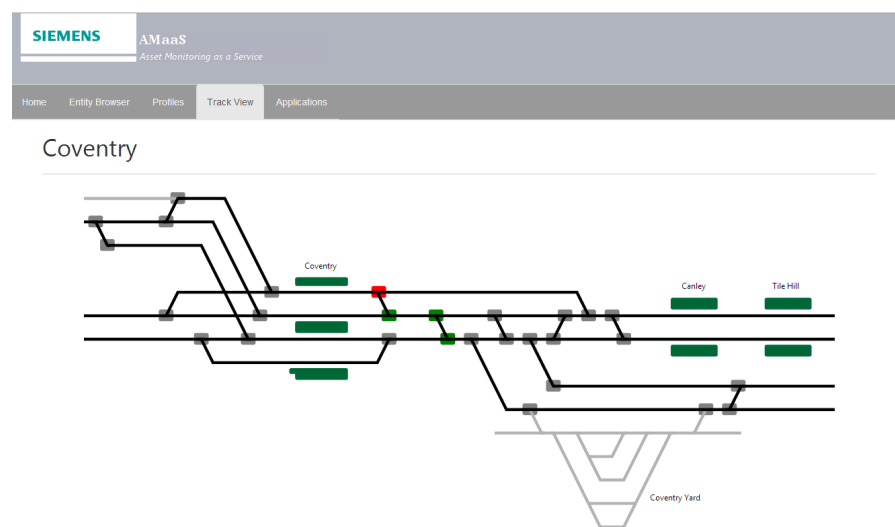


Figure 6.14: Screenshot of AMaaS Track View

### 6.2.5.3 Integration and Inference between PCM System and Infrastructure Model

In order to enable reasoning on data across the various systems in the AMaaS demonstrator—resulting in a richer knowledge of the system as a whole—links were manually created between the asset monitoring (PCM) dataset and the transcribed asset infrastructure information. Asset monitoring equipment was linked to infrastructure through the `u:monitors` relation, which, when used in conjunction with the

<sup>12</sup> <http://phd.jtutcher.co.uk/futro/tracklayout>

design patterns explained below, facilitates inference of infrastructure condition.

#### INFERENCE OF ASSET CONDITION

With new knowledge of the the relationships between assets and asset monitors in place, AMaaS can use links between systems to infer information about the assets themselves. The most useful of these link asset monitor observations to asset hardware, as illustrated in [Table 6.4](#):

Table 6.4: Example of Asset Data Contextualisation using Asset Monitoring System Data

Asset (Asserted)	Asset Monitor	Asset (Inferred)
Westlock Points Machine Installed: 1992 Maintained: 2014 Location: B29 6PQ	Points Monitor Observation: 20:00 Health: Faulty	Westlock Points Machine Installed: 1992 Maintained: 2014 Location: B29 6PQ <b>Observation: 20:00</b> <b>Health: Faulty</b>

Given two entities linked by a `u:monitors` relation, it follows that any observation made by a monitoring device is an observation of the health of the asset it monitors. The `amaas:associatedObservation` models this using the following DL axioms:

$$\begin{aligned} \text{associatedObservation} &\sqsubseteq \\ &\quad \text{monitoredBy} \circ \text{observedEvent} \\ \top &\sqsubseteq \forall \text{associatedObservation} \text{Observation} \end{aligned}$$

This first axiom states using a property chain that *any observation made directly by a device that monitors an asset, is an associated observation of the asset itself*, whilst the second restricts the property's range. The full turtle serialisation of this entity is shown in [Listing 6.1](#).

This pattern has the result that clients can query asset condition using the `amaas:associatedObservation` relation, and retrieve all asset observations related to the subject asset. In the demonstrator, this is exploited both in the track view and in the entity browser, with

```

:associatedObservation
  a owl:ObjectProperty ;
  rdfs:comment "An indirect observation of an entity (through
  ↪ another monitoring method etc) {@en} - this is how assets
  ↪ are linked to their observations through assetmonitors"@en
  ↪ ;
  rdfs:domain u:IndependentThing ;
  rdfs:label "Associated Observation"^^xsd:string ;
  rdfs:range u:Observation ;
  owl:propertyChainAxiom ( :monitoredBy :observedEvent ) .

```

Listing 6.1: OWL Axiom Asserting Observation Relation Properties

current status being shown using a SPARQL query.

As a result of the use of the 3D paradigm in RaCoOn, the creation of an OWL relation linking an asset to its most recent observation is not achievable in the AMaaS system. Rule reasoning, however, does allow this, and a `amaas:currentObservation` predicate was created using Stardog Rules to infer the value of the most recent observation on an entity. For the purposes of the demonstrator, the AMaaS application assumes that the most recent observation recorded is current, and does not consider validity periods for measurements. This issue would need to be considered further in a commercial implementation of the system. The Stardog rule used to implement this is shown in [Listing 6.2](#).

```

@prefix rule: <tag:stardog:api:rule:> .
[] a rule:SPARQLRule ;
  rule:content """
  PREFIX :<urn:test:>
  IF {
    SELECT ?asset ?condition (MAX(?tstamp) as ?date)
    where {
      ?asset amaas:associatedObservation ?o .
      ?o amaas:startTime ?tstamp .
      ?o amaas:calculatedCondition ?condition
    } GROUP BY ?asset
  } THEN {
    ?asset amaas:currentCondition ?condition
  }""" .

```

Listing 6.2: Stardog Rule for Inference of Current Asset Condition

## INFERENCE OF FAULT

Although diagnoses of health for differing assets can be queried using the patterns described above, the Track View application and alarms panel present in AMaaS require a more generalised diagnosis for each asset of ‘available/unavailable’. Inference was used to achieve this by creating a new set of axioms to define healthy and failed observations as follows:

$$\begin{aligned}
 \text{HealthyCondition} &\sqsubseteq \text{Condition}, \\
 \text{FailedCondition} &\sqsubseteq \text{Condition}, \\
 \text{AcknowledgedCondition} &\sqsubseteq \text{Condition}, \\
 \\ 
 \text{HealthyStateObservation} &\equiv \text{Observation} \\
 &\quad \sqcap (\exists \text{observedCondition}.\text{HealthyCondition}), \\
 \text{FailedStateObservation} &\equiv \text{Observation} \\
 &\quad \sqcap (\exists \text{observedCondition}.\text{FailedCondition})
 \end{aligned}$$

After first manually classifying existing individuals of type `amaas:Condition` as either `HealthyCondition` or `FailedCondition`, inference could be used to populate the classes `amaas:HealthyStateObservation` and `amaas:FailedStateObservation` with the appropriate observation instances. Using a backward-chaining reasoner (as discussed in [Section 2.6.4.1](#)), these classes become populated whenever an application queries for such individuals.

Membership of the `amaas:AcknowledgedCondition` class is asserted on observations which are still (potentially) in a faulty state, but whose status has been acknowledged by a system operator, to allow the application to discount observations that have already been dealt with.

As a result of the inference options above, AMaaS front end applications are able to directly query for available and unavailable assets, rather than rely on bespoke interfaces between systems. As the platform grows, its ontology can be further extended to encompass more cues to assert the status of an asset, and the front end application will continue to function without the need for significant updates.

## ASSET DEPENDENCIES

In many operational scenarios, the failure of a railway asset leads to other systems and infrastructure being put out of use due to interdependencies between components. For example, if a railway switch

fails to move when requested, routes cannot be set across it and the connected tracks may become unusable<sup>13</sup>. Failing utilities can also pose problems to maintainers—a power outage can adversely affect the operation of many parts of a system. The interdependence of assets was modelled in the AMaaS ontology, allowing users to instantly understand what parts of the railway are at risk or affected by a faulty component. Assets are first linked together using the transitive `amaas:dependsOn`, such that if asset A depends on asset B’s operation, and asset B depends on asset C, a fault with C will affect assets A and B. This example is shown in Figure 6.15, where the `amaas:dependentObservation` property is inferred on dependent assets.

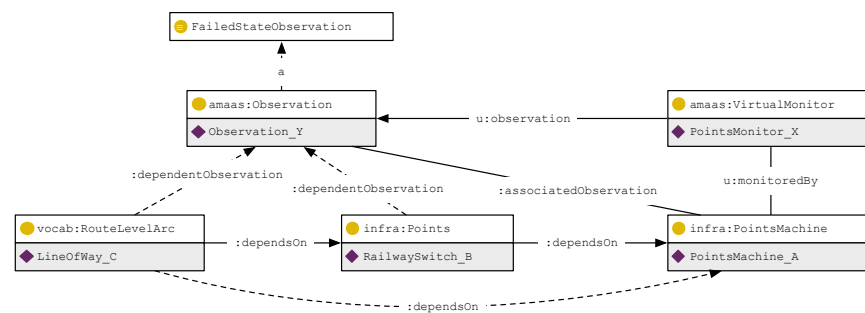


Figure 6.15: Asset Dependency and Fault Inheritance Design Pattern

In the AMaaS demonstrator, dependence reasoning of this type can be seen when a PCM fault is observed. Having introduced triples into the model that assert dependencies between railway switches and railway track, the track view displays unavailable track elements in red when points faults are observed. The same pattern could also be used to establish availability of rolling stock in a maintenance application, allowing staff to quickly establish the state of a train based on the information reported by its subsystems.

### 6.2.6 Stages 5 & 6: Integration of Timetable Data and Inference of Rolling Stock Faults

The final use case for the AMaaS demonstrator was to show how knowledge of railway scheduling, combined with infrastructure and asset monitoring data, can bring immediate benefit to signallers and maintainers. Firstly, the AMaaS asset monitoring system was extended to encompass a simulated Wheel Impact Load Detector, and then pub-

<sup>13</sup> In practice, points are often ‘locked’ in one position rather than taken out of use completely.

licly available Working Timetable (WTT) data was mapped into the AMaaS ontology. WILDs are used across the UK rail infrastructure to check for troublesome rolling stock wheels, and monitor trains as they pass over a set of sensors. By exploiting existing timetable data, the AMaaS system was able to infer which WILD observations corresponded to which timetabled train services, solving a known data integration problem for rolling stock maintainers and infrastructure managers across the industry.

#### 6.2.6.1 *The Case for Inclusion of Wheel Impact Load Detectors*

WILDs are used by railway infrastructure managers to warn of and detect destructive or dangerous wheel faults in operational railway rolling stock. They are installed as part of the wayside infrastructure, and usually take the form of instrumented sleepers, installed in sections of the network that experience large volumes of traffic. Strain sensors placed on the rail head measure the impact load of passing wheels, with large impact loads usually indicating wheel flats that require (sometimes immediate) corrective maintenance.

Current WILD systems in use within the UK, such as Gotcha [129] provide only information about the number of axles of each passing train, their spacing, and individual axle load. Measurements exceeding a dangerous threshold cause the system to register an alarm with the signaller, who then alerts train drivers to take their trains out of service. Vehicle maintainers who have access to the Gotcha data must then manually cross-reference train schedule data with wheel impact load records, identify high load measurements, and then infer which wheel(s) caused the alarm.

#### 6.2.6.2 *Adding Wheel Impact Monitors to AMaaS*

In the demonstration system, a simulated WILD sensor was added to the ontology as part of stage 4. Unlike the PCM hardware, data was not taken from Siemens devices and simulators, but instead derived from historic WheelChex<sup>14</sup> data already held at the University of Birmingham as a component of previous projects (see [Appendix D](#)). Given the pre-defined asset monitoring design pattern, extending the application to include WILD asset monitors was simple and consisted of the following stages:

---

<sup>14</sup> WheelChex is the brand name for a Wheel Impact Load Detector product sold by AEA Technology and fitted across the UK rail network.

1. A new RDF file, containing extensions to the AMaaS T-box was created. `amaas:EWILD` was created as a subclass of `amaas:EAssetMonitor`, as a class for all WILD devices.
2. An instance of `amaas:EWILD`, `amaas:CoventryWILD` was created, to represent the new WILD device, a sensor installed line-side outside Coventry rail station.
3. Manual XML to RDF mappings for WILD data were created as `amaas:Observation` instances. WILD data was placed in another RDF file and loaded into Stardog.

The use of `amaas:Observation` allows the AMaaS application to treat WILD measurements in the same way as it does PCM measurements, even though the underlying data differs between the two systems. The track view, for example, is able to display WILD faults without any further application logic being necessary.

For WILD measurements, post-analysis metadata is stored for each observation, including wheel impact values for each axle. Once the WILD observation is entered into the AMaaS system, worker nodes diagnose high wheel impacts based on threshold values, and a diagnosis asserted on each observation. High WILD observations are shown in the AMaaS front end in the same way as PCM observations, with faulty WILD events shown both in the track view, entity browser, and alarms panel.

#### 6.2.6.3 *Mapping Timetable Data*

To show that wayside WILD data can be correlated with rolling stock asset identities, a timetable of scheduled services was obtained, and used as an indication of actual train movements through time. Historic WILD measurements used in AMaaS were re-played to create artificial Class 390 wheel impact faults<sup>15</sup>, and ontology inference used to identify likely passing trains.

UK timetable data is provided by ATOC publicly, as ASCII data files. Using open source software tools and the RaCoOn ontology, the March 2014 Working Timetable was mapped into RDF for use in the AMaaS application.

---

<sup>15</sup> British Rail Class 390 rolling stock are commonly known as Virgin Pendolino trains, and form a large majority of high speed rolling stock travelling through the AMaaS demonstrator infrastructure.

## MAPPING TOOLS: CIFREADER AND OPENREFINE

Mapping of Working Timetable data was aided by the use of several software tools: CIFReader, by Tom Cairns<sup>16</sup>, MySQL, and OpenRefine<sup>17</sup> with the Digital Enterprise Research Institute (DERI) RDF extension<sup>18</sup>. Mapping was done in several stages:

1. WTT data was downloaded from ATOC<sup>19</sup> as a Common Interface Format (CIF) bundle, containing the entire UK's working timetable.
2. The CIF bundle was imported into CIFReader, and mapped to a SQL database
3. Data was loaded into MySQL for querying and exporting.
4. SQL queries were constructed and used to create single table exports of:
  - Any services timetabled to run through Coventry station
  - All railway station locations
5. These single tables were loaded into OpenRefine, and data cleansing applied. Location records were matched to existing location URIs in the AMaaS data store using OpenRefine's RDF reconciliation tool.
6. The DERI RDF mapping tool for OpenRefine, shown in [Figure 6.16](#) was used to build RDF mappings according to design patterns shown in [Chapter 5](#). Extensions to the core design patterns were made to include more specific timings (timetabled vs. public arrival and departure times) and further rolling stock information. Mappings from the CIF schema to RDF (via OpenRefine) are shown in [Table 6.5](#) and [Figure 6.16](#).

The resulting files produced by OpenRefine contained details of all services running through Coventry Railway station, and totalled around 20k triples. Services were linked (through RDF reconciliation) to existing infrastructure, allowing the ontology to infer which services travel over which pieces of infrastructure. [Figure 6.16](#) also shows the RDF mappings configured in OpenRefine.

<sup>16</sup> <https://github.com/swlines/CIFReader>

<sup>17</sup> <http://openrefine.org/>

<sup>18</sup> <http://refine.deri.ie/>

<sup>19</sup> <http://data.atoc.org/>



Table 6.5: Working Timetable Attribute Mappings

CIF Attribute	OpenRefine Attribute	Description
Record ID	id	Unique key per train service (timetabled train)
location_type	location_type (URI)	Origin, intermediate, or destination Point
tiploc_code	tiploc_uri	Location URI, reconciled from known stations
arrival	arrTime	Timetabled arrival time
public_arrival	pubarrTime	Published arrival time
pass	passTime	Timetabled passing time <sup>21</sup>
departure	depTime	Timetabled departure time
public_departure	pubDeparture	Published departure time
order	ordTime	Aggregated time for ordering nodes
	location_order	Location index on route
platform	platform	Platform number of calling station to be used

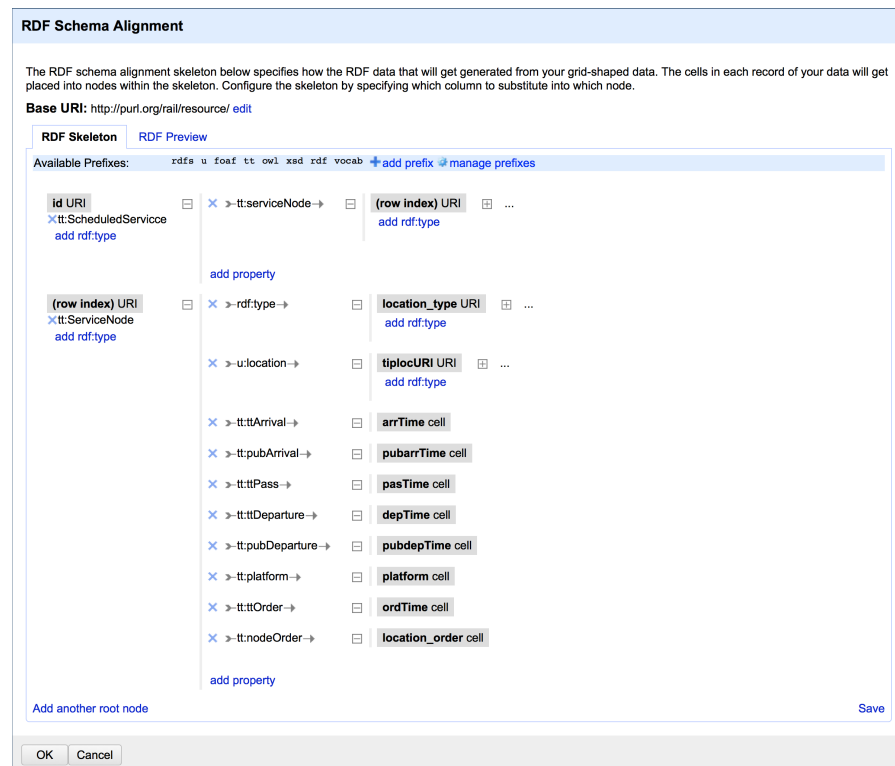


Figure 6.16: Screenshot of OpenRefine RDF Mapping Plug-in

## ROLLING STOCK DESIGN

CIF Working Timetable files also contain some information about rolling stock used on each train service, an extract of which is shown in [Table 6.6](#). For the AMaaS demonstrator, and in lieu of detailed information on the actual vehicle formations involved, some of the rolling stock identifiers shown in schedule data were mapped to representative rolling stock types, according to the rolling stock ontology elements discussed in [Chapter 5](#). The inclusion of formation information allowed additional logic to be used in the Train Finder view, and for trains to be identified based on axle count.

Table 6.6: Selected Records from CIF ‘Schedules’ Schema Table

unique_id	date_from	category	train_id...	power	tim- ing_load	speed	rs_id
G217...	08/12/ 13	OO	2O50	EMU		75	ME134900
L382...	08/12/ 13	OO	9A38	EMU	375	75	LO971800
C107...	09/12/ 13	XX	1P00	DMU	V	125	XC478000

Entities of type `rs:TrainSet` were created for several known Virgin Pendolino Class 390 train sets, using information gathered and mapped from Wikipedia. A `tt:TrainConsist` entity was created for several scheduled service entities to represent each relevant rolling stock ID, and this entity linked to both the `tt:ScheduledService` and the relevant `rs:TrainSet` trains. Each `rs:TrainSet` that was created comprised several carriages, enabling the number of axles of each vehicle to be deduced through rule reasoning. [Figure 6.17](#) shows a design pattern encompassing all of the above.

#### 6.2.6.4 Inference of Train Position and User Interface

The final step in linking wayside fault observations to passing trains was implemented in the AMaaS ‘Train Finder’ view, shown in [Figure 6.18](#).

The Train Finder is displayed by default as a view on WILD measurements, and can be accessed either through the Entity Browser or through the Track View interfaces. From the Train Finder view, users can select any one of a number of WILD observations, and view wheel

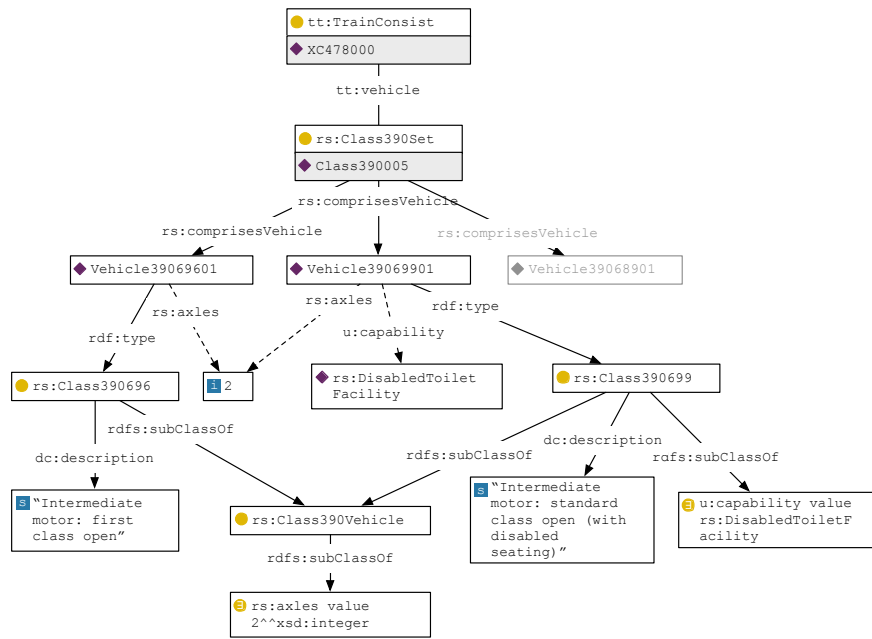


Figure 6.17: AMaaS Rolling Stock Design Pattern & Example Data

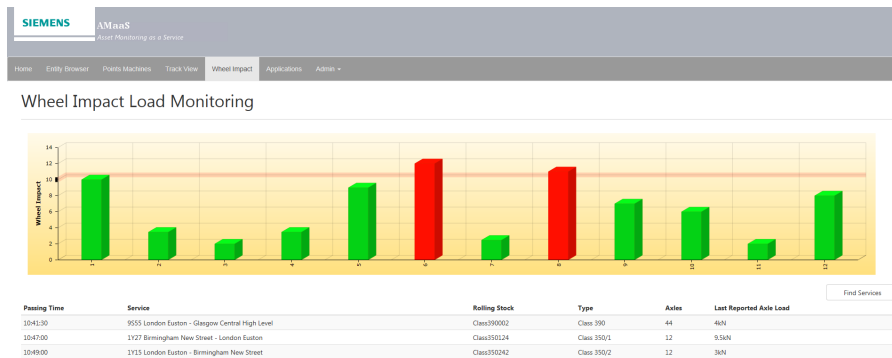


Figure 6.18: Screenshot Showing AMaaS Train Finder View

impact measurements on a graph. High wheel impacts are shown both in the ‘alarms’ panel, and highlighted in red in the Train Finder view. After selecting an event, clicking the *Find Trains* button finds one or more matching trains for each WILD observation. Each train can then be viewed in the entity browser for further analysis.

