ABSTRACT

Title of dissertation: MULTI-DOMAIN SEMANTIC MODELS AND REASONING FOR SAFETY-CRITICAL URBAN OPERATIONS

Sachraa G. Borjigin, Doctor of Philosophy, 2022

Dissertation directed by: Associate Professor Mark A. Austin
Department of Civil and Environmental Engineering, and Institute for Systems Research

This dissertation explores development of an integrated framework for multi-domain semantic modeling and reasoning, coupled to OptaPlanner, a real-time planning and optimization tool. The investigation is motivated by abstractions from three safety-critical urban application domains: (a) small urban network operations, (b) airplane taxiing operations, and (c) planning and scheduling for disaster evacuation. From a systems modeling perspective, all three domains share the common interests of decision making supported by high-level situational awareness, and effective planning to avoid schedule conflicts and to assure system's safety. Our integrated approach uses knowledge-based representation and reasoning: (1) to understand the relationships among the physical entities in each application domain that is complicated by other relevant participating domains, (2) reason semantic graphs with external events, and (3) transform and update semantic graphs in response to these external events for making further decisions. We investigate usages of temporal, spatial and graph theories, and understand what role ontologies play in deriving appropriate semantic models for urban applications.

Semantic modeling and reasoning capabilities in the dissertation work are handled by Apache Jena and Jena Rules; temporal knowledge and reasoning are driven by time ontology and Allen's Algebra for temporal relations; Spatial knowledge and reasoning are supported by spatial ontology and class that interfaces (AbstractGeometry) with the Java Topology Suite (JTS); The JGraphT is used to handle graph structures and conduct associated graph analyses, which is a Java library describing graph theory data structures and algorithms. Systems integration of these elements is adopted in the Whistle environment, a small scripting language that is able to process complex data types (i.e., physical units and quantities). While the scope of this dissertation is limited to the three case study applications, we expect that the new knowledge will set us on a pathway to assembly and planning for behavior scenarios across a wide range of urban applications.

Keywords: multi-domain semantic modeling, reasoning, ontologies and rules, real-time planning, safety-critical operations, urban systems

MULTI-DOMAIN SEMANTIC MODELS AND REASONING FOR SAFETY-CRITICAL URBAN OPERATIONS

by

Sachraa G. Borjigin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Advisory Committee:
Professor Mark A. Austin, Chair/Advisor
Professor Deb A. Niemeier
Professor Bilal M. Ayyub
Professor Allison C. Reilly
Professor Jefferey W. Herrmann, Dean's Representative

For

My parents, who make me a man

Tsolmon, who holds my hand

Acknowledgments

I owe my deepest gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I would like to thank my advisor, Prof. Mark Austin, for giving me an invaluable opportunity to work on challenging and extremely interesting projects over the past four years. He has always made himself available for help and advice, and there has never been an occasion when I've knocked on his door and he hasn't given me time. I still remember the first day when I met him after switching from the Structures to Civil Systems group, where he told me that I would learn a lot in the coming years, and he kept his promise. Thank you, Dr. Austin. It has been an honor to work with him and learn from such an extraordinary individual.

My special thanks also goes to Prof. Deb Niemeier. Without the countless conversations about work and life, and various research projects and experiences at Maryland Transportation Institute (MTI), I could not reach to this special milestone and complete this dissertation work. Thank you so much, Dr. Niemeier. You have made it special for me to spend the last two years at UMD, and learn about how to become a professional researcher with all the hard and soft skills that one should have. You are always there for me, and for that I am forever grateful.

I would also like to show my appreciation to Prof. Allison Reilly, for the valuable discussions and suggestions shared towards my dissertation work. With your support and guidance, this dissertation has been undoubtedly improved and I felt much confident about this work. Thank you so much for the warm words. In addition, I would like to

acknowledge my dissertation committee to Prof. Bilal Ayyub and Prof. Jeffrey Herrmann for sharing their perspectives and sparing their invaluable time to review the manuscript and providing valuable insights. I strongly believe that this dissertation has been improved as a result of their comments and suggestions.

Thank you to other important professors (Richard Marciano, Peter Chang, Chung Fu), staff members, fellow students and friends at the Department of Civil and Environmental Engineering. In particular, I would like to thank Jongmin Park, Maria Coelho, Yeming Hao, Yalda Saadat, Hamed Ghaedi, Nneoma Ugwu, Heather Stewart, and others who have shared their valuable thoughts and spared time with me. Every moment with every one of you at UMD counts toward the completion of this dissertation and I sincerely thank you from the bottom of my heart. Acknowledgement also goes to Parastoo Delgoshaei and Leonard Petnga, the former graduate students at Dr. Austin's lab. The essence of this dissertation work is built on the basis of the software foundation in *Whistle* environment where the former students contributed to. It is impossible to remember all, and I apologize to those I've inadvertently left out.

I would have never gone near this stage of my life without my family's help and support. I owe my deepest appreciation to my wife Tsolmon Hereid, parents Ganggamuren and Mönggöntsetseg, grandma Daguula, parents-in-law Nyamsambuu and Serjimöödög, brother-in-law Ayonga and his wife, my uncle's family in Boston - Bagna, Jenny, Hana and Tana, my cousin in Mongolia Tsuuria Bayuud, and all the Borjigin and Hereid families who have always stood by my side, made themselves available, and helped me conquer those difficult times during these years. Words cannot express the gratitude I owe you. All supports from family members who live in Mongolia, Inner Mongolia, China, Japan and

USA. Because of the wonderful moments, delightful conversations, and priceless supports from you, this valuable dissertation work was possible. I hope that this dissertation as well as many upcoming research works from me can contribute to the various engineering communities and hopefully, it will also benefit the general public.

Lastly, I know that all good things come from God who has undoubtedly shined his light upon me. For this, I am blessed and grateful.

Table of Contents

List of Tables vii

List of Figures viii

Glossary of Terms ix

List of Abbreviates xii

1 Introduction 1
- 1.1 Problem Statement . . . 1
- 1.2 Operating Systems Perspective of Urban Decision Making . . . 3
- 1.3 Multi-Domain Semantic Modeling and Reasoning . . . 4
- 1.4 Three Motivating Applications of Safety-Critical Urban Planning . . . 8
  - 1.4.1 Safety Assessment of Vehicles at Traffic Intersections . . . 8
  - 1.4.2 Planning of Taxiway Operations at Airports . . . 10
  - 1.4.3 Modeling of Wildfire Evacuations . . . 14
- 1.5 Synthesis of Common and Disparate Modeling Requirements . . . 18
- 1.6 Research Scope and Objectives . . . 20
- 1.7 Research Contributions . . . 21
- 1.8 Dissertation Outline . . . 23

2 Background and Related Work 24
- 2.1 Foundations of Model-based Systems Engineering (MBSE) . . . 24
  - 2.1.1 State-of-the-Art MBSE with SysML . . . 24
  - 2.1.2 Transition to MBSE with Semantics . . . 27
- 2.2 Graphs and Graph Analysis . . . 29
- 2.3 Semantic Web Modeling and Technologies . . . 32
  - 2.3.1 Semantic Web Vision . . . 32
  - 2.3.2 Technical Infrastructure . . . 33
  - 2.3.3 Ontologies, Individuals, Axioms and Reasoning . . . 34
  - 2.3.4 Apache Jena and Jena Rules . . . 37
- 2.4 Whistle Scripting Language . . . 40
  - 2.4.1 Visualization of Multi-Domain Urban Data . . . 40

2.4.2 OpenStreetMap, System and Pathway Data Models 43
2.4.3 Data-Driven Generation of Individuals 47
2.5 Working Example: Modeling and Analysis of a Tiny Urban Network . . . 48
2.5.1 Problem Description 48
2.5.2 Load Problem Description into SystemDataModel 48
2.5.3 Define Tiny Urban Network Ontology and Semantic Graph 52
2.5.4 Add Individuals to Semantic Graph 53
2.5.5 Query System Data Model and Semantic Model 55
2.5.6 Specify Jena Rules 58
2.5.7 Event-Driven Graph Transformations 60
2.5.8 Minimum Cost Pathway Analysis (with JGraphT) 62

3 Proposed Methodology 67
3.1 Recent Advances in Multi-Domain Semantic Modeling 67
3.1.1 Multi-Domain Architectural Framework 67
3.1.2 Concurrent Development of Data Models, Ontologies, Rules . . . 69
3.1.3 Data-Driven Synthesis of Behavior 69
3.2 Proposed Methodology 72
3.2.1 Integration of Multi-Domain Semantic Modeling and Planning . . 73
3.2.2 Customized Data Models and Planning 75

4 Semantic Modeling and Reasoning for Time and Space 77
4.1 Time Ontologies, Rules and Reasoning 77
4.2 Space Ontologies, Rules and Reasoning 81
4.3 Working Examples 87
4.3.1 Semantic Models of Time and Time-based Reasoning 87
4.3.2 Semantic Models of Space and Space-based Reasoning 98

5 Real-Time Planning for Urban Operations 106
5.1 Real-World Planning and Replanning 106
5.2 Constraint Satisfaction Problems 108
5.3 Software Library Support for Real-Time Planning 110
5.4 OptaPlanner: A Constraint Satisfaction Solver 113
5.5 Working Example: Path Planning on a Small Network 115
5.5.1 Problem Description 115
5.5.2 Step-by-Step Solution Procedure 117
5.5.3 Sample Input and Output 120
5.5.4 Tiny Urban Network: Drool Scoring Rules 121
5.5.5 Tiny Urban Network Pathway Results 124

6 Case Study A: Multi-Domain Semantic Modeling and Planning for Airport Taxiway Operations 130
6.1 Introduction 130
6.1.1 Problem Statement 130
6.1.2 Literature Review 133

6.1.3 Scope and Objectives . . . 138
6.2 Integrated Model and Solution Procedure . . . 139
6.3 Modeling Airport Taxiway Operations and Airport Operational Safety . . . 144
6.3.1 Modeling Airport Taxiway Operations . . . 144
6.3.2 Modeling Airport Operational Safety . . . 144
6.4 Simplified Strategies for Taxiway Operations . . . 146
6.4.1 Framework for Spatio-Temporal Analysis. . . . 146
6.4.2 Three Airport Taxiway Strategies. . . . 147
6.5 Multi-domain Semantic Model and Event-based Reasoning . . . 149
6.5.1 Generation of Airport Operation Individuals in Semantic Graphs . 149
6.5.2 Airport Taxiway Operations Ontology and Jena Rules . . . 150
6.5.3 Software Problem Setup . . . 153
6.6 Formulation of Planning Problem . . . 156
6.6.1 Use Cases and Scenarios . . . 156
6.6.2 Requirements and Constraints . . . 158
6.6.3 Planning Objectives . . . 159
6.7 Planning Airport Taxiway Operations . . . 160
6.7.1 Problem Description . . . 160
6.7.2 Class Diagram . . . 161
6.7.3 Drool scoring rules for taxiway operations. . . . 162
6.8 Results and Discussion . . . 170

7 Case Study B: Multi-Domain Semantic Modeling and Planning for Wildfire Evacuation 178
7.1 Introduction . . . 178
7.1.1 Problem Statement . . . 178
7.1.2 Literature Review . . . 181
7.1.3 Scope and Objectives . . . 186
7.2 Integrated Framework and Solution Procedure . . . 187
7.3 Modeling Wildfire Spread and Wildfire/Infrastructure Impact . . . 192
7.3.1 Simplified Wildfire Spread Model . . . 192
7.3.2 Modeling Impact of Wildfires on Urban Infrastructure . . . 193
7.4 Simplified Strategies of Evacuation . . . 194
7.4.1 Framework for Spatio-Temporal Analysis . . . 194
7.4.2 Three Wildfire Evacuation Strategies . . . 195
7.5 Multi-Domain Semantic Model and Event-based Reasoning . . . 198
7.5.1 Generation of Wildfire Simulation Individuals . . . 199
7.5.2 Simplified Wildfire Evacuation Ontology and Jena Rules . . . 200
7.5.3 Software Problem Setup . . . 202
7.6 Formulation of Planning Problem . . . 205
7.6.1 Use Cases and Scenarios . . . 205
7.6.2 Requirements and Constraints . . . 207
7.6.3 Planning Objectives . . . 207
7.7 Planning for Wildfire Evacuation . . . 209
7.7.1 Problem Description . . . 209

7.7.2 Class Diagram . . . 210
7.7.3 Evacuation Scoring Rules Modeled in Drools . . . 212
7.7.4 Fire Sequences . . . 218
7.8 Results and Discussion . . . 219

8 Conclusion and Future Research 233
8.1 Conclusion . . . 233
8.2 Future Research . . . 234
8.2.1 Path Planning and Trajectory Adjustment . . . 236
8.2.2 Urban Safety System Components and Architectures . . . 238
8.2.3 Urban System Agent Communications and Interactions . . . 239

A Tiny Small Urban Network System 241
A.1 System Data Model Representation (TinyUrbanNetwork.xml) . . . 241
A.2 Print Urban Network System Data Model . . . 245

B Tiny Urban Network Semantic Model: Ontology, Rules and Graph Query 248
B.1 Urban Network Ontology (umd-urbannet.owl) . . . 248
B.2 Urban Network Jena Rules (umd.urbannet.rules) . . . 252
B.3 Print Urban Network Semantic Model . . . 253

Bibliography 256

List of Tables

5.1 Summary of software libraries that support development of planning applications. . . . 111

6.1 Summary of literature on airport operations (part 1/2). . . . 135
6.2 Summary of literature on airport operations (part 2/2). . . . 136
6.3 Data types used in the Airport Case Study. . . . 150
6.4 Use Cases and Scenarios of Airport Taxiway Operations 157
6.5 Hard and soft constraints for the airport taxiway operations problem. . . . 163
6.6 Summary of results for taxiway operations (before and after coordination). 177

7.1 Summary of literature on wildfire (disaster) evacuation modeling (part 1/2). 184
7.2 Summary of literature on wildfire (disaster) evacuation modeling (part 2/2). 185
7.3 Fire spread rate as a function of FMC and wind speed (Source: [9]). . . . 193
7.4 Data types used in the Wildfire Case Study. . . . 200
7.5 Use cases and scenarios for wildfire evacuation operations. . . . 205
7.6 Hard and soft constraints for the Wildfire Evacuation Problem. . . . 212
7.7 Summary of results for wildfire evacuation modeling (before and after obstruction assessment). . . . 231

List of Figures

1.1 Annotated view of participating domains and traffic behavior at main entrance to UMD. (Source: [29]). . . . 2
1.2 Simplified operating systems architecture for urban system planning and control. . . . 4
1.3 Preliminary framework for semantic analysis and rule-based reasoning [42]. 5
1.4 Semantic modeling for control of traffic intersection behaviors (main entrance to UMD) (Source: [29]). . . . 5
1.5 High-level framework for multi-domain semantic modeling. . . . 6
1.6 Plan view of traffic intersection at main entrance to UMD. . . . 9
1.7 Spatio-temporal modeling of a vehicle making a left-hand turn at a traffic intersection. . . . 9
1.8 Multi-domain view of OpenStreetMap data for Baltimore Washington International (BWI) Airport annotated with pathways for taxiway operations. 11
1.9 Panoramic view of Colorado June 2018 Wildfire [99]. . . . 15
1.10 Multi-layer visualization of Paradise, CA, with town boundary. . . . 15
1.11 Sequences of decision making and planning in the development and operation of systems in the urban environment. . . . 20

2.1 Framework for frontend development of systems. . . . 25
2.2 Pillars of SysML: structure, behavior, requirements and parametric diagrams. 25
2.3 MBSE capability: transition from MBSE to Semantics. . . . 27
2.4 Classification of graph types. . . . 30
2.5 Draft of graph ontology mirroring architecture of JGraphT software. . . . 30
2.6 Technologies in the Semantic Web Layer Cake [50] 33
2.7 Sample inference rules. . . . 36
2.8 Time-based evolution of semantic graph: a simple evacuee example. . . . 38
2.9 Framework for forward chaining of facts and results of builtin functions to assertions (new facts). . . . 39
2.10 Multi-domain view of BWI airport. . . . 41
2.11 Plan view of hydraulic network system. . . . 41
2.12 Fragments of XML in open street map and system data model formats. . . . 42
2.13 Abstract representation of a component model. . . . 45

2.14 Pathway data model and its interaction with domain data model and jena semantic model. . . . 46
2.15 Data-driven approach to generation of individuals in semantic graphs. . . 47
2.16 Tiny town network structure with undirected edges. . . . 49
2.17 Tiny town network structure with directed weighted edges and minimum weighted cost pathway from node A to node L. . . . 49
2.18 Minimum weighted cost (all) pathways analysis from sets of sources (nodes A, B and C) to sets of destinations (nodes L, M and N). . . . 49
2.19 Portion of code for the urban network input file (part a). . . . 50
2.20 Portion of code for the urban network input file (part b). . . . 51
2.21 Tiny town network structure, created with System Data Model. . . . 51
2.22 Simplified urban network ontology with its classes, object and data properties. . . . 53
2.23 Fragment of code to manually assemble urban network ontology and semantic graph (part a). . . . 54
2.24 Fragment of code to manually assemble urban network ontology and semantic graph (part b). . . . 55
2.25 Fragment of code to add individuals to the urban network semantic graph. 56
2.26 Load and query the urban network data model (system data model). . . . 57
2.27 Load and query the urban network ontology and semantic graph (jena semantic model). . . . 58
2.28 Fragment of code to set up semantic model, visitor and initiate rules execution. . . . 59
2.29 Jena rules for the tiny urban network. . . . 61
2.30 Fragment of code to set up simple graph analysis on the tiny urban network. 63
2.31 Fragment of Whistle code to set up directed graph analysis on tiny urban network. . . . 64
2.32 JGraphT output: computes and ranks all paths connecting sets of nodes A, B, C and L, M, N (part a). . . . 65
2.33 JGraphT output: computes and ranks all paths connecting sets of nodes A, B, C and L, M, N (part b). . . . 66

3.1 Architectural template for multi-domain semantic modeling and reasoning. 68
3.2 Data-driven synthesis of system behavior and structure (simplified). . . . 70
3.3 Data-driven synthesis of system behavior and structure (detailed). Adapted from [31–33]. . . . 70
3.4 Architectural template for integration of multi-domain semantic modeling and reasoning with computational support for real-time planning. . . . 74
3.5 Preliminary framework for integration of multi-domain semantic modeling with OptaPlanner. . . . 74

4.1 Left: Ontology for modeling time instances and time intervals, Right: Allen's algebra for relationships among time intervals. . . . 78
4.2 Time ontology and its data and object properties. . . . 80
4.3 Jena rules reasoning with time ontology. . . . 80

4.4 Types of geometry supported in Java Topology Suite. 84
4.5 Abbreviated representation of spatial (geometry) ontology and associated data and object properties. 84
4.6 Summary of spatial relations between two geometric objects. 86
4.7 Simplified airport network ontology with its classes, object and data properties (reasoning with time). 88
4.8 Fragment of code to manually set up airport network semantic model with temporal aspects (part a). 89
4.9 Fragment of code to manually set up airport network semantic model with temporal aspects (part b). 90
4.10 Fragment of code to add individuals and properties (related to time) to the airport network semantic model (part a). 92
4.11 Fragment of code to add individuals and properties (related to time) to the airport network semantic model (part a). 93
4.12 Load and query the airport network data model related to time. 94
4.13 Software code to import airport network ontology, add and execute Jena rules. 95
4.14 Simplified Jena rules for the airport network operations (time). 97
4.15 Simplified airport network ontology with its classes, object and data properties (reasoning with time + space). 98
4.16 Fragment of code to manually set up airport network semantic graph with spatial aspects (part a). 99
4.17 Fragment of code to manually set up airport network semantic graph with spatial aspects (part b). 100
4.18 Fragment of code to add individuals and properties (related to space) to the airport network semantic model. 102
4.19 Load and query the airport network semantic graph related to space. . . . 103
4.20 Simplified Jena rules for the airport network operations (space). 105

5.1 Collage of OptaPlanner scheduling and planning problems [116]. 112
5.2 Vehicle routing app example (Source: [116]). 112
5.3 Path planning on a tiny town network structure. 116
5.4 Path planning avoids disruptions on a tiny town network structure. 116
5.5 Class diagram for the tiny town network structure. 118
5.6 Fragment of code for the OptaPlanner Sample Input. 119
5.7 Fragment of code for the OptaPlanner Sample Output. 121
5.8 Fragment of code for the OptaPlanner constraints on shortest path. 122
5.9 Fragment of code for the OptaPlanner constraints on edge capacity and entity schedules. 123
5.10 Fragment of code for the OptaPlanner constraints on capacity of safe areas (nodes). 124
5.11 Fragment of code for the OptaPlanner constraints on delay costs. 125
5.12 Urban Network Problem: Fragment of OptaPlanner output for Person A. . 126
5.13 Urban Network Problem: Fragment of code OptaPlanner output for Persons B and C. 127

5.14 Time and location chart for Persons A, B and C (Results for Figure 5.3). . 129
5.15 Time and location chart for Persons A, B and C (Results for Figure 5.4). . 129

6.1 Multi-domain semantic modeling for airport taxiway operations. 131
6.2 Overview of OptaPlanner input/output for airport taxiway operations. . . 132
6.3 Multi-domain modeling: plane, obstruction and taxiways. 140
6.4 Integrated framework and solution procedure. 141
6.5 Model Dependencies for Airport Taxiway Operations. 143
6.6 Framework for spatio-temporal analysis. 146
6.7 Taxiway strategy 1: plane 1, plane 3, and plane 2. 148
6.8 Taxiway strategy 2: plane 2, plane 3, and plane 1 148
6.9 Taxiway strategy 3: plane 3, plane 2, and plane 1 148
6.10 Simplified schematic for airplane, taxiway, runway, gateway, maintenance ontology classes and properties. 151
6.11 Abbreviated list of Jena Rules for airport taxiway operations. 152
6.12 Fragment of code to set up the terminal and holding positions with the airport operation data model. 153
6.13 Fragment of code to extract useful information from OSM (OpenStreetMap). 154
6.14 Assemble the airport operations semantic model, populate ontologies, and execute graph transformation. 155
6.15 Assemble the airport operation semantic model, populate ontologies, and execute graph transformation. 156
6.16 Taxiway planning on an airport network structure. 160
6.17 OptaPlanner class diagram for airport taxiway operations. 162
6.18 Taxiway operations with planning hard constraints view and results. . . . 165
6.19 Taxiway operations with planning soft constraints view and results. 166
6.20 Fragment of code for airport taxiway operations hard constraints. 167
6.21 Fragment of code for airport taxiway operations soft constraints. 169
6.22 Fragment of OptaPlanner output for Airplane I. 171
6.23 Fragment of OptaPlanner outputs for Airplane II and III. 172
6.24 Changes in semantic representation for planning and replanning airplanes. 173
6.25 Summary of taxiway operations. 174
6.26 Semantic graph transformation for taxiway operations. 176

7.1 Multi-domain semantic model and reasoning infrastructure in wildfire evacuation modeling. 179
7.2 Overview of OptaPlanner input/output for wildfire evacuation. 180
7.3 Multi-domain modeling: fire, vegetation and road segments. 188
7.4 Integrated framework and solution procedure. 189
7.5 Model dependencies for wildfire evacuation modeling. 191
7.6 Framework for Spatio-Temporal Analysis 195
7.7 Evacuation strategy 1: Play it safe (evacuate before fire arrives). 197
7.8 Evacuation strategy 2: Stay as long as possible (evacuate just-in-time). . . 197
7.9 Evacuation strategy 3: Stay and fight (evacuate after the fire is extinguished). 197

7.10 Simplified schematic for wildfire, person, house, vegetation and shelter ontology classes and properties. 201
7.11 Abbreviated list of Jena Rules for wildfire evacuation. 201
7.12 Portion of code to set up the house and shelter locations with the wildfire data model. 202
7.13 Portion of code to extract useful information from OSM. 203
7.14 Assemble the wildfire semantic model, populate ontologies, and execute graph transformation. 204
7.15 Assemble the wildfire semantic model, populate ontologies, and execute graph transformation. 204
7.16 Evacuation planning on a urban road network structure. 209
7.17 OptaPlanner Class Diagram: Wildfire Evacuation 211
7.18 Small town with planning hard constraints view and results. 214
7.19 Small town with planning soft constraints view and results. 215
7.20 Fragment of code for wildfire evacuation hard constraints. 216
7.21 Fragment of code for wildfire evacuation soft constraints. 217
7.22 Small town evacuation with sequence of fire at various times. 218
7.23 Fragment of OptaPlanner output for Evacuees 1 and 2. 221
7.24 Fragment of OptaPlanner output for Evacuees 3 and 4. 222
7.25 OptaPlanner Output for the New Evacuation Route of Evacuee 1. 223
7.26 OptaPlanner Output for the New Evacuation Route of Evacuee 2. 224
7.27 Changes in semantic representation for planning and replanning evacuation. 225
7.28 Small town evacuation planning view and results. 226
7.29 Small town evacuation replanning view and results. 228
7.30 Semantic graph transformation for wildfire evacuations 230

8.1 Path planning and trajectory adjustment. 237

Glossary of Terms

This glossary provides definitions of key terms employed in this work:

Abstract Ontology Model: is an abstract infrastructure for an ontology model.

Action: (Effect) is the response given to stimuli in a transition, and will normally corresponds to an activity performed during the transition in the statechart

Apache Jena: A free and open source Java framework for building Semantic Web and Linked Data applications.

API: (Application program interface) is a set of routines, protocols, and tools for building software applications. An API species how software components should interact.

Axioms: are logical statements about the relationships between properties and/or classes in the domain.

Bi-directional Association: Refers to a symmetric dependency between two classes.

Block Diagram: A SysML diagram, the represents the principal components of a system and the structural design that connects them together.

Class Diagram: A UML diagram, which focuses on different classes of the software systems and their connection with respect to each other.

Composite Hierarchy Design Pattern: provides a flexible way to create tree structures of arbitrary complexity, while enabling every element in the structure to operate with a uniform interface.

Constraint Satisfaction Problem: are mathematical questions defined as a set of objects whose state must satisfy a number of constraints or limitations [158].

Constraint: A design constraint refers to some limitation on the conditions under which a system is developed.

DataType Property: DataType Property defines the relation between instances of classes and literal values, i.e., String using the Protégé tool.

Description logic: (DL) is a family of logic-based knowledge representation languages that can be used to represent the terminological knowledge of an application domain in a structured way.

Event: Stimuli that may cause a transition from one state to another state in statechart. There are four main categories of events: Signal, time, change and call events.

Extended Markup Language (XML): The extensible Markup Language provides the fundamental layer for representation and management of data on the Web.

Facts: are typically expressed by binary relations between data elements, whereas higher order relations are expressed as collections of binary relations. Typically binary relations have the form of triples: Object-RelationType-Object. For example: the `Eiffel Tower <isLocatedIn> Paris` [139].

Finite State Machine: A finite-state machine (FSM) is a mathematical model of computation for an abstract machine defined in terms of a finite number of states and transitions, and sequences of input events that will be consumed during the machine's operation.

Individual: Is a semantic web terminology that represents an instance of a class in the ontology.

JavaFX: A set of graphics and media packages for creating and delivering desktop applications.

Java Topology Suite: is an open source Java Library for creating and manipulating vector geometry [86].

Jena Rules: The Jena inference subsystem that is designed to allow a range of inference engines or reasoners to be plugged into Jena.

JGraphT: a Java library of graph theory data structures and algorithms [87].

Model-Based Systems Engineering: Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases (INCOSE-TP-2004-004-02, Version 2.03, September 2007).

Model-View-Controller (MVC): Is a system design pattern that separates the representation of information from the user's interaction with it.

ObjectType Property: ObjectType Property defines the relation between instances (individuals) of two classes in semantic web terminology using protégé tool.

Ontology: A model that describes what entities exist in a design domain, and how such entities are related.

Ontology Class: A placeholder for an entity in the system design. An ontology class may have some dataType or objectType properties.

Ontology Instance: An ontology instance is a specific realization of any ontology class object. An object may be varied in a number of ways. Each realized variation of that object is an instance. The creation of a realized instance is called instantiation.

OpenStreetMap: provides map data for thousands of web sites, mobile apps, and hardware devices [114].

OptaPlanner: An easy-to-use, open source constraint solver with AI algorithms [116].

Ontology Web Language: The Web Ontology Language (OWL) is a knowledge representation languages for defining ontologies.

Planning Problem: A planning problem has an optimal goal, based on limited resources and under specific constraints. Optimal goals can be any number of things [129].

Reasoner (Rule Engine): A semantic reasoner, reasoning engine, rules engine, or simply a reasoner, is a piece of software able to infer logical consequences from a set of asserted facts or axioms.

Reasoning: To infer new statements based on set of asserted facts in the ontology.

Requirement: A textual representation representing derived from users statement of need, or system functionality limitations.

Resource Decription Framework: Provides support for modeling graphs of resources on the Web.

Rule Checking: A mechanism that ensures existing data in the ontology is consistent with rules defined over the ontology. A rule engine often performs this task.

Semantic Web: Refers to W3C's vision of the Web of linked data.

Semantic Web Layer Cake: An informal term used to describe the stack of technologies used in the implementation of the Semantic Web.

Semantic Web Technologies: Semantic Web technologies provide features to build vocabularies, and develop rule repositories and ontologies.

Software Design Patterns: In software engineering a design pattern is a general reusable solution description to a recurring problem.

State: is a description of the status of a system that is waiting to execute a transition.

Statechart: In model-based systems engineering, state machines (statecharts) are used to describe the state-dependent behavior of a system component throughout its life cycle.

SysML: The Systems Modeling Language (SysML) is a graphical modeling language used to define models of systems structure and system behavior.

Transition: A transition is a set of actions to be executed when a condition is fulfilled or when an event is received.

Unified Modeling Language: UML is a graphical modeling language used to define mainly software systems structure and behavior.

View: Visual representation of the model in MVC design architecture.

Visitor Design Pattern: provides a mechanism to separate an algorithm (i.e. system functionality) from the object structure on which it operates.

WayPoint: provides a list of nodes and ways generated from the wildfire pathway simulation.

Workspaces: Requirement, ontology and engineering workspaces are created to store various viewpoints and their associated models.

List of Abbreviations

AI: Artificial Intelligence.

TFMP: Air Traffic Flow Management Problem.

ANSI: American National Standards Institute.

API: Application Programming Interface.

APMS: Airport Pavement Management System.

CPS: Cyber-Physical System.

CSP: Constraint Satisfaction Problem.

DL: Description Logic.

DOM: Document Object Model.

DRL: Drools.

DT: Digital Twin.

ERP: Enterprise Resource Planning.

FMC: Fuel Moisture Content.

FP: Functional Programming.

FSR: Fire Spread Rate.

GAP: Gate Assignment Problem.

GIS: Geographic Information System.

GUI: Graphical User Interfaces.

IFC: Industry Foundation Classes.

ifcOWL: Industry Foundation Classes Ontology.

IP: Integer Programming.

JAXB: XML Binding for Java.

JVM: Java Virtual Machine.

JTS: Java Topology Suite.

LiDAR: Light Detection and Ranging.

LP: Linear programming.

ML: Machine Learning.

MIP: Mixed Integer programming.

MVC: Model-View-Controller.

MBSE: Model-Based Systems Engineering.

NAS: National Airspace System.

NLP: Natural Language Processing.

NP: Nondeterministic Polynomial time.

ODE: Ordinary Differential Equation.

OOP: Object Oriented Programming.

OSM: Open Street Map.

OR: Operations Research.

OWL: Web Ontology Language.

RDF: Resource Description Framework.

RDFS: Resource Description Framework Schema.

SAX: Simple API for XML Parsing.

SPARQL: Simple Protocol and RDF Query Language.

SysML: System Modeling Language.

UMD: University of Maryland.

UML: Unified Modeling Language.

URI: Uniform Resource Identifier.

W3C: World Wide Web Consortium.

WKT: Well Known Text.

XML: Extensible Mark-up Language.

XSLT: Extensible Stylesheet Language Transformations.

Chapter 1: Introduction

1.1 Problem Statement

The basic premise of information-age system design is that modern techologies (e.g., computer software, sensing, wireless communications, new materials) can work together to expand the range of functionality and performance of engineering systems. In the case of civil infrastructure, these advances in technology have opened doors to the replacement of aging urban infrastructure with new types of urban systems comprising physical networks connected to cyber components (data, information, software) for decision making [122]. The expectation is that next-generation urban systems will be defined by superior levels of performance, new forms of functionality, transparency in allocation of resources and decision making, and good economics over long time horizons. To achieve these benefits, next-generation urban systems will make increased use of automation, where human involvement for managemement of system functionality is replaced (or partially replaced) by software automation. Complicating factors in the design and implementation of these new types of systems include: the presence of heterogeneous content (e.g., input from multiple disciplines); network structures and operations that are spatial, dynamic, and affected by environmental phenomena; and behaviors that need to

be controlled both on the individual and group level.

Small Urban Example. As a first step toward understanding what might be involved in dealing with these new opportunities, Figure 1.1 shows a street view of traffic at the Baltimore Avenue and Campus Drive intersection near the University of Maryland, College Park.

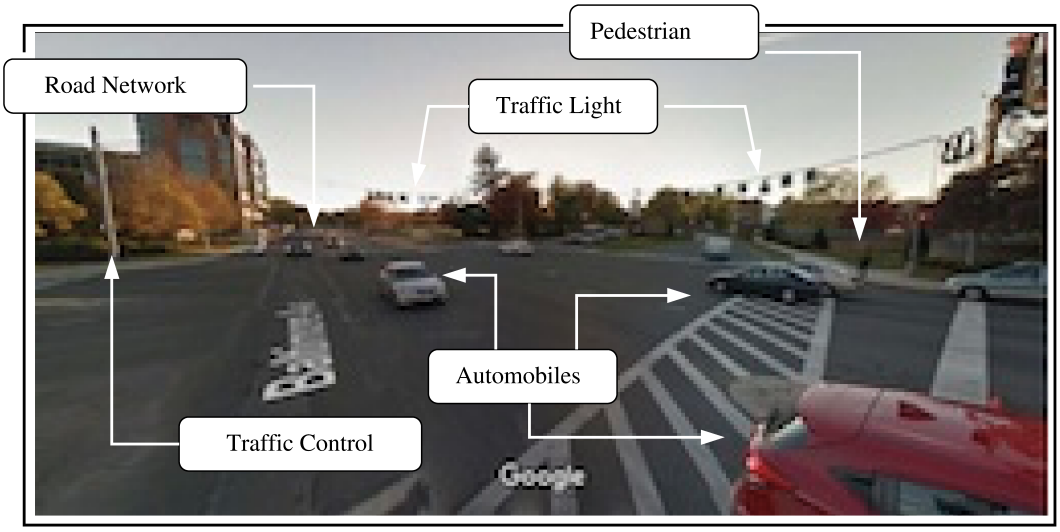


Figure 1.1: Annotated view of participating domains and traffic behavior at main entrance to UMD. (Source: [29]).

From an operational standpoint, drivers want to traverse the traffic intersection in an efficient manner and without causing (or becoming involved in) a traffic accident. Mission success means either being in the right place at the right time or taking the right action to be in the right place at the right time, while moving efficiently to complete a task. To this end, a well run system will: (1) sense the system state and surrounding environment, (2) look ahead and anticipate events, and (3) take action to control the system behavior in a timely manner. From a traffic management standpoint, a well run system will also dynamically adapt its operations to handle abnormal circumstances such as a loss of traffic

control operations during an electrical storm, and/or clearing the roadway for emergency vehicles. For normal and abnormal operations alike, traffic managers and individual drivers seek safety, and an optimal balance of performance and fairness. The latter suggest the need for an operating system-like services to manage behaviors and resources in urban environments. As illustrated in Figure 1.1, sources of complication include a multitude of participating domains (e.g., road networks, traffic lights, pedestrians, other vehicles), multiple types of data (e.g., visual, audio), network structures that are spatial and interwoven, behaviors that are distributed and concurrent, and many interdependencies among the coupled urban subsystems.

Looking beyond this small example, assurance of urban safety is a complex and difficult problem. From a civil engineering perspective, the main areas of interest are comprimised safety due to natural and human induced events (e.g., wildfires, flooding, failures of aging infrastructure, terrorist attack, etc). Without a loss of generality, it is clear that good solutions require attention to a large range of issues spanning multiple physical and cyber domains, multiple levels of spatial and temporal detail, and human considerations [128, 145, 168].

1.2 Operating Systems Perspective of Urban Decision Making

In computing circles, an operating system is software that is responsible for the management of a computer's basic functions and resources, including scheduling of tasks and execution of applications.

Figure 1.2 shows a high-level variation on this theme – an operating systems ar-

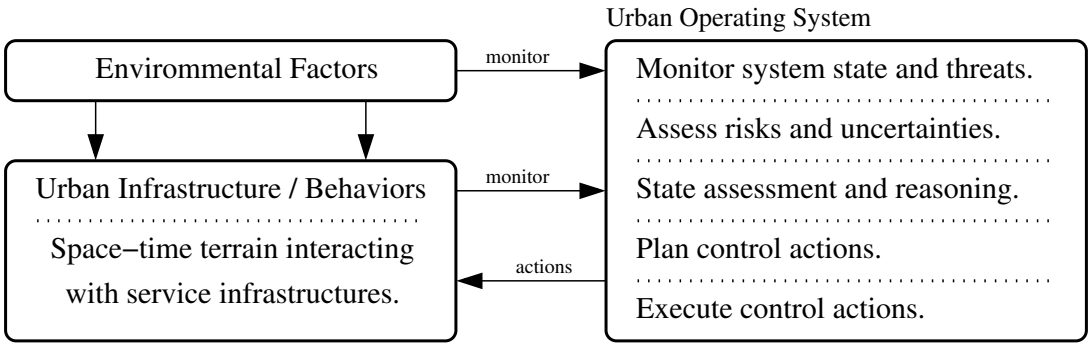


Figure 1.2: Simplified operating systems architecture for urban system planning and control.

chitecture responsible for management of behaviors and resources in urban environments, and urban system planning and control. Such systems will need to sense the surrounding environment, and systematically transform streams of data into knowledge relevant for decision making. As indicated by the annotations in Figure 1.1, these decision making procedures will need to operate across multiple domains.

1.3 Multi-Domain Semantic Modeling and Reasoning

This investigation explores the role that multi-domain semantic modeling and reasoning can play in design and planning of safetly critical urban operations.

Figures 1.3 and 1.4 show the semantic modeling and reasoning framework proposed by Delgoshaei [42] and refined by Coelho [33]. This framework illustrates the higher-level information that is required for the rule-based reasoning framework. The framework shows four main sections including design rules and reasoner, textual requirements, ontologies and models, as well as engineering models. Capabilities of this framework include system structures of engineering models can be treated as composite

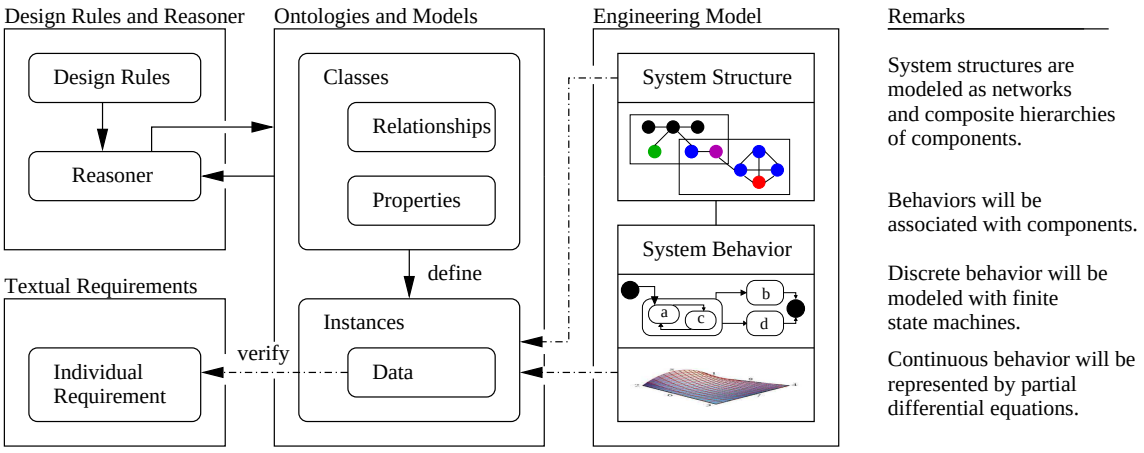


Figure 1.3: Preliminary framework for semantic analysis and rule-based reasoning [42].