The Train Finder view works as follows:

- Upon loading the view with a valid WILD detector URI as arguments, a SPARQL query is triggered to load recent WILD observations.
- Key metadata from each observation is displayed in table form. Maximum wheel impact load, diagnosis, and number of axles are all included. Calculation of maximum wheel impact is performed by the triplestore at query time.
- A second query uses the location on infrastructure of the WILD device to calculate the traffic crossing it. The query filters traffic to those vehicles within a 2 minute window around the time of the measurement, and to those vehicles that match the number of axles seen by the WILD observation.
- Vehicle axle counts are calculated through a Stardog rule, using knowledge of a train’s composition found through the rolling stock ontology.
- Returned vehicle URIs are displayed on the web page, allowing users to validate WILD faults on rolling stock manually.

Although automatic assertion of rolling stock faults from WILD devices is possible using reasoning, this was not implemented for several reasons. Firstly, the trade-off between expressivity and performance must be considered when implementing ontology-based systems; implementing a rule to associate WILD observations with rolling stock is possible but is computationally expensive.

The backward-chaining architecture of the Stardog reasoner does not assert inferred triples when triggered, and so each lookup of rolling stock statuses using such a rule would require a new query to complete, slowing the system significantly. An alternative approach, likely to be undertaken in future work, is to implement a worker node to run such queries when new WILD observations are recorded, and assert diagnosis data on rolling stock directly. Whilst this does expose the system to inconsistency, it is significantly less computationally

expensive to perform. Using such an agent would also allow further vehicle identification algorithms to be employed to provide extra accuracy in ambiguous cases, such as where more advanced deduction approaches are required than those provided by OWL reasoning.

### 6.3 THE TRAIN LOCATOR APPLICATION

This section describes design and implementation of the second demonstrator created as part of the FuTRO Universal Data Challenge project, which explores methods for maximising the reach and utility of information from railway subsystems, and how to ensure compatibility between them through upgrades and evolution.

Using RTPI as a case study, this demonstrator shows how data from two separate passenger information systems can be combined to provide greater resilience during degraded service and increased accuracy during normal operation. It shows how such a system removes the need for data semantics to be encoded in each application, and how they can thus continue to function as input data sources change and evolve.

#### 6.3.1 *Motives for Second Demonstrator*

Until recently, customer information systems for railways in the UK were designed and run completely separately, with bespoke interfaces created between data sources as required, (shown in [Figure 1.2](#)). ATOC's DARWIN [209] will soon provide a unified customer information system and data model across the UK [53, 183], and whilst the system, due in April 2015, is undoubtedly an improvement on previous practise in the industry, the use of an ontology-based semantic data model for the same task could provide even greater benefits. The Train Locator application produced as the second FuTRO demonstrator shows a number of benefits of using such a model for this purpose, as well as mechanisms for aiding data integration.

The demonstrator aims to show how an ontology-based solution can be used as an alternative to traditional approaches in passenger information systems, and how they can greatly reduce effort and expense in maintaining such systems as they grow and change over time. The project considers the following aims and associated benefits:

- Use of an RDF-based system for describing live train running data, and for providing data to two passenger information applications:
  1. A mock-up ‘Live Departure Boards’ application, intended to mimic station platform displays and provide arrivals and departures information.
  2. A ‘Map View’, intended for travellers wishing to know their geographic location more accurately.
- Integration and enrichment of train running and passenger information data from disparate and/or upgraded information systems.
- Resilience of legacy systems in response to environmental change—use of ontology reasoning to allow legacy systems to run on data from new systems without modification.
- Performance in degraded operation—inference of ‘best guess’ location data for use in train location systems when high resolution positioning information is unavailable.

The Train Locator demonstrator is designed around a single railway route, where two separate (fictional) passenger information systems are used, each based on a single, disparate source data set. Most commercial customer information systems in the UK depend upon track circuits (or train describer systems) and mass detectors to find train locations and report that information to customers, but fail to take advantage of the higher locational accuracy provided to TOCs by GPS positioning devices present on some trains. The demonstrator utilises both track circuit and GPS mileage data, and provides a number of mappings between them for differing applications.

The application also identifies and documents fundamental data integration design patterns, with the expectation that such design patterns will also find uses in other enterprise and railway linked data applications.

### 6.3.2 *Design*

Like the AMaaS demonstrator, the Train Locator was designed according to a storyboard, to show a series of benefits by leading the user through an application in stages. As the FuTRO calls themselves placed very few restrictions on the team in terms of pre-defined requirements, the creation of storyboards allowed the expected benefits

of each demonstrator to be highlighted and worked towards. The storyboard for the Train Locator demonstration is shown in [Figure 6.19](#).

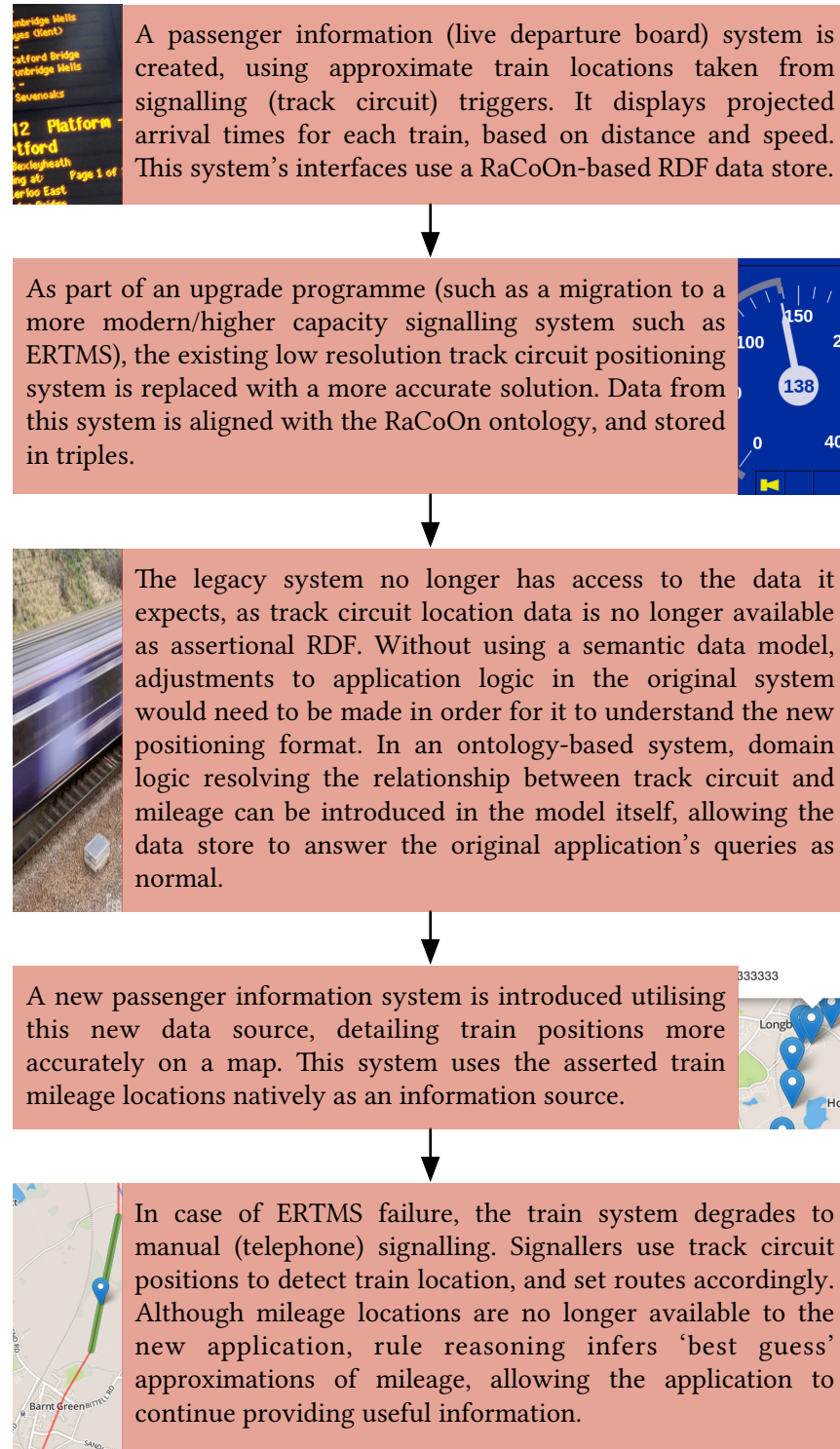


Figure 6.19: System Design Storyboard for FuTRO Train Locator Application<sup>22</sup>

### 6.3.2.1 *System Components*

Derived from the storyboard above, three separate application views were created. Each draws on data provided by the ontology system, and behaves differently depending on available input data.

- **The Legacy Departure Board System.** In this view, a user can select a train station and observe a simulation of a platform-based passenger information board, including departure point, destination location, and the scheduled/expected timings of services.
- **Train Position Map.** The train position map shows the map locations of each train on the network. Train chainage information stored in the ontology is mapped to GPS co-ordinates, and located on a satellite map of the countryside. Where chainage information is not available, ontology inference is used to fall back and infer position information by other means.
- **Entity Information View (Using Linked Data & Inference).** The final screen presents all knowledge available about an entity to the user, whether through assertion or inference. In the case of train services, inference is used to provide information about the rolling stock itself as well as the train service; for locations, reasoning provides additional information such as touching/neighbouring entities and line reference information.

To allow end users to observe how the system reacts to data sources becoming available or unavailable, several configuration options are presented on the main page of the web application as follows:

- **Track circuit data on/off.** This switch controls the system's access to legacy train location information, as typically provided by a Train Describer service.
- **High resolution positioning data on/off** controls access to more accurate, GPS or European Rail Traffic Management System (ERTMS)-style positioning data intended for use by the mapping view.
- **Reasoning on/off.** Configures the inference behaviour of the triplestore itself, in order that users can see the effect of the RDFS, OWL, and rule reasoning axioms across the demonstrator. Reasoning allows views to draw upon data from multiple

---

<sup>22</sup> Photo attribution: <https://www.flickr.com/photos/joshtechfission/8901326919/>, <http://www.ianbritton.co.uk/>



sources, and disabling it illustrates how certain views can only work with their original data source.

The behaviour of the system under differing data source availability is described in [Table 6.7](#), and each sub-application's behaviour is discussed further in [Section 6.3.3](#).

Table 6.7: Matrix of Train Locator System Behaviour Using Different Data Sources

	Track Circuit Data	Mileage (Moving Block) Data
Departure Board View	Asserted ('real') track circuit data.	Inferred track circuit data based on train mileage.
Train Map View	Inferred (approximate) train location based on known track circuit positions.	Asserted ('real') mileage data.
Train Map View (with all datasets available)	Rule reasoning chooses optimum location object for the task.	

### 6.3.2.2 System Architecture

The Train Locator system was designed with a simple system architecture based on the Model-View-Controller (MVC) software design paradigm. A data model containing static and dynamic (simulator) data is used as a single, centralised data source, whilst a server/client-based web front end presents information to users. The key components that make up this architecture are shown in [Figure 6.20](#).

- **Static Data & Train Locator Ontology.** RDF files for infrastructure/timetable data and OWL files for the Train Locator (TLOC) ontology are used for reference.
- **RDF Triple Store.** As in the AMaaS system, Clark & Parsia Stardog was used as a data store.
- **Separate Stardog Rules** were encoded as RDF, stored in the Stardog triplestore, and are activated and deactivated by the demonstrator.

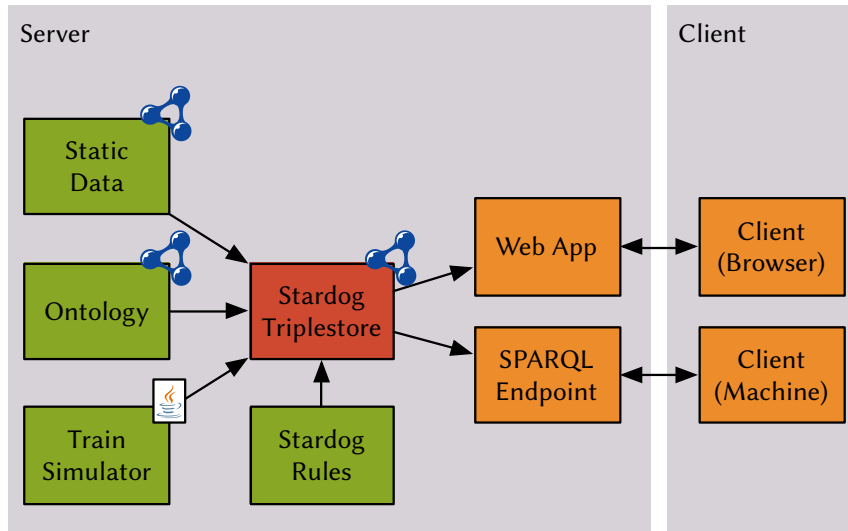


Figure 6.20: FuTRO Train Locator Key Components

- **Train Simulator.** The train simulator is a Java application that generates fictional train movements over a given railway using infrastructure information provided by the triple store.
- **TLOC Web Application.** The TLOC web application, contains both control and views for the demonstrator. It emulates the two demonstration use cases discussed earlier:
- **Client-side,** front end code presents information to users, fires SPARQL queries for updated information, and stores session data.

Interface technology choice was determined by FuTRO's requirement that demonstrators should be easily understandable to a wide railway user base. This requirement again led to the decision to build a web-based demonstrator, relying on users' computers and web browsers to act as interfaces to the system. Further technological decisions centred around understandability of the demonstrator's architecture for readers and ease of development; widely-supported and well-tooled technologies were adopted where possible.

#### NAMED GRAPHS

To show live integration between several data sources, it was necessary to keep simulated data (for mileage and track circuit locations) logically separate for each application, so users could explore the effects that the availability or otherwise of such information has on the

application, rather than by storing them in separate applications. This was achieved by storing data in four separate RDF graphs<sup>23</sup> within a single Stardog database. This approach facilitated simpler implementation of the system whilst allowing the logic used across it to remain similar to if physically diverse data stores were used.

### 6.3.3 *Front End Application Implementation*

The Train Locator web application provided all of the application logic and user interface presentation for the demonstrator. It consisted of two components: a server-side application to serve HTML pages (views) and data (API) to users' browsers, and client-side javascript code which contained further logic for querying dynamically updating data. When users requested a page in the demonstrator, the application responded by serving a web page to the user's browser, which then made further web requests to populate each page with current data from the Train Locator API. An example of this interaction is shown in [Figure 6.21](#), which demonstrates two requests to the 'mapper' view. To suit the rapid development of this demonstrator, widely used open source web application stacks and standardised interface protocols were used:

- **Express**<sup>24</sup> is a MVC javascript web framework built on Node.js<sup>25</sup>, and was used for the server-side element of the web application. Express includes built-in route and view management, and enabled easy development of both the sub-application views and the JSON API utilised in Asynchronous Javascript and HTML (AJAX) requests. The Stardog.js library<sup>26</sup> was used by the application to read RDF from the triple store.
- **HTML, CSS, and client-side javascript** were used to create the front end. When served to the web browser, these web pages load system data by issuing AJAX calls to the Train Locator API, using W3C standard interfaces and best practice techniques.
- The Train Locator API implemented **REST-ful** web services, allowing clients to issue it requests to retrieve data for each view. The API provides a number of services, each of which return data in a structure tailored to each client-side view.

---

<sup>23</sup> The ontology itself formed one graph, whilst static infrastructure data, track circuit data, and mileage data formed the other three respectively.

<sup>24</sup> <http://expressjs.com>

<sup>25</sup> <http://nodejs.org>

<sup>26</sup> <https://github.com/clarkparsia/stardog.js>

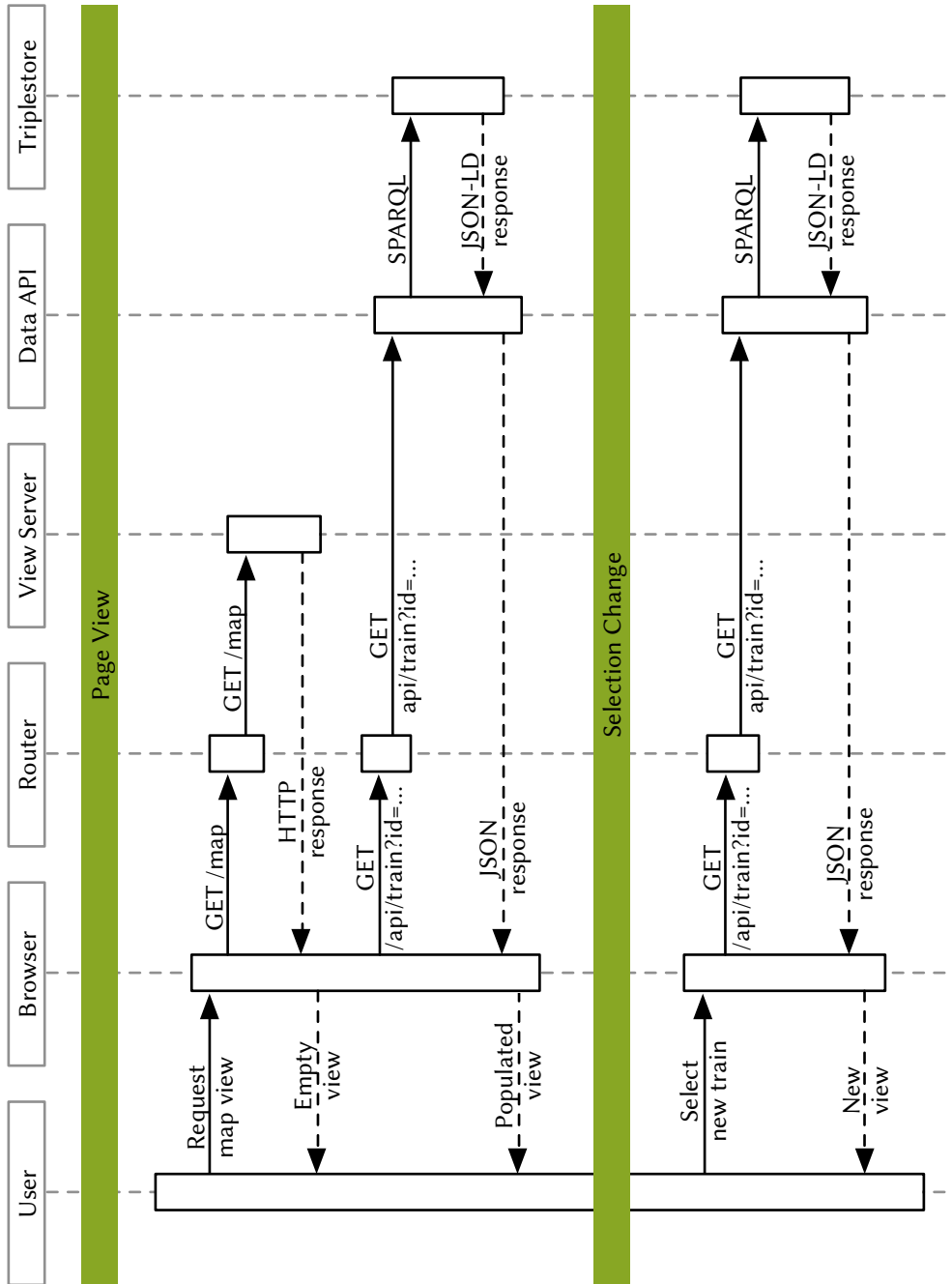


Figure 6.21: UML Sequence Diagram Showing High Level Data Flow for Request of Live Departure Board View

- Session cookies were used to keep track of application state, such that multiple users can use the system simultaneously without interaction.

Table 6.8: Views Provided by Train Locator Application

View	Function	Calls
/	Configuration page; entity view	/api/, /api/?id=...
/track	Track circuit view	/api/list?id=..., /api/?id=..., /api/track/?id=...
/train	<b>Live train mapper</b> view	/api/map/route/, /api/train/, /api/train/?id=...
/route	Track circuit & route view	/api/map/circuits/, /api/map/route/
/dep	<b>Departure boards</b> view	/api/dep/, /api/dep/?id=..., /api/?id=...

The web application includes five views, including one for each sub-application, as shown in Table 6.8. Each view requests data from the API, which ultimately calls the triple store. Table 6.9 lists and describes the application's supporting API functions, and results vary depending on the current session's selected data sources and reasoning capabilities.

As a result, the scenarios outlined in the application storyboard are shown in the application by alteration of data source availability by the user.

#### 6.3.4 Source Data and Simulation

The Train Locator application depends on a variety of static and dynamic data sources. Like AMaaS, the data supporting the application was acquired through both publicly available sources and through creation of fictional sources where real data would be inappropriate, difficult to obtain, or overly complex for the needs of the project. After deciding on one railway line to use as a demonstration scenario for the Train Locator application, requirements for data to drive both applications can be listed as follows:

- **Railway line topology**, including interconnections between stations, track circuit locations, and line direction.

Table 6.9: Train Locator API Calls and associated functionality

API Route	Query parameter	Function
/	(none)	Get current configuration
/	id [URI]	RDF entity information
/list	id [URI]	List all entities of type 'id'
/dep	(none)	Get list of stations for departure board view
/dep	id [URI]	Get station departure board info for 'id'
/map/ circuits	(none)	Get all track circuits in use by system (for map view)
/map/ route	(none)	Get all routes for map view
/track	id [URI]	Get details of track circuit with URI 'id'
/train	(none)	List all trains currently in system
/train	id [URI]	Get information about train with URI 'id'

- **Railway geography**, including line of way positioning, railway station positioning, and rolling stock positioning.
- Dynamic (changing) **rolling stock location** data.

Static data used by the system was mapped from various non-ontological resources into RDF form, as detailed in the following section.

#### 6.3.4.1 Infrastructure Data Model and Sources

The static infrastructure data used in the demonstrator was taken from three disparate railway data sources: the ATOC Working Timetable, Network Rail geographic location data, and additional crowd-sourced information taken from DBpedia. Using the RaCoOn ontology and a selection of RDF tools, data from each source was taken and combined into a single RDF file detailing all train stations (and their topologic connections) on the line between Birmingham New Street and Cardiff Central station<sup>27</sup>. The process that was undertaken is shown in [Figure 6.22](#).

An example of a fully enriched station is shown in [Listing 6.3](#). This information is shown to users on the live departure boards stage, and

<sup>27</sup> OpenRefine provides RDF entity resolution by attempting to match cell contents to the `rdf:labels` of entities in a knowledge base.

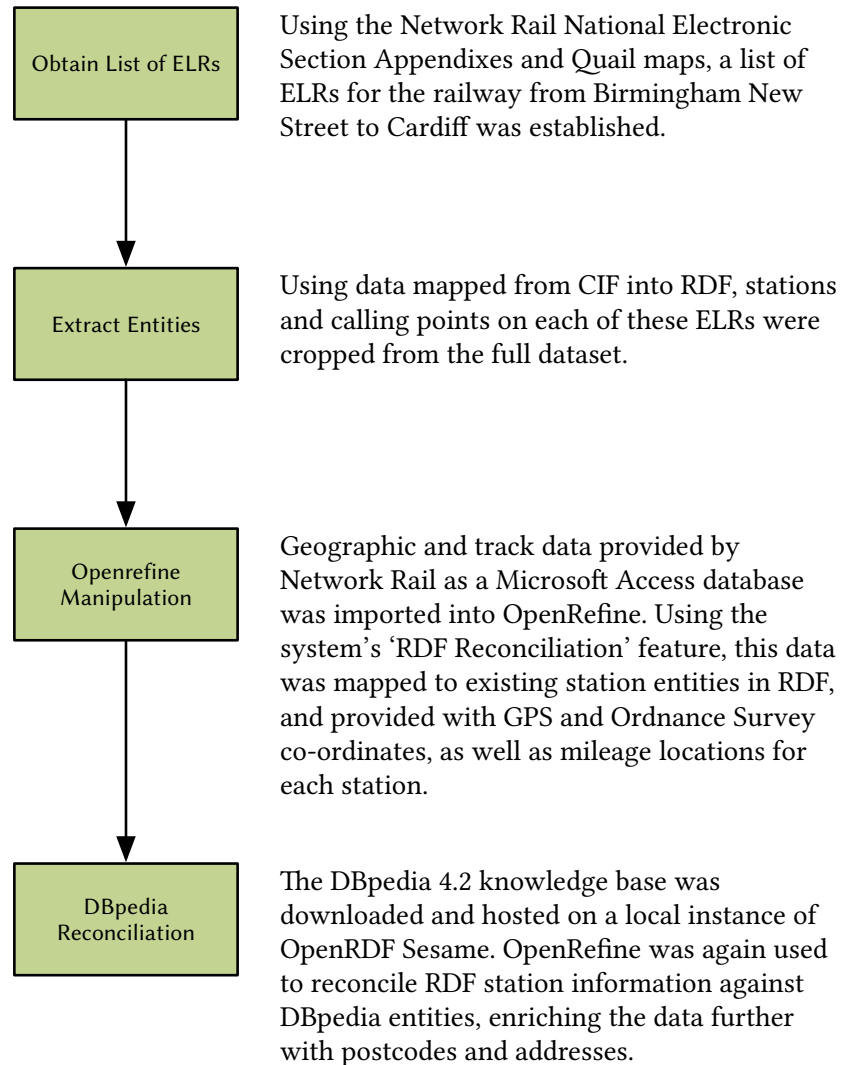


Figure 6.22: Train Locator Infrastructure Data Mapping Workflow

demonstrates the potential for further integration, for instance with multi-modal transport planning systems.

```

:CardiffCentralCDFStation
  a          vocab:Station ;
  rdfs:label "Cardiff Central"^^xsd:string , "Cardiff
→ Central"@en ;
  dc:description "Cardiff Central"@en ;
  is:tiploc   :TiplocCRDFCEN ;
  is:tiplocCode "CRDFCEN" ;
  is:county   "Cardiff - Caerdydd"^^xsd:string ;
  is:crs      :CRSLocationCDF ;
  is:district "Cardiff - Caerdydd"^^xsd:string ;
  is:govRegion "Wales - Cymru"^^xsd:string ;
  is:locationString "CF10 1EP, UK"^^xsd:string ;
  is:nlcCode "389900" , "3899"^^xsd:string ;
  is:nuts2Code "UKL2"^^xsd:string ;
  is:nutsRegion "East Wales"^^xsd:string ;
  is:owner     :TOCArrivaTrainsWales ;
  is:stanox    :StanoxLocation77301 ;

```

Listing 6.3: Extract of RDF Station Information from Train Locator Knowledge Base

#### 6.3.4.2 *Simulation Data Patterns*

The purpose of the train simulator is to provide a fictional source of train movement data to the application. Whilst initially this was attempted using a system that ‘re-plays’ timestamped RDF triples in and out of Stardog, the final approach taken was to physically model the track. Its output was designed according to the patterns and model described in [Chapter 5](#) and [Section 6.2.5.1](#), with several slight extensions. Its operation is beyond the scope of this thesis and will only be described briefly; a full code listing is available online<sup>28</sup>.

On startup, the simulator queries Stardog using Apache Jena<sup>29</sup> for infrastructure data about the line it is configured to simulate, finding track circuits and locations. It then creates objects for the railway system, track circuits, and trains, and positions a set number of trains randomly along the railway (in both up and down directions). The positions of these trains are then added as assertions into the Stardog data store using Apache Jena, and updated based on an assumed speed every ten seconds.

<sup>28</sup> <http://phd.jtutcher.co.uk/simulator>

<sup>29</sup> <http://jena.apache.org/>



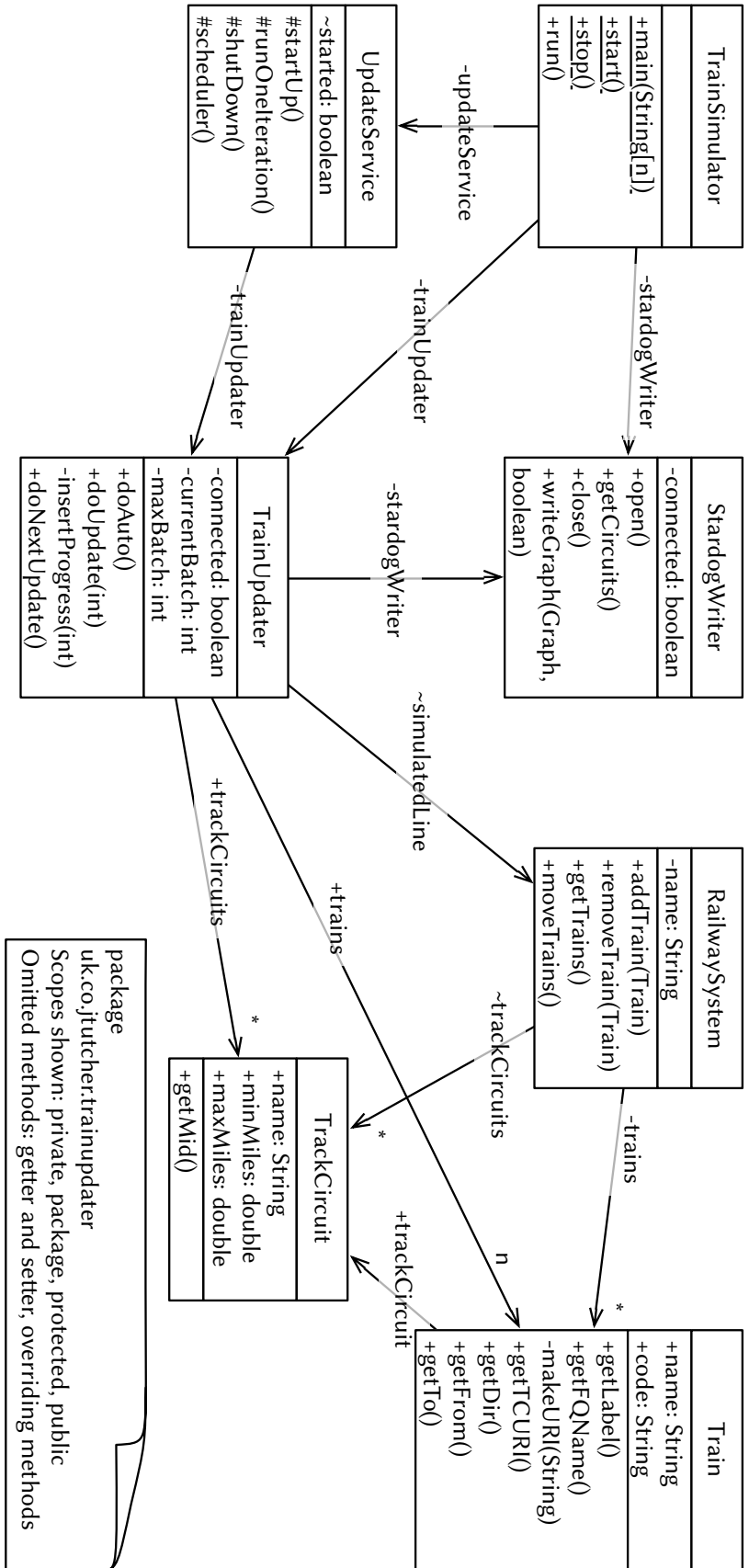


Figure 6.23: UML Class Diagram Showing Structure of FuTRO Train Simulator

On each update, the simulator erases the previous triples and asserts new ones; observation times are not recorded. Trains that reach the end of a line change direction and travel in the opposite direction. This simulation, whilst simple, provides enough information for the demonstrator to operate properly.

The simulator outputs positions to two named graphs in Stardog; one simulating the Track circuit data source, and one simulating the Mileage data source. The design pattern used can be seen in [Figure 6.24](#):

- Both types of assertions rely on the `tt:ServiceNode` design pattern. Each train is an instance of `tt:ServiceInstance`, and new `tt:ServiceNodes` indicating the location of the train are created at each update. The simulator also asserts train origin and destination information, and a headcode identifier.
- **Mileage** locations are asserted through the `rcn:RailwayMileage` entity and used by the map view application. They are stored in named graph `http://purl.org/ub/demo/graph/miles`
- **Track Circuit** locations are asserted through the `rcn:tcPos` object property and stored in named graph `http://purl.org/ub/demo/graph/track`. Each track circuit exists in the infrastructure data set, and is located on a railway line by its `is:-chainage` values.

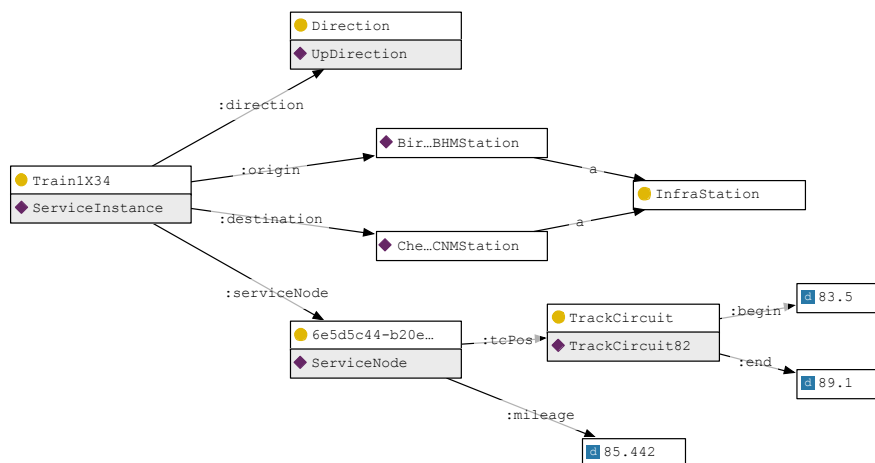


Figure 6.24: Design Pattern Used for Train Locations in Train Locator Demonstrator

### 6.3.5 *Live Departure Boards View & Reasoning*

The *Live Departure Board* view shown in [Figure 6.25](#) is the first of two sub-applications in the Train Locator demonstrator. It is a simplified mock-up of platform information screens found across the UK railway, and provides information to users on trains and times approaching a station. The Live Departure Board (LDB) view relies upon provision of track circuit and train describer data to present users with approximations of arrival times, and emulates the behaviour of typical RTPI platform level systems.

The LDB view can operate using either explicitly asserted track circuit data (generated by the train simulator) or, in its absence, using data inferred by the ontology derived from train chainages. In the case of an upgrade from fixed block to moving block signalling, this ability allows the application to continue functioning without modification, despite the physical changes to the railway network.

#### 6.3.5.1 *Legacy Operation*

The LDB view is shown in [Figure 6.25](#). To use it, a user first selects a train station (shown top) from a pre-populated list. The application requests data for that station from the web API, then displays station information and the upcoming departures. Each upcoming train's ID, origin, destination and expected arrival time is shown, and each entity can be clicked for further information.

This application calculates expected arrival times based on a static metric for time distance between stations. A new relation, `demo:-timing`, was added to each track circuit entity within the ontology, showing a time in minutes (between 0 and 70) that represent a train's expected progress through the system. The live departure board application queries the timings present at the trains' track circuit locations, and at the station itself, and displays the difference between these values as expected arrival time<sup>30</sup>.

The query employed in the application to retrieve trains' forecast arrival times is given in pseudo-code ([Listing 6.4](#)) and then as SPARQL in [Listing 6.5](#).

---

<sup>30</sup> This method of forecasting train arrivals is over-simplified and does not take into account varying train speeds, timetable, or any other factors. It is intended only to illustrate the basic concept, and is unlikely to reflect real the way in which real customer information systems predict timings.

Train Mapper | **Departure Boards View** | Route View | Train Locations (Standard) | Track Circuits

## Departure Board - Selly Oak

Select Station:  
Selly Oak

### Departure Board

Service	From	To	Expected
Train1X34	Cheltenham Spa	Birmingham New Street	7 mins, 20seconds
Train2N99	Birmingham New Street	Cheltenham Spa	24 mins, 4seconds
Train2U40	Cheltenham Spa	Birmingham New Street	34 mins, 27seconds

### Station Information

Label	Selly Oak
County	West Midlands
CRS	SLY
District	Birmingham
ELR	BAGS1
Location String	West Midlands
Location String	B29 6DW, UK
Nic Code	110500
Nomenclature Of Territorial Units For Statistics Code	UKG3
Nuts Region	West Midlands
Owner	London Midland Trains
Stanox	STANOX 65502
TIPOLOC	SELYOAK




Figure 6.25: Screenshot of Train Locator Live Departure Board View

```

SELECT all services WHERE
  each service has a current node [node] and a direction
  → [dir]
  each [node] has a location of track circuit [node_tc]
  [station_tc] is the track circuit located at current train
  → station
  [node_tc] and [station_tc] both have a timing value
  → attached
  [time] is [node_tc] minus [station_tc] or the other way
  → round, depending on direction
  FILTER out all trains that have passed the station already

```

Listing 6.4: Natural Language Query for Train Forecast in Train Locator Application

```

SELECT DISTINCT ?service ?label ?nodeloc ?nodelabel ?time ?dir
↪ ?from ?to WHERE {
  BIND (ex:current_station as ?station) .
  ?station is:tiploc ?tiploc .
  ?tiploc is:mileage [ u:value ?tmileobj ] .
  ?tc a is:TrackCircuit ;
  is:minLocation ?tiploc ;
  demo:timing ?sTime .
  ?service a tt:ServiceInstance ;
  tt:serviceNode ?node;
  rdfs:label ?label;
  tt:origin ?from ;
  tt:destination ?to ;
  tt:direction ?dir.
  ?node is:tcPos ?nodeloc .
  ?nodeloc demo:timing ?tTime ;
  rdfs:label ?nodelabel ;
  is:minLocation [ is:mileage [ u:value ?mileage ] ] .
  FILTER (((?mileage <= ?tmiles) && ?dir = tt:UpDirection) ||
↪ ((?mileage >= ?tmiles) && ?dir = tt:DownDirection)) .
  BIND ((?tTime - ?sTime) AS ?time)
} ORDER BY ASC (?service)

```

Listing 6.5: SPARQL Query for Train Forecast in Train Locator Application

The `is:mileage` property shown above represents the linear position of the entity along the railway track. `rcn:RailwayMileage` entities represent this value in miles and chains<sup>31</sup>. In normal operation, the system depends upon knowing track circuit locations to predicate times.

### 6.3.5.2 Upgraded Operation

A key aim of the Train Locator demonstrator was to show how reasoning can allow legacy systems to continue operating as the data structures around them change. In the case of the LDB view, this is demonstrated by the fictional upgrade of a train signalling/position reporting system. Instead of assertions about the track circuits occupied, the new system instead contains an exact mileage value for each train, linked to the `tt:ServiceNode` entity. Without the origi-

<sup>31</sup> Railway running distances are measured in *miles and chains*, as a result of the Victorian construction of the railway, during which time construction was measured using fixed length chains of 22 yards. This unit of measurement continues to be used, and values usually measured as ‘miles from London’. Chainage zero points are also recorded, and can be seen in Quail maps (diagrams produced for the UK railway industry that show infrastructure and signalling on a railway line, as well as locations for every major component).

nal data source, no triples of type `is:TrackCircuit` exist, and the query shown in the previous section returns no results. The application fails, as it is unable to use the `is:mileage` values associated with each train.

Whilst in this case it is trivial to modify the application to use new data, a real railway system with multiple subsystems (such as DARWIN) may require costly modification or the continued maintenance of the legacy data source itself. Using the existing semantic data model, a rule asserting the relationship between track circuit locations and mileage locations can be implemented, allowing systems to continue functioning with no modifications. This rule calculates the track circuit of a train based on its mileage, and is shown in [Listing 6.6](#).

```

IF {
  ?node a tt:ServiceNode .
  ?node u:location ?nodeloc .
  ?nodeloc is:elr ?elr .
  ?nodeloc is:mileageLocation [u:value ?mileage] .
  ?tcPos a is:TrackCircuitLocation .
  ?tcPos is:elr ?elr .

  ?tcPos is:minLocation [u:value ?min] .
  ?tcPos is:maxLocation [u:value ?max] .
  ?tc is:tcPos ?tcPos .

  FILTER(?mileage < ?max && ?mileage > ?min)
} THEN {
  ?node is:trackCircuit ?tc .
}

```

Listing 6.6: Stardog Rule to Assert Track Circuit Mileage in Train Locator Application

With reasoning enabled, the LDB application's query on `is:TrackCircuit` now triggers execution of the above rule, returning an inferred track circuit location for the train queried. Thus, the application is provided with the data it expects, and is able to continue running as normal.

### 6.3.5.3 Legacy Design vs. Semantic Design

In the LDB view, reasoning is used to aid compatibility and infer track circuit locations for trains. However, information is lost in this conversion and the accuracy of arrival estimates suffer due to the appli-

cation taking only explicit track circuits as location cues, as shown in Figure 6.26.

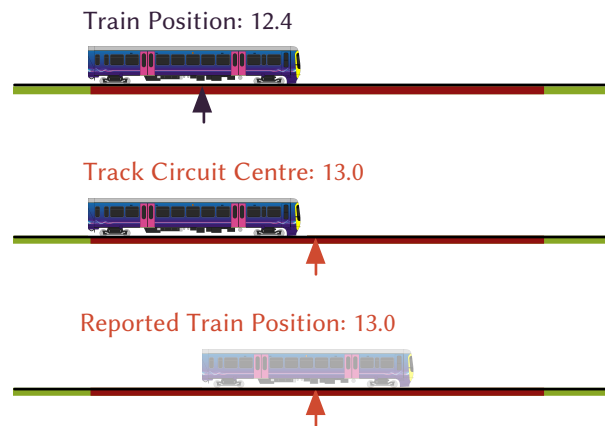


Figure 6.26: Demonstration of Train Location as Reported by Train Locator

A more accurate approach in a system known to be using semantic models would be to rely on more fundamental concepts for information retrieval. Consider that in the application logic, what is necessary for arrival time estimation (in this case) is the track length between the train and the station. The ‘mileage’ of the train could be requested instead, but this assumes that mileages and references will not change over time.

Another alternative could be to request `u:position`, use the model to establish and convert between measurement units, and present the result. Furthermore, a `is:lengthBetween` predicate could be used, providing the distance between two locations inferred by a rule that utilises route knowledge and railway network graph structure. Here, the trade-off between modelling correctness and pragmatism must again be considered, as discussed in Section 4.5.6.

### 6.3.6 Train Mapper View & Reasoning

The second sub-application, the *Train Mapper* view, allows users to geographically view a railway network and the position of trains within it as they progress in near real time. It illustrates how fine-grained chainage data generated by moving block control systems may be used for passenger information—users curious about their journeys or the journeys of others can look up trains and see their position rather than simply their expected arrival time into the next station. Its operation and interaction with the application ontology also shows

how semantic data models and reasoning can be used to allow such systems to continue operating in times of degraded operation, by providing ‘best guess’ data when a primary source is unavailable.

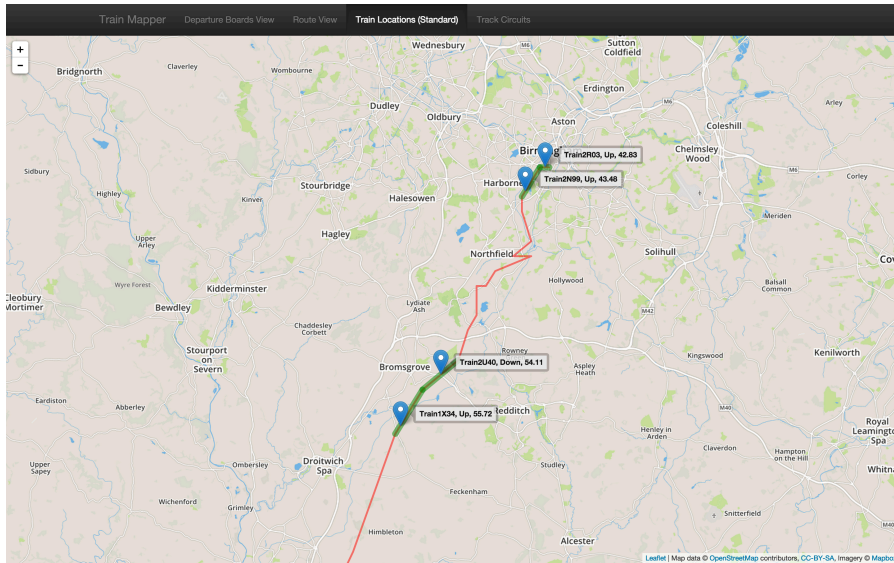


Figure 6.27: Screenshot of Train Locator Map View

The Train Mapper view is shown in [Figure 6.27](#). In normal operation, the map view uses chainage data provided by the simulator to establish train position down a route. The line of way is interpolated between known geographic locations given by the static infrastructure data (as explained in [Section 6.3.4](#)), and the position of track circuits and trains is calculated by the application based on mileage data. In this demonstrator, reasoning is used to establish the correct location data to use, rather than directly referencing asserted data, allowing the model to dictate what data is best used by the demonstrator.

#### 6.3.6.1 Prioritisation of and ‘Best Guess’ Location Data

Reasoning allows the data model to return the most accurate location information available for a particular asset: either an accurate mileage measurement for train location, or a ‘best guess’ assertion from other sources. A new design pattern and Stardog rules are used for this purpose, allowing the ontology to reason in a ‘closed world’ manner over data that is present. It is implemented as follows:

- A new predicate, `tloc:preferredLocation` is declared, representing the inferred most accurate location for an entity.



- Another predicate, `tloc:preferredOver`, is asserted across location classes that are preferred (more accurate) over other location classes.
- A Stardog rule infers the most accurate location, based on the transitive `tloc:preferredOver` properties as `tloc:preferredLocation` when triggered. The rule (in Stardog rules syntax) is shown in Listing 6.7:

```

IF {
  ?entity u:location ?location1 .
  ?entity u:location ?location2 .
  ?location1 a ?locationClass1 .
  ?location2 a ?locationClass2 .
  ?locationClass1 tloc:preferredOver ?locationClass2 .
} THEN {
  ?entity tloc:preferredLocation ?locationClass1 .
}
    
```

Listing 6.7: Stardog Rule to Present Preferred Data in Train Locator Application

This pattern for prioritising locations is illustrated in Figure 6.28, and by designing the Train Mapper to query using the `tloc:preferredLocation` predicate, the model is able to continually resolve the most appropriate location for the application. The demonstrator makes the assumption that whatever entity is returned will have an associated (asserted or inferred) mileage property, and by querying the entity’s `rdf:type`, it is possible to gain more information about the accuracy of the data.