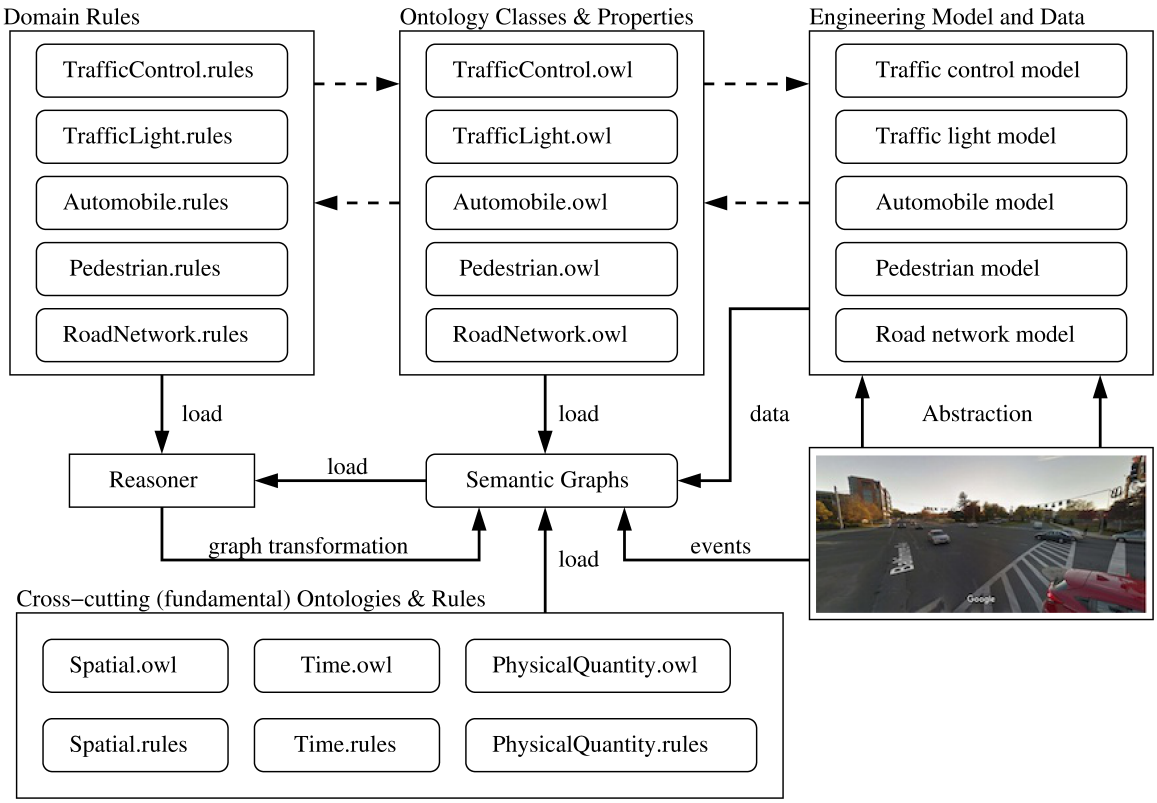


Figure 1.4: Semantic modeling for control of traffic intersection behaviors (main entrance to UMD) (Source: [29]).

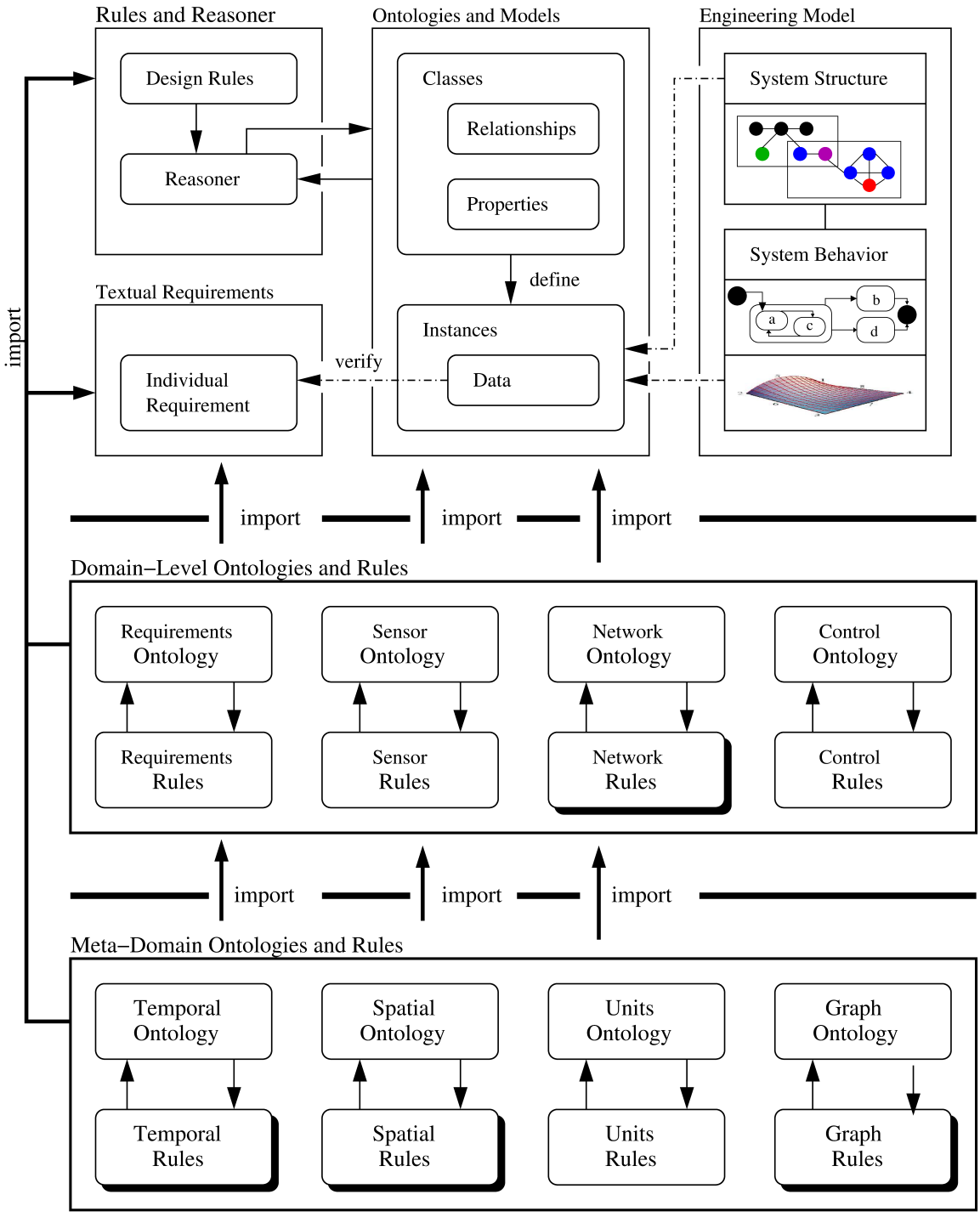


Figure 1.5: High-level framework for multi-domain semantic modeling.

hierarchies of components where the system behaviors can associated with. The discrete system behavior can be modeled with finite state machines and the continuous behavior can be represented by partial differential equations. Figure 1.4 shows the details of a traffic intersection domain that is inserted to the Figure 1.3. The three upper parts of the model are constructed side-by-side meaning the jena rules, ontologies and data model shall be considered for each individual domain simultaneously that will be added to the framework. The lower part of the Figure shows the cross-cutting ontologies and rules that are needed for all of the domains defined above. After all information is added to the `Semantic Graphs`, shown in the center, the events (center-right) that are detected from the data model can initiate graph transformation with the help of the reasoner (center-left).

Figure 1.5 builds upon the early work of Delgoshaei and Coelho, and targets multi-domain semantic modeling and reasoning on urban operations. Looking ahead, domain-specific ontologies and rules are needed for sensor, network and control operations. Urban network ontologies can be viewed as an extension of graphs. To ensure that actions occur in the right place and right time, semantic models and reasoning capability are needed for the meta-domains of space and time.

1.4 Three Motivating Applications of Safety-Critical Urban Planning

The research plan is motivated by behavior modeling and decision making problems in three seemingly distinct, yet similar safety-critical application domains: (1) Safety assessment of vehicles at traffic intersections, (2) Planning and control of planes taxiing at airports, and (3) Timely evacuation of urban residents threatened by wildfires. The three domains are distinct in the sense that they operate across a range of temporal and spatial scales, and mechanisms of control. And, yet, these domains are similar in the sense that behaviors can be viewed as operations on graph structures. It is expected that graphs and support for spatial and temporal modeling will play a pivotal role in keeping these systems safe.

1.4.1 Safety Assessment of Vehicles at Traffic Intersections

Safety Concerns at Traffic Intersections. Despite the abovementioned advances in technology, accidents at traffic intersections claim around 2,000 lives annually within the US alone [80, 121].

Spatio-Temporal Modeling of Traffic Behavior at Intersections. Figure 1.6 shows a plan view of the traffic intersection at main entrance to the University of Maryland (UMD). And Figure 1.7 shows a framework for spatio-temporal modeling of a vehicle making a left-hand turn at a traffic intersection.

Traffic behavior can be viewed as control movement of multiple vehicles on graphs. The control works when vehicles can traverse the intersection without being involved in

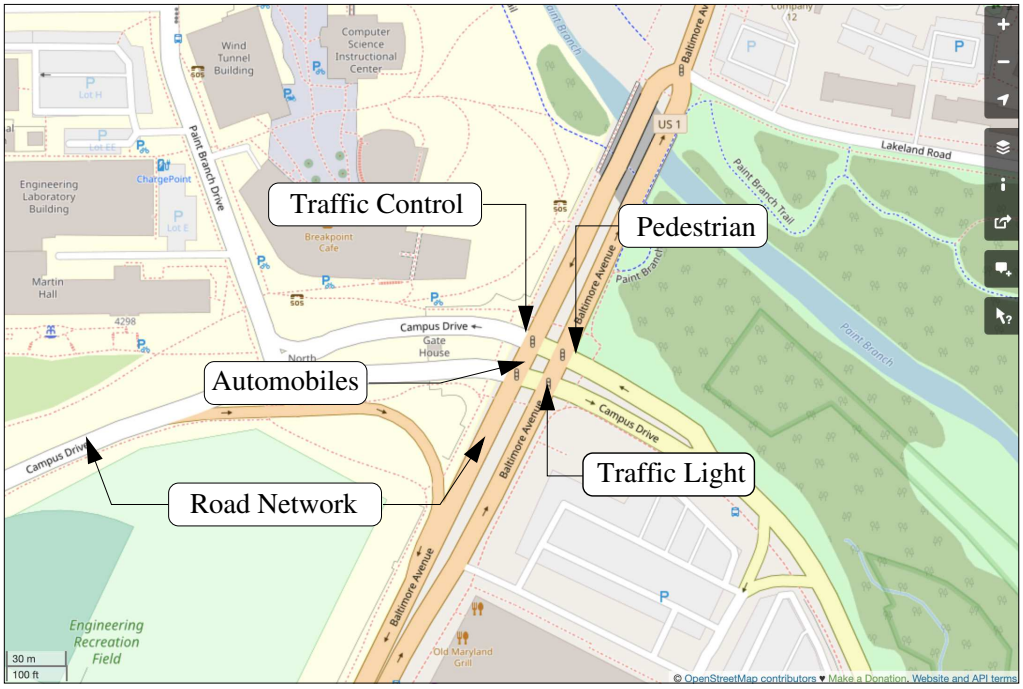


Figure 1.6: Plan view of traffic intersection at main entrance to UMD.

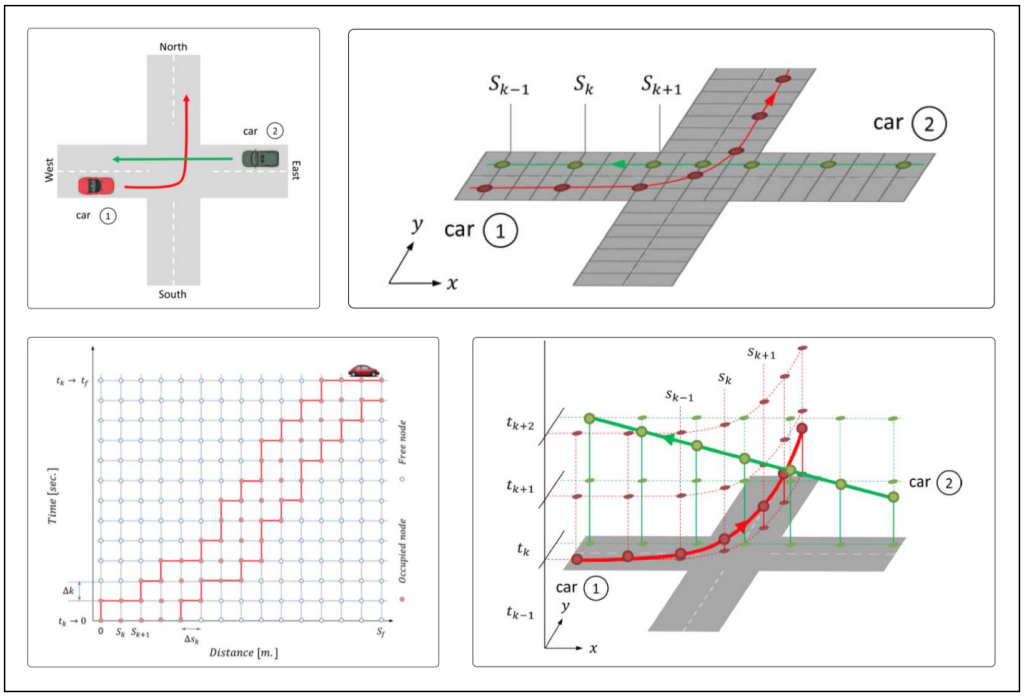


Figure 1.7: Spatio-temporal modeling of a vehicle making a left-hand turn at a traffic intersection.

an accident. From a mathematical standpoint, safety at traffic intersections boils down to whether spatial constraints and temporal trajectories of vehicle are well spaced. This is a space-time reservation model – only one vehicle is permitted to occupy a block of space at any point in time.

1.4.2 Planning of Taxiway Operations at Airports

Safety Concerns at Airports. As aviation systems become progressively crowded, an emerging concern is their diminished ability to deal with heavy work loads and enhanced ground safety concerns [152]. The Federal Aviation Administration (FAA) reports that since 1990, six runway collisions have resulted in 63 deaths. During the time period 2003 through 2006, fifty-four (54) percent of incursions were caused by pilot errors and (29) percent were caused by air traffic controller errors [151, 152]. State-of-the-art solutions have been unable to prevent the frequent occurrence of aircraft wings and tails clipping on airport taxiways all around the world [2, 90, 98].

Spatio-Temporal Modeling of Taxiway Operations. Spatio-temporal modeling of taxiway operations requires similar knowledge and solution strategies as the traffic intersection domain. The ultimate goal of modeling taxiway operations is to plan airplane schedules carefully without causing any accidents and conflicts.

Figure 1.8 is a plan view of layers of data/information extracted from open street map (OSM) for Baltimore Washington International (BWI) airport. The required capabilities of securing safety of the system contains monitoring, evaluating, reasoning and taking corresponding actions. The Figure also shows the information within the

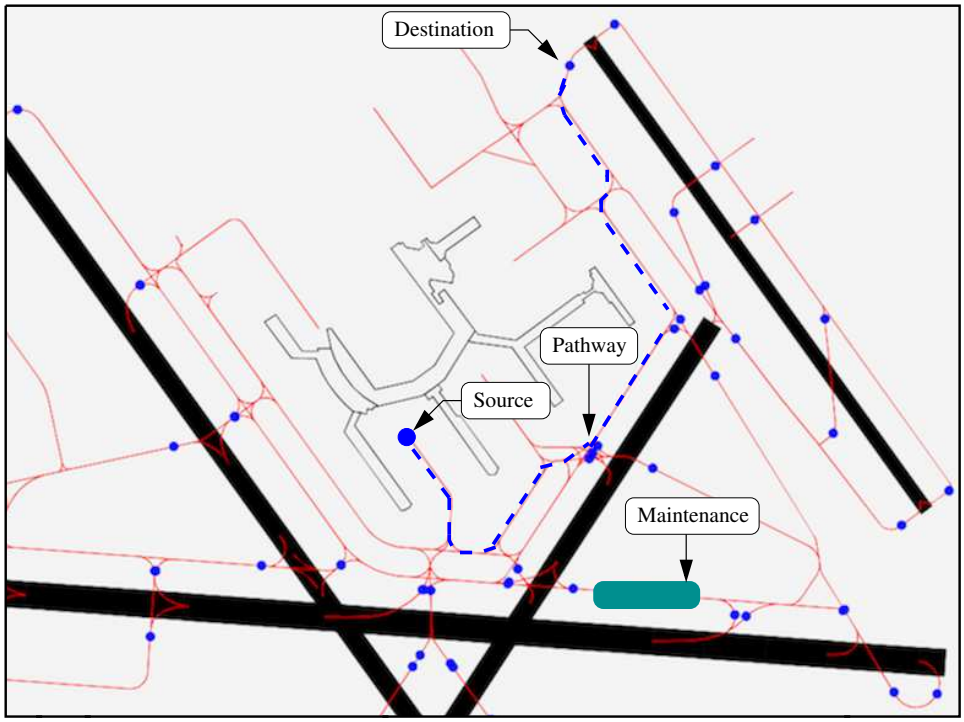


Figure 1.8: Multi-domain view of OpenStreetMap data for Baltimore Washington International (BWI) Airport annotated with pathways for taxiway operations.

airport region, such as buildings, highways, plane holding positions, and railway station, which can all be factored for planning and scheduling purposes.

Figure 1.8 indicates a case of a trajectory-adjustment in a dynamic taxiway environment. The information required for adjusting with dynamic environment is highlighted in blue. At the airport taxiway intersections, there is a shared space which can only be accessed by one airplane at a time. Therefore, gathering useful information in multiple participating domains, efficient planning and control on schedules for each airplane can minimize accidents and reduce time conflicts.

Semantic Modeling and Rule-Based Reasoning with Airport Operations Domain. The airport operations domain design will employ semantic descriptions of the involved domains. This approach allows ontologies and rule-based reasoning to enable require-

ments validation and communication between multiple domains. The defined textual requirements for the domain of airport operation initiates the safety control with rule-based mechanism. The ontology models and rules bring the textual requirements with the engineering models and provide a platform for simulating the development of system structures, and adjustments to system structure and system behaviors over time. The airport operations level of ontologies and rules interprets some domain-specific ontologies that are associated with airport operations and having such information is useful to understand potential risky factors that can endanger the taxiway safety operations.

State-of-the-Art Airport Taxiway Operations. Studies [22] indicate that within the U.S. alone, approximately 700 to 1100 airplanes are delayed by more than 15 minutes per day. These statistics point to a strong need for efficient scheduling of for aircraft take-offs and landings [19]. Recent advances in the mathematical modeling of taxiway operations [110] have come at the problem from an operations research perspective. For example, Balakrishnan et al. (2012) [15] use an an integer programming (IP) formulation to evaluate ground operations at Dallas-Fort Worth Airport. Similarly, Beasley et al. (2000) [19] used mixed-integer zero-one formulation to schedule landings for single and multiple runway scenarios, and common issues covering number of landing planes, precedence constraints, runway workload balance and so on. In a third study, Adacher et al. (2018) [3] looked at routing and scheduling problem for the ground operations and computed through alternative graph modeling method, with a case study on *Malpensa Airport, Italy.* Two objective functions are used to minimize taxiing delay and pollution emission cased by additional plane waiting; Finally, Bersimas et al. (1998) [22] constructed a model to

consider capacity-related elements from NAS (e.g., departure, arrival, sector capacity) and further extended it to consider dependencies on airport enroute runway capacity. This approach aims to solve a large-scale integer programming (IP) and obtain strong formulations for managing air Traffic Flow Management Problem (FMP).

Modeling frameworks have also been constructed for the airport operations. Yin et al. (2022) [167] built a joint apron-runway assignment framework for effective airport network operations with reconfiguration of O-D problems. Scala et al. (2021) [135] proposed an integrated framework that synthesizes simulation and optimization, and resolve scheduling problems considering uncertainties. The opt-sim algorithmic framework is applied on airport capacity management to reduce conflicts through tuning key modeling parameters. Tang et al. (2019) [146] presented simulation and spatial-temporal modeling framework to predict collisions and safety-critical events on ramp areas during surface operations. Historical data is also used for stochastic simulation models and four collision scenarios are considered to support the spatial-temporal simulations.