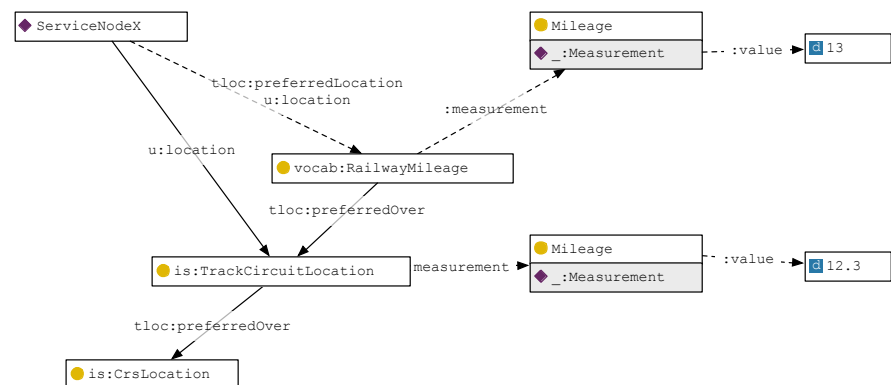


Figure 6.28: Pattern for Prioritising Knowledge in TLOC Ontology

The results of this reasoning can be seen in the Train Mapper by viewing the simulator with mileage data, track circuit data, and both. The design pattern used here is a generic ODP that could be used for other applications, examples of which are:

- Estimation of passenger numbers on trains arriving at stations for route prioritisation purposes. Low certainty estimates (based on headcode or service type) could be returned if more accurate estimates (train consist details, ticket reservations) are known.
- Fault reporting using fault hierarchies. In many situations it is useful for faults to be asserted at a particular level of granularity. `ex:DoorFault` may be a superclass of `ex:DoorMotorFault`. Using a variation on this pattern, the most specific fault type can always be displayed rather than the most generic class.

#### 6.4 SUMMARY

Through the description and implementation of the two use cases shown in [Section 6.2](#) and [Section 6.3](#), two novel applications of semantic data models in the railway domain have been demonstrated, and have shown that:

- Ontologies can be used to facilitate syntax-agnostic applications that adapt to changing sources and formats, as demonstrated by the Train Locator demonstrator.
- ‘Common sense reasoning’ can be used to enrich existing information and add business value in the context of an asset monitoring system.
- A pre-defined, standard ontology can be used to integrate data between diverse industrial software systems across the railway, as shown by both demonstrators.
- Systems built on the RDF/OWL technology stack can be implemented using well-supported software development tools, and can easily be incorporated with existing enterprise software systems.

The AMaaS project implemented a railway asset monitoring system using RDF to both represent its operational data and the state of the system itself. The use of a semantic data model to describe the AMaaS system architecture allowed new types of asset monitoring devices to be incorporated into the system with no change to the application

itself, through using OWL logic and reasoning. This illustrates the power of using this approach for data integration; as new and legacy components come and go, storing logic and semantics in a domain data model rather than hard-coding them into applications allows a great deal of flexibility, and reduces the effort needed to maintain and evolve the system.

The demonstrator described in [Section 6.3](#) further exploited this capability, and showed how logic captured in a semantic data model can be used in conjunction with reasoning to present information from a single data source to multiple data consumers in different ways. In abstracting semantics away from individual applications, legacy information systems can be kept running effectively, without any need for individual system components to be altered. Taking this idea further, the train locator demonstrator also showed how rules in OWL can be used to prioritise, canonicalise, and present multiple conflicting sources of data as a single source, such that applications can continue functioning in degraded states using ‘common sense’ reasoning, much as a human operator carrying out the same role might.

Both demonstrators show small proof-of-concept ideas that illustrate the real-life gains to be had from using semantic data models to contextualise data across heterogeneous systems. The work undertaken in the FuTRO project is currently being developed further in collaboration with several new railway asset monitoring suppliers, in order to build new ontology extensions and facilitate full data sharing between products from these vendors. A further summary of the work undertaken in these projects will be provided in the next and final chapter of this thesis.

## CONCLUSIONS DRAWN AND FURTHER WORK

---

The work undertaken in this thesis serves to provide a baseline set of methods, models, and reference implementations for those seeking to further implement semantic data models for data integration across industrial domains. This chapter provides a summary and set of conclusions drawn from the work undertaken in chapters four, five, and six. [Section 7.1](#) outlines the findings and key contributions made; [Section 7.2](#) discusses known limitations of the work undertaken and in the use of semantic models in the railway industry in general. Finally, [Section 7.3](#) suggests further work that could be undertaken based on the findings of this thesis, and describes projects already underway to do so.

### 7.1 KEY FINDINGS AND CONTRIBUTIONS MADE

This thesis has made novel contributions to knowledge in three areas, each written as one of the previous chapters: by providing a new methodology for creating industrial data models ([Chapter 4](#)), defining a new set of railway ontologies ([Chapter 5](#)), and by providing a set of demonstration implementations based on known use cases ([Chapter 6](#)). These contributions are discussed as follows.

#### 7.1.1 *RaCoOn Methodology*

In [Chapter 4](#), a new ontology engineering methodology was presented based on current state-of-the-art. It was designed to allow the creation pragmatic industrial domain models, and achieved this by describing the following:

- A method for scoping domain ontologies, based on analysis of existing resources and evaluation against possible use cases.
- A design approach based on iteration and scope refinement, and driven by existing resources and expert knowledge.
- A set of ontology design patterns to encourage the creation of modular and pragmatic OWL models.
- An approach to validation of industrial domain models based on state-of-the-art and ‘in use’ testing.

The methodology was used in the creation of the RaCoOn ontologies, and was therefore designed with the challenges of creating a railway domain ontology in mind. In particular, the main challenge addressed by the methodology described here was that of designing a domain ontology to support *future* interoperability between applications, rather than to address the needs of any particular initial system(s). Consequently, the methodology works from a set of very high level requirements, and draws upon guidance by domain experts and knowledge present in existing information resources to assist in both scoping and conceptualising models.

### 7.1.2 *The RaCoOn Ontologies*

[Chapter 5](#) detailed the design and implementation of a set of modular ontologies to support data interoperability in the railway industry based on requirements given in [Chapter 4](#). The chapter also outlined how these ontologies were created according to the methodology shown in the same chapter, and included the following contributions:

- A cross-domain ontology for pragmatically representing non-railway-specific concepts in RDF and OWL, such as temporal data, provenance, composition, and entity type.
- A set of design patterns for representing these concepts in subject-specific ontologies, addressing trade-offs between expressivity and functionality.
- A set of railway domain ontologies that model railway infrastructure, geography, and signalling concepts, as well as basic rolling stock and maintenance information
- Documentation patterns to express mappings of terms between non-ontological resources (principally RailML) and the core ontologies.
- Validation of the above models using approaches described in [Chapter 4](#), including descriptions of a set of validation workshops.

The ontologies presented draw on current state-of-the-art in modelling high level concepts, and reuses terminology and semantics from existing industrial models in the rail domain. Ideas for time representation, quantities and units, mereology, and provenance come from well-known ‘gold standard’ ontologies, and some railway specific concepts are drawn from RailML and RailTopoModel.

### 7.1.3 *The FuTRO Case Studies*

The final chapter of this thesis studied two use cases for data sharing across the railway industry. The first of these described in [Chapter 4](#) shows an ontology-based system for integrating heterogeneous remote condition monitoring information, developed in collaboration with Siemens, using data from existing products. This section presented novelty in several areas:

- Description of an AMaaS task ontology and set of patterns for representing railway asset monitoring data in OWL, addressing the following subjects:
  - Topology of the AMaaS system, allowing sensors and measurements to be abstracted away from physical or logical organisation of lineside equipment.
  - Measurement and diagnostic (good/bad) data, and methods for pointing to fine-grained data stored in other repositories.
  - Data acquisition equipment modelling, allowing integration of new types of equipment using existing semantics.
- Demonstration of the use and extension of RaCoOn infrastructure concepts and DL reasoning to map condition monitoring equipment and state onto railway infrastructure. The AMaaS system showed how sensors in the asset monitoring system could be associated with track in a network model and infer network state and condition.
- Demonstration of the extension of the AMaaS system to include new Wheel Impact load Measurement (WILM) equipment and data, using existing applications.
- Extension of the RaCoOn ontology to include timetable information, and mapping of real-world data into the newly created subdomain model.
- Demonstration of fault and condition inference on rolling stock using location information and rule reasoning, allowing the integration of data between the infrastructure and rolling stock subdomains.

The second case study showed proof-of-concept for data integration in a railway passenger information system. Utilising ontology and rule reasoning, it showed how diverse and legacy passenger information systems could be integrated and work together, and made the following contributions:

- Extension of the RaCoOn ontology to store train running and passenger information data, using existing infrastructure concepts.
- Implementation of an RDF-based application to store and display train running and passenger information data from two heterogeneous data sources at different levels of accuracy (GPS-based and track circuit-based).
- Design and implementation of a set of rules to allow conversion of location information between these two levels of accuracy, and subsequent representation as common RDF concepts.
- Web-based demonstration of how this integrated data can be used to power a ‘legacy’ system, using ontology reasoning to present data from either source in its expected representation.
- Implementation of design patterns and rules to allow accurate information to take precedence over less accurate information.
- Demonstration of these rules in a web application, showing how systems continue functioning in a ‘graceful degradation’ state when high accuracy information is not available.

## 7.2 LIMITATIONS OF APPROACHES TAKEN

When discussing new information technology techniques and processes, it is tempting to describe them as ‘silver bullets’—solutions that magically solve all of an organisation’s needs with seemingly little effort or risk. As well as inherent limitations in the capabilities of the technologies adopted in this thesis, the methodologies and ontologies designed have some limitations. These are outlined below.

### 7.2.1 *RaCoOn Methodology*

#### SCALABILITY

The methodology described suits modelling of domains that can be broken into submodules, each of which can be feasibly understood and managed by human ontology engineers. The iterative approach shown relies upon manual processes, and may constitute a huge amount of work for those wishing to model areas requiring large vocabularies in domains such as medicine. In these circumstances, semi-automated approaches to knowledge acquisition should be used.

#### RELIANCE ON DOMAIN EXPERTS

The RaCoOn methodology requires that domain experts are available to steer the direction of modelling and act as sources for model knowledge, and are available to interact directly with ontology engineers. It is realised that such experts may not be available in some circumstances, and that in some organisations a collaborative approach to ontology building may be more appropriate. In these cases, collaborative methodologies exist to assist with ontology building, and may be combined with the methodology described here.

#### VALIDATION

The RaCoOn methodology is designed for the creation of non-application-specific domain models, and does not assume an exact scope at project start. As such, validation against low level functional requirements is not undertaken, and objective assessment of ontology coverage is not possible. The validation approaches suggested do provide a measure of domain coverage and fitness-for-purpose, but do not utilise formal requirements validation technique such as those suggested in the NeON methodology[203].

#### CHANGE CONTROL

Change management of ontology modules is not explicitly addressed as part of the described methodology. The use of subdomain-specific disjoint modules provides a degree of reusability and maintainability, as well as an easy way of asserting ownership, but does not guarantee compatibility between modules as they change and evolve. It is likely that software engineering change control techniques such as package management and build control could assist in this regard.

#### 7.2.2 *The RaCoOn Ontologies*

#### EXPRESSIVITY

Many of the benefits of using ontological models in information systems arise from their ability to infer new knowledge from existing data. Whilst some of this inference can be done in an efficient manner, much of the OWL DL language requires reasoning algorithms that do not scale to large volumes of data. In the RaCoOn models, a trade-off between reasoning performance and scalability is made in order to provide a 'best fit' point, and DL axioms are only asserted



where they add significant value to the model.

The models do not provide full coverage of domain semantics in order to allow for more efficient reason in most situations. They therefore exhibit two limitations owing to their chosen expressivity:

- Some applications may require axioms that are not present. These must be implemented in separate models or in application logic, which lessens the data exchange advantages of the models.
- Applications requiring DL reasoning may not perform well at scale. Axioms present in the ontology may not be required by the applications, and alternative approaches to computing inferences may be required.

Ongoing research in ‘web-scale’ reasoning techniques combined with state-of-the-art RDF graph storage technology is likely to bring increased performance in the future, but using reasoning over large datasets using highly expressive OWL models is currently a significant challenge. It is suggested that a combination of DL materialisation and rule reasoning using less expressive profiles may be appropriate for applications using the RaCoOn ontologies at scale.

#### COVERAGE

The RaCoOn models do not provide coverage of knowledge of the whole railway domain, for practicality reasons and owing to the diminishing benefit of modelling parts of the industry that may have less use for a model to support data exchange. Future mapping of other railway domain models such as the RFA may encourage further adoption, and could be undertaken on demand as necessary.

#### REUSABILITY

A stated aim of the RaCoOn models was to provide reusable domain models for other applications, whilst maintaining a practical model that could be used in software systems. Thus, a number of parts of the models exchange full semantic correctness for brevity in representation. Whilst it is likely that the semantics expressed in the models will be sufficient for most applications (or extendible as necessary), there may still be some instances where this is not the case.

#### CONFLICTING DATA AND VALIDATION

The majority of design patterns specified in the RaCoOn ontologies do not enforce or check the validity of data input. This is by design;

the assertion of such axioms could severely impede the performance of open world DL reasoners. However, it places a dependency upon correct and non-conflicting data being entered in the models, as no mechanism for resolving conflicting data is provided. In an application, rule reasoning could be used to assert further (closed world) constraints axioms over knowledge requiring validation, and could prevent or control the entry of conflicting data.

### 7.2.3 *Other Limitations*

#### 7.2.3.1 *Architecture and Distribution*

The practical implications of implementing systems in RDF and OWL at scale are only briefly considered in [Chapter 6](#), where the availability of efficient, federated query systems are assumed. This is not a trivial assumption, and the architecture of a multi-stakeholder system would need to be carefully considered prior to implementation.

Data sharing on the semantic web shares this limitation: to make use of another dataset, one must either download it in its entirety to interact with locally, or rely on the provider's processing power and availability using a SPARQL endpoint. Possible alternative approaches include bespoke Service-oriented Architectures as used in [Chapter 6](#), more HTTP-based linked data best practice solutions such as Apache Marmotta and the Linked Data Platform [229], and Linked Data Fragments [215], a system which uses smaller data dumps to facilitate local querying of federated data.

#### 7.2.3.2 *Security, Data Value, and Business Case*

The issue of data ownership and value is touched upon in [Chapter 1](#). Whilst this thesis assumes a set of use cases and stakeholders that have a desire to cooperate and share data, many real-world stakeholders suffer disincentives from doing so. Although an argument for Open Data asserts that the value of sharing data often exceeds the value lost by giving it away 'for free', this argument may not hold in enterprise environments. From progress in the railway domain seen so far, it is likely that the business case for data sharing will come through mandate from key stakeholders rather than through a voluntary consensus, as witnessed by progress arising from recent EU interoperability legislation [58].

A related issue is that of data security. Secure transmission of data itself is not considered in this thesis, but is likely to draw upon known approaches for TCP/IP stacks when implementing Service-oriented Architectures or federated queries. A more challenging prospect is

that of access control to RDF data stores. In a multi stake-holder system, organisations may wish to restrict the actions of certain actors across a dataset, for instance by providing only a subset of data to external parties, whilst allowing internal users to view or modify all data. Currently, no standard approach for providing these rules exist, although proprietary approaches are implemented in triplestores such as Stardog<sup>1</sup>, Virtuoso<sup>2</sup>, and Oracle<sup>3</sup>, and other methods are presented in the literature [158].

### 7.3 PLANNED AND POSSIBLE FURTHER WORK

The initial outputs of this PhD lend themselves to a number of further projects in two interlinked areas:

- Development of subdomain and application railway models.
- Development of architecture, platforms, and tools to support future deployment.

The work undertaken in this thesis aimed to create both a domain model and a methodology for building similar such models in the railway industry. As such, a key piece of future work is the extension of this ontology to provide wider coverage of other railway application areas, and to build data integration implementations based on further industrial use cases.

#### 7.3.1 Possible Extensions

In addition to projects underway, this thesis raises new questions that could be addressed in further work. Suggestions for such work are as follows:

- **Application of state-of-the-art stream reasoning to support real-time decision support in railway systems.** The use of stream reasoning to integrate heterogeneous realtime data sets has recently been shown for applications in Smart Cities [206] and sensor networks [46, 227] amongst other areas. The use of RDF streams could allow real time decision support in railway applications, by integrating data about timetabling, maintenance, and external transportation systems.

<sup>1</sup> [http://docs.stardog.com/#\\_permissions](http://docs.stardog.com/#_permissions)

<sup>2</sup> <http://docs.openlinksw.com/virtuoso/rdfgraphsecurity.html>

<sup>3</sup> <http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/rdfsemantic-graph-1902016.html>

- **Formalisation of recent EU-mandated interoperability conceptual models into RDF and OWL.** As described in [Chapter 3](#), railway undertakings around Europe are currently developing systems to allow interoperability of some passenger, infrastructure, and freight information according to a common conceptual model. The use of OWL rather than XML for data exchange in these areas would allow companies to additionally share data not demanded by the specification, such that other stakeholders with an interest in the additional data could use it. Extension of RaCoOn to fully encompass the TSI TAP and TAF frameworks would facilitate this use.
- **Use of rail core ontology and transformed UK railway data in Open Data applications.** Although not widely discussed in this thesis, applications and research into using railway data for public services such as journey planning is growing, with services such as Realtime Trains<sup>4</sup> and the Transport API<sup>5</sup> using openly available rail data to provide passengers with services not offered by railway companies themselves. These applications provide benefits to both the rail industry and application developers, and the development of linked datasets based on RaCoOn ontologies could aid reuse in the public domain even further.

### 7.3.2 *Work Currently Underway*

As a direct extension to the work discussed in this thesis, a two year collaborative industry-funded project is now being undertaken as a continuation of the FuTRO feasibility study described in [Chapter 6](#). The new project will implement several ontology extensions and implement data sharing applications in three new areas, and also aims to increase awareness and skillsets in RDF/OWL modelling through work with industry project partners and development of tutorial materials. Key outputs will address several of the limitations described in [Section 7.2](#):

1. A reference software architecture for the fusion of railway data in RDF/OWL.
2. A set of standard processes, tutorials, and reference implementations illustrating the use of this architecture and the rail core ontologies.

---

<sup>4</sup> <http://www.realtimetrains.co.uk/>

<sup>5</sup> <http://www.transportapi.com/>

3. A set of ontology extensions and implementations for a range of rail industry asset types:
  - Track-side Equipment (with Siemens ITS)
  - On-board Equipment (with Arrowvale Electronics)
  - Traffic Management (with IRG)

The project is supported and funded by Future Railway, and collaborators include and outputs from it will be disseminated across the railway industry and academia. It is likely that the resulting increase in Technology Readiness Level for the application of semantic web tools and technologies across the railway will allow stakeholders to begin implementing such systems in earnest, and to realise the benefits stated throughout this thesis.



## LIST OF CODE AND ONTOLOGIES HOSTED ONLINE

---

Rather than providing full code and OWL listings in print, source code for many of the applications and ontologies designed as part of this thesis are available online, and can be accessed by the following URLs:

- <http://phd.jtutcher.co.uk/examples/ontologies>
  - RaCoOn Ontologies (Redirect)
- <http://phd.jtutcher.co.uk/examples/railml2owl>
  - Automated RailML XSD2OWL Mapping Results
- <http://phd.jtutcher.co.uk/futro/tracklayout>
  - AMaaS RDFa SVG Track Diagram
- <http://phd.jtutcher.co.uk/simulator>
  - FuTRO Train Locator Simulator Application (in Java)



## REFERENCE DIAGRAMS AND LISTS

## B.1 LIST OF CURIE PREFIXES USED THROUGHOUT THESIS

The following list shows all URI prefixes used throughout this thesis as well as their corresponding full URIs. For new ontologies, a list of high level namespaces is given in [Table B.1](#).

Table B.1: Ownership Denoted by URI Namespaces

Organisation or ontology	Domain or URI
Siemens /AMaaS	<a href="http://amaas-siemens.com/">http://amaas-siemens.com/</a>
RaCoOn Railway Ontologies	<a href="http://purl.org/rail/">http://purl.org/rail/</a>
RaCoOn Cross-domain Ontologies	<a href="http://purl.org/ub/">http://purl.org/ub/</a>

Prefix	Full URI
amaas:	<a href="http://amaas-siemens.com/ontology/">http://amaas-siemens.com/ontology/</a>
amres:	<a href="http://amaas-siemens.com/resource/">http://amaas-siemens.com/resource/</a>
co:	<a href="http://purl.org/co/">http://purl.org/co/</a>
core:	<a href="http://purl.org/rail/core/">http://purl.org/rail/core/</a>
core3d:	<a href="http://purl.org/rail/core/3d/">http://purl.org/rail/core/3d/</a>
corecv:	<a href="http://purl.org/rail/core/vocab/">http://purl.org/rail/core/vocab/</a>
dbp:	<a href="http://dbpedia.org/ontology/">http://dbpedia.org/ontology/</a>
dc:	<a href="http://purl.org/dc/elements/1.1/">http://purl.org/dc/elements/1.1/</a>
dcam:	<a href="http://purl.org/dc/dcam/">http://purl.org/dc/dcam/</a>
dct:	<a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a>
doc:	<a href="http://purl.org/ub/doc/">http://purl.org/ub/doc/</a>
dul:	<a href="http://www.loa-cnr.it/ontologies/DUL.owl#">http://www.loa-cnr.it/ontologies/DUL.owl#</a>
ex:	urn:example
foaf:	<a href="http://www.loa-cnr.it/ontologies/DUL.owl#">http://www.loa-cnr.it/ontologies/DUL.owl#</a>
geo:	<a href="http://www.opengis.net/ont/geosparql#">http://www.opengis.net/ont/geosparql#</a>
gml:	<a href="http://www.opengis.net/ont/gml#">http://www.opengis.net/ont/gml#</a>
is:	<a href="http://purl.org/rail/is/">http://purl.org/rail/is/</a>



owl:	<a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a>
po:	<a href="http://www.loa-cnr.it/ontologies/DUL.owl#">http://www.loa-cnr.it/ontologies/DUL.owl#</a>
prov:	<a href="http://www.w3.org/ns/prov#">http://www.w3.org/ns/prov#</a>
qudt:	<a href="http://qudt.org/schema/qudt#">http://qudt.org/schema/qudt#</a>
rcn:	<a href="http://purl.org/rail/core/vocab/">http://purl.org/rail/core/vocab/</a>
rdf:	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a>
rdfs:	<a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a>
res:	<a href="http://purl.org/rail/resource/">http://purl.org/rail/resource/</a>
rs:	<a href="http://purl.org/rail/rs/">http://purl.org/rail/rs/</a>
rule:	tag:stardog:api:rule:
sf:	<a href="http://www.opengis.net/ont/sf#">http://www.opengis.net/ont/sf#</a>
skos:	<a href="http://www.w3.org/2004/02/skos/core#">http://www.w3.org/2004/02/skos/core#</a>
time:	<a href="http://www.w3.org/2006/time#">http://www.w3.org/2006/time#</a>
tloc:	<a href="http://purl.org/ub/demo/ontology/">http://purl.org/ub/demo/ontology/</a>
tt:	<a href="http://purl.org/rail/tt/">http://purl.org/rail/tt/</a>
tzont:	<a href="http://www.w3.org/2006/timezone#">http://www.w3.org/2006/timezone#</a>
u:	<a href="http://purl.org/ub/upper/">http://purl.org/ub/upper/</a>
u3d:	<a href="http://purl.org/ub/upper/3d/">http://purl.org/ub/upper/3d/</a>
u4d:	<a href="http://purl.org/ub/upper/4d/">http://purl.org/ub/upper/4d/</a>
ucv:	<a href="http://purl.org/ub/upper/cv/">http://purl.org/ub/upper/cv/</a>
unit:	<a href="http://qudt.org/vocab/unit#">http://qudt.org/vocab/unit#</a>
vocab:	<a href="http://purl.org/rail/core/vocab/">http://purl.org/rail/core/vocab/</a>
wgspos:	<a href="http://www.w3.org/2003/01/geo/wgs84_pos#">http://www.w3.org/2003/01/geo/wgs84_pos#</a>
xml:	<a href="http://www.w3.org/XML/1998/namespace">http://www.w3.org/XML/1998/namespace</a>
xs:	<a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a>
xsd:	<a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a>

## B.2 ISO 15926 EXPRESS-G NOTATION DIAGRAMS

Minimal explanation of the EXPRESS-G graphical modelling language used in ISO 15926 is provided from International Standards Organisation [112, p. 21].