Review Studies on Airport Operations: These include: Lei et al. (2020) [96] conducted a review on runway capacity evaluation models and distinguished between five runway capacity modes: run-slip structure, parallel runway capacity, near-parallel runway, cross-runway capacity, single and parrallel runway; aircraft gate assignment problem's (GAP's) formulation types, objective classification and solution method categorization are reviewed by Das et al. (2020) [39]. The authors mentioned that no standard formulation is found across literature but common objective(s) is set to cover one or more of: passenger-, aircraft- and robustness-oriented aspects; Shone et al. (2021) [140] reviewed on stochastic

modeling applications for air traffic management through demand, capacity and air traffic congestion modeling. The current OR approaches include: stochastic optimal control, analytically queuing theory, stochastic integer programming and robust optimization.

1.4.3 Modeling of Wildfire Evacuations

Safety Concerns due to Wildfires. Given that one in three U.S. homes are now located within the wildland urban interface, the risk of catastrophic (destructive and costly) loss is significant [125]. During recent years, the combination of severe drought and wildfires have affected the US western states tremendously [43]. Two prominent recent events are the June 2018 Wildfire in Colorado (see Figure 1.9) and the 2018 California camp fire [5].

Spatio-Temporal Modeling of Wildfire/Urban Interactions. Spatio-temporal modeling for the planning of evacuation operations requires similar knowledge and solution strategies as the previous two domains. The ultimate goal is to create schedules for evacuation to a shelter away from the wildfire impact.

Semantic Modeling and Rule-Based Reasoning for Evacuation Operations. A key premise of this dissertation is that semantic modeling and rule-based reasoning can work together with planning software (details to follow) to plan evacuation operations. Solutions to this problem are complicated by the need to work with semantic concepts and data from multiple domains.

Figure 1.10 shows, for example, the data file with the layers of information that are stored with the OSM (i.e., amenities, buildings). Individual layers of information can

Figure 1.9: Panoramic view of Colorado June 2018 Wildfire [99].

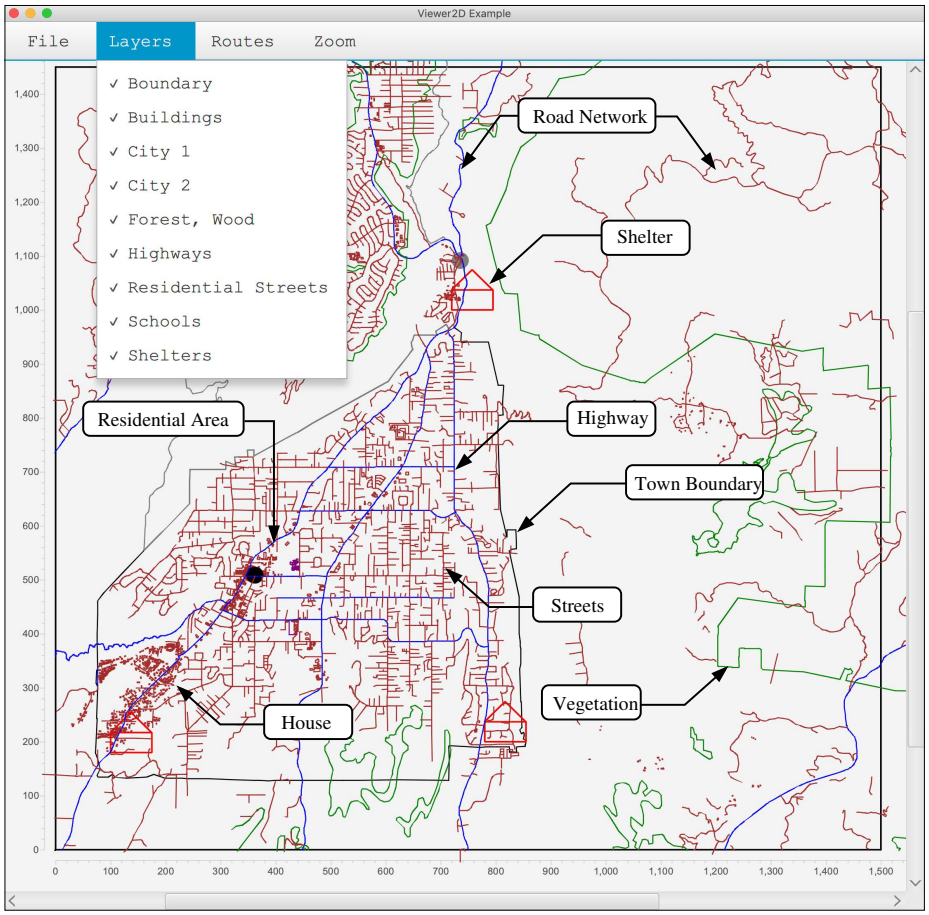


Figure 1.10: Multi-layer visualization of Paradise, CA, with town boundary.

be extracted and used whenever it's needed, which makes it capable of modeling dynamic evacuation processes with the real-time information from the wildfire-engineering model (i.e., locations commuters, spread of fire).

State-of-the-Art Wildfire Evacuation. In state-of-the-art studies of wildfire evacuation procedures, "departure time" serves as a proxy for understanding and analyzing evacuee behavior. Traditional wildfire evacuation analysis includes analysis of departure time via pre-determined distributions and S-curves [36, 104, 163]. However, evacuee departure timing can be affected by multiple reasons. For example, fire dynamics, evacuee characteristics, interactions between evacuees, and others [64, 68, 70], which traditional approaches found difficult to address multiple of these simultaneously. Also, many wildfire evacuation studies rely on favorable and ideal situations for modeling purposes while the extremeness of fast-moving dire wildfire aspect is not well considered [35]. Some wildfire evacuation models are aimed to capture wildfire development and interactions between fire and built environment, and consider dynamic complexity for evacuees [53].

Evacuation models also include survey-based approaches for rapid-onset hazards (e.g., wildfires, tsunamis, hurricanes) modeling [68–70, 105, 107, 162]. Agent-based modeling (ABM) techniques [68–70] can be adopted to simulate wildfire evacuation with the support of survey data describing social characteristics and behaviors of agents. For instance, Wilmot and Mei (2004) [162] conducted perception and behavioral surveys through computer-based telephone interviews for alternative trip generations in hurricane evacuations; an integration of survey and GPS-based can also help to determine evacuation route decisions [105]; Trainor et al. (2013) [150] proposed a merged approach of empirical and

theoretical insights into transportation evacuation modeling and to improve transportation modeling assumptions with improved real world mapping. The social equity aspect of evacuation research can also be modeled with ABMs. Grajdura et al. (2021) [70] stated that finding out fire sooner relates to certain groups and communities (e.g., white, higher income, smart-phone availability and younger ages).

A closely related problem is hazardous evacuation. Reilly et al. (2021) [130] described that a common knowledge (boundary object) is preferred in hazards research to avoid ambiguous coordination and interaction among researchers from various disciplines, who shall work towards a shared goal. Clearly, that the current and traditional scientific approaches of modeling wildfire evacuation is lack of understanding the hierarchy relationships between the physical individuals and shared common knowledge among these interdependent systems are commonly not well defined. This deficiency can be complemented with multi-domain semantic modeling where it defines relationships and data associations behind these physical models and systems.

1.5 Synthesis of Common and Disparate Modeling Requirements

Here is a list of abstractions and problems common to all three applications:

1. **Spatial Domain.** All three applications involve issues related to concept of space or spatial constraint. Analyzing the relations of physical components in each system shows whether some are interfering others' access zones. Spatial relations show many aspects of relations for two geometry objects. For example, two objects can have different combinations of dynamic and static statuses. This enables capturing relations between a moving object and a non-moving object, such as a person and a building, a car and a road.

2. **Temporal Domain.** The concept of time relates to all urban safety issues. Timing is extremely important for safety issues. Two objects happen to appear at the same location but at different time instants, which can avoid collision. Temporal domain regularly involves knowledge of start-time, end-time, time-instant and time-interval. These summarizes the important knowledge in urban domain, such as, when an object leaves, how long it travels and when it returns. Having temporal knowledge of multiple physical units can help to manage schedules and to avoid conflicts.

3. **Graph Domain.** Here, all three urban applications involve notions of graphs. Graph analysis simplifies the issues of geometry shapes and relations in the urban network such that relations among different objects can be described and used together regardless of their shapes and locations. A Java topology suite named - JGraphT is able to conduct graph analysis on various levels. For instance, graph structure,

graph comparisons, graph generation, graph traversal and graph cloning can be used for describing urban systems. In other words, urban network is a simplified graph that has additional capabilities, and these can be used for conducting graph analysis as feature parameters.

4. **Planning and Control.** Major urban issues require well defined schedules and controls in multiple levels. Having a planning and control tool is necessary as it is able to show the relations of temporal aspects for physical entities. This helps to further facilitate the decision-making process as it checks, for example, schedules of other individuals and then respond to changes as execution of actions if any schedule conflict is detected.

5. **Combination of two or more of above domains.** The current developing urban system requires information merged from various domains. For example, in order to avoid two car collapse, we need to assure they do not appear at the same time or at the same location. This phrase implies the coexistence of temporal and spatial aspects in urban settings. In addition, graph analysis enables the geometrical analysis on physical entities (i.e., length, width, height, connectivity) and physical units can provide consistency of measurements. All of these domain-neutral knowledge will be used simultaneously for assuring safety of urban systems.

1.6 Research Scope and Objectives

This dissertation explores development of an integrated framework for multi-domain semantic modeling and reasoning, coupled to OptaPlanner, a real-time planning and optimization tool.

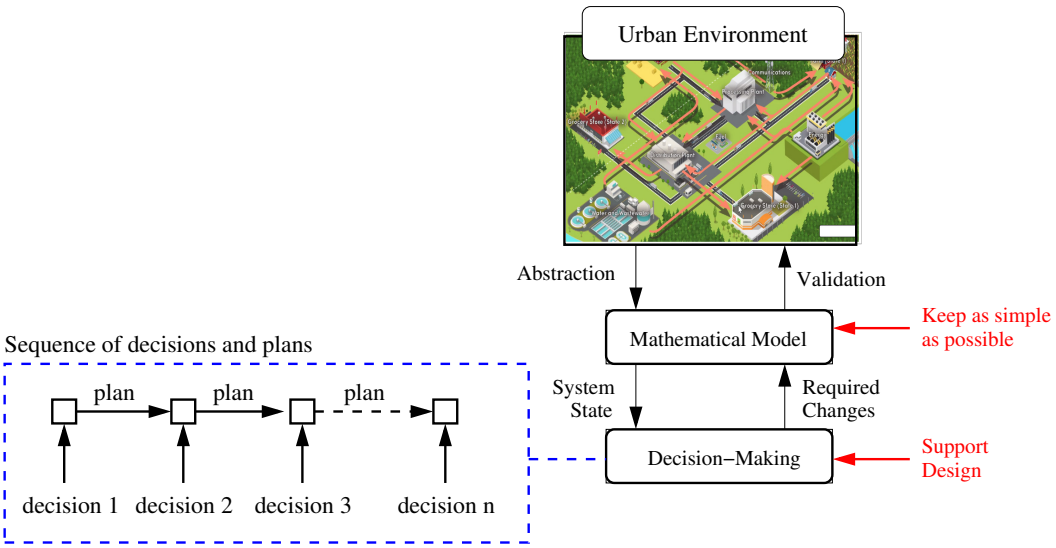


Figure 1.11: Sequences of decision making and planning in the development and operation of systems in the urban environment.

As illustrated in Figure 1.11, solutions to this problem are complicated by the need for sequences decision making and planning coupled to abstractions of a real-world problem domain. This investigation is motivated by abstractions from three safety-critical urban application domains: (a) small urban network operations, (b) airplane taxiing operations, and (c) planning and scheduling for disaster evacuation. From a systems modeling perspective, all three domains share the common interests of decision making supported by high-level situational awareness, and effective planning to avoid schedule conflicts and to assure system's safety.

Research Hypothesis: In order to address the scientific questions, the following scientific hypothesis will be addressed: Planning for urban operations can be supervised by the multi-domain semantic modeling approach.

Research Questions: This dissertation research will focus on the following questions:

1. How to integrate the multi-domain semantic modeling framework with real-time planning?
2. How to use the integrated framework to promote urban operations safety?
3. How can multi-domain modeling and procedures for real-time planning work together to assist decision-making?
4. How to embed the knowledge representation within the multi-domain semantic modeling and constraint-solving?

1.7 Research Contributions

For the past four years, the author has conducted research in ontological models and systems for urban safety, with motivating applications drawn from a variety of urban domains. Along the way, the author has worked with Prof. Deb Niemeier on several research projects relating to wildfires, meatpacking plants, transit bus and EV industries [68, 69].

The primary contribution of this dissertation is development and preliminary assessment of an integrated framework for multi-domain semantic modeling and real-time planning of urban models and operations. The framework and associated problem solving

methodology contribute to the improvement of decision-making in urban operations in three ways: (1) Increasing safety, (2) Mitigation of schedule conflicts in real-time, and (3) Optimal allocation of resources in real-time. The investigation is motivated by three applications: (1) Shortest path and all-paths planning on a small urban network (see Chapters 2 and 5), (2) Aircraft taxiway operations (see Chapter 6), and (3) Wildfire evacuations (see Chapter 7). Defining characteristics of the set of applications include: limited resources, the need for high-levels of situational awareness, and effective planning to avoid schedule conflicts and assurance of system's safety.

Proposed Framework for Semantic Modeling/Reasoning + Planning. The proposed framework will use a combination of domain-specific ontologies and rules coupled with meta-domain ontologies and rules (e.g., for modeling and reasoning with time and space) to model and make decisions in-real time – or near real-time – in response to external events (derived from physical/data models). These events trigger semantic graphs transformations, which, in turn, cause the planner to dynamically adapt plans based on the changes captured in the semantic graphs. Graph transformations can update data attributes of physical entities and assign or remove relationships between these entities. Updated attributes and relationships are sent to the planner for the purposes of replanning. Once planning is completed for this round, then the planning results are returned back to the semantic graphs and merged to the corresponding physical entities for a new round of reasoning since additional events can also trigger planning again. This way, the integrated knowledge-based framework has the ability (1) to understand the relationships among the physical entities in each application domain with its surrounding participating domains,

(2) reason in real-time (near-time) to incoming events, and (3) transform and update decisions for assuring safety and optimizing urban operations. The proposed approach uses a variety of strategies to increase system safety (e.g., plan evacuation strategies to avoid wildfires; plan taxiway operations to avoid collisions).

1.8 Dissertation Outline

The remainder of the dissertation proceeds as follows: Chapter 2 introduces theories, languages and tools used in the Semantic Web. These tools and languages will be adopted extensively in our studies of time, space, and graphs, as well as applications that can be built with the capabilities. Chapter 3 introduces the proposed research methodology –spatio-temporal modeling and reasoning on urban safeties, multi-domain semantic approaches to the urban applications, and corresponding solution procedure. Chapter 4 introduces semantic modeling for time and space with ontologies, rules and reasoning individually and coherently. Two working examples of utilizing time- and space-based reasoning are included at the end of the chapter. Chapter 5 discusses real-time urban planning, and the mathematical formulation of planning problems with OptaPlanner. Chapters 6 and 7 cover the multi-domain semantic modeling and planning approach for airport taxiway operations and wildfire evacuations, respectively. Chapter 8 provides the conclusion and summary of key findings in this dissertation, as well as potential directions for future work.

Chapter 2: Background and Related Work

This chapter covers related work in foundations of model-based systems engineering (MBSE), graph theory and analysis, semantic web modeling and technologies. We introduce the Whistle scripting language [41,42] and describe its capabilities for handling urban data in a variety of formats (e.g., open street map, system and pathway data models). The chapter concludes with semantic and graph-based analysis of a tiny urban network.

2.1 Foundations of Model-based Systems Engineering (MBSE)

2.1.1 State-of-the-Art MBSE with SysML

In the conceptual development, design, construction, verification, and operation of engineering systems, the central goal of model-based systems engineering (MBSE) is development of models, as opposed to documents [82]. Models can be developed for a wide variety of purposes: to represent a simplified concept, phenomenon, structure, system or a relationship. To keep the size of these models in check, knowledge abstraction removes from consideration details that are not needed in decision making processes.

Frontend Development of Systems. Figure 2.1 shows the system operation concept to simplified models for system structure and behavior, system-level design, system require-

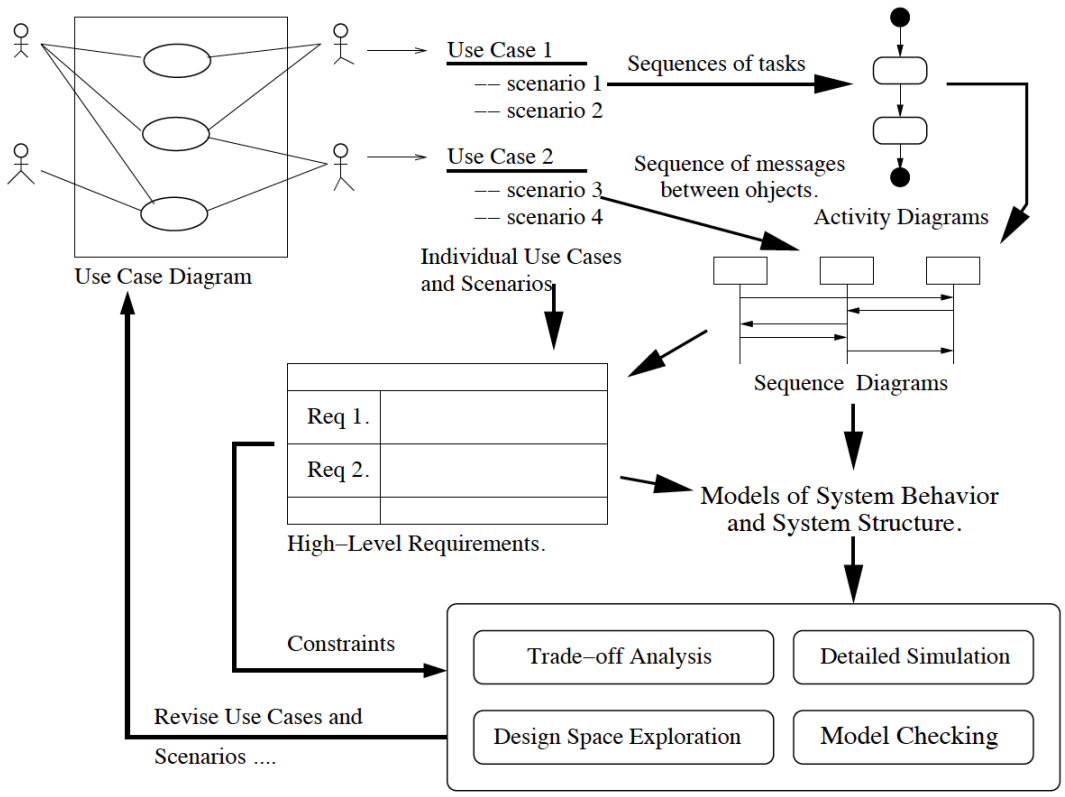


Figure 2.1: Framework for frontend development of systems.