Figure B.1 shows how entities, subtypes, and relationships are represented.

Figure B.2 provides a legend for symbols used when representing individuals and design patterns.

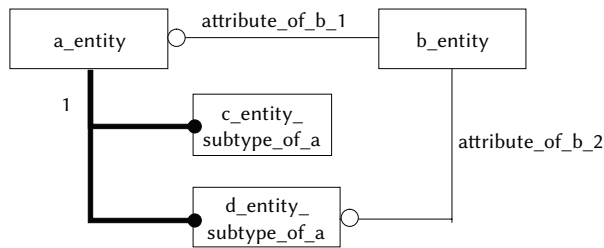


Figure B.1: EXPRESS-G Model Diagram

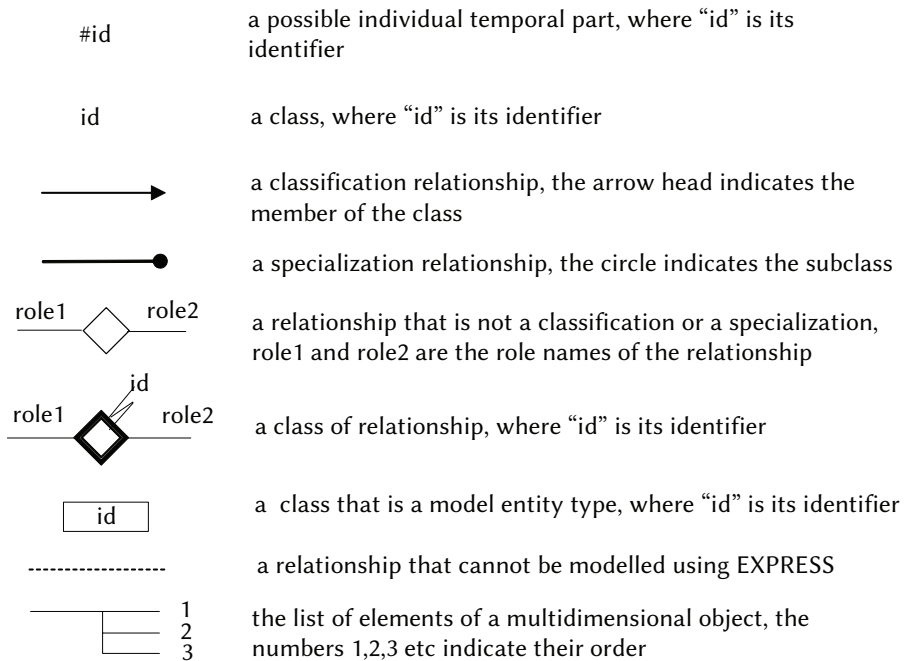


Figure B.2: EXPRESS-G Instance Diagram



## RACoon ONTOLOGY RESOURCES

---

### C.1 RACoon ONTOLOGY CLASS TERMS

[Section C.1.1](#) and [Section C.1.2](#) list the full set of class concepts identified and modelled in the RaCoOn ontologies. Full ontologies can be found online, as detailed in [Appendix A](#).

#### c.1.1 *List of Cross-domain Ontology Terms*

u:AbsoluteLocation	u:Stakeholder
u:Agent	u:TopologicalThing
u:Area	u:Unit
u:Characteristic	co:Bag
u:ContinuousMeasurement	co:Collection
u:Customer	co:Item
u:DataConcept	co:List
u:DependentThing	co:ListItem
u:Dialect	co:Set
u:DomainSpecificThing	doc:AnnotationODP
u:Event	doc:EntityDescription
u:Function	doc:MetaThing
u:IndependentThing	doc:ODP
u:InformationThing	doc:OntologyViewAffiliation
u:Location	doc:PresentationODP
u:Measurement	owl:Nothing
u:NonSpatialThing	owl:Thing
u:Observation	prov:Activity
u:Organisation	prov:Agent
u:Person	prov:Dictionary
u:PhysicalAgent	prov:Entity
u:PhysicalDimension	prov:Influence
u:Place	prov:InstantaneousEvent
u:ProvenanceConcept	prov:Location
u:RelativePosition	prov:Organization
u:Resource	prov:Person
u:Role	prov:Role
u:SpatialThing	

C.1.2 *List of Railway Domain Ontology Terms*

rcn:AbsoluteChainage	rcn:ElectrificationChange
rcn:RCMElement	rcn:TrackPlacedElement
rcn:AmberSignalAspect	rcn:ELRArc
rcn:ATPProtect	rcn:ELRNode
rcn:AxleCounter	rcn:Engineer's Line Reference (ELR)
rcn:AxleWeight	rcn:EMU
rcn:ChainageMeasurement	rcn:EnumeratedCharacteristic
rcn:Balise	rcn:ETCSL0Standard
rcn:BaliseGroup	rcn:ETCSL1Standard
rcn:BlockingSignal	rcn:ETCSCharacteristicSignal
rcn:Bogie	rcn:ETCSL2Standard
rcn:Border	rcn:ETCSL3Standard
rcn:BrakingCharacteristic	rcn:ETCSSignallingChange
rcn:RSBrakingSystem	rcn:ETCSStandard
rcn:BranchLine	rcn:ExitSignal
rcn:Bridge	rcn:ExternalInterface
rcn:BufferStop	rcn:FeatureSignal
rcn:Building	rcn:RSFireSystem
rcn:CharacteristicChange	rcn:FixedSpeedSign
rcn:RSCarriage	rcn:FourAspectSignal
rcn:ChainageZero	rcn:FreightService
rcn:CharacteristicChangeSignal	rcn:FreightTrain
rcn:CivilThing	rcn:OrientedElement
rcn:CombinedSignal	rcn:GreenSignalAspect
rcn:CommunicationsDevice	rcn:HomeSignal
rcn:ComputerInterlocking	rcn:RCMHABD
rcn:ConnectingTrack	rcn:IDCode
rcn:Contract	rcn:IdentityConcept
rcn:Crossing	rcn:IDOrganisation
rcn:Customer	rcn:IDPerson
rcn:Cutting	rcn:InfraControlElement
rcn:DataSource	rcn:InfrastructureConcept
rcn:RSDieselLoco	rcn:InfrastructureModel
rcn:DieselPropulsion	rcn:Interlocking
rcn:DieselTractionCharacteristic	rcn:Junction
rcn:DirectionCapability	rcn:IntermediateSignal
rcn:DistantSignal	rcn:RSInternalSystem
rcn:DMU	rcn:Lease
rcn:RSDoorSystem	rcn:LevelCrossing
rcn:DoubleAmberSignalAspect	rcn:LineDetailArc
rcn:RSElectricLoco	rcn:LineDetailNode
rcn:ElectricPropulsion	rcn:LineLevelArc
rcn:ElectricalTractionCharacteristic	

rcn:LineLevelNode	rcn:RailwayDomainConcept
rcn:LORLine	rcn:RailwaySystem
rcn:LinearPosition	rcn:RailwayLine
rcn:RelativeTrackPosition	rcn:RailwayLocomotive
rcn>LoadingGauge	rcn:RailwayMaintainer
rcn:LocalRoute	rcn:RailwayMileage
rcn:LocatedThing	rcn:RailwayMileageRef
rcn:Location	rcn:RailwayOperationalIncident
rcn:RSLocomotive	rcn:Signal
rcn:MainLine	rcn:RailwaySignaller
rcn:MainSignal	rcn:RailwaySpecificEvent
rcn:MainTrack	rcn:RailwayStaff
rcn:MaintenanceSystem	rcn:Station
rcn:MandatoryBrakingSpeedChange	rcn:RailwayTicket
rcn:MandatoryStopSpeedChange	rcn:RailwayTrain
rcn:MechanicalInterlocking	rcn:RailwayUnit
rcn:MeterloadMeasurement	rcn:RailwayVehicle
rcn:MovementAuthoritySignal	rcn:RCMWheelchex
rcn:RSMultipleUnit	rcn:RedSignalAspect
rcn:RailwayMultipleUnit	rcn:Regulator
rcn:RSMultipleUnitTrainSet	rcn:RelayInterlocking
rcn:NetworkArc	rcn:RepeaterSignal
rcn:NUTS2	rcn:RSCharacteristic
rcn:TrainFormation	rcn:RollingStockComponent
rcn:OHLEPropulsion	rcn:RollingStockConcept
rcn:OpenEnd	rcn:ROSCO
rcn:OCP	rcn:RSPart
rcn:PassengerInformationSystem	rcn:RSPowerSystem
rcn:PassengerService	rcn:RSPropulsionSystem
rcn:PassengerTrain	rcn:RollingStockThing
rcn:PhysicalThing	rcn:Route
rcn:Platform	rcn:RouteAbstraction
rcn:Points	rcn:RouteArc
rcn:PointsMachine	rcn:RouteBoundary
rcn:RCMPoints	rcn:RouteConcept
rcn:Position	rcn:RouteGraphConcept
rcn:PowerTransmission	rcn:RouteNode
rcn:RSPushPullTrainSet	rcn:RouteTerminus
rcn:RailStandard	rcn:RSBogie
rcn:RailwayAssetConcept	rcn:RSDieselMU
rcn:RailwayBusinessEvent	rcn:RSElectricMU
rcn:RailwayFunction	rcn:RSTrainSet
rcn:RailwayCarriage	rcn:RSVehicleType
rcn:RailwayCharacteristic	rcn:RuleCodeElement
rcn:RailwayConditionMonitoringSystem	rcn:SafetyCharacteristic
rcn:RailwayContract	rcn:SILCharacteristic

rcn:SafetySystem	rcn:TrackAssociatedElement
rcn:SecondaryTrack	rcn:TrackAxleWeight
rcn:Service	rcn:TrackCharacteristic
rcn:ServiceCharacteristic	rcn:TrackCircuit
rcn:ServiceConcept	rcn:TrackCircuitLocation
rcn:ShuntingSignal	rcn:TrackComms
rcn:SideOfTrackPosition	rcn:TrackCondition
rcn:SidingTrack	rcn:Electrification
rcn:SignalAspect	rcn:TrackElement
rcn:SignalBox	rcn:TrackGauge
rcn:SignalGroup	rcn:TrackGaugeCapability
rcn:SignalThing	rcn:TrackGradient
rcn:SignalWithAspect	rcn:TrackOperationMode
rcn:SignalWithLocationFunction	rcn:TrackRelatedPosition
rcn:SignalWithRole	rcn:TrackRadius
rcn:SignallingChange	rcn:TrackRadiusChange
rcn:SignallingStandard	rcn:TrackSection
rcn:Slope	rcn:TrackServiceProvider
rcn:SSIInterlocking	rcn:TrackSide
rcn:SpacialMeasurement	rcn:TrackSignallingMethod
rcn:SpeedCapability	rcn:TractionCharacteristic
rcn:GradientChange	rcn:TractionPackage
rcn:SpeedChange	rcn:TrainConductor
rcn:SpeedProfile	rcn:TrainControlCapability
rcn:SpeedRange	rcn:TrainControlStandard
rcn:SpeedSign	rcn:TrainDetector
rcn:StandardsCapability	rcn:TrainDriver
rcn:StationTrack	rcn:TrainOperatingCompany
rcn:StationWithWGS84	rcn:TrainProtectCapability
rcn:Status	rcn:TrainProtectStandard
rcn:Stop	rcn:TrainProtectionInfrastructureElement
rcn:StopPost	rcn:TrainStopProtect
rcn:Subcontractor	rcn:Tunnel
rcn:Switch	rcn:UKSignal
rcn:SwitchPosition	rcn:ViewConcept
rcn:SwitchableSignal	rcn:ViewGraphPosition
rcn:SwitchableSpeedSign	rcn:WeatherEvent
rcn:TestClass	rcn:MassMeasurement
rcn:ThirdRailPropulsion	rcn:WGS84Measurement
rcn:ThreeAspectSignal	rcn:WGS84Location
rcn:TiltCharacteristic	rcn:WILD
rcn:GeodesicLocation	
rcn:Track	

## C.2 VALIDATION WORKSHOP RESULTS

C.2.1 *High Level and Subdomain Concepts Elicited From RaCoOn Workshops*C.2.1.1 *Results from Edgbaston Workshop*

- Operations
  - Routing
  - Planning
  - Maintenance
  - Passenger Flow
  - Stations
  - Ticketing
  - Food and Beverages
  - Work Rotas
  - Retail
  - Fueling
  - Passenger Information Provision
  - Signalling
  - Inspection
  - Driving
  - Signage
  - Timetabling
  - External Communication and Media
- Infrastructure
  - Laws
  - Suspension
  - Bogie
  - Acceleration
  - HVAC
  - OTMR
- Range
- Configuration Flexibility
- Unit Flexibility
- Depot
- Customer Information System
- Speed
- Automatic Train Protection
- Air Conditioning
- Train and Passenger
- Traction Control
- Windows
- Aesthetics
- Mass
- Recoverability
- Doors
- Facilities /Auxiliaries
- Gauge
- Brakes
- Braking Capacity
- Maintenance
- Wheelsets
- Access
- Power Source
- Energy Storage
- Contact System



- Track Gauge
  - Loading Gauge
  - Structure Gauge
  - Kinematic Gauge
  - Warning Systems
  - Couplers
  - Tilt
  - Common Carrier
- Rolling Stock
    - Drains
    - Ballast
    - Tamping
    - Grinding
    - Inspection
    - Gradients
    - Switches & Crossings
    - Cabling
    - Formation
    - Renewal
    - Track Quality
    - Condition Monitoring
    - Mapping
    - Transformers
    - Depot
    - Conductor Rail
    - Overhead Line Electrification
    - Track
    - Bridges
    - Tunnels
    - Civil Works
    - Maintenance
    - Cuttings
- Embankments
  - Station
  - Points
  - Signals
  - Interlocking
  - GSMR Equipment
  - Track Circuits
  - Axle Counters
- People
    - Employees
    - Design
    - Research
    - Station Staff
    - Cleaning
    - Passengers
    - Journalists /Media
    - Drivers
    - Conductors
    - Level Crossing Operators
    - Maintenance
    - Demographic
    - First Class
    - Pedestrians
    - Ticket Sales Staff
    - Dispatcher
    - HR
    - Families
    - Emergency Services
    - Politicians
    - Standard Class
    - Health and Safety
    - Maintenance Staff
    - Signaller

- Salaries
- Car Drivers
- Regulation
  - Good Will
  - Directive
  - Standards
  - Inspection
  - Economic Regulation
  - Credit Rating
  - Authority
  - Corporate Policy
  - Rules
  - Education and Training
  - Corporate Culture
  - General Policy
  - Manuals
  - Train Control
  - Citizenship
  - Laws
  - Stakeholders
  - Public Relations
  - Assurance
  - Approvals Boards
  - Regulatory Bodies
- Commercial /Social
  - Contracts
  - Rate of Return
  - Borrowing
  - Incentives
  - Project Management
  - DDA
  - HLOS
  - SOFA
  - Business Aim
  - Performance
  - Pricing
  - Fundraising
  - Carbon Accounting
  - Value of Asset Base
  - ‘Cool’ Railway
  - Image
  - Sustainability
  - Demand
  - Class Mix
  - Suicide
  - Industrial Relations
  - Age Profile
  - Employment Level
  - NIMBYs

#### c.2.1.2 *Results from Chippenham Workshop*

- Service & Operations
  - Timetable Design
  - Timetable Delivery
  - Staff Rotas & Diagrams
  - Rolling Stock Diagrams
  - Providing Information
  - Management of Perturbations

- Passenger Safety (BTP, CCTV etc)
- On Train Catering
- Selling Tickets
- Managing Reservations
- Revenue Protection
- Passenger Experience
- Assets & Infrastructure/Maintenance
  - Permanent Way
  - Civil Engineering (Cuttings, Tunnels, Bridges)
  - Electrification
  - Signalling (Signal sets, control centres)
  - IT (Fibre, Computers etc)
  - Buildings
  - Rolling Stock
  - Vehicles (Cars, Vans, etc)
  - Redundant Asset Maintenance
  - Gardening
  - Cleaners (Train/Station)
  - Rolling Stock Maintenance
  - Signalling Maintenance
  - Building Maintenance
- Customers
  - Passengers
- Freight
- Information Consumers
- Potential Customers
- Train Operators
- Advertisers
- The Queen
- Topic
- External Stakeholders
  - NIMBYs
  - Shareholders
  - Farmers
  - Emergency Services
  - Tenants
  - Sufferers of Interference (Noise/EM)
  - Taxi Firms
  - Level Crossing Users
  - Bus Service Operators
  - Other Railways
  - Thieves
  - Airports
  - Unions
  - Metro Systems
- Organisation & Governance/Business
  - Regulatory
  - Train Operating Companies (TOCs)
  - ROSCOs
  - Infrastructure Companies
  - Penalties
  - Rolling Stock Builders
  - Public Subsidy

- Staff
- Signalling Suppliers
- Public Relations
- Topic
- Resources
  - Pollution/Waste
  - Water, Air
  - Power, Energy, Fuel
  - Land/Space
  - Money/Finance
  - Staff
  - Raw Materials
  - Suppliers
  - Intellectual Property
  - Legal System
  - Research, Academia
  - Train-spotters
  - Property Leases
  - Food

### c.2.2 *Domain Interactions Elicited From RaCoOn Workshops*

Table C.1 and Table C.1 show the full list of interactions between concepts identified during two RaCoOn Validation Workshops, held at the University of Birmingham, Edgbaston and at Invensys Rail Group, Chippenham respectively.

Table C.1: Railway Domain Interactions Elicited From Edgbaston Validation Workshop

Subject	Object	Relationship
People	Commercial	related to
People	Commercial	citizenship & perception
People	Commercial	sustainability & politicians
People	Commercial	travel class & comfort
People	Regulation	culture
People	Regulation	rules
People	Infrastructure	maintenance staff
People	Infrastructure	station staff
People	Infrastructure	signalling staff
People	Infrastructure	car drivers
People	Operations	maintenance staff
People	Operations	train drivers
People	Rolling stock	train drivers
People	Rolling stock	maintenance staff
People	Rolling stock	dispatcher
Commercial	Regulation	commercial regulation
Regulation	Rolling stock	design and standards
Regulation	Operations	related to
Regulation	Infrastructure	related to
Infrastructure	Operations	maintenance & infrastructure
Infrastructure	Operations	related to
Infrastructure	Operations	passengers
Infrastructure	Rolling stock	gauge
Infrastructure	Rolling stock	structures
Infrastructure	Rolling stock	OHLE (electrification)
Infrastructure	Rolling stock	wheel-rail interface
Infrastructure	Rolling stock	bogie & wheelset
Infrastructure	Rolling stock	body & suspension
Operations	Rolling stock	maintenance
Operations	Rolling stock	cargo

Table C.2: Table Showing Railway Domain Interactions Elicited From Chippenham Validation Workshop

Subject	Object	Relationship
Organisation	Service	related to
Organisation	Resources	regulations
Organisation	Resources	staff
Organisation	Resources	pollution
Organisation	Stakeholders	public relations
Organisation	Customers	related to
Organisation	Assets	related to
Organisation	Assets	interface managed by
Service	Resources	constrained by
Service	Resources	related to
Service	Stakeholders	related to
Service	Customers	used by
Service	Customers	demand quality of
Service	Assets	requires
Service	Assets	constrained by
Stakeholders	Resources	suffer
Stakeholders	Assets	affect
Customer	Assets	use (infrastructure)
Customer	Assets	affected by reliability of
Assets	Resources	requires
Assets	Resources	requires (investment)

### C.3 RACOON RAILSYS VALIDATION

Status 1	Status 2	Elem Attribute	Usage	Type	Disambiguation	Core Mapping	Extension Mapping	Notes
MAPPABLE	RELEVANT	line						
EXTENSION	IRRELEVANT	dependentLines	required		Railway lines which this depends on for simulation	(Not mapped)	New independentLines property	Application-specific
MAPPABLE	RELEVANT	lineID	required	xs:NMTOKEN		uid		
MAPPABLE	RELEVANT	name	required			rdfs:label		
EXTENSION	RELEVANT	trafficRouting	required	xs:NCName	How trains are routed - on the left or right	(Not mapped)	New vocab:Characteristics set	Core ontology has no way of representing traffic routing.
MAPPABLE	RELEVANT	type	required	xs:NCName		rdftype (subclass of RouteArc)		
EXTENSION	IRRELEVANT	transformation			View-based co-ordinate transformation of line	(Not mapped)	New ex:Transformation class, or re-use	Application-specific: Co-ordinate transforms are presentation level
DUPLICATE	DUPLICATE	dx	required	xs:double				
DUPLICATE	DUPLICATE	dy	required	xs:double				
DUPLICATE	DUPLICATE	line	required	xs:NMTOKEN				
DUPLICATE	DUPLICATE	m11	required	xs:double				
DUPLICATE	DUPLICATE	m12	required	xs:double				
DUPLICATE	DUPLICATE	m21	required	xs:double				
DUPLICATE	DUPLICATE	m22	required	xs:double				
MAPPABLE	RELEVANT	version				Graph level provenance: dc:version		
MAPPABLE	RELEVANT	comment	required			Graph level rdfs:comment		
EXTENSION	RELEVANT	major	required	xs:integer		(Not mapped)	Extensions to doc: ontology to incorporate additional metadata	
EXTENSION	RELEVANT	minor	required	xs:integer		(Not mapped)	Extensions to doc: ontology to incorporate additional metadata	
EXTENSION	RELEVANT	patchlevel	required	xs:integer		(Not mapped)	Extensions to doc: ontology to incorporate additional metadata	
MAPPABLE	RELEVANT	station				vocab:Station		
MAPPABLE	RELEVANT	kilometre		xs:decimal		vocab:RelativePosition		
MAPPABLE	RELEVANT	name	required			rdfs:label		
MAPPABLE	RELEVANT	stationID	required	xs:NCName		uid		
MAPPABLE	RELEVANT	xNet	required	xs:double	Local positioning	[vocab:GeodesicPosition vocab:lat]		Required' because Railsys is a simulator
MAPPABLE	RELEVANT	xWGS84		xs:double		[vocab:WGS84Pos vocab:lat]		