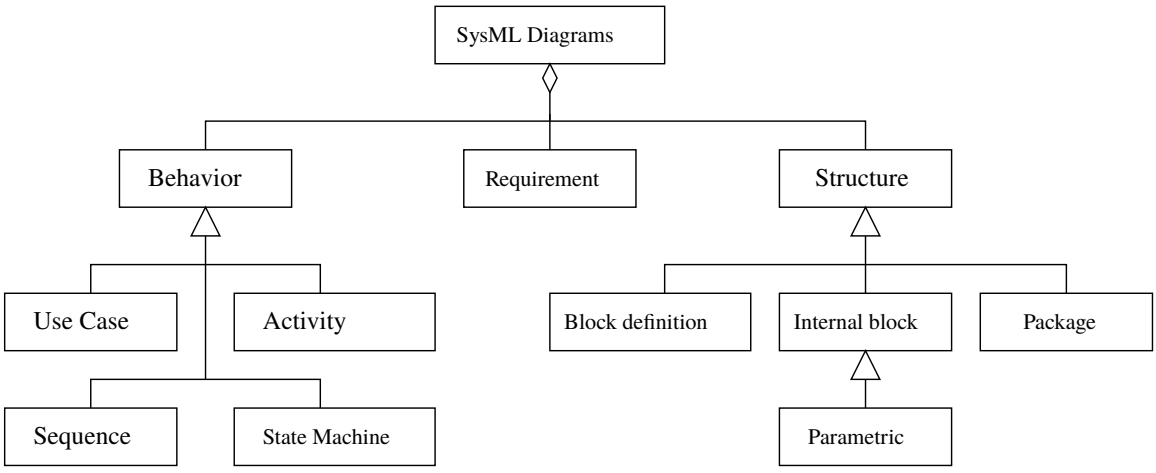


Figure 2.2: Pillars of SysML: structure, behavior, requirements and parametric diagrams.

ments as well as various use cases and diagrams (i.e., activity diagram, sequence diagram). At this early stage of system development, system descriptions will be written as design requirements and mathematical constraints because it regularly does not exist this early in the design phase. System functionality is described by use cases, and scenarios of these various use cases will imply design requirements, objects, object interactions and interfaces. Mathematical constraints and textual requirements can be derived from the communication and structure of objects in models that describes system's functionality. System structure is based on the collections of interconnected subsystems and objects, and limited by the system environment where the main system must exist. The system-level design is built upon the mapping between subsystems and objects in system structure onto the fragments of system functionality. Therefore, the functionality of each component and subsystem of a system is defined by the behavior-to-structure mapping. In addition, the textual requirements are used to evaluate the system performance and characteristics of the system level design.

Dealing with Design Complexity. State-of-the art MBSE procedures deal with design complexity through disciplined approaches to decomposition/composition, separation of concerns, development along the disciplinary lines, and formal approaches to system validation, verification and integration [11, 13, 83, 103].

State-of-the-Art MBSE with SysML. The Systems Modeling Language (SysML) [56] provides standardized formalisms (e.g., the various types of diagrams in SysML) to capture requirements and create semi-formal models of system structure and behavior (shown in Figure 2.2). Visual modeling abstractions such as SysML provide a means

for the development stakeholders to communicate ideas in a domain-neutral manner. Traceability mechanisms allow for connectivity of tentative designs to stakeholder needs.

2.1.2 Transition to MBSE with Semantics

From SysML to Semantics. While semi-formal languages such as SysML provide for efficient communication of ideas among team members, support for formal analysis of representations is weak.

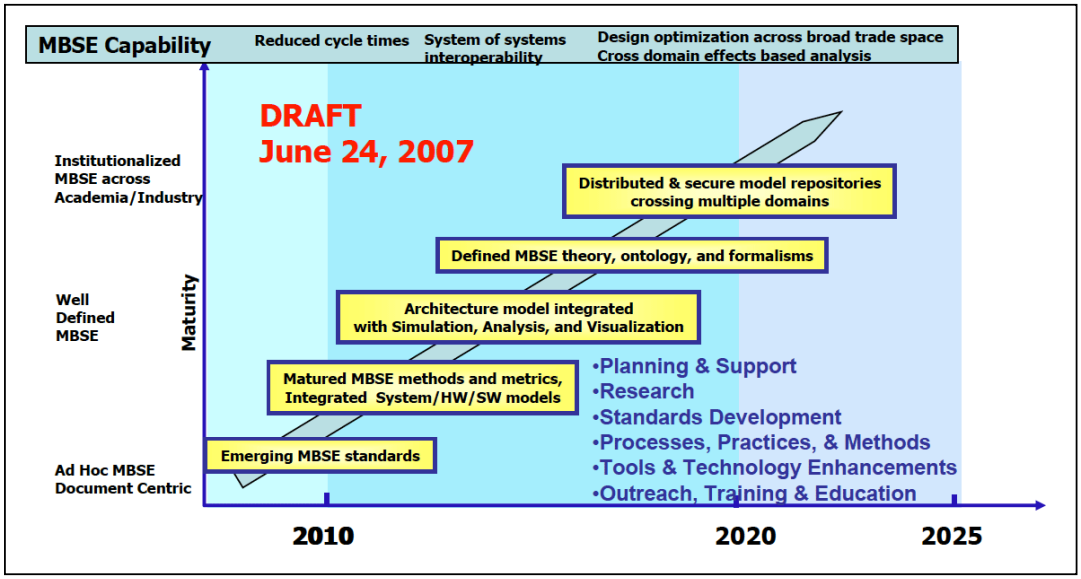


Figure 2.3: MBSE capability: transition from MBSE to Semantics.

Thus, over the past decade (see Figure 2.3), the systems engineering community has been transitioning from MBSE with SysML to MBSE with semantics, integrated system models, knowledge representaion with ontologies and semantic graphs, and rule-based reasoning.

State-of-the-Art Control Architectures. Software control architecture are commonly described with the model-view-controller (MVC) design pattern [160]. This separation of

concerns simplifies the challenge of understanding the internal representation of a system. The 4D-RCS is a model architecture that describes how various software components are organized and identified for military unmanned vehicles [159]. It uses real-time control system (RCS) and has been adopted for various kinds of machine controls nowadays (i.e., autonomous vehicle control). In related work, some recent studies on control architectures are designed for analyzing industrial control hardware with reinforcement learning frameworks [137], reconfigurable manufacturing systems in production capacity and functionality [38], detecting false data injection attacks in networked control systems [60], local control of power converter [45] and many others. This indicates that the adaptability of a well-developed software control architecture can be employed in many areas, especially during the smart city digital twin era to solve for existing problems in urban settings.

2.2 Graphs and Graph Analysis

Graphs are powerful and efficient knowledge representation techniques that appear in many areas of engineering [34]. Within Civil Engineering, for example, transportation and utility network structures can be modeled as graphs. System engineers use a wide variety of graph structures to represent system architectures, the relationship of requirements models to system architectures, and to facilitate tradeoff studies [52, 108]. In the Computer Science field, graph techniques are used to represent the data structure of semantic relations among objects, otherwise known as semantic networks [95].

Mathematical Definition. A graph is a data structure consisting of:

- A set of vertices (or nodes).
- A set of edges (or arcs) connecting the vertices.

Mathematically, a graph G = (V, E), where V is a set of vertices, E = set of edges, and each edge is formed from pair of distinct vertices in V. V and E are usually taken to be finite.

Classification of Graph Types. Figure 2.4 shows the connectivity relationship in six types of graph. A simple graph is an undirected, unweighted graph that does not contain multiple edges and loops. A multi-graph allows having nodes connected with multiple edges. Pseudographs allow self-loop edges. Graphs can also be classified as directed or undirected, weighted or unweighted, connected and/or unconnected.

Graph Theory. Graph theory is the study mathematical structures used to model pairwise relations between objects. Three well-known problems on graphs are: (1) Hamiltonian

Classification of Graph Types

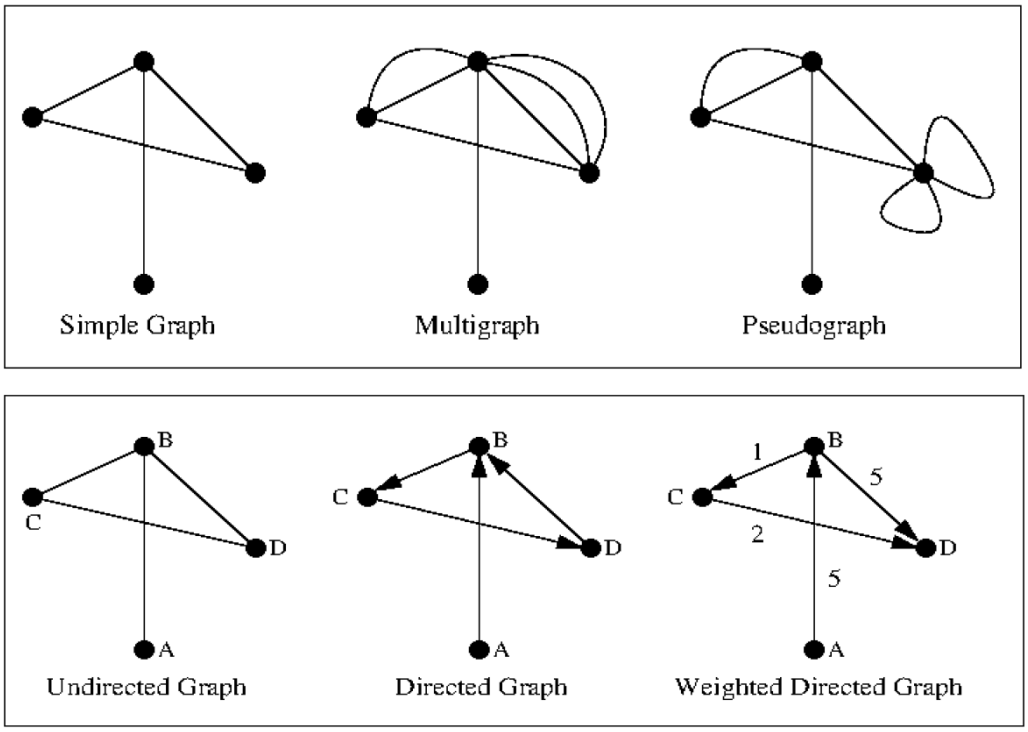


Figure 2.4: Classification of graph types.

Graph Ontology (Mirrors JGraphT Software Architecture)

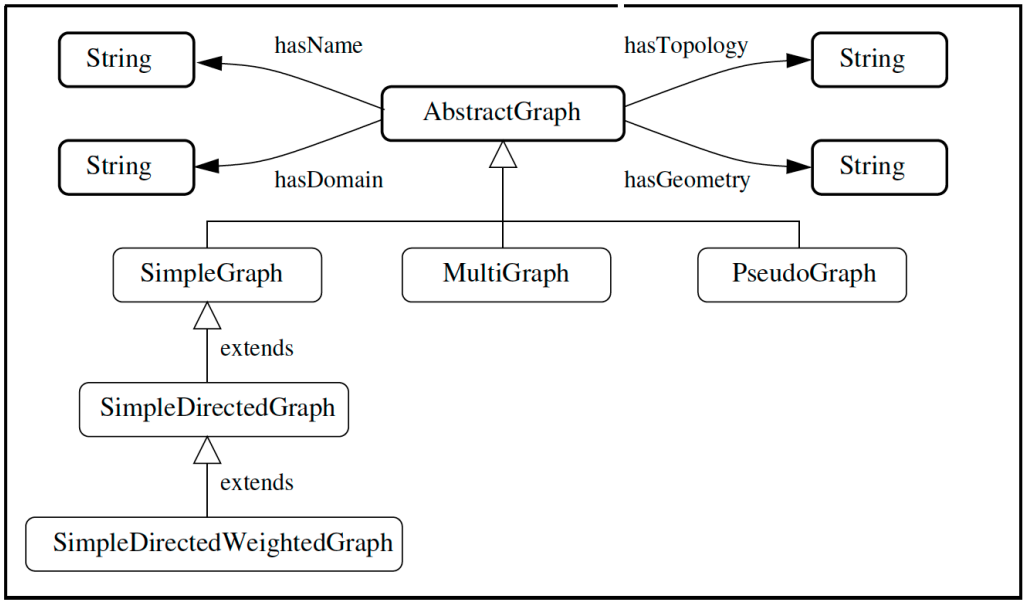


Figure 2.5: Draft of graph ontology mirroring architecture of JGraphT software.

cycle problem (Is it possible to traverse each of the vertices of a graph exactly once, starting and ending at the same vertex?), (2) Chinese postman problem (Find the shortest round-trip path in a graph that visits every edge at least once (road in a mail route)), and (3) Planar graph problem (Is it possible to draw the edges of a graph in such a way that edges do not cross?).

Graph Analysis with JGraphT. JGraphT [87] is an open source Java library for the representation and analysis of graph structures. Computational support is provided for path finding, clique detection, isomorphism detection, coloring, common ancestors, tours, connectivity, matching, cycle detection, partitions, cuts, flows, centrality, spanning, and so forth.

Simplified Graph Ontology. Some relevant research work of using graph-based approach and graph domain are included below. Graph topology initiated a new metric on linear and time-invariant system [155], it shows a clear frequency response interpretation as well as parametric uncertainty to be considered in the metric. An automatic construction of domain ontology from domain corpus is proposed using a graph-based approach [79]. The utilization of graph domain knowledge is not as popular as graph-based analysis on urban safety issues. However, the importance of understanding the multi-domain infrastructural issues with graph domain knowledge is crucial as it closes gaps of some areas that the spatial and other domain-neutral knowledge cannot cover. From a graph standpoint, semantic modeling framework with graph domain of knowledge allows capturing problems in a larger graph scale (see Figure 2.5).

2.3 Semantic Web Modeling and Technologies

2.3.1 Semantic Web Vision

In the late 1980s, Tim Berners-Lee invented the World Wide Web for the scientific community for automatic information-sharing [20]. The two main goals were identified as: (1) To make the Web a collaborative medium, and (2) To make the Web understandable and automatically processable by machines.

Over the past three decades, the first part of this vision has come to pass. The Web is provided as a medium for presenting data and contents to humans, and people can retrieve and render information with the help of machines. At the same time, humans are expected to interpret and understand the meaning of the contents. The purpose of having the Semantic Web is to create a semantic data structure that gives machines to access and share information, thus form a communication of knowledge between various machines, and then new knowledge is automatically discovered. To reach this goal, it requires mechanisms that can initiate the introduction, coordination, and sharing formal semantics of data. It also needs abilities to reason and draw conclusions (inference) through the semantics of data with associated problem domains' knowledge (ontologies). We also note that knowledge can help to formalize mathematical and philosophical analyses [14] which can be implemented within the semantic web modeling for specific problems.

2.3.2 Technical Infrastructure

Figure 2.6 shows the technical infrastructure that supports the Semantic Web vision, and the foundation upon which we hope to build our system-behavior models.

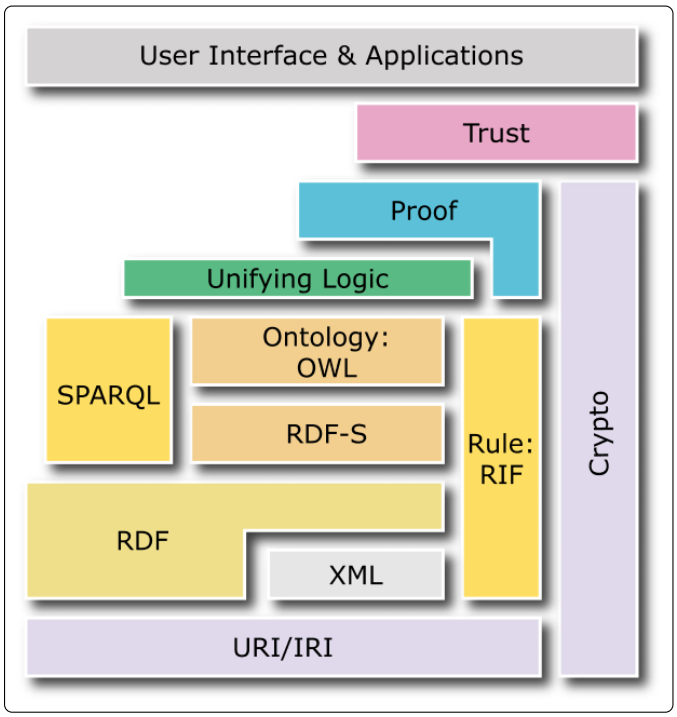


Figure 2.6: Technologies in the Semantic Web Layer Cake [50]

Each new layer builds, exploits and uses capabilities of the layers below. In short, the bottom layer is constructed of Uniform Resource Identifier (URI) and Unicode. URI and Unicode provide capability for linking documents, identifying resources on the Web, and providing 16-bit representation for multi-lingual languages. The extensible Markup Language (XML), on the other hand, is an open standard which describes how to define and utilize simple tree-based data structures within a plain text (human readable format). XML is a meta-language (or set of rules) for defining industry- or domain-specific markup

languages. It can also be used to filter, sort and re-purpose the data for different devices using the eXtensible Stylesheet Language Transformation (XSLT) [147, 164].

Semantic Web applications are able to gather information from various sources, and the context of urban planning, merge and organize these data sources in order to have better decision making. There is no easy way for tree structures to be merged, however. The resource description framework (RDF) solves this issue through allowing for the representation of graphs of data on the web and that's because graphs can always be merged. The web ontology language (OWL) provides for semantic descriptions of underlying data. Having XML, RDF and OWL all together, it allows for the implementation of reasoning that provide whether or not assertions are true or false. This dissertation uses combination of all three for semantic modeling as well as urban safety decision making.

2.3.3 Ontologies, Individuals, Axioms and Reasoning

Ontologies. An ontology is an explicit and formal representation of the associated concepts of a specific domain in terms of classes as well as their relations which is called as "object properties." The classes in the knowledge domain can have attributes that are stored as "data properties" and they also have specific data types and values. Ontologies define taxonomical (hierarchical) relationships between classes that result in the inheritance of object and datatype properties for the subclass, and inherited from superclass. Following is an example in wildfire evacuation ontology

- **Classes:** Evacuee, Shelter, Fire

- **Data properties:** hasAge (double), hasLocation (string), hasGeometry (string), isEvacuated (boolean)
- **Classes:** evacuateTo

Individuals. Individuals are instances of ontology concepts. They are the existing data in the domain.

- **Individuals:** EvacueeI, EvacueeII, ShelterI, ShelterII, FireI, FireII

Axioms. Axioms represent logical statements of the relationships among data and object properties, as well as classes under the same domain. These statements are asserted to be true in the domain that is being described. One common paradigm to assert an axiom is in triple-based format - <*subject, predicate, object*>. In an inferring process, axiomatic systems can represent formal models better than non-axiomatic systems because the systems are composed of axioms. Many logic systems fall under the axiomatic category, for example, first-order and descriptive logic (DL) that is the logical formalism for ontologies defined in OWL.

- **Stored Axioms:** *<:Evacuee :evacuateTo :Shelter>*
- **Stored Facts::** *<:EvacueeI :evacuateTo :ShelterII>*

Reasoning. One of the most important benefits of utilizing semantic modeling and ontologies is using a reasoner to generate and derive additional facts about the concepts being modeled. In the previous example: assertions *<:EvacueeI :evacuateTo :ShelterII>*

and <*:ShelterII :hasLocation (lat01,lon01)*> infers that <*:EvacueeI :isEvacuated :true*> and <*:EvacueeI :hasLocation (lat01,lon01)*> based on the following inference based rule shown in Figure 2.7. This inference based system with rules are mechanisms to derive new information based on the existing data stored in the ontology in the form of: <*conditions*> then <*consequent*>.

```
// -- An inference based rule that infers an evacuee is evacuated
// -- and the location is also stored for this evacuee

(?EvacueeI rdf:Type :evacuee) (?EvacueeI :evacuateTo ?ShelterII)
(?ShelterII :hasLocation ?location01) ->
(?EvacueeI: isEvacuated true)
(?EvacueeI: hasLocation ?location01)

Stored facts :
<:EvacueeI :isEvacuated true>
<:EvacueeI :hasLocation :(string)>
```

Figure 2.7: Sample inference rules.

Ontology descriptions have varying features and capabilities. The purpose of having them varying is to define ontologies that include classes, data and object properties, and their relationships to encode the semantic of the problem domain in a way such it can be processed by machines. In other words, these languages can be provided as a standard and especially an unambiguous way for machines to effectively understand and reason about contextual information and/or context that may refer to an existing entity of the domain. In essence, these contextual information shape the knowledge of the domain machine processable.

2.3.4 Apache Jena and Jena Rules

Not all technologies on the semantic web that are standardized. Some are emergent ones that are used mostly for horizontal and vertical integration of multiple layers of the stack. The Application Programming Interfaces (APIs) are used to complete integration tasks and Jena handles this process. Our prototype software implementation makes extensive use of Apache Jena and Jena Rules.

Apache Jena. Apache Jena [10] is a widely used open source Java framework for constructing Semantic Web and linked data applications. It also provides APIs for developing code that are capable of handling RDF (resource description framework), RDFS, OWL (web ontology language) and SPARQL (support for query of RDF graphs). Jena uses a rule-based reasoning approach, which is the classic technique to logic-based reasoning where the knowledge-based system is developed by deduction, induction, abduction or choices from a starting set of data and rules. A unifying description logic (DL) is needed for horizontal integration of the top layers of stacks and provide the rigorous, formal support needed by applications. In another words, Jena inference features allow a range of inference engines and reasoners to be used on semantic models.

Jena Rules. The Jena inference subsystem is designed to allow a range of inference engines or reasoners to be plugged into Jena. Jena Rules is one such engine. Reasoners provide a means to derive additional RDF assertions which are entailed from some base RDF together with any optional ontology information and the axioms and rules associated with the reasoner. Jena Rules use facts and assertions described in OWL to infer additional

facts from instance data and class descriptions. Such inferences result in structural transformations to the semantic graph model (see Figure 2.8).

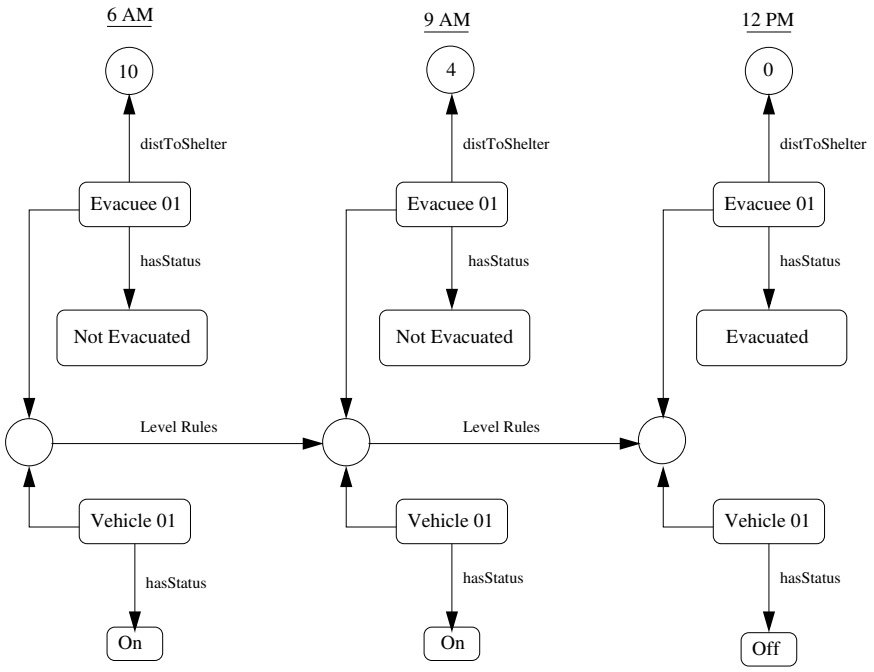


Figure 2.8: Time-based evolution of semantic graph: a simple evacuee example.

Apache Jena provides support for the development of builtin functions that can link to external software programs and streams of data sensed in the real world, thereby extending its reasoning capability beyond what is possible with the basic data types provided in OWL.

Figure 2.9 shows, for example, the essential details for forward and backward chaining driven by data collected from an airport operations setting. To combat the lack of support for complex data types, such as those needed to represent data for spatial and temporal reasoning, we adopt a strategy of embedding the relevant data in character strings, and then designing builtin functions and external software that can parse the data into spatial/temporal models, and then make the reasoning computations that are required.

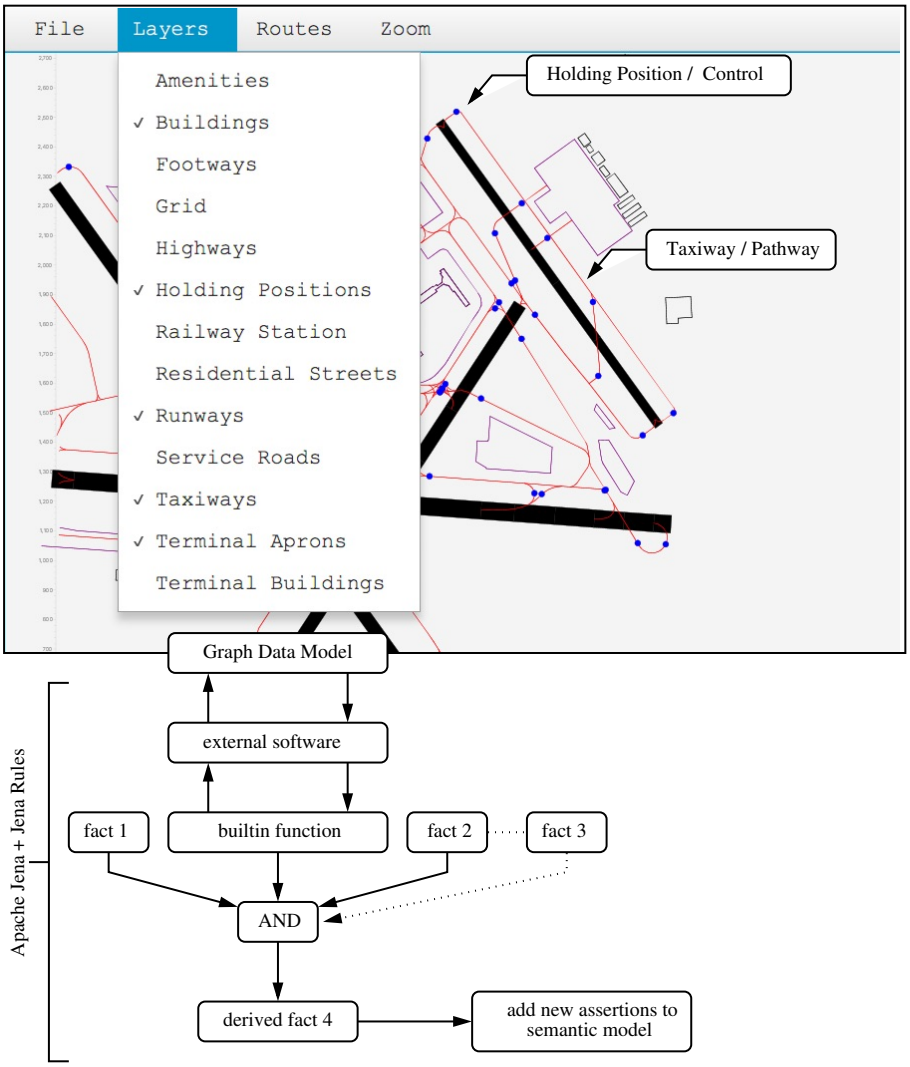


Figure 2.9: Framework for forward chaining of facts and results of builtin functions to assertions (new facts).

2.4 Whistle Scripting Language

Whistle [41, 42, 157] is a Java-enabled scripting language for the definition and assembly of semantic models of urban networks, execution of simulation and planning procedures, and graphical display of urban network structures and and behavior performance.

The Whistle core provides functionality for basic programming (e.g., definition of physical quantities, looping and branching, math computations) and import of Java classes, working with Java collections (e.g., arrays and hashsets), and implementation of software design patterns and two-dimensional visualization. For the specification and modeling of urban applications, the composite hierarchy and visitor design patterns are particularly useful. The Whistle utilities include wrapper interfaces to external packages such as JGraphT [87, 102] (for graph theory data structures and algorithms), Apache Jena (for semantic modeling), the Java Topology Suite (for spatial modeling and reasoning), and OpenStreetMap for modeling of urban domains.

2.4.1 Visualization of Multi-Domain Urban Data

Individual layers of urban data are modeled as composite hierarchies, and organized into stacks for visualization purposes. Figure 2.10 shows, for example, a multi-domain view of OpenStreetMap (OSM) data for Baltimore Washington International (BWI) Airport, organized into thirteen layers. Similarly, Figure 2.11 shows a plan view of an hydraulic network, reservoir and water tank, with the geometries and visualization

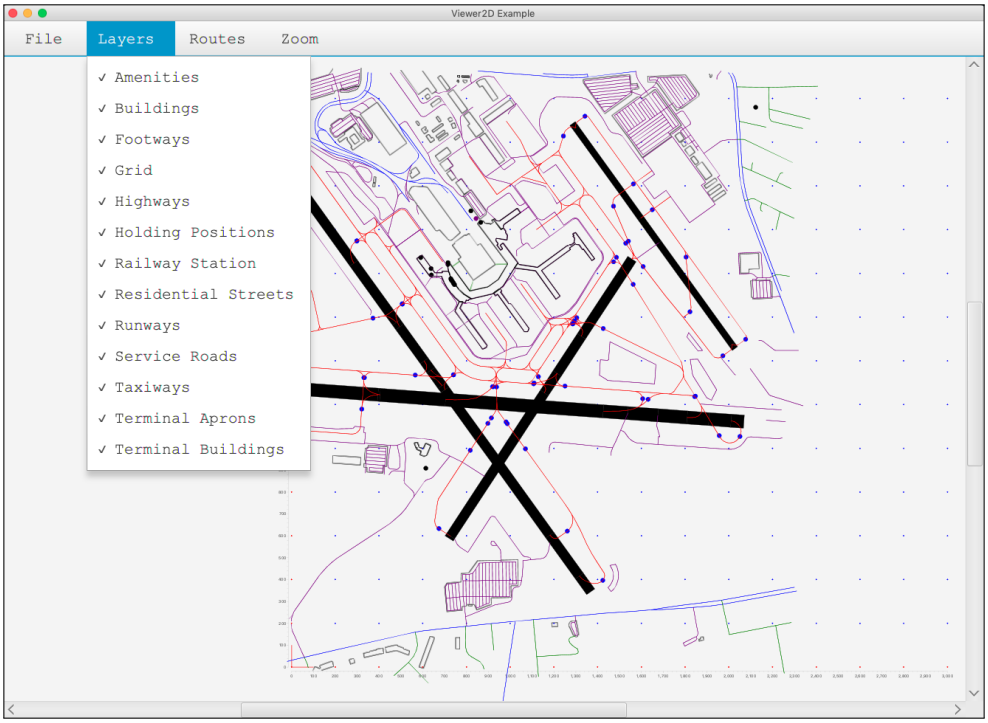


Figure 2.10: Multi-domain view of BWI airport.

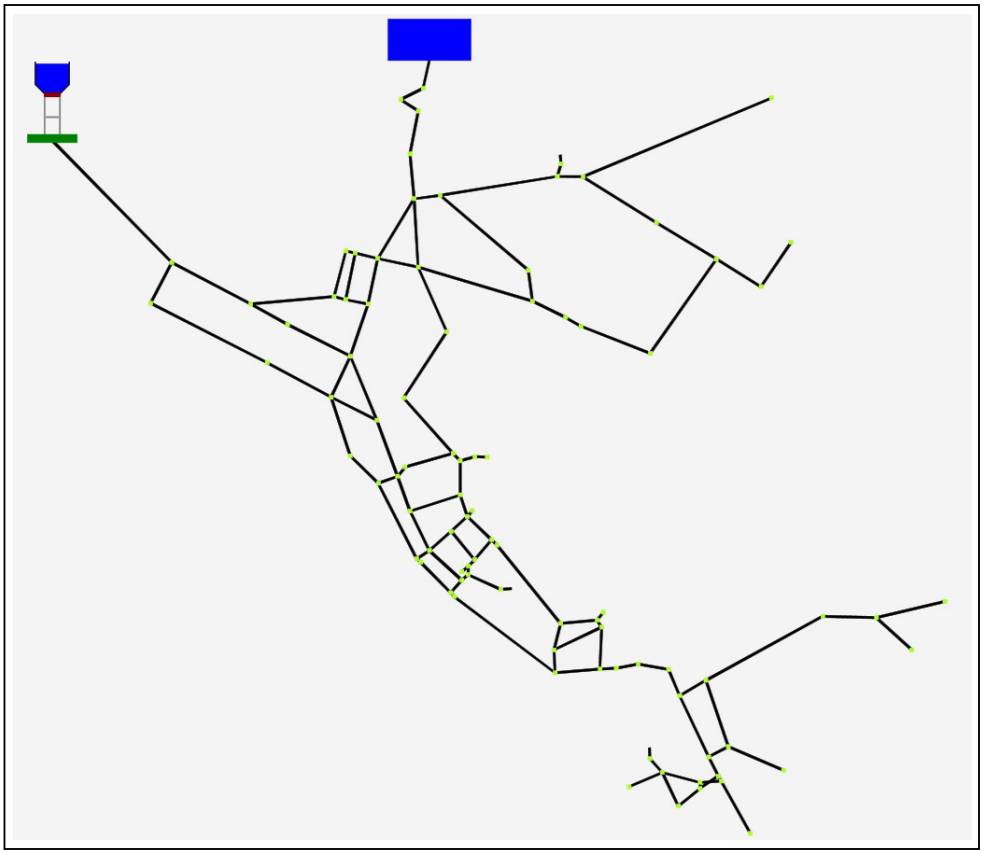


Figure 2.11: Plan view of hydraulic network system.

Fragment of XML in Open Street Map

```
<?xml version="1.0" encoding="UTF-8"?>
<osm>
   <bounds minlat="39.1605000" minlon="-76.6814000"
           maxlat="39.1898000" maxlon="-76.6503000"/>
   <node id="33051350" lat="39.1623308" lon="-76.6726108"/>
   <node id="33051337" lat="39.1590716" lon="-76.6893368"/>
   <node id="33051331" lat="39.1584560" lon="-76.6958300"/>
   <way id="01" >
       <node id="33051350" >
       <node id="33051337" >
       <node id="33051331" >
   </way>
</osm>
```

Fragment of XML in System Data Model

```
<component ID = "C03" x = "400.0" y = "200.0">
   <description text="Elevated Water Tank" />
   <attribute key = "type"      value = "Tank" />
   <attribute key = "elevation" value = "700.0" units ="ft" />
   <attribute key = "area"      value = "400.0" />
   <attribute key = "initlevel" value =   "2.0" />
   <attribute key = "minlevel"  value =   "0.0" />
   <attribute key = "maxlevel"  value =  "20.0" />

   <!-- Visual description of water tank -->

   <compoundshape ID = "Water-Tank-Shape01">
      <shape type = "Polygon">
          <attribute key =   "level" value = "48.0"/>
          <attribute key =   "color" value = "blue"/>
          <attribute key = "opacity" value = "1.0"/>
          <node ID="n01" x =  "0.0" y = "170.0" type="Point" />
          <node ID="n02" x =  "0.0" y = "120.0" type="Point" />
          <node ID="n03" x = "20.0" y = "100.0" type="Point" />
          <node ID="n04" x = "60.0" y = "100.0" type="Point" />
          <node ID="n05" x = "80.0" y = "120.0" type="Point" />
      </shape>

      ... details of shapes removed ...

   </compoundshape>
</component>
```

Figure 2.12: Fragments of XML in open street map and system data model formats.

aspects specified in a system data model file.

2.4.2 OpenStreetMap, System and Pathway Data Models

The Whistle utilities include data model support for geographic, system and pathway data.

OpenStreetMap Data Model. OpenStreetMap (OSM) is a free and open geographic database of the world [114]. Whistle provides computational support for import of OSM XML files into an OpenStreetSap data model, organization of data into domain-specific groups (e.g., groups of buildings, residential streets, etc), and construction of composite hierarchy workspaces for visualization purposes. The OpenStreetMap data model also hosts visitors.

OpenStreetMap Primary Tags. With only three primary tags (i.e., <node>, <way>, <relation>) and their attributes (i.e., <attribute>), OSM can represent the structure of very large urban systems and logical relationships among urban entities. To see how this works in practice, the upper half of Figure 2.12 shows the definition of three nodes and one way in OSM. Nodes represent any kind of point type feature (or named point of interest). Ways are an ordered lists of nodes, often representing linear features such as boundaries, roadways or pipelines. A relation provides a means to logically organize things into groups that naturally belong together. The attributes of nodes, ways and relations are stored as key-value pairs in hash maps. The downside of a relatively flat, but general data storage is that the corresponding data files can be very large (tens of millions of lines of

XML). OSM makes no attempt to describe the visualization aspects (e.g., shape, color) of geodata. Likewise, it makes no attempt to describe behaviors.

System Data Model. As illustrated in Figures 3.1, semantic graphs are populated with individuals (i.e., urban data) by visiting one or more data models. One potential downside of the proposed approach is the burden it places on a developers to create data models for the variety of sources from which data will be mined.

The system data model is an experimental software that aims to provide a single XML data format and parser for reading and storing system structure data and system behavior data. The goals are to build upon OSM with sets of tags to describe components and networks, their attributes and parameters, specifications and constraints, and statechart behaviors, and explore the use of JAXB (as opposed to SAX or DOM in OpenStreetMap) for parsing and processing component and network data models into Whistle.

System Data Model Tag Extensions. The system data model borrows the <node>, <way>, <relation> and <attribute> tags from OSM, and adds support for components (i.e., <component>), specifications and constraints (i.e., <specification> and <constraint>), system parameters (i.e., <parameter>), and component-level behaviors (i.e., <behavior>).

Components are abstractions that simplify system modeling by bundling groups of elements into cohesive units that have a well-defined boundary, and support input and output flows through ports (see Figure 2.13). Within the component, attributes and parameters are described with the <attribute> and <parameter> tags. The system data model assumes that component-level behaviors (i.e., <behavior> tag) can be adequately described by

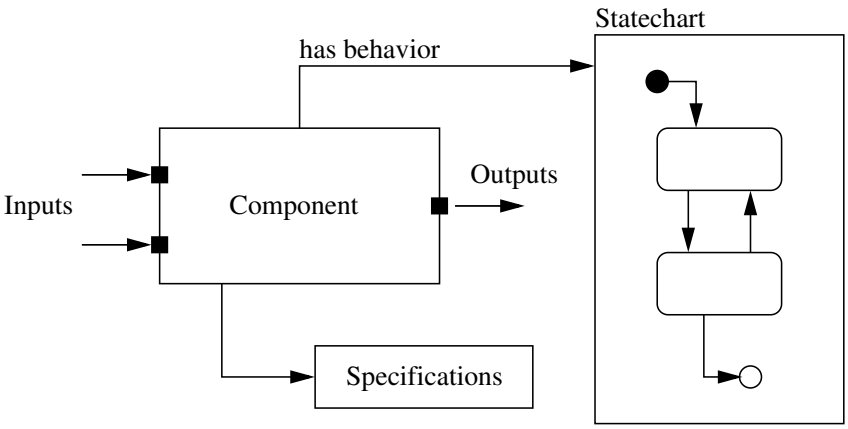


Figure 2.13: Abstract representation of a component model.

either statecharts or patterns of loading that will be applied to a component. Component performance can be evaluated with respect to mathematical constraints. Specifications for attainable levels of component performance can include performance curves (e.g., to describe head-flow relationships in a water pump).

We add the tags <compoundshape> and <shape> (e.g., see the lower half of Figure 2.12) to control the way in which the visual aspects of data elements are organized and drawn. Shape attributes specify parameters for the sizing (e.g., width and height) and displaying (e.g., opacity, color, depth level) an entity (e.g., circle, square, linestring, polygon and multipolygon). A compound shape is simply a list of simple shapes enclosed within a compound shape tag.

The system data model also supports representation and evaluation of mathematical constraints. Equality, inequality and logical constraints are defined by sets of parameters (i.e., name and value) and expressions stored in a character string format. In this project, constraints are extended to include premise-action rules.

Pathway Data Model. As shown in figure 2.14 pathway data model stores a list of `WayPoint` that is being generated through simulation. Based on the urban domain problem, various pathways shall be generated, simulated and then added to the pathway data model. Eventually, the data stored in pathway data model will be sent to the `Domain Data Model` which later interacts with the semantic graph defined with `Jena Semantic Model`.

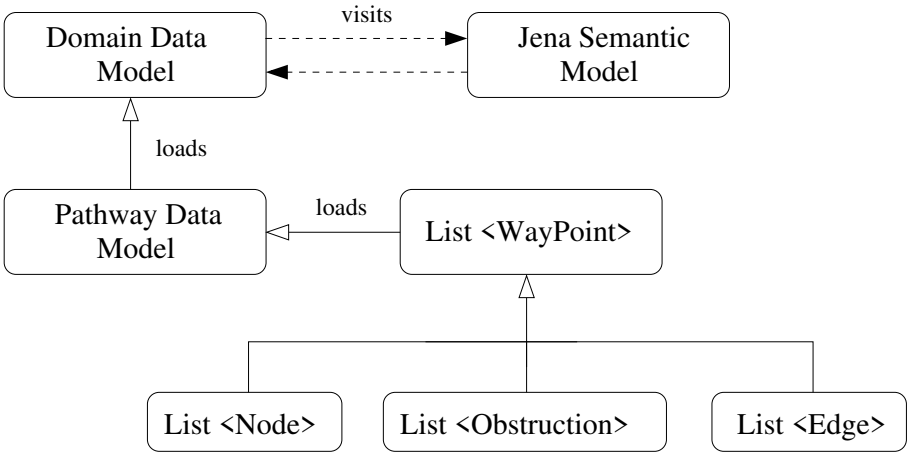


Figure 2.14: Pathway data model and its interaction with domain data model and jena semantic model.