Figure C.1: Railsys RaCoOn Transformation Validation Spreadsheet



Status 1	Status 2	Elemt Attribute	Usage	Type	Disambiguation	Core Mapping	Extension Mapping	Notes
MAPPABLE	RELEVANT	yNet	required	xs:double	Local positioning	[vocab:GeodesicPosition vocab:lon]		Required because Ralys is a simulator
MAPPABLE	RELEVANT	yWCS84		xs:double		[vocab:WGS84Pod vocab:lon]		
MAPPABLE	RELEVANT	crossing				vocab:Crossing		
MAPPABLE	RELEVANT	description				d:description		
DUPPLICATE	DUPPLICATE	endOfBranchTrack	required	xs:integer		(Not mapped)		Unitless measure of crossing dimensions
DUPPLICATE	DUPPLICATE	endOfMainTrack	required	xs:integer		(Not mapped)		Unitless measure of crossing dimensions
DUPPLICATE	DUPPLICATE	kilometre	required	xs:decimal		(Duplicate)		
DUPPLICATE	DUPPLICATE	name	required	xs:integer		(Duplicate)		
MAPPABLE	RELEVANT	modelID	required	xs:integer	association between infrastructure and graph arc	ODP: Route Graph Linking		
DUPPLICATE	DUPPLICATE	startOfBranchTrack	required	xs:integer		(Not mapped)		Unitless measure of crossing dimensions
DUPPLICATE	DUPPLICATE	startOfMainTrack	required	xs:integer		(Not mapped)		Unitless measure of crossing dimensions
MAPPABLE	RELEVANT	type	required	xs:NCName	enum of crossing types		subclassing of vocab:crossing is possible	
MAPPABLE	RELEVANT	x	required	xs:decimal	Diagrammatic x position	ODP: Presentation Views		
DUPPLICATE	DUPPLICATE	xNet	required	xs:decimal		(Duplicate)		
DUPPLICATE	DUPPLICATE	yWCS84	required	xs:decimal		(Duplicate)		
MAPPABLE	RELEVANT	y	required	xs:decimal	Diagrammatic y position	ODP: Presentation Views		
DUPPLICATE	DUPPLICATE	yNet	required	xs:decimal		(Duplicate)		
DUPPLICATE	DUPPLICATE	yWCS84	required	xs:decimal		(Duplicate)		
MAPPABLE	RELEVANT	kilometreInconsistency			Models a step change in mileage	ODP: Mileage Zeroing		
DUPPLICATE	DUPPLICATE	description				(Duplicate)		
DUPPLICATE	DUPPLICATE	kilometre		xs:decimal		(Duplicate)		
DUPPLICATE	DUPPLICATE	name		xs:decimal		(Duplicate)		
MAPPABLE	RELEVANT	newKilometre	required	xs:decimal		ODP: Mileage Zeroing		
DUPPLICATE	DUPPLICATE	modelID	required	xs:integer		(Duplicate)		
MAPPABLE	RELEVANT	referenceNode	required	xs:integer		ODP: Mileage Zeroing		
DUPPLICATE	DUPPLICATE	x	required	xs:decimal		(Duplicate)		
DUPPLICATE	DUPPLICATE	xNet	required	xs:decimal		(Duplicate)		

Status 1	Status 2	Elem Attribute	Usage	Type	Disambiguation	Core Mapping	Extension Mapping	Notes
DUPLICATE	DUPLICATE	xWGS84	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	y	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	yNet	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	yWGS84	required	xs:decimal		(Duplicate)		
MAPPABLE	RELEVANT	inode				vocab:RouteNode		Railsys considers graph nodes and infrastructure nodes identical
EXTENSION	IRRELEVANT	aspectTrigger		xs:boolean	In Railsys, whether passing this node triggers a signalling change if this is a balise or not	(Not mapped)		Extendable using <vocab:Signal>
MAPPABLE	RELEVANT	balise		xs:boolean		ODP: Route Graph Linking		
DUPLICATE	DUPLICATE	description				(Duplicate)		
DUPLICATE	DUPLICATE	kilometre		xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	name				(Duplicate)		
DUPLICATE	DUPLICATE	nodeID	required	xs:integer		(Duplicate)		
EXTENSION	IRRELEVANT	presignal		xs:boolean	Signalling-specific signal type	(Not mapped)		Extendable using <vocab:Signal>
MAPPABLE	RELEVANT	referenceNode		xs:NMTOKEN	A way of combining information from two nodes	owl:sameAs		
EXTENSION	IRRELEVANT	releaseContact		xs:boolean	Signalling-specific attribute	(Not mapped)	vocab:characteristic property on	
EXTENSION	IRRELEVANT	sightingDistance		xs:boolean	Signalling-specific attribute	(Not mapped)	vocab:characteristic property on	
EXTENSION	IRRELEVANT	with1000HzMagnet		xs:boolean	Balise-specific attribute	(Not mapped)	vocab:characteristic property on	
DUPLICATE	DUPLICATE	x	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	xNet	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	xWGS84	required	xs:double		(Duplicate)		
DUPLICATE	DUPLICATE	y	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	yNet	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	yWGS84	required	xs:decimal		(Duplicate)		
MAPPABLE	RELEVANT	signal		xs:double		vocab:Signal		
DUPLICATE	DUPLICATE	description				(Duplicate)		
EXTENSION	RELEVANT	interlockingMachine	required		The type of interlocking used by this entity	(Not mapped)	vocab:characteristic property for interlocking type	
MAPPABLE	RELEVANT	kilometre		xs:decimal				
MAPPABLE	RELEVANT	name						

Status 1	Status 2	Elemt	Attribute	Usage	Type	Disambiguation	Core Mapping	Extension Mapping	Notes
MAPPABLE	RELEVANT		modelID	required	xs:integer				
MAPPABLE	RELEVANT		referenceNode	required	xs:NMTOKEN				
MAPPABLE	RELEVANT		type	required	xs:NCName				
MAPPABLE	RELEVANT		x	required	xs:decimal				
MAPPABLE	RELEVANT		xNet	required	xs:decimal				
MAPPABLE	RELEVANT		xWGS84	required	xs:double				
MAPPABLE	RELEVANT		y	required	xs:decimal				
MAPPABLE	RELEVANT		yNet	required	xs:decimal				
MAPPABLE	RELEVANT		yWGS84	required	xs:double				
MAPPABLE	RELEVANT		speedIndicator	required	xs:double		vocab:CharacteristicChanges\$ signal		
DUPLICATE	DUPLICATE		description	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE		kilometre	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE		name	required	xs:anyURI		(Duplicate)		
DUPLICATE	DUPLICATE		nodeID	required	xs:integer		(Duplicate)		
DUPLICATE	DUPLICATE		referenceNode	required	xs:NMTOKEN		(Duplicate)		
DUPLICATE	DUPLICATE		x	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE		xNet	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE		xWGS84	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE		y	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE		yNet	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE		yWGS84	required	xs:decimal		(Duplicate)		
MAPPABLE	RELEVANT		stationBorder	required	xs:decimal	The border between a network and a station node	ODP: Graph Generalisation Pattern		
DUPLICATE	DUPLICATE		description	required	xs:NCName	Locked', 'Alternative', or 'Regular'	(Duplicate)		
MAPPABLE	RELEVANT		exitType	required	xs:NCName		ODP: Graph Generalisation Pattern		
DUPLICATE	DUPLICATE		kilometre	required	xs:decimal		(Duplicate)		
MAPPABLE	RELEVANT		lineID	required	xs:NMTOKEN		ODP: Graph Generalisation Pattern		
DUPLICATE	DUPLICATE		name	required	xs:NCName		(Duplicate)		
EXTENSION	IRRELEVANT		netBorder	required	xs:boolean	Represents whether this is at the edge of a network	(Not mapped)	new subclass of vocab:Station for ex:BorderStation	Network permitters are not mapped in the core ontology
DUPLICATE	DUPLICATE		nodeID	required	xs:integer		(Duplicate)		
DUPLICATE	DUPLICATE		referenceNode	required	xs:integer		(Duplicate)		

Status 1	Status 2	Elem Attribute	Usage	Type	Disambiguation	Core Mapping	Extension Mapping	Notes
DUPLICATE	DUPLICATE	stationID	required	xs:NCName		(Duplicate)		
DUPLICATE	DUPLICATE	trackNumber	required		The track number of a track passing through a station		New concepts needed to map from a station point of view	
DUPLICATE	DUPLICATE	x	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	xNet	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	xWGS84	required	xs:double		(Duplicate)		
DUPLICATE	DUPLICATE	y	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	yNet	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	yWGS84	required	xs:double		(Duplicate)		
MAPPABLE	RELEVANT	switch				vocab:Switch and vocab:RouteNode		
DUPLICATE	DUPLICATE	description				(Duplicate)		
MAPPABLE	RELEVANT	endOfBranchTrack	required	xs:integer	Graph node of branch track	ODP: Routing Pattern		Unitless measure of switch dimensions
MAPPABLE	RELEVANT	endOfMainTrack	required	xs:integer	Graph node of main track	ODP: Routing Pattern		Unitless measure of switch dimensions
DUPLICATE	DUPLICATE	kilometre	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	name	required	xs:decimal		(Duplicate)		
MAPPABLE	RELEVANT	nodeID	required	xs:integer	Graph node of this switch	ODP: Routing Pattern		
DUPLICATE	DUPLICATE	start	required	xs:integer	Graph node of switch entrance	ODP: Routing Pattern		
DUPLICATE	DUPLICATE	x	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	xNet	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	xWGS84	required	xs:double		(Duplicate)		
DUPLICATE	DUPLICATE	y	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	yNet	required	xs:decimal		(Duplicate)		
DUPLICATE	DUPLICATE	yWGS84	required	xs:double		(Duplicate)		
NEW	RELEVANT	track			A station track; the track served by a platform	(Not mapped)	New concepts needed to map from a station point of view	This tag allows additional assertion of informatio between a track at a station. Semantically, this information should be represented using vocab:Track and vocab:Station interactions
DUPLICATE	DUPLICATE	description						

Status 1	Status 2	Elem. Attribute	Usage	Type	Disambiguation	Core Mapping	Extension Mapping	Notes
DUPLICATE	DUPLICATE	InterlockingMachine			The type of entity this is from an interlocking perspective			
DUPLICATE	DUPLICATE	kilometre		xs:decimal				
DUPLICATE	DUPLICATE	length		xs:integer				
DUPLICATE	DUPLICATE	name	required	xs:integer				
DUPLICATE	DUPLICATE	nodeID	required	xs:integer				
DUPLICATE	DUPLICATE	referenceNode	required	xs:integer				
DUPLICATE	DUPLICATE	stationID	required	xs:NCName	The UID of a station			
DUPLICATE	DUPLICATE	trackID	required	xs:NCName	The local ID of a platform track			
DUPLICATE	DUPLICATE	type		xs:NCName				
DUPLICATE	DUPLICATE	x	required	xs:decimal				
DUPLICATE	DUPLICATE	xNet	required	xs:decimal				
DUPLICATE	DUPLICATE	xMCS84	required	xs:decimal				
DUPLICATE	DUPLICATE	y	required	xs:double				
DUPLICATE	DUPLICATE	yNet	required	xs:decimal				
DUPLICATE	DUPLICATE	yMCS84	required	xs:decimal				
MAPPABLE	RELEVANT	platform		xs:double	Used as a sub-attribute of track	vocab:Platform		Platforms are modelled separately in the RaCoOn ontology
MAPPABLE	RELEVANT	endPos	required	xs:integer	The position of a platform relative to a station's absolute reference	vocab:TrackRelatedPosition		
MAPPABLE	RELEVANT	left	required	xs:boolean	The alignment of a platform (left or right)	vocab:alignment		
MAPPABLE	RELEVANT	open	required	xs:boolean	The status of the platform (open/closed)	vocab:status		
MAPPABLE	RELEVANT	startPos	required	xs:integer		vocab:TrackRelatedPosition		
MAPPABLE	RELEVANT	width	required	xs:integer		unwidth		
EXTENSION	IRRELEVANT	dependentLink			List of dependencies for links in route setting	(Not mapped)	Extension of udependsOn property	
DUPLICATE	DUPLICATE	active	required	xs:boolean	Whether the track is in use			

Status 1	Status 2	Elem Attribute	Usage	Type	Disambiguation	Core Mapping	Extension Mapping	Notes
DUPLICATE	DUPLICATE	direction	required	xs:NCName	Enum of direction capability			
DUPLICATE	DUPLICATE	mode	required	xs:NCName				
DUPLICATE	DUPLICATE	source	required	xs:integer				
DUPLICATE	DUPLICATE	target	required	xs:NMTOKEN				
MAPPABLE	RELEVANT	blockSection			Representation of signaling blocks as aggregates of track nodes	ODP: Graph Generalisation Pattern [vocab:routeConcept <vocab:SignalBlock>		
MAPPABLE	RELEVANT	nodes	required			ODP: Graph Generalisation Pattern		
MAPPABLE	RELEVANT	presignal		xs:integer		vocab:SignalBlock vocab:hasPart <vocab:PreSignal>		
MAPPABLE	RELEVANT	signallingSystem		xs:NCName		vocab:SignalBlock vocab:capability <vocab:SignallingSystem>		
MAPPABLE	RELEVANT	speedIndicators				vocab:SignalBlock vocab:routeConcept <vocab:CapabilityChangeSignal>		
EXTENSION	IRRELEVANT	slipDistance			Models the 'override' distance	(Not mapped)	Model through vocab:Characteristic class	
DUPLICATE	DUPLICATE	complete	required	xs:boolean				
DUPLICATE	DUPLICATE	name	required	xs:NMTOKEN				
DUPLICATE	DUPLICATE	nodes	required					
DUPLICATE	DUPLICATE	publicName						
DUPLICATE	DUPLICATE	signal	required	xs:NMTOKEN				
DUPLICATE	DUPLICATE	vmax	required	xs:integer				
DUPLICATE	DUPLICATE	warnroute	required	xs:boolean				
EXTENSION	IRRELEVANT	approachControlRoute			Pre-set route for station approach	(Not mapped)	Routes to be represented by an OWL list of individual :RouteNodes	Application-specific: the core ontology does not cover routing
DUPLICATE	DUPLICATE	complete	required	xs:boolean				
DUPLICATE	DUPLICATE	controlTyp		xs:NCName				

Status 1	Status 2	Element Attribute	Usage	Type	Disambiguation	Core Mapping	Extension Mapping	Notes
DUPLICATE	DUPLICATE	description	required	xs:NMTOKEN				
DUPLICATE	DUPLICATE	name	required	xs:NMTOKEN				
DUPLICATE	DUPLICATE	nodes	required					
DUPLICATE	DUPLICATE	publicName						
DUPLICATE	DUPLICATE	signal	required	xs:NMTOKEN				
EXTENSION	IRRELEVANT	stationRoute				(Not mapped)		Routes to be represented by an OWL list of individual :RouteNodes
DUPLICATE	DUPLICATE	continuousMainTrack		xs:boolean				Application-specific: the core ontology does not cover routing
DUPLICATE	DUPLICATE	name	required					
DUPLICATE	DUPLICATE	nodes	required					Mapped by datatype property rdfs:label
DUPLICATE	DUPLICATE	priorities	required					OWL list items New OWL class: Priority, referencing rolling stock types and priority values
DUPLICATE	DUPLICATE	stationID	required	xs:Ncname				Mapped through vocab:route
DUPLICATE	DUPLICATE	usableBy	required					Mapped through vocab:route
EXTENSION	IRRELEVANT	netLimitInfo				(Not mapped)		New owl:Property relation :RouteArc attribute to be extended with ex:limitInfo predicate, and linked to :RouteNode entity
DUPLICATE	DUPLICATE	lineID	required	xs:NMTOKEN				Application-specific: the core ontology does not cover simulation
DUPLICATE	DUPLICATE	stationID		xs:Ncname				Mapped through :limitInfo properties
DUPLICATE	DUPLICATE	stationName						Mapped through :limitInfo properties
MAAPPABLE	RELEVANT	link				vocab:RouteArc		
MAAPPABLE	RELEVANT	bidirectional		xs:boolean		ODP: Track Topology Pattern		
MAAPPABLE	RELEVANT	direction		xs:Ncname		ODP: Track Topology Pattern		
MAAPPABLE	RELEVANT	gradient		xs:decimal		ODP: Track Characteristic		
MAAPPABLE	RELEVANT	length		xs:integer		Pattern urlength		

Status 1	Status 2	Elem/Attribute	Usage	Type	Disambiguation	Core Mapping	Extension Mapping	Notes
MAPPABLE	RELEVANT	source	required	xs:NMTOKEN	Node	vocabstartNode		
MAPPABLE	RELEVANT	target	required	xs:NMTOKEN	Node	vocabendNode		
MAPPABLE	RELEVANT	vmax		xs:integer	Maximum speed	ODP: Track Characteristic Pattern		
MAPPABLE	RELEVANT	vmaxr		xs:integer	Maximum speed (reverse)	ODP: Track Characteristic Pattern		





## FUTRO IMPLEMENTATION NOTES

This appendix provides details and notes of particular aspects of the FuTRO projects described in [Chapter 6](#).

## D.1 AMAAS TRACK LAYOUT GRAPHICS

[Figure D.1](#) shows the track layout diagram used to identify assets in the AMaas demonstrator. It is encoded as an SVG diagram, with additional RDFa markup to allow for the identification of components.

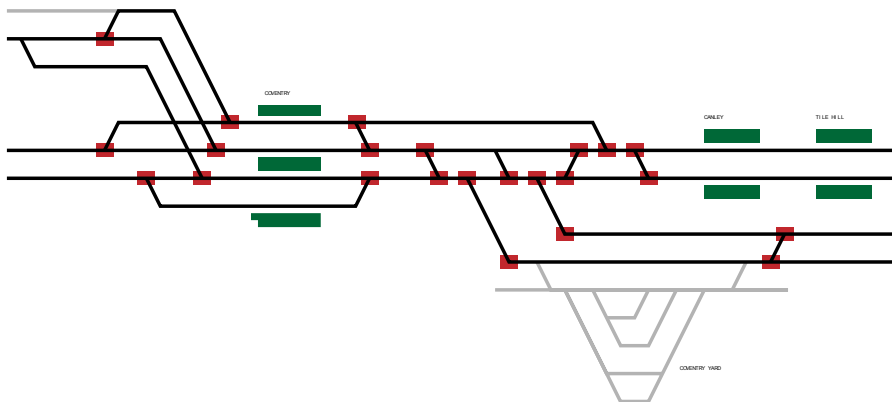


Figure D.1: SVG Track Diagram Used in FuTRO AMaas Project

```

<svg version="1.1" class="trackmap">
  <!-- CSS definitions removed -->
  <g id="Points">
    <rect rx="5" ry="5" x="780" y="304" class="points"
    ↪ width="40" height="32" about="amres:TM14BPoints0"
    ↪ typeof="vocab:Points"></rect>
    <!-- and more -->
  </g>
  <g id="Stations">
    <rect rx="5" ry="5" x="576" y="400" class="station"
    ↪ width="144" height="32" about="amres:TM14BStations0"
    ↪ typeof="vocab:Station"></rect>
    <!-- and more -->
  </g>
  <g id="CNN">
    <g id="CNN_Switches">
      <polyline class="low" points="1272,448 1280,448
    ↪ 1312,384 1320,384 " about="amres:TM14BCNNCNN_Switches0"
    ↪ typeof="vocab:LineDetailArc"></polyline>
      <!-- and more -->
    </g>
    <!-- more line definitions removed -->
  </g>
  <!-- more line definitions removed -->
  <!-- labels removed -->
  <g id="LoWNodes">
    <line class="nodes" x1="32" x2="32" y1="128" y2="128"
    ↪ about="amres:TM14BLoWNodes0"
    ↪ typeof="vocab:LineDetailNode"></line>
    <line class="nodes" x1="0" x2="0" y1="128" y2="128"
    ↪ about="amres:TM14BLoWNodes1"
    ↪ typeof="vocab:LineDetailNode"></line>
    <!-- more nodes -->
  </g>
</svg>

```

Listing D.1: Extract from RDFa Enriched SVG Code

## D.2 LEGACY WHEELCHEX DATA SNIPPETS

Listing D.2 shows an anonymised extract from an operational WILD system, which was used as a basis for the encoding of WILD data within the AMaaS project (see Section 6.2).

```

<?xml version="1.0" ?>
<WheelChexFullTrain
  → xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <WheelChexSite Name="Dallam">
    <Track_Name Name="Down Slow">
      <Train Train_Time="2008-01-01T12:00:00"
  → Train_Speed="97" Train_AxleCount="12" Train_Length="65"
  → Train_Tonnage="1.5E002" Track_NormalDirection="true">
        <Axle Axle_Number="1" Distance_From_1st="0"
  → AxleSpeed="97">
          <Wheel Train_Side="Left" Wheel_Load="60"
  → Wheel_Impact="72" DynamicRatio="1.2"></Wheel>
          <Wheel Train_Side="Right" Wheel_Load="67"
  → Wheel_Impact="76" DynamicRatio="1.1"></Wheel>
        </Axle>
        <Axle Axle_Number="2" Distance_From_1st="2.5"
  → AxleSpeed="96">
          <Wheel Train_Side="Left" Wheel_Load="61"
  → Wheel_Impact="70" DynamicRatio="1.1"></Wheel>
          <Wheel Train_Side="Right" Wheel_Load="65"
  → Wheel_Impact="75" DynamicRatio="1.2"></Wheel>
        </Axle>
        <!-- more axles -->
        <Axle Axle_Number="12" Distance_From_1st="65"
  → AxleSpeed="96">
          <Wheel Train_Side="Left" Wheel_Load="54"
  → Wheel_Impact="68" DynamicRatio="1.2"></Wheel>
          <Wheel Train_Side="Right" Wheel_Load="72"
  → Wheel_Impact="82" DynamicRatio="1.1"></Wheel>
        </Axle>
      </Train>
    </Track_Name>
  </WheelChexSite>
</WheelChexFullTrain>

```

Listing D.2: Example Wheelchex XML Data File

## D.3 AMAAS STARDOG RULES AND QUERIES

Listing D.3 shows the rule used in implementation of AMaaS to infer the current condition of a particular railway asset. Listing D.4 shows how knowledge in the triplestore can be exploited to create links between infrastructure assets and rolling stock.

```

@prefix rule: <tag:stardog:api:rule:> .

[] a rule:SPARQLRule ;
  rule:content """
    PREFIX :<urn:test:>
    IF {
      SELECT ?asset ?condition (MAX(?tstamp)
↪ as ?date)
      where {
        ?asset amaas:indirectObservation ?o .
        ?o amaas:startTime ?tstamp .
        ?o amaas:calculatedCondition ?condition
      }
      GROUP BY ?asset
    }
    THEN {
      ?asset amaas:currentCondition
↪ ?condition
    }""" .