In order to initiate planning capability in urban domains, pathway data model is created as a temporary bridge to store the actual list of points that define the pathway, which is needed in the domain data model for capturing events and making associated decisions. The reason of having a pathway data model helps simplifying the process of integrating various formats of pathway models to the domain data model. Within the list of way points, there are three lists of objects that are part of the way points. They are `Node, Edge and Obstruction`. This process greatly promotes a hierarchy relationship within the pathway data model that a list of nodes, a list of obstructions and a list of edges can be

extracted separately to fulfill the purposes of smart planning. It means that selected data can be selected to feed to the domain data model in order to reduce the computational cost by feeding in unnecessary data sets.

2.4.3 Data-Driven Generation of Individuals

Semantic models are the composition of ontologies, rules and data in the proposed framework. Figure 2.15 illustrates a data-driven approach to the generation of individuals in semantic graphs.

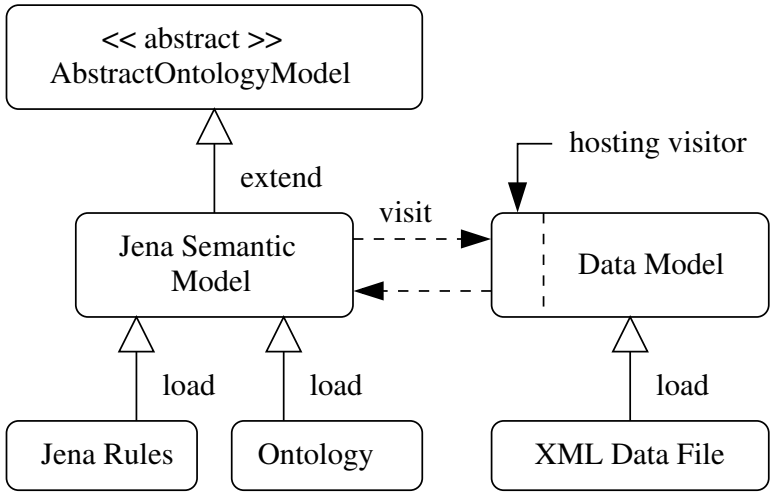


Figure 2.15: Data-driven approach to generation of individuals in semantic graphs.

First, data is imported into Java Object data models using JAXB, the XML binding for Java. After the ontologies and rules have been loaded into the Jena Semantic Model, the semantic model creates instances of the relevant OWL ontologies by visiting the data model and gathering information on the individuals within a particular domain (e.g., building, sensor, occupant). Once the data has been transferred to the Jena Semantic Model and used to create an ontology instance, the rules are applied.

2.5 Working Example: Modeling and Analysis of a Tiny Urban Network

2.5.1 Problem Description

This section exercises the system data model, semantic model and reasoning, and graph analysis capabilities of JGraphT on a tiny urban network test problem. Figure 2.16 shows the network infrastructure modeled with undirected edges. Figure 2.17 shows the same network infrastructure modeled with weighted directed edges, along with a minimum weighted cost pathway from node A to node L. And Figure 2.18 shows the network infrastructure modeled with weighted directed edges after a flood obstruction. We will compute a minimum weighted cost (all) pathways analysis from sets of sources (nodes A, B and C) to sets of destinations (nodes L, M and N). Finally, the exercise demonstrates use of Whistle and, in particular, what information is required and used in order to import data sources to the knowledge-based framework which supports reasoning and derivation of new knowledge.

2.5.2 Load Problem Description into SystemDataModel

The tiny network problem is defined in an XML file (see Figures 2.19 and 2.20) as lists of nodes and ways. The fragment of code below defines three nodes and two edges. The coordinates of each node is defined within the node object and the way object can be defined by assigning two node references which are two end nodes of an edge. A relation object is also shown which can store a list of ways to define, in this case, a pre-assigned pathway.

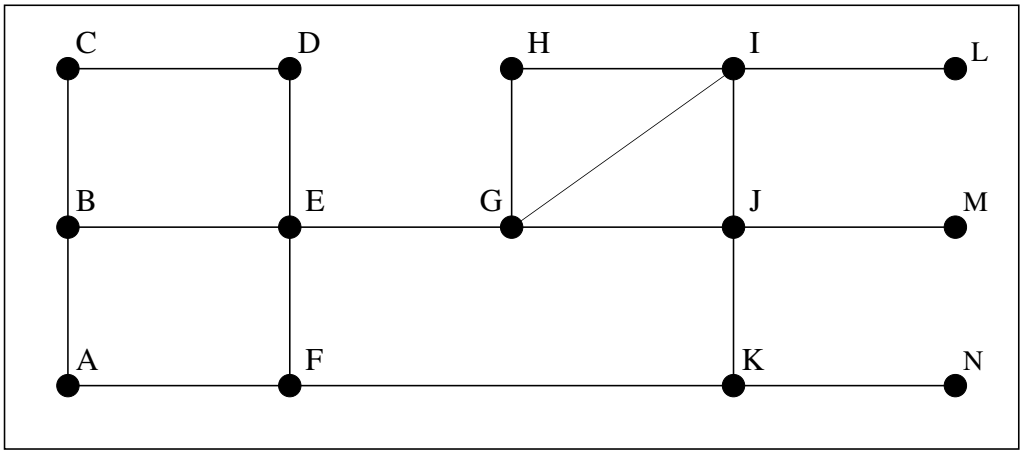


Figure 2.16: Tiny town network structure with undirected edges.

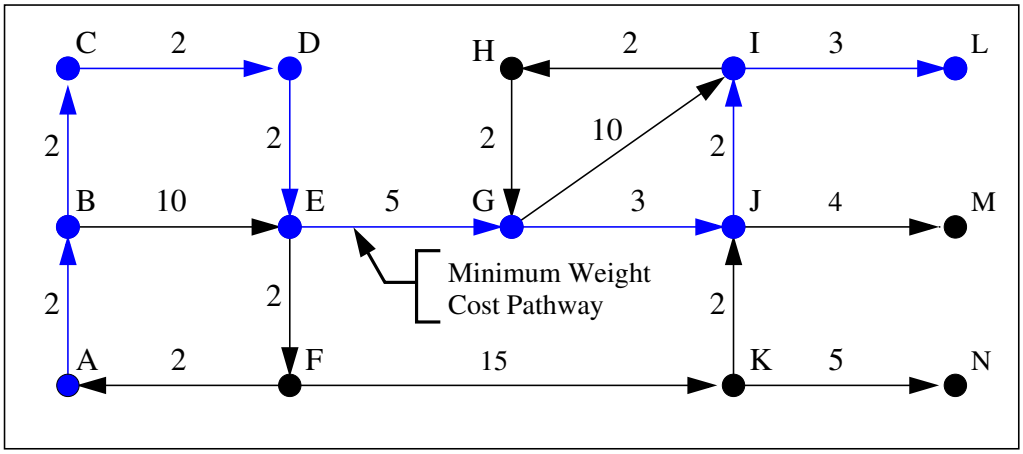


Figure 2.17: Tiny town network structure with directed weighted edges and minimum weighted cost pathway from node A to node L.

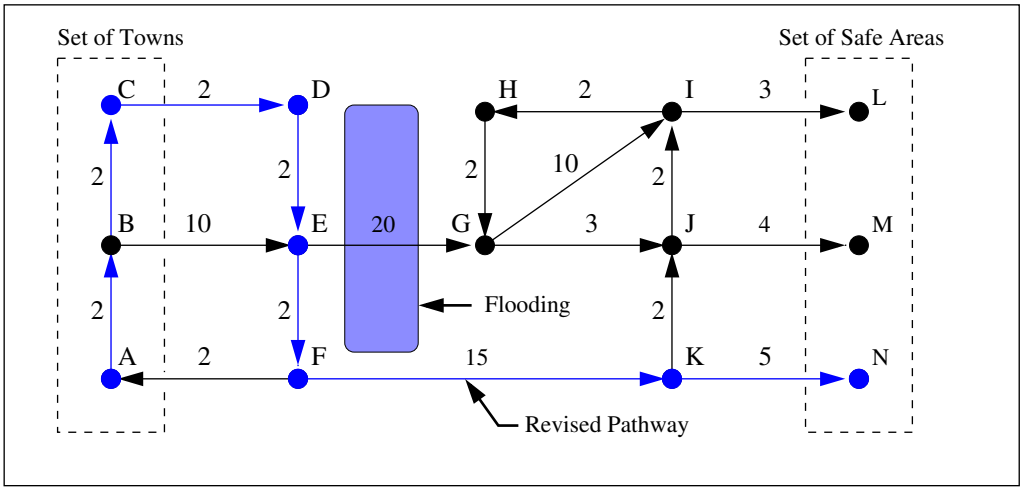


Figure 2.18: Minimum weighted cost (all) pathways analysis from sets of sources (nodes A, B and C) to sets of destinations (nodes L, M and N).

Abbreviated Fragment of Software Code:

```
// Define nodes ...

  <node ID = "001" x =   "0.0" y =   "0.0" type="Point"/>
  <node ID = "002" x =   "0.0" y = "100.0" type="Point"/>
  <node ID = "003" x =   "0.0" y = "200.0" type="Point"/>

  ... details of other nodes removed ...

// Define edges ...

  <way ID="001">
     <attribute key = "type" value = "Edge"/>
          <attribute key = "weight" value = "2.0"/>
     <node ID="001" />
     <node ID="002" />
     <shape type = "LineString">
          <attribute key = "width" value =  "1.5"/>
          <attribute key = "color" value = "black"/>
     </shape>
  </way>

  <way ID="002">
     <attribute key = "type" value = "Edge"/>
          <attribute key = "weight" value = "2.0"/>
     <node ID="002" />
     <node ID="003" />
     <shape type = "LineString">
          <attribute key = "width" value =  "1.5"/>
          <attribute key = "color" value = "black"/>
     </shape>
  </way>

  ... details of other ways removed ...
```

Figure 2.19: Portion of code for the urban network input file (part a).

Abbreviated Fragment of XML Code:

```
// Define relations ...

<relation ID="001">
   <description text="Pathway 01 for the Urban Network." />
   <attribute key = "type" value = "Pathway"/>
   <attribute key = "end1" value =    "001" />
   <attribute key = "end2" value =    "014" />
   <attribute key = "length"    value = "800" />
   <attribute key = "status"    value = "Open" />
   <way ID="001" />
   <way ID="002" />
   <way ID="003" />
   <way ID="004" />
   <way ID="005" />
   <way ID="009" />
   <way ID="018" />
   <shape type = "MultiPolygon">
     <attribute key = "width" value =  "5.0"/>
     <attribute key = "color" value = "Orange"/>
   </shape>
</relation>
```

Figure 2.20: Portion of code for the urban network input file (part b).

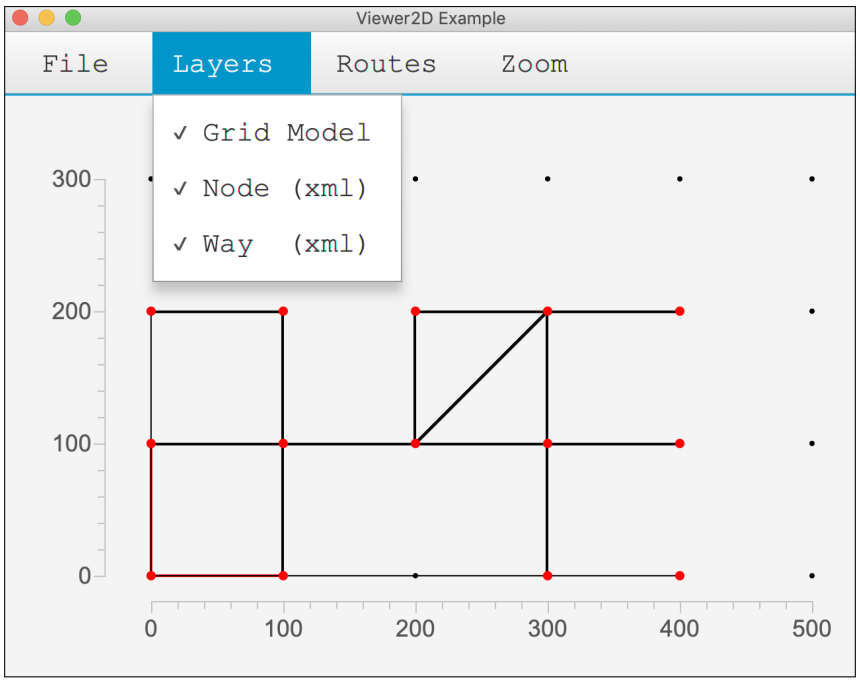


Figure 2.21: Tiny town network structure, created with System Data Model.

Figure 2.21 shows the geometry replication of the town using System Data Model in Whistle. The detailed data file (in XML) is attached in the appendix where it contains the nodes and edges of this simple urban network. The weight of each edge is also stored as an attribute shown in the fragment of source data code above (under `way` object). In the Figure, the nodes are red dots and the edges are black lines.

2.5.3 Define Tiny Urban Network Ontology and Semantic Graph

Figure 2.22 shows a simplified family ontology of its classes, object properties and data properties. There are six classes - `UrbanNetwork, Component` and `Node, Edge, Pathway` and `Obstruction` that are subclass of `Component`. The `UrbanNetwork` has an object property of `hasComponent` which connects it to `Component`; The `Component` has a data property of `hasStatus`; The `Edge` and `Obstruction` both have a data property of `hasWeight` which defines the weight of an edge or an obstruction.

The software code in Figures 2.23 and 2.24 show the assembly of urban network ontology classes and properties which results into a hierarchy for the `UrbanNetwork`. Relationships between these six classes are settle by the middle portion of the code where subclasses are added with the `addSubClass()` method. Furthermore, the data properties are also assigned where the `createDatatypeProperty()` is used to create a data property, and `setDomain()` and `setRange()` methods are used to define the domain it belongs to (in this case - `Edge`) and property type (i.e., `XSD.double`). Because of the defined ontology, the subclasses of `Component` will automatically inherit a data property of `hasStatus` because of the inheritance relationship between subclass and superclass.

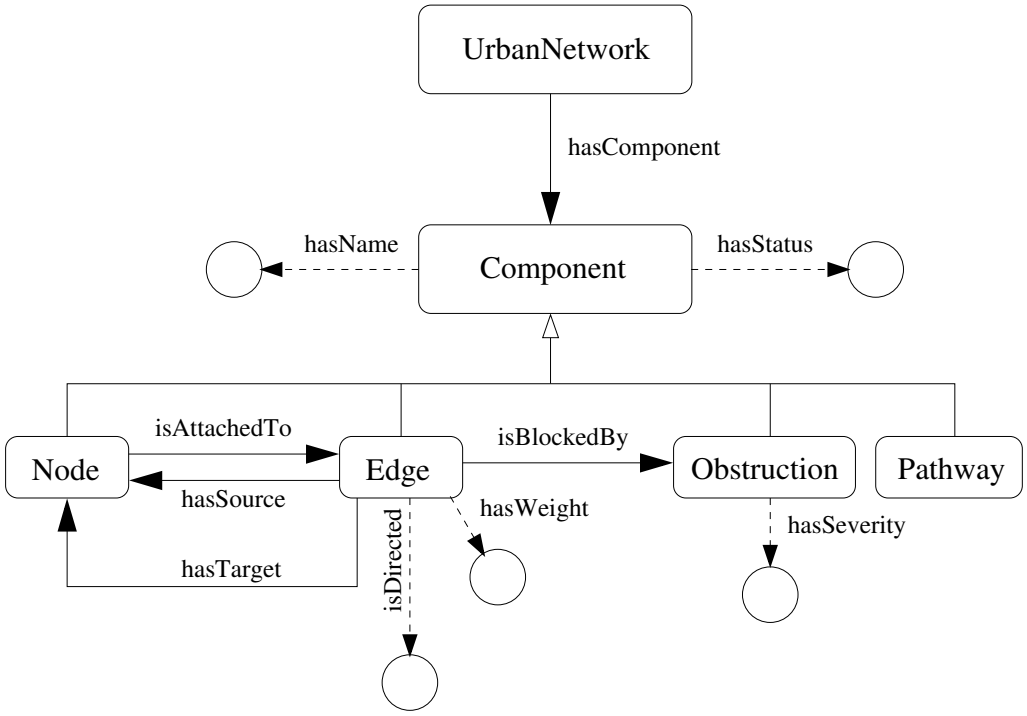


Figure 2.22: Simplified urban network ontology with its classes, object and data properties.

2.5.4 Add Individuals to Semantic Graph

Following with the assembly of ontology graph, urban network ontology individuals, data properties of these individuals, and object properties among them shall be defined with the urban network ontology (see Figure 2.25). The code below starts with establishing a name space for the urban network ontology, constructing an ontology model (graph) which stores individuals and their data and object properties, and then creating an individual for class `Edge` `Edge#01` that has a data property of `hasWeight` and the value is assigned as `5.0`, with type `XSDDatatype.XSDdouble`. Similarly, an individual `Obstruction#01` has a data property of `hasSeverity` is added and the corresponding value is `3`, and with a datatype of `XSDDatatype.XSDinteger`. We will explain the following sections how Jena can support querying of semantic graphs subject to a wide

Abbreviated Fragment of software Code:

```
// Define classes in the urban network ontology ...

   urbannetwork = model.createClass( ns + "UrbanNetwork" );
   component    = model.createClass( ns + "Component" );
   node         = model.createClass( ns + "Node" );
   edge         = model.createClass( ns + "Edge" );
   obstruction  = model.createClass( ns + "Obstruction" );
   pathway      = model.createClass( ns + "Pathway" );

// Define relationships among classes ...

   component.addSubClass ( pathway );
   component.addSubClass ( edge );
   component.addSubClass ( node );
   component.addSubClass ( obstruction );

// Create object properties for the class Component ...

   hasComponent = model.createObjectProperty( ns + "hasComponent" );
   hasComponent.setDomain( urbannetwork );
   hasComponent.setRange( component );

// Create object propertt "isBlockedBy" for the class Edge ...

   isBlockedBy = model.createObjectProperty( ns + "isBlockedBy" );
   isBlockedBy.setDomain( edge );
   isBlockedBy.setRange( obstruction );

// Create object property "hasSource" for the class Edge ...

   hasSource = model.createObjectProperty( ns + "hasSource" );
   hasSource.setDomain( edge );
   hasSource.setRange( node );

// Create object property "hasTarget" for the class Edge ...

   hasTarget = model.createObjectProperty( ns + "hasTarget" );
   hasTarget.setDomain( edge );
   hasTarget.setRange( node );

// Create object property "hasEdges" for the class Node ...

   hasEdges = model.createObjectProperty( ns + "hasEdges" );
   hasEdges.setDomain( node );
   hasEdges.setRange( edge );
```

Figure 2.23: Fragment of code to manually assemble urban network ontology and semantic graph (part a).

Abbreviated Fragment of software Code:

```
// Create data property "hasWeight" for the class Edge ...

   hasWeight = model.createDatatypeProperty( ns + "hasWeight" );
   hasWeight.setDomain( edge );
   hasWeight.setRange( XSD.double );

// Create data property "isDirected" for the class Edge ...

   isDirected = model.createDatatypeProperty( ns + "isDirected" );
   isDirected.setDomain( edge );
   isDirected.setRange( XSD.boolean );

// Create data properties for the class Obstruction ...

   hasSeverity = model.createDatatypeProperty( ns + "hasSeverity" );
   hasSeverity.setDomain( obstruction );
   hasSeverity.setRange( XSD.integer );
```

Figure 2.24: Fragment of code to manually assemble urban network ontology and semantic graph (part b).

range of search criteria.

2.5.5 Query System Data Model and Semantic Model

Computational support is provided for querying the contents of system data models and semantic graphs.