```

Listing D.3: Stardog Rule for Deriving Current Asset Condition in AMaaS

```

construct { ?service ?p ?o } where {
  amaas:CoventryWILD amaas:sited ?l .
  ?service a tt:ScheduledService ;
    tt:serviceNode ?origin ;
    tt:serviceNode ?terminus ;
      tt:serviceNode ?n ;
      tt:runningDay time:Monday ;
  tt:consist ?rollingStock ;
  ?p ?o;
    u:id ?id .
  ?origin a tt:OriginNode ;
    u:location [rdfs:label ?olabel ].
  ?terminus a tt:TerminusNode ;
    u:location [rdfs:label ?tlabel ].
  OPTIONAL{?rollingStock a [rdfs:label ?type]}.
  OPTIONAL{?rollingStock is:axles ?axles ; is:axleLoad ?load }.
  ?n u:location ?l .
  ?n tt:ttOrder ?time .
  FILTER (?time < "13:10:00Z"^^xsd:time && ?time >
    ↪ "12:50:00Z"^^xsd:time)
} ORDER BY ASC(?time)

```

Listing D.4: SPARQL Query for Wheel Impact Load Detector Traffic Inference



## BIBLIOGRAPHY

---

- [1] B. Adida et al. *RDFa Core 1.1 - Third Edition*. 2015 (cit. on p. 46).
- [2] H. Alani, C. Brewster, and N. Shadbolt. “Ranking ontologies with AKTiveRank”. In: *The Semantic Web - ISWC 2006*. Lecture Notes in Computer Science 4273 (2006). Ed. by I. Cruz et al., pp. 1–15 (cit. on p. 37).
- [3] T. Albrecht and M. Dasigi. “ON-TIME–A framework for integrated railway network operation management”. In: *Transport Research Arena (TRA) 5th Conference: Transport Solutions from Research to Deployment*. 2014 (cit. on p. 73).
- [4] K. Alexander et al. *Describing Linked Datasets with the VoID Vocabulary*. 2011 (cit. on p. 142).
- [5] D. Allemang and J. A. Hendler. *Semantic web for the working ontologist: modeling in RDF, RDFS and OWL*. Morgan Kaufmann, 2008 (cit. on pp. 56, 63, 122, 139, 140).
- [6] S. Auer and H. Herre. “Perspectives of Systems Informatics: 6th International Andrei Ershov Memorial Conference, PSI 2006, Novosibirsk, Russia, June 27-30, 2006. Revised Papers”. In: ed. by I. Virbitskaite and A. Voronkov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Chap. RapidOWL -, pp. 424–430 (cit. on p. 33).
- [7] F. Baader, I. Horrocks, and U. Sattler. “Description Logics as Ontology Languages For The Semantic Web”. In: *Mechanizing Mathematical Reasoning*. Ed. by D. Hutter and W. Stephan. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 228–248 (cit. on p. 26).
- [8] F. Baader et al., eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. New York, NY, USA: Cambridge University Press, 2003 (cit. on p. 52).
- [9] C. Ballard et al. *Data modelling techniques for Data Warehousing*. 1998 (cit. on p. 14).
- [10] S. Batsakis, K. Stravoskoufos, and E. G. M. Petrakis. “Temporal Reasoning for Supporting Temporal Queries in OWL 2.0”. In: *Knowledge-Based and Intelligent Information and Engineering Systems*. Ed. by A. König et al. Vol. 6881. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 558–567 (cit. on p. 90).



- [11] D. Beckett. *RDF 1.1 N-Triples* (cit. on p. 45).
- [12] D. Beckett. *RDF/XML Syntax Specification (Revised)*. 2004 (cit. on p. 45).
- [13] D. Beckett et al. *Turtle - Terse RDF Triple Language - W3C Candidate Recommendation*. 2013 (cit. on p. 45).
- [14] P. M. Beevers and D. Fox. *TAF TAP Implementation Strategy*. Tech. rep. November. Network Rail, 2013 (cit. on pp. 3, 75).
- [15] C. R. Bell. “The business case for remote monitoring applications”. In: *Proceedings of the 4th IET International Conference on Railway Condition Monitoring (RCM 2008)*. Derby: Institution of Engineering and Technology, 2008, pp. 42–47 (cit. on p. 4).
- [16] Bentley. *Bentley OPTRAM*. 2015 (cit. on p. 71).
- [17] T. Berners-Lee. *Linked Data - Design Issues*. 2006 (cit. on p. 42).
- [18] T. Berners-Lee and M. Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Texere, 2001 (cit. on p. 38).
- [19] N. Besinovic et al. “Integrated Decision Support Tools for Disruption Management”. In: *RailTokyo2015: 6th International Conference on Railway Operations Modelling and Analysis, Narashino, Japan, 23-26 March 2015*. 2015 (cit. on p. 73).
- [20] M. Birbeck and S. McCarron. *CURIE Syntax 1.0*. 2010 (cit. on p. 44).
- [21] C. Bizer et al. “The Semantic Web – ISWC 2013: 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II”. In: ed. by H. Alani et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Chap. Deployment, pp. 17–32 (cit. on p. 46).
- [22] H. Bohring and S. Auer. “Mapping XML to OWL Ontologies”. In: *Leipziger Informatik-Tage, volume 72 of LNI*. GI, 2005, pp. 147–156 (cit. on p. 184).
- [23] R. Brachman and H. Levesque. *Knowledge Representation and Reasoning*. San Francisco: Morgan Kaufmann, 2004, p. 381 (cit. on pp. 26, 29, 56).
- [24] C. Brewster et al. “Data driven ontology evaluation”. In: *Proceedings of the 4th International Conference on Language Resources and Evaluation*. Lisbon, 2004 (cit. on p. 37).
- [25] D. Brickley. *W3C Basic Geo Vocabulary*. 2003 (cit. on pp. 125, 176, 177).

- [26] D. Brickley and R. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*. 2004 (cit. on pp. 43, 50).
- [27] R. Budden. *GIS and Information Management on Crossrail C122 Bored Tunnels contract*. 2011 (cit. on p. 165).
- [28] M. Bunge. *Treatise on Basic Philosophy: Ontology I: the Furniture of the World*. Springer Science & Business Media, 1977, p. 354 (cit. on p. 23).
- [29] M. A. Bunge. *Treatise on Basic Philosophy Volume 1: Semantics I - Sense and Reference*. Dordrecht: Springer Netherlands, 1974, p. 208 (cit. on p. 23).
- [30] A. Burton-Jones et al. “A Semiotic Metrics Suite for Assessing the Quality of Ontologies”. In: *Data & Knowledge Engineering* 55.1 (Oct. 2005), pp. 84–102 (cit. on pp. 36, 37).
- [31] J. Cardoso. “The Semantic Web Vision: Where Are We?” In: *IEEE Intelligent Systems* 22.5 (Sept. 2007), pp. 84–88 (cit. on pp. 30, 31).
- [32] S. Chadwick. *Layout Description Language (LDL) Specification*. Tech. rep. Invensys Rail Group, 2007 (cit. on p. 77).
- [33] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins. “What are ontologies, and why do we need them?” In: *IEEE Intelligent Systems* 14.1 (Jan. 1999), pp. 20–26 (cit. on p. 5).
- [34] E. Christensen et al. *Web Services Description Language (WSDL) 1.1*. Tech. rep. World Wide Web Consortium, 2001 (cit. on p. 16).
- [35] P. Ciccarese and S. Peroni. “The Collections Ontology: creating and handling collections in OWL 2 DL frameworks”. In: *Semantic Web* 5.6 (2013), pp. 515–529 (cit. on p. 189).
- [36] M. Ciocoiu, D. S. Nau, and M. Gruninger. “Ontologies for Integrating Engineering Applications”. In: *Journal of Computing and Information Science in Engineering* 1.1 (2001), pp. 12–22 (cit. on p. 21).
- [37] Clark & Parsia. *Stardog: The Enterprise Graph Database*. 2014 (cit. on pp. 56, 57, 144).
- [38] M. Compton et al. “The SSN Ontology of the W3C Semantic Sensor Network Incubator Group”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 17 (Dec. 2012), pp. 25–32 (cit. on p. 222).

- [39] J. Conesa and A. Olivé. “A General Method for Pruning OWL Ontologies”. In: *On the Move to Meaningful Internet Systems 2004: Coop.* Ed. by R. Meersman and Z. Tari. Vol. 3291. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 981–998 (cit. on p. 127).
- [40] O. Corcho, M. Poveda-Villalón, and A. Gómez-Pérez. “Ontology engineering in the era of linked data”. In: *Bulletin of the American Society for Information Science and Technology* 41.4 (2015), pp. 13–17 (cit. on p. 33).
- [41] M. Courtot et al. “MIREOT: The Minimum Information to Reference an External Ontology Term”. In: *Appl. Ontol.* 6.1 (Jan. 2011), pp. 23–33 (cit. on p. 127).
- [42] E. Dahlström et al. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. Tech. rep. World Wide Web Consortium, 2011 (cit. on p. 16).
- [43] M. D’Aquin. “Modularizing Ontologies”. In: *Ontology Engineering in a Networked World*. Ed. by M. Suárez-Figueroa et al. Berlin: Springer Berlin Heidelberg, 2012. Chap. 10, pp. 213–233 (cit. on p. 34).
- [44] M. D’Aquin. “Modularizing ontologies”. In: *Ontology Engineering in a Networked World*. Ed. by M. Suárez-Figueroa et al. Berlin: Springer Berlin Heidelberg, 2011. Chap. 7 (cit. on pp. 127, 128, 184).
- [45] A. De Nicola and M. Missikoff. “A Lightweight Methodology for Rapid Ontology Engineering”. In: *Commun. ACM* 59.3 (Feb. 2016), pp. 79–86 (cit. on p. 33).
- [46] E. Della Valle et al. “It’s a Streaming World! Reasoning upon Rapidly Changing Information”. In: *IEEE Intelligent Systems* 24.6 (2009) (cit. on p. 272).
- [47] K. Dentler et al. “Comparison of Reasoners for Large Ontologies in the OWL 2 EL Profile”. In: *Semantic Web* (2011) (cit. on p. 55).
- [48] Department for Transport. *How People Travel (Mode): Table NTS0303, Average number of Trips (Trip Rates) by Main Mode: England, Since 1995*. 2014 (cit. on p. 1).
- [49] C. Dickerson and D. N. Mavris. *Architecture and Principals of Systems Engineering*. CRC Press, 2010, p. 451 (cit. on pp. 14, 101).
- [50] M. Dimitrov. *Using the OWLIM triplestore to power BBC’s 2010 World Cup site*. 2010 (cit. on p. 5).

- [51] L. Dodds and I. Davis. “Linked data patterns”. In: *A pattern catalogue for modelling, publishing, and consuming Linked Data* (2011) (cit. on pp. [122](#), [123](#), [140](#), [168](#)).
- [52] P. Doran, V. Tamma, and L. Iannone. “Ontology Module Extraction for Ontology Reuse: An Ontology Engineering Perspective”. In: *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*. CIKM ’07. New York, NY, USA: ACM, 2007, pp. 61–70 (cit. on p. [35](#)).
- [53] J. Durk. *Developing the Rail Industry’s Customer Information Strategy to Provide Accurate, Consistent, Timely and Relevant Information*. London, 2014 (cit. on pp. [3](#), [242](#)).
- [54] O. Erling. “Virtuoso, A Hybrid RDMS/Graph Column Store”. In: *IEEE Computer Society Bulletin of the Technical Committee on Data Engineering* 35.1 (2012), pp. 3–8 (cit. on p. [57](#)).
- [55] S. Etchell, D. Phillips, and B. Ward. *Remote Condition Monitoring of London Underground Track Circuits*. Tech. rep. Railway Safety and Standards Board, 2014 (cit. on p. [208](#)).
- [56] D. W. Etherington. *Reasoning with Incomplete Information*. San Francisco: Morgan Kaufmann Publishers Inc., Feb. 1988 (cit. on p. [65](#)).
- [57] European Commission. *Commission Regulation (EU) No 1305/2014*. Tech. rep. European Commission, 2014, pp. 66–87 (cit. on pp. [73](#), [74](#)).
- [58] European Commission. *TAF-TSI Master Plan*. Tech. rep. January. European Commission, 2013 (cit. on pp. [76](#), [271](#)).
- [59] European Railway Agency. *Rail System Register Of Infrastructure - Final Report*. Tech. rep. European Railway Agency, 2010 (cit. on pp. [3](#), [72](#), [75](#)).
- [60] R. A. Falbo et al. “Developing Software For and With Reuse: an Ontological Approach”. In: *International Conference on Computer Science, Software Engineering, Information Technology, E-Business and Applications, CSITeA 2002*. ACIS, 2002, pp. 311–316 (cit. on pp. [12](#), [124](#)).
- [61] M. Ferdinand, C. Zirpins, and D. Trastour. “Lifting XML Schema to OWL”. In: *Web Engineering*. Ed. by N. Koch, P. Fraternali, and M. Wirsing. Vol. 3140. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 354–358 (cit. on p. [184](#)).

- [62] J. D. Fernández et al. “Binary RDF representation for publication and exchange (HDT)”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 19 (2013), pp. 22–41 (cit. on pp. 45, 219).
- [63] M. Fernández, A. Gómez-Pérez, and N. Juristo. “METHONTOLOGY: From Ontological Art Towards Ontological Engineering”. In: *Proceedings of AAAI97 Spring Symposium Series, Workshop on Ontological Engineering*. 1997 (cit. on pp. 30, 31, 36).
- [64] M. Fernández-López, A. Gómez-Pérez, and M. C. Suárez-Figueroa. “Selecting and Customizing a Mereology Ontology for Its Reuse in a Pharmaceutical Product Ontology”. In: *Proceedings of the 2008 Conference on Formal Ontology in Information Systems: Proceedings of the Fifth International Conference (FOIS 2008)*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2008, pp. 181–194 (cit. on p. 157).
- [65] M. Fernández-López, M. Suárez-Figueroa, and A. Gómez-Pérez. “Ontology Development by Reuse”. In: *Ontology Engineering in a Networked World*. Ed. by M. C. Suárez-Figueroa et al. Springer Berlin Heidelberg, 2012, pp. 147–170 (cit. on pp. 121, 124).
- [66] Fiatech. *Introduction to ISO 15926*. Tech. rep. Fiatech, 2011 (cit. on p. 82).
- [67] K. Forsberg and H. Mooz. “The Relationship of System Engineering to the Project Cycle”. In: *INCOSE International Symposium* 1.1 (1991), pp. 57–65 (cit. on p. 102).
- [68] F. Fuchs et al. “Towards Semantics-based Monitoring of Large-Scale Industrial Systems”. In: *2006 International Conference on Computational Intelligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA’06)* (2006), pp. 261–261 (cit. on p. 78).
- [69] A. Gangemi. “Ontology Design Patterns for Semantic Web Content”. In: *The Semantic Web—ISWC 2005* (2005) (cit. on p. 120).
- [70] A. Gangemi and V. Presutti. “Ontology Design Patterns”. In: *Handbook on Ontologies, 2nd Edition*. Springer, 2009, pp. 221–243 (cit. on pp. 81, 122, 157).
- [71] A. Gangemi and V. Presutti. *OntologyDesignPatterns.org*. 2008 (cit. on pp. 122, 140).
- [72] A. Gangemi et al. “Modelling ontology evaluation and validation”. In: *The semantic web: research and applications*. 2006, pp. 140–154 (cit. on p. 36).

- [73] R. García. “A Semantic Web Approach to Digital Rights Management”. PhD Thesis. Universitat Pompeu Fabra, 2006 (cit. on p. 184).
- [74] M. Genesereth et al. *Knowledge Interchange Format Version 3.0 Reference Manual*. 1992 (cit. on p. 51).
- [75] J. H. Gennari et al. “The evolution of Protégé: an environment for knowledge-based systems development”. In: *International Journal of Human-Computer Studies* 58.1 (Jan. 2003), pp. 89–123 (cit. on p. 61).
- [76] Y. Gil and S. Miles. *PROV Model Primer*. 2015 (cit. on p. 142).
- [77] Graffica. *HERMES Rail Simulation Platform*. 2015 (cit. on p. 165).
- [78] B. C. Grau et al. “A logical framework for modular integration of ontologies”. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*. 2007, pp. 298–303 (cit. on p. 35).
- [79] P. Grenon and B. Smith. *SNAP and SPAN: Towards dynamic spatial ontology*. 2004 (cit. on p. 86).
- [80] P. Grenon, B. Smith, and L. Goldberg. “Biodynamic Ontology: Applying BFO in the Biomedical Domain”. In: *Studies in Health and Technology Informatics*. IOS Press, 2004, pp. 20–38 (cit. on p. 86).
- [81] S. Grimm, B. Motik, and C. Preist. “The Semantic Web: Research and Applications: 3rd European Semantic Web Conference, ESWC 2006 Budva, Montenegro, June 11-14, 2006 Proceedings”. In: ed. by Y. Sure and J. Domingue. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. Chap. Matching S, pp. 575–589 (cit. on p. 58).
- [82] S. Groom. *Personal Communication*. 2014 (cit. on p. 69).
- [83] T. R. Gruber. “A Translation Approach to Portable Ontology Specifications”. In: *Knowledge Acquisition* 5:April (1993), pp. 199–220 (cit. on p. 24).
- [84] T. R. Gruber. “Toward principles for the design of ontologies used for knowledge sharing”. In: *International Journal of Human-Computer Studies* 43:5-6 (Nov. 1995), pp. 907–928. arXiv: [0701907v3](https://arxiv.org/abs/0701907v3) [math] (cit. on pp. 5, 33, 99).
- [85] M. Gruninger and M. S. Fox. “The Role of Competency Questions in Enterprise Engineering”. In: *Proceedings of the IFIP WG5* (1994) (cit. on p. 30).

- [86] N. Guarino, ed. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*. 1st. Amsterdam: IOS Press, 1998 (cit. on pp. [28](#), [113](#)).
- [87] N. Guarino and C. Welty. “Evaluating ontological decisions with OntoClean”. In: *Communications of the ACM* 45.2 (Feb. 2002), pp. 61–65 (cit. on p. [37](#)).
- [88] R. V. Guha. *Light at the End of the Tunnel - Slides Presented at the 12th International Semantic Web Conference (ISWC) 2013*. 2013 (cit. on p. [41](#)).
- [89] G. Guizzardi. “Ontological Foundations For Structural Conceptual Models”. PhD Thesis. Telematica Institut, 2005 (cit. on p. [21](#)).
- [90] H. Guler and S. Jovanovic. “The Application of Modern GIS Technology in the Development of Railway Asset Management Systems”. In: *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*. Vol. 5. IEEE, 2004, pp. 4153–4158 (cit. on p. [165](#)).
- [91] C. Gutierrez, C. Hurtado, and A. Vaisman. “Temporal RDF”. In: *The Semantic Web: Research and Applications* (2005), pp. 93–107 (cit. on p. [90](#)).
- [92] V. Haarslev et al. “The RacerPro Knowledge Representation and Reasoning System”. In: *Semantic Web 11* (2011), pp. 1–5 (cit. on p. [57](#)).
- [93] A. Halevy, N. Ashish, and D. Bitton. “Enterprise information integration: successes, challenges and controversies.” In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (2005) (cit. on p. [20](#)).
- [94] T. Halpin and T. Morgan. *Information Modeling and Relational Databases*. San Francisco: Morgan Kaufmann Publishers Inc., Mar. 2008 (cit. on pp. [10](#), [12](#)).
- [95] F. van Harmelen et al. *Handbook of Knowledge Representation*. San Diego, USA: Elsevier Science, 2007 (cit. on p. [52](#)).
- [96] J. Hartmann, R. Palma, and Y. Sure. “OMV – ontology metadata vocabulary”. In: *ISWC 2005 Workshop on Ontology Patterns for the Semantic Web*. 2005 (cit. on p. [142](#)).
- [97] J. Hastings et al. “Interdisciplinary perspectives on the development, integration, and application of cognitive ontologies”. In: *Frontiers in Neuroinformatics* 8 (June 2014), p. 62 (cit. on p. [86](#)).

- [98] T. Heath and C. Bizer. “Linked Data: Evolving the Web into a Global Data Space”. English. In: *Synthesis Lectures on the Semantic Web: Theory and Technology* 1.1 (Feb. 2011), pp. 1–136 (cit. on p. 42).
- [99] M. Hepp and J. de Bruijn. “GenTax: A generic methodology for deriving OWL and RDF-S ontologies from hierarchical classifications, thesauri, and inconsistent taxonomies”. In: *The Semantic Web: Research and Applications* (2007) (cit. on p. 30).
- [100] M. Hepp. “GoodRelations: An Ontology for Describing Products and Services Offers on the Web”. In: *Knowledge Engineering: Practice and Patterns*. Ed. by A. Gangemi and J. Euzenat. Vol. 5268. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, pp. 329–346 (cit. on p. 40).
- [101] I. Hickson. *HTML Microdata*. 2013 (cit. on p. 46).
- [102] E. F. Hill. *Jess in Action: Java Rule-Based Systems*. Greenwich, CT, USA: Manning Publications Co., 2003 (cit. on p. 57).
- [103] P. Hitzler et al. “Knowledge Representation for the Semantic Web Part I : OWL 2”. In: *Knowledge Creation Diffusion Utilization* (2009) (cit. on p. 57).
- [104] A. Hogan. “Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora”. PhD Thesis. NUI Galway, 2011 (cit. on p. 129).
- [105] M. Horridge. *OWL 2 Validator*. 2009 (cit. on p. 126).
- [106] M. Horridge et al. “The Manchester OWL Syntax”. In: *Proceedings of the 2006 OWL Experiences and Directions Workshop (OWLED2006)*. 2006 (cit. on p. 52).
- [107] R. Hull and R. King. “Semantic database modeling: Survey, applications, and research issues”. In: *ACM Computing Surveys (CSUR)* 19.3 (1987), pp. 201–260 (cit. on p. 4).
- [108] InteGRail. *InteGRail - Intelligent Integration of Railway Systems*. 2011 (cit. on pp. 77, 78).
- [109] InteGRail Consortium. *Refined Conceptualization Model and Services for Intelligent Monitoring Part II/II. Railway Domain Ontology: Proposal for Standardisation*. Tech. rep. InteGRail Consortium, 2009 (cit. on p. 78).
- [110] International Standards Organisation. *BSI ISO 13374-2:2012 - Condition Monitoring And Diagnostics Of Machines - Data Processing, Communication, And Presentation*. Tech. rep. International Standards Organisation, 2007 (cit. on p. 84).



- [111] International Standards Organisation. *ISO 10303-11 2004: Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual*. Tech. rep. International Organization for Standardization, 1994 (cit. on p. 51).
- [112] International Standards Organisation. *ISO15926-2: Industrial Automation Systems and Integration – Integration of Life-Cycle Data for Process Plants Including Oil and Gas Production Facilities: Part 2: Data Model*. Tech. rep. International Standards Organisation, 2003 (cit. on p. 278).
- [113] International Union Of Railways. *Feasibility Report - UIC Rail-TopoModel*. Tech. rep. September. International Union of Railways, 2013 (cit. on p. 75).
- [114] International Union Of Railways. *UIC RailTopoModel - Railway Network Description*. Tech. rep. International Union Of Railways, 2013 (cit. on pp. 75, 173).
- [115] Invensys Rail Group. *WESTCAD Scaleable train supervision (Product Literature)*. 2010 (cit. on pp. 210, 227).
- [116] N. Iscoe, G. B. Williams, and G. Arango. “Domain Modeling for Software Engineering”. In: *Proceedings of the 13th International Conference on Software Engineering*. Austin: IEEE Computer Society Press, May 1991, pp. 340–343 (cit. on pp. 11, 36).
- [117] K. Janowicz and M. Compton. “The Stimulus-Sensor-Observation Ontology Design Pattern and its Integration into the Semantic Sensor Network Ontology.” In: *SSN (2010)* (cit. on p. 222).
- [118] Y. Kalfoglou and M. Schorlemmer. “Ontology Mapping: the State of the Art”. In: *The knowledge engineering ... 18.1* (2003), pp. 1–31 (cit. on p. 135).
- [119] E. Kharlamov et al. “The Semantic Web – ISWC 2014: 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I”. In: ed. by P. Mika et al. Cham: Springer International Publishing, 2014. Chap. How Semant, pp. 601–619 (cit. on p. 85).
- [120] M. R. Khondoker and P. Mueller. “Comparing Ontology Development Tools Based on an Online Survey”. In: *Proceedings of the World Congress on Engineering 2010 I* (2010) (cit. on p. 61).

- [121] A. Kiryakov, D. Ognyanov, and D. Manov. “OWLIM – A Pragmatic Semantic Repository for OWL”. In: *Web Information Systems Engineering – WISE 2005 Workshops*. Ed. by M. Dean et al. Springer Berlin Heidelberg, 2005, pp. 182–192 (cit. on pp. 56, 57).
- [122] G. Klyne, J. J. Carroll, and B. McBride. *RDF 1.1 Concepts and Abstract Syntax*. 2014 (cit. on p. 43).
- [123] H. Knublauch. *SPIN JavaScript Functions (SPINx)*. 2010 (cit. on p. 56).
- [124] R. Kontchakov, F. Wolter, and M. Zakharyashev. “Logic-based ontology comparison and module extraction, with an application to DL-Lite”. In: *Artificial Intelligence* 174.15 (Oct. 2010), pp. 1093–1141 (cit. on p. 35).
- [125] A. Kossiakoff et al. *Systems engineering principles and practice*. Holboken, New Jersey: John Wiley & Sons, 2011 (cit. on p. 10).
- [126] M. Lebold and K. Reichard. “OSA-CBM architecture development with emphasis on XML implementations”. In: *Proceedings of the Maintenance And Reliability Conference (2002)* (cit. on p. 85).
- [127] R. Lewis and C. Roberts. “Using non-monotonic reasoning to manage uncertainty in railway asset diagnostics”. In: *Expert Systems with Applications* 37.5 (May 2010), pp. 3616–3623 (cit. on p. 79).
- [128] R. Lewis et al. “Using Ontology to Integrate Railway Condition Monitoring Data”. In: *Railway Condition Monitoring, 2006. The Institution of Engineering and Technology International Conference on Railway Condition Monitoring*. 2006, pp. 149–155 (cit. on pp. 78, 79).
- [129] Lloyd’s Register Rail Europe BV. *Gotcha Monitoring Systems*. Utrecht, 2011 (cit. on p. 235).
- [130] M. J. Loux. *Metaphysics: A Contemporary Introduction*. Third. Abingdon, Oxon: Routledge, 2006, p. 328 (cit. on p. 87).
- [131] A. Lozano-Tello and A. Gómez-Pérez. “Ontometric: A method to choose the appropriate ontology”. In: *Journal of Database Management* 2.15 (2004), pp. 1–18 (cit. on p. 36).

- [132] A. Maedche and S. Staab. “Measuring Similarity between Ontologies”. In: *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*. Ed. by A. Gómez-Pérez and V. R. Benjamins. Vol. 2473. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 251–263 (cit. on p. 37).
- [133] A. Margara et al. “Streaming the web: Reasoning over dynamic data”. In: *Semantic Web 25* (2014) (cit. on p. 90).
- [134] V. Mascardi, V. Cordi, and P. Rosso. “A Comparison of Upper Ontologies”. In: *Woa* (2007), pp. 55–64 (cit. on p. 86).
- [135] M. N. Meenachi and M. S. Baba. “Web Ontology Editors for the Semantic web: A Survey”. In: *International Journal of Computer Applications* 53.12 (2012) (cit. on p. 61).
- [136] P. N. Mendes et al. “DBpedia Spotlight: Shedding Light on the Web of Documents”. In: *Proceedings of the 7th International Conference on Semantic Systems. I-Semantics '11*. New York, NY, USA: ACM, 2011, pp. 1–8 (cit. on p. 22).
- [137] G. A. Miller. “WordNet: a lexical database for English”. In: *Communications of the ACM* 38.11 (Nov. 1995), pp. 39–41 (cit. on p. 25).
- [138] G. E. Modoni, M. Sacco, and W. Terkaj. “A Survey of RDF Store Solutions”. In: *20th International Conference on Engineering, Technology and Innovation, ICE'14* (2014), pp. 1–7 (cit. on p. 215).
- [139] M. Morsey et al. “Usage-Centric Benchmarking of RDF Triple Stores”. In: *AAAI Conference on Artificial Intelligence* 26 (2012) (cit. on p. 215).
- [140] B. Motik, I. Horrocks, and U. Sattler. “Bridging the gap between OWL and relational databases”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.2 (Apr. 2009), pp. 74–89 (cit. on p. 58).
- [141] B. Motik, U. Sattler, and R. Studer. “Query Answering for OWL-DL with Rules”. In: *Journal of Web Semantics*. Springer, 2004, pp. 549–563 (cit. on p. 56).
- [142] B. Motik et al. *OWL 2 Web Ontology Language Profiles*. 2009 (cit. on p. 55).
- [143] B. Motik et al. *OWL 2 Web Ontology Language XML Serialization (Second Edition)*. 2012 (cit. on p. 52).

- [144] National Rail Enquiries. *Response to ORR Consultation on Real Time Train Information*. Tech. rep. National Rail Enquiries, 2012 (cit. on p. 71).
- [145] Network Rail. *Asset Management Strategy*. Tech. rep. October. Network Rail, 2014 (cit. on p. 71).
- [146] Network Rail. *Network Rail Technical Strategy*. Tech. rep. London: Network Rail, 2013 (cit. on pp. 72, 93).
- [147] N. Noy et al. *Defining N-ary Relations on the Semantic Web*. 2006 (cit. on p. 48).
- [148] N. Noy, M. Sintek, and S. Decker. “Creating Semantic Web Contents With Protégé-2000”. In: *IEEE Intelligent Systems* 16.2 (2001) (cit. on p. 61).
- [149] L. Obrst. “Ontologies for semantically interoperable systems”. In: *Proceedings of the twelfth international conference on Information and knowledge management - CIKM '03*. New York, New York, USA: ACM Press, Nov. 2003, p. 366 (cit. on p. 25).
- [150] OECD. *Infrastructure Investment (Indicator)*. 2015 (cit. on p. 1).
- [151] Office of Rail Regulation. *An Overview of the British Rail Industry*. 2014 (cit. on p. 2).
- [152] Office of Rail Regulation. *National Rail Trends 2010-2011 Yearbook*. Tech. rep. London, UK: Office of Rail Regulation, 2011 (cit. on p. 1).
- [153] Ordnance Survey. *A Guide to Coordinate Systems in Great Britain*. Tech. rep. Ordnance Survey, 2015 (cit. on p. 176).
- [154] A. Ouskel and A. Sheth. “Semantic Interoperability in Global Information Systems”. In: *SIGMOD Record* 28.1 (1999), pp. 5–12 (cit. on p. 12).
- [155] T. Özacar, Ö. Öztürk, and M. O. Ünalır. “ANEMONE: An environment for modular ontology development”. In: *Data & Knowledge Engineering* 70.6 (June 2011), pp. 504–526 (cit. on p. 33).
- [156] L. Page et al. *The PageRank citation ranking: Bringing order to the web*. 1999 (cit. on p. 38).
- [157] J. Z. Pan and I. Horrocks. “RDFS (FA): Connecting RDF(S) and OWL DL”. In: *Knowledge Creation Diffusion Utilization* 19.2 (2007), pp. 192–206 (cit. on p. 127).
- [158] V. Papakonstantinou et al. “Access Control for RDF Graphs Using Abstract Models”. In: *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*. SACMAT '12. New York, NY, USA: ACM, 2012, pp. 103–112 (cit. on p. 272).

- [159] C. Parent and S. Spaccapietra. “An Overview of Modularity”. In: *Modular Ontologies*. Ed. by H. Stuckenschmidt, C. Parent, and S. Spaccapietra. Vol. 5445. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 5–23 (cit. on p. 114).
- [160] B. Parsia and E. Sirin. “Pellet: An OWL DL Reasoner”. In: *Proceedings of the International Workshop on Description Logics*. Citeseer, 2004 (cit. on p. 57).
- [161] J. Pathak, T. M. Johnson, and C. G. Chute. “Survey of Modular Ontology Techniques and Their Applications in the Biomedical Domain”. In: *Integrated Computer-Aided Engineering - Selected papers from the IEEE Conference on Information Reuse and Integration (IRI)* 16.3 (Aug. 2009), pp. 225–242 (cit. on p. 34).
- [162] H. S. Pinto and J. P. Martins. “Ontologies: How can They be Built?” In: *Knowledge and Information Systems* 6.4 (Mar. 2004), pp. 441–464–464 (cit. on pp. 29, 36).
- [163] H. S. Pinto and J. P. Martins. “Reusing Ontologies”. In: *Proceedings of the AAAI 2000 Spring Symposium on Bringing Knowledge to Business Processes*. Karlsruhe, Germany, 2000, pp. 77–84 (cit. on p. 124).
- [164] H. S. Pinto, S. Staab, and C. Tempich. “DILIGENT: Towards a fine-grained methodology for Distributed, Loosely-controlled and evolInG”. In: *Proceedings of the 16th ...* (2004) (cit. on pp. 30, 31).
- [165] H. S. Pinto, C. Tempich, and S. Staab. “Ontology engineering and evolution in a distributed world using DILIGENT”. In: *Handbook on ontologies* (2009) (cit. on p. 36).
- [166] N. Plum. *TRAK Enterprise Architecture Framework*. 2008 (cit. on p. 80).
- [167] R. Porzel and R. Malaka. “A Task-based Approach For Ontology Evaluation”. In: *Proceedings of ECA 2004 Workshop On Ontology Learning and Population*. Valencia, Spain, 2004 (cit. on p. 37).
- [168] M. Poveda-Villalón, M. C. Suárez-Figueroa, and A. Gómez-Pérez. “Validating Ontologies with OOPS!” In: *Knowledge Engineering and Knowledge Management*. Ed. by A. Teije et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 267–381 (cit. on pp. 37, 134).
- [169] R. S. Pressman. *Software engineering: a practitioner’s approach*. Palgrave Macmillan, 2005 (cit. on p. 11).

- [170] V. Presutti et al. “Pattern-Based Ontology Design”. In: *Ontology Engineering in a Networked World*. Ed. by M. C. Suárez-Figueroa et al. Springer Berlin Heidelberg, 2012, pp. 35–64 (cit. on p. 97).
- [171] E. Prud’hommeaux. *W3C RDF Validation Service*. 2006 (cit. on p. 126).
- [172] E. Prud’hommeaux, A. Seaborne, and A. Seaborne. *SPARQL Query Language for RDF*. 2008 (cit. on p. 43).
- [173] A. Radtke and J.-P. Bendfeldt. “Handling of railway operation problems with RailSys”. In: *Proceedings of the 5th World Congress on Rail Research*. 2001 (cit. on pp. 165, 199).
- [174] E. Rahm. “Towards Large-Scale Schema and Ontology Matching”. In: *Schema Matching and Mapping*. Ed. by Z. Bellahsene, A. Bonifati, and E. Rahm. Data-Centric Systems and Applications. Springer Berlin Heidelberg, 2011, pp. 3–27 (cit. on p. 135).
- [175] Rail Safety and Standards Board. *The Railway Technical Strategy 2012*. Tech. rep. Rail Safety and Standards Board, 2012 (cit. on pp. 1, 93).
- [176] Rail Safety and Standards Board and Railway Safety And Standards Board. *The Railway Functional Architecture*. 2013 (cit. on pp. 13, 80).
- [177] Railml.org. *Home - railML.org*. 2011 (cit. on pp. 159, 182).
- [178] Y. Raimond et al. “Semantic Web Use Cases and Case Studies Case Study : Use of Semantic Web Technologies on the BBC Web Sites”. In: *Linking Enterprise Data (2010)* (cit. on p. 5).
- [179] Y. Raimond et al. “The Music Ontology”. In: *ISMIR 2007: 8th International Conference on Music Information Retrieval 8 (2007)*, pp. 417–422 (cit. on p. 27).
- [180] A. L. Rector. “Modularisation of Domain Ontologies Implemented in Description Logics and Related Formalisms Including OWL”. In: *Proceedings of the 2Nd International Conference on Knowledge Capture. K-CAP ’03*. New York, NY, USA: ACM, 2003, pp. 121–128 (cit. on pp. 34, 35).
- [181] A. Rector et al. “Engineering use cases for modular development of ontologies in OWL”. In: *Applied Ontology 7.2 (2012)*, pp. 113–132 (cit. on pp. 34, 35).

- [182] A. Rector et al. “OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns”. In: *Engineering Knowledge in the Age of the Semantic Web*. Springer Berlin Heidelberg, 2004, pp. 63–81 (cit. on p. 66).
- [183] O. o. R. Regulation. *Passenger Information*. Tech. rep. December. Office of Rail Regulation, 2012 (cit. on p. 242).
- [184] C. Roberts et al. *Rail Research UK Feasibility Account: The Specification of a System-wide Data Framework for the Railway Industry—Final Report*. Tech. rep. University of Birmingham, 2011 (cit. on pp. 2, 42, 93, 104, 107, 116).
- [185] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd Editio. Pearson, 2010 (cit. on p. 57).
- [186] M. Sabou et al. “Evaluating the Semantic Web: A Task-Based Approach”. In: *The Semantic Web*. Ed. by K. Aberer et al. Vol. 4825. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 423–437 (cit. on p. 37).
- [187] E. Santos et al. “Ontology Alignment Repair through Modularization and Confidence-Based Heuristics”. In: *PLoS ONE* 10.12 (Dec. 2015). Ed. by P. Csermely, e0144807 (cit. on p. 35).
- [188] J. Seidenberg and A. Rector. “Web Ontology Segmentation: Analysis, Classification and Use”. In: *Proceedings of the 15th International Conference on World Wide Web*. WWW ’06. New York, NY, USA: ACM, 2006, pp. 13–22 (cit. on p. 35).
- [189] L. Seremeti and A. Kameas. “A Task-Based Ontology Engineering Approach for Novice Ontology Developers”. In: *2009 Fourth Balkan Conference in Informatics* (2009), pp. 85–89 (cit. on p. 37).
- [190] R. Shearer, B. Motik, and I. Horrocks. “Hermit: A highly-efficient owl reasoner”. In: *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*. Ed. by A. Ruttenberg, U. Sattler, and C. Dolbear. Karlsruhe, Germany: Citeseer, 2008 (cit. on p. 57).
- [191] A. P. Sheth. “Changing focus on interoperability in information systems: from system, syntax, structure to semantics”. In: *Interoperating geographic information systems* (1999) (cit. on p. 20).

- [192] A. P. Sheth and V. Kashyap. “So Far (Schematically) yet So Near (Semantically)”. In: *Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)*. Amsterdam: North-Holland Publishing Co., Nov. 1992, pp. 283–312 (cit. on p. 11).
- [193] E. Simperl and M. Luczak-Rösch. “Collaborative ontology engineering: a survey”. English. In: *The Knowledge Engineering Review* 29.01 (May 2014), pp. 101–131 (cit. on pp. 30, 61).
- [194] E. Simperl et al. “Achieving maturity: the state of practice in ontology engineering in 2009”. In: *On the Move to Meaningful Internet Systems: OTM 2009*. Springer, 2009, pp. 983–991 (cit. on pp. 30, 31).
- [195] D. Smallbone. *Review of Asset Information Strategy - Phase 2: ORBIS*. Tech. rep. Office of Rail Regulation, 2012 (cit. on p. 65).
- [196] B. Smith and C. Welty. “Ontology: Towards a new synthesis”. In: *Formal Ontology in Information Systems* (2001) (cit. on p. 24).
- [197] J. Smith. *A Rail Perspective - Intelligent Infrastructure: Network Rail’s Strategy for RCM*. Tech. rep. November. Network Rail, 2011 (cit. on pp. 68, 85).
- [198] M. Sporny et al. *JSON-LD 1.0*. 2014 (cit. on p. 46).
- [199] S. Staab and R. Studer. *Handbook on Ontologies*. 2nd. Springer Publishing Company, Incorporated, 2009 (cit. on p. 51).
- [200] M. Q. Stearns et al. “SNOMED clinical terms: overview of the development process and project status.” In: *Proceedings / AMIA ... Annual Symposium. AMIA Symposium* (Jan. 2001), pp. 662–6 (cit. on p. 5).
- [201] H. Stuckenschmidt and A. Schlicht. *Modular Ontologies*. Vol. 5445. 2009, pp. 5–23 (cit. on pp. 33, 35).
- [202] M. C. Suárez-Figueroa. “NeOn Methodology for Building Ontology Networks: Specification, Scheduling and Reuse”. PhD Thesis. Universidad Politécnica de Madrid, 2010, p. 268 (cit. on p. 32).
- [203] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López. “The NeON Methodology For Ontology Engineering”. In: *Ontology Engineering in a Networked World*. Ed. by M. C. Suárez-Figueroa et al. Springer Berlin Heidelberg, 2012. Chap. 1, pp. 9–34 (cit. on pp. 30, 32, 97, 124, 269).
- [204] Y. Sure, S. Staab, and R. Studer. “On-to-knowledge methodology (OTKM)”. In: *Handbook on ontologies* (2004) (cit. on p. 30).



- [205] M. Szluinska et al. *Exploring Intelligent Mobility–The Data Challenge*. Tech. rep. Transport Systems Catapult, 2014, p. 28 (cit. on pp. [43](#), [104](#), [108](#), [109](#)).
- [206] S. Tallevi-Diotallevi et al. “Real-Time Urban Monitoring in Dublin Using Semantic and Stream Technologies”. In: *The Semantic Web – ISWC 2013*. Ed. by H. Alani et al. Vol. 8219. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 178–194 (cit. on p. [272](#)).
- [207] J. Tao et al. “Integrity Constraints in OWL”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI) (2010)*, pp. 1443–1448 (cit. on p. [59](#)).
- [208] TAP Phase Two Project Team. *TAP TSI Phase Two Master Plan*. Tech. rep. DG Move, 2013 (cit. on p. [76](#)).
- [209] Thales. *DARWIN National Real Time Database*. Vélizy, 2009 (cit. on p. [242](#)).
- [210] Transmitton. *Fastflex RTU*. 2005 (cit. on p. [212](#)).
- [211] J. Tutchter, J. M. Easton, and C. Roberts. “Enabling Data Integration in the Rail Industry Using RDF and OWL: The RaCoOn Ontology”. In: *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering* (2015), F4015001 (cit. on p. [124](#)).
- [212] UK Government. *Realising the Potential of GB Rail - Report of the Rail Value For Money Study*. Tech. rep. May 2011. London, UK: UK Government, 2011 (cit. on pp. [1](#), [2](#)).
- [213] M. Uschold and M. Grüninger. “Ontologies: principles, methods, and applications”. In: *Knowledge Engineering Review* 11.2 (1996), pp. 93–155 (cit. on p. [121](#)).
- [214] M. Uschold and M. King. “Towards a Methodology for Building Ontologies”. In: *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*. Montreal, 1995 (cit. on p. [30](#)).
- [215] R. Verborgh et al. “Web-Scale Querying through Linked Data Fragments”. In: *Proceedings of the 7th Workshop on Linked Data on the Web (LDOW2014) at the 23rd International World Wide Web Conference (WWW2014)*. 2014 (cit. on p. [271](#)).
- [216] F. B. Vernadat. *Enterprise Modeling and Integration*. Chapman & Hall, 1996, p. 513 (cit. on p. [11](#)).

- [217] S. Verstichel et al. “Efficient data integration in the railway domain through an ontology-based methodology”. In: *Transportation Research Part C: Emerging Technologies* (Nov. 2010) (cit. on pp. 78, 79).
- [218] R. Volz, J. Kleb, and W. Mueller. “Towards ontology-based disambiguation of geographical identifiers”. In: *CEUR Workshop Proceedings* 249 (2007) (cit. on p. 22).
- [219] E. M. Voorhees. “Using WordNet to Disambiguate Word Senses for Text Retrieval”. In: *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '93. New York, NY, USA: ACM, 1993, pp. 171–180 (cit. on p. 22).
- [220] W3C Data Shapes Working Group. *W3C Data Shapes*. 2015 (cit. on p. 81).
- [221] W3C OWL Working Group. *OWL 2 Web Ontology Language Document Overview*. 2009 (cit. on pp. 43, 51, 52).
- [222] G. Walker and P. Godwin. “Vision into Network Rail’s Intelligent Infrastructure Monitoring Project”. In: *Intelligent Infrastructure In Rail - Predict And Prevent*. London: Institution of Engineering and Technology, 2013 (cit. on pp. 4, 68).
- [223] T. D. Wang et al. *A Survey of the Web Ontology Landscape*. 2006 (cit. on p. 126).
- [224] P. Warren. *Ontology Users ’ Survey – Summary of Results*. Tech. rep. June. Milton Keynes: Knowledge Media Institute, 2013, pp. 1–20 (cit. on p. 61).
- [225] C. Welty, R. Fikes, and S. Makarios. “A reusable ontology for fluents in OWL”. In: *Formal Ontology in Information Systems. Proceedings of the 3rd International Conference–FOIS* (2006), pp. 226–236 (cit. on pp. 65, 89, 155).
- [226] M. West. *Developing High Quality Data Models*. Morgan Kaufmann Publishers Inc., 2011 (cit. on pp. 5, 9, 158).
- [227] K. Whitehouse, F. Zhao, and J. Liu. “Semantic Streams: A framework for composable semantic interpretation of sensor data”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3868 LNCS (2006), pp. 5–20 (cit. on p. 272).

- [228] M. D. Wilkinson, B. Vandervalk, and L. McCarthy. “The Semantic Automated Discovery and Integration (SADI) Web service Design-Pattern, API and Reference Implementation.” In: *Journal of biomedical semantics* 2.1 (Jan. 2011), p. 8 (cit. on p. 219).
- [229] World Wide Web Consortium. *Linked Data Platform 1.0*. 2015 (cit. on pp. 43, 219, 271).
- [230] L. Yu. “Follow Your Nose: A Basic Semantic Web Agent”. In: *A Developer’s Guide to the Semantic Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 533–557 (cit. on p. 44).
- [231] A. Zaveri et al. “Quality assessment for Linked Data: A Survey”. In: *Semantic Web* 7.1 (Mar. 2015). Ed. by P. Hitzler, pp. 63–93 (cit. on p. 33).
- [232] L. Zhou, Q. E. Booker, and D. Zhang. “ROD - Toward Rapid Ontology Development for Underdeveloped Domains”. In: *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. Jan. 2002, pp. 957–965 (cit. on p. 30).