Query System Data Model. In this section, querying for the system data model is shown in Figure 2.26. First, the `SystemDataModel` object is created and assigned as `sdm01`. Then, the `getData()` method in `SystemDataModel` class is called in order to extract the data file of `data/umd-urban-network01.xml` and save it under the `sdm01` object. The next line of code shows the printing method to print the data in the object. The second

Abbreviated fragment of Software Code:

```
// Namespace for tiny urban network ...

   String ns = "http://cee.umd.edu/urban#";

// Create ontology model (a graph) ...

   OntModel model = ModelFactory.createOntologyModel();

// Retrieve classes and associated properties from semantic model ...

   OntClass urbannetwork = jsm.findNamedClass("UrbanNetwork");

// Add "edge01" to the urban network graph model ...

   Individual edge01 = urbannetwork.createIndividual( ns + "Edge#01" );
   model.add( edge01 );

// Create statement:   Edge#01's hasWeight "5.0"

   Literal weight01 = model.createTypedLiteral( "5.0", XSDDatatype.XSDdouble );
   Statement   ebd = model.createStatement( edge01, hasWeight, weight01 );
   model.add ( ebd );

// Add "obstruction01" to the urban network graph model ...

   Individual obstruction01=urbannetwork.createIndividual(ns + "Obstruction#01");
   model.add( obstruction01 );

// Create statement:   Obstruction#01's hasSeverity "3"

   Literal severity01 = model.createTypedLiteral( "3", XSDDatatype.XSDinteger );
   Statement sbd = model.createStatement(obstruction01, hasSeverity, severity01);
   model.add ( sbd );
```

Figure 2.25: Fragment of code to add individuals to the urban network semantic graph.

portion of the code shows the querying of various components that are saved under the `sdm01` object. To accomplish that, the `getComponents(), getNodes(), getWays()` and `getRelations()` methods are used to extract the corresponding list of objects. Appendix **??** shows the details of the system data model where Appendix A.1 shows the details of the input `XML` file and Appendix A.2 shows querying of the semantic data model.

Abbreviated Fragment of Software Code:

```
// Import and print sdm01 Urban Network Data

   SystemDataModel sdm01 = new SystemDataModel();
   sdm01.getData ( "data/umd-urban-network01.xml" );
   System.out.println( sdm01.toString() );

// Query sdm01 for a list of components, nodes, edges and pathways

   List components = sdm01.getComponents();
   List nodes      = sdm01.getNodes();
   List edges      = sdm01.getWays());
   List pathway    = sdm01.getRelations();
```

Figure 2.26: Load and query the urban network data model (system data model).

Query Semantic Graph. Querying semantic graph (see Figure 2.27) helps to visualize the list of classes, individuals, object and data properties that are associated with the classes. In the code below, it is shown that the ontology file is stored in `JenaSemanticModel` class. This is completed via the `loadOntology()` method stored in the class. After defining the semantic model, the second portion of the code shows how are the classes, individuals and properties in the semantic graph is being queried and printed. The details of software input and output can be found in Appendix B; the ontology file of the Urban Network Semantic Model can be found in Appendix B.1, the Jena rules can be found in

Appendix B.2 and printed outcomes of bottom portion of the below code can be referred to Appendix B.3.

Abbreviated Fragment of Software Code:

```
// Define an empty JenaSemanticModel for Urban Network System

   JenaSemanticModel jsm01=new JenaSemanticModel("Urban Network Semantic Model");

// Import the Semantic Model for Urban Network System

   jsm01.loadOntology ("file:ontology/umd-urban-network.owl");

// Print the Urban Network Semantic Model

   System.out.println( jsm01.toString() );

// Print named classes, individuals and properties in the semantic graph...

          jsm01.printNamedClasses();
          jsm01.printIndividuals();
          jsm01.printStatements();
```

Figure 2.27: Load and query the urban network ontology and semantic graph (jena semantic model).

2.5.6 Specify Jena Rules

Once the System Data Model is assembled correctly, the semantic graph, using `JenaSemanticModel` class, is created and the `loadOntology()` method is used to feed the ontology file into the semantic graph (in this case - `jsm01`) as shown in Figure 2.28. Then the corresponding visitor class is created in order to let the Jena Semantic Model (object) to visit the individuals in the `SystemDataModel`, and merge the individuals with the semantic correlations. Once the individuals and the corresponding semantic relationships are populated, the Jena Semantic Model can use the `addRules()` method to

add the corresponding rules to the semantic model, then use the `executeRules()` method to accomplish the semantic graph transformation. Details can be referred to Appendix B.

Abbreviated Fragment of Software Code:

```
// Create Semantic Model for Urban Network System

   JenaSemanticModel jsm01 = new JenaSemanticModel
                               ("Urban Network Semantic Model");

   jsm01.loadOntology ("file:ontology/umd-urban-network.owl");

// Create System Data Model Urban Network Ontology Visitor

   SystemDataModelUrbanNetworkOntologyVisitor visitor01 = new
                                    SystemDataModelUrbanNetworkOntologyVisitor();

   visitor01.add( jsm01 );

// Visit System Data Model Accepts Urban Network Ontology Visitor

   sdm01.accept ( visitor01 );

// Add and execute rules

   jsm01.addRules ( "rules/umd-urban-network.rules" );
   jsm01.executeRules();
```

Figure 2.28: Fragment of code to set up semantic model, visitor and initiate rules execution.

2.5.7 Event-Driven Graph Transformations

From the previous indicated facts and a set of three rules, graph transformations are enabled. In Figure 2.29 Rule 01 below shows how to propagate class hierarchy relationships with jena rules. This sets the subclass of a super class will inherit all the properties that are associated with the super class. Rule 02 finds the relationship between an obstruction and an edge object. The `getObsInEdge()` built-in function is

called in order to evaluate their spatial relationships and if an intersection is captured, an object property relationship of `isBlockedBy` will be assigned from an edge object to an obstruction object. Rule 03 further identifies the weight which shall be updated when an edge intersects with an obstruction. This rule also shows that when the weight of an edge needs to be updated, the previous weight value will be deleted (i.e., with the syntax `remove(3)` below) and the new value will be assigned afterwards. This is important since it will guarantee that no duplicated `hasWeight` data property will be assigned to an individual edge. Rule 04 sets up the relationships among nodes and edges for directed graphs. It means that if an edge `hasSource` points to *node01* and `hasTarget` points to *node02* which has class `Node`, then the data property of this edge `isDirected` is turned to `true`. Also, the object property of `hasEdges` is assigned from the two nodes to this edge.

Abbreviated Fragment of Software Code:

```
@prefix ur: <http://cee.umd.edu/urban#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

// Rule 01: Propagate class hierarchy relationships ...

[ rdfs01: (?x rdfs:subClassOf ?y), notEqual(?x,?y) ->
           (?a rdf:type ?y) <- (?a rdf:type ?x)] ]

// Rule 02: Obstruction and Edge Rule ...

[ Urban01: (?e rdf:type ur:Edge) (?o rdf:type ur:Obstruction)
           (?e ur:hasEdgeGeometry ?eg) (?o ur:hasJSTGeometry ?og)
           getObsInEdge(?og,?eg,?t) equal(?t, "true"^^xs:boolean) ->
           (?e ur:isBlockedBy ?o) print(?o,'isBlockedBy',?e,?t)]

// Rule 03: Computes and Updates Edge's Weight ...

[ Urban02: (?e rdf:type ur:Edge) (?o rdf:type ur:Obstruction)
           (?e ur:isBlockedBy ?o) (?e ur:hasWeight ?ew)
           (?o ur:hasSeverity ?os) computesEdge(?ew,?os,?newWeight) ->
            remove(3), (?e ur:hasWeight ?newWeight)
            print(?e,'hasWeight',?hasWeight)]

// Rule 04: Edges and Nodes for Directed Graph ...

[ Urban03: (?e rdf:type ur:Edge) (?n1 rdf:type ur:Node) (?n2 rdf:type ur:Node)
            notEqual(?n1,?n2), (?e ur:hasSource ?n1) (?e ur:hasTarget ?n2) ->
           (?e ur:isDirected "true"^^xs:boolean)
           (?n1 ur:hasEdges ?e) (?n2 ur:hasEdges ?e) ]
```

Figure 2.29: Jena rules for the tiny urban network.

2.5.8 Minimum Cost Pathway Analysis (with JGraphT)

This section introduces `JGraphT` [87] and how it handles pathway analysis in Figures 2.17 and 2.18. The pathway analysis with `JGraphT` provides a benchmark against which the pathway planning with `OptaPlanner` (introduced in Section 5.4) can be assessed.

Minimum Weighted Cost Pathway Analysis. Figure 2.17 indicates a graph analysis on the tiny town defined previously. The blue arrow lines show the short path for the graph with the defined weights on edges which are labeled accordingly. Figure 2.30 shows the software setup of the problem using `JGraphT`. The first line indicates the inclusion of the `SimpleGraph` class; the second line shows the creation of the class and named the object as `graph01`; then, the following several lines show how odes and edges are being added to the object `graph01` with methods of `addVertex()` and `addEdge()`; the final line shows the use of `shortestPath()` method using two input variables that define the origin and destination node for the defined shortest path problem.

Minimum Weighted Cost All-Pathways Analysis. Figure 2.18 indicates a case where the origin and the destination of a shortest path problem can have multiple numbers. In Figure 2.31, we define that the nodes `A, B, and C` are the origins and the `L, M, and N` are the destinations. In addition, an obstruction at edge `EG` is defined which changes the weight from 5 to 20. The below block of code shows how this problem setting is handled with `JGraphT`. Different than the previous problem setting in terms of the multiple origins and destinations, the `HashSet` Java collections class is adopted to define sets of origins

Abbreviated Fragment of Software Code:

```
// Importing the SimpleGraph class from JGraphT.

    import whistle.util.jgrapht.SimpleGraph;

// Creating a SimpleGraph Object, and adding vertices (or nodes) and edges.

    graph01 = SimpleGraph("test");

    graph01.addVertex("A");
    graph01.addVertex("B");
    graph01.addVertex("C");

    ... other vertices removed ...

    graph01.addEdge("A", "B");
    graph01.addEdge("B", "C");
    graph01.addEdge("B", "E");

    ... other edges removed ...

// Finding shortest path: node A --> node L ...

    graph01.shortestPath( "A", "L" );
```

Figure 2.30: Fragment of code to set up simple graph analysis on the tiny urban network.

and destinations (bottom of the code). The `SimpleDirectedWeightedGraph` is used in order to solve the directed graph with a support method of `allDirectedPaths()` which takes sets of origins and destinations defined with the `Hashset` class. This way, the program is able to handle multiple nodes for the origin-destination problem and return all shortest paths linked with these nodes.

Figures 2.32 and 2.33 indicates the printed graph of the above network and number of vertices and edges. The second part of the outcomes indicate the results of the paths between these source nodes and target nodes. Then, the second portion of the code shows the sorted results of all paths based on the minimum cost.

Abbreviated Fragment of Software Code:

```
// Importi the HashSet class from Whistle collections.

    import whistle.util.collections.HashSet;

// Importing the SimpleDirectedWeightedGraph class from JGraphT.

    import whistle.util.jgrapht.SimpleDirectedWeightedGraph;

// Creating a SimpleDirectedWeightedGraph Object and adding vertices.

    graph02 = SimpleDirectedWeightedGraph("Road Network");
    graph02.addVertex("A");
    graph02.addVertex("B");
    graph02.addVertex("C");

    ... other vertices removed ...

// Adding edges with weights defined.

    graph02.addEdge("A", "B",  "2");
    graph02.addEdge("B", "C",  "2");
    graph02.addEdge("B", "E", "10");
    graph02.addEdge("E", "G", "20");

    ... other edges removed ...

// Using HashSet class to define sets of origins and destinations.

    origins01 = HashSet("Rural Towns");
    origins01.add( "A" );
    origins01.add( "B" );
    origins01.add( "C" );

    destinations01 = HashSet("Safe Areas");
    destinations01.add( "L" );
    destinations01.add( "M" );
    destinations01.add( "N" );

// Finding shortest paths: nodes A, B, C --> nodes L, M, N.

    graph02.allDirectedPaths( origins01, destinations01 );
```

Figure 2.31: Fragment of Whistle code to set up directed graph analysis on tiny urban network.

Abbreviated Fragment of Whistle Code:

```
--- Part 01: Create simple directed weighted graph ...
--- ================================================================================

--- Graph(Road Network): ([A, B, C, D, E, F, G, H, I, J, K, L, M, N],
                         [(A,B), (B,C), (B,E), (C,D), (D,E), (E,F),
                          (F,A), (E,G), (F,K), (G,J), (J,I), (I,H),
                          (H,G), (G,I), (K,J), (I,L), (J,M), (K,N)])
--- No vertices = 14 ...
--- No edges    = 18 ...

--- Part 02: All directed paths computation    ...
--- ================================================================================
--- HashSet (cloned): [A, B, C] ...
--- HashSet (cloned): [L, M, N] ...
--- No paths = 30 ...
--- Path  1: [(A : B), (B : E), (E : G), (G : I), (I : L)] ...
--- Path  2: [(A : B), (B : E), (E : G), (G : J), (J : M)] ...
--- Path  3: [(A : B), (B : E), (E : G), (G : J), (J : I), (I : L)] ...
--- Path  4: [(A : B), (B : E), (E : F), (F : K), (K : N)] ...
--- Path  5: [(A : B), (B : E), (E : F), (F : K), (K : J), (J : M)] ...
--- Path  6: [(A : B), (B : E), (E : F), (F : K), (K : J), (J : I), (I : L)] ...
--- Path  7: [(A : B), (B : C), (C : D), (D : E), (E : G), (G : I), (I : L)] ...
--- Path  8: [(A : B), (B : C), (C : D), (D : E), (E : G), (G : J), (J : M)] ...
--- Path  9: [(A : B), (B : C), (C : D), (D : E), (E : G), (G : J), (J : I),
              (I : L)]
--- Path 10: [(A : B), (B : C), (C : D), (D : E), (E : F), (F : K), (K : N)] ...
--- Path 11: [(A : B), (B : C), (C : D), (D : E), (E : F), (F : K), (K : J),
              (J : M)]
--- Path 12: [(A : B), (B : C), (C : D), (D : E), (E : F), (F : K),(K : J),
              (J : I), (I : L)]
--- Path 13: [(B : C), (C : D), (D : E), (E : G), (G : I), (I : L)] ...
--- Path 14: [(B : C), (C : D), (D : E), (E : G), (G : J), (J : M)] ...
--- Path 15: [(B : C), (C : D), (D : E), (E : G), (G : J), (J : I), (I : L)] ...
--- Path 16: [(B : C), (C : D), (D : E), (E : F), (F : K), (K : N)] ...
--- Path 17: [(B : C), (C : D), (D : E), (E : F), (F : K), (K : J), (J : M)] ...
--- Path 18: [(B : C), (C : D), (D : E), (E : F), (F : K), (K : J), (J : I),
              (I : L)]
--- Path 19: [(B : E), (E : G), (G : I), (I : L)] ...
--- Path 20: [(B : E), (E : G), (G : J), (J : M)] ...
--- Path 21: [(B : E), (E : G), (G : J), (J : I), (I : L)] ...
--- Path 22: [(B : E), (E : F), (F : K), (K : N)] ...
--- Path 23: [(B : E), (E : F), (F : K), (K : J), (J : M)] ...
--- Path 24: [(B : E), (E : F), (F : K), (K : J), (J : I), (I : L)] ...
--- Path 25: [(C : D), (D : E), (E : G), (G : I), (I : L)] ...
--- Path 26: [(C : D), (D : E), (E : G), (G : J), (J : M)] ...
--- Path 27: [(C : D), (D : E), (E : G), (G : J), (J : I), (I : L)] ...
--- Path 28: [(C : D), (D : E), (E : F), (F : K), (K : N)] ...
--- Path 29: [(C : D), (D : E), (E : F), (F : K), (K : J), (J : M)] ...
--- Path 30: [(C : D), (D : E), (E : F), (F : K), (K : J), (J : I), (I : L)] ...
```

Figure 2.32: JGraphT output: computes and ranks all paths connecting sets of nodes A, B, C and L, M, N (part a).

Abbreviated Fragment of Whistle Code:

```
--- Sort paths by cost of traversal ...
--- ========================================================================
--- Path  1: 16.00: [(C : D), (D : E), (E : G), (G : J), (J : M)] ...
--- Path  2: 17.00: [(C : D), (D : E), (E : G), (G : J), (J : I), (I : L)] ...
--- Path  3: 18.00: [(B : C), (C : D), (D : E), (E : G), (G : J), (J : M)] ...
--- Path  4: 19.00: [(B : C), (C : D), (D : E), (E : G), (G : J), (J : I),
              (I : L)] ...
--- Path  5: 20.00: [(A : B), (B : C), (C : D), (D : E), (E : G), (G : J),
              (J : M)] ...
--- Path  6: 21.00: [(A : B), (B : C), (C : D), (D : E), (E : G), (G : J),
              (J : I), (I : L)] ...
--- Path  7: 22.00: [(B : E), (E : G), (G : J), (J : M)] ...
--- Path  8: 22.00: [(C : D), (D : E), (E : G), (G : I), (I : L)] ...
--- Path  9: 23.00: [(B : E), (E : G), (G : J), (J : I), (I : L)] ...
--- Path 10: 24.00: [(A : B), (B : E), (E : G), (G : J), (J : M)] ...
--- Path 11: 24.00: [(B : C), (C : D), (D : E), (E : G), (G : I), (I : L)] ...
--- Path 12: 25.00: [(A : B), (B : E), (E : G), (G : J), (J : I), (I : L)] ...
--- Path 13: 26.00: [(A : B), (B : C), (C : D), (D : E), (E : G), (G : I),
              (I : L)] ...
--- Path 14: 26.00: [(C : D), (D : E), (E : F), (F : K), (K : N)] ...
--- Path 15: 27.00: [(C : D), (D : E), (E : F), (F : K), (K : J), (J : M)] ...
--- Path 16: 28.00: [(B : C), (C : D), (D : E), (E : F), (F : K), (K : N)] ...
--- Path 17: 28.00: [(B : E), (E : G), (G : I), (I : L)] ...
--- Path 18: 28.00: [(C : D), (D : E), (E : F), (F : K), (K : J), (J : I),
              (I : L)] ...
--- Path 19: 29.00: [(B : C), (C : D), (D : E), (E : F), (F : K), (K : J),
              (J : M)] ...
--- Path 20: 30.00: [(A : B), (B : E), (E : G), (G : I), (I : L)] ...
--- Path 21: 30.00: [(A : B), (B : C), (C : D), (D : E), (E : F), (F : K),
              (K : N)] ...
--- Path 22: 30.00: [(B : C), (C : D), (D : E), (E : F), (F : K), (K : J),
               (J : I), (I : L)] ...
--- Path 23: 31.00: [(A : B), (B : C), (C : D), (D : E), (E : F), (F : K),
              (K : J), (J : M)] ...
--- Path 24: 32.00: [(A : B), (B : C), (C : D), (D : E), (E : F), (F : K),
              (K : J), (J : I), (I : L)] ...
--- Path 25: 32.00: [(B : E), (E : F), (F : K), (K : N)] ...
--- Path 26: 33.00: [(B : E), (E : F), (F : K), (K : J), (J : M)] ...
--- Path 27: 34.00: [(A : B), (B : E), (E : F), (F : K), (K : N)] ...
--- Path 28: 34.00: [(B : E), (E : F), (F : K), (K : J), (J : I), (I : L)] ...
--- Path 29: 35.00: [(A : B), (B : E), (E : F), (F : K), (K : J),
              (J : M)] ...
--- Path 30: 36.00: [(A : B), (B : E), (E : F), (F : K), (K : J),
              (J : I), (I : L)] ...
--- ========================================================================
```

Figure 2.33: JGraphT output: computes and ranks all paths connecting sets of nodes A, B, C and L, M, N (part b).

Chapter 3: Proposed Methodology

This chapter describes an architectural template for multi-domain semantic modeling and reasoning. We then propose a methodology to extend this capabiility by adding support for real-time (near-time) planning.

3.1 Recent Advances in Multi-Domain Semantic Modeling

3.1.1 Multi-Domain Architectural Framework

Multi-domain semantic models are assembled from mixtures of domain-neutral and domain-specific ontologies and rules. The meta-domain ontologies and rules were used to determine spatial relations between sensors and rooms in buildings [40]. Meta-domain includes temporal, spatial, physical unit as well as graph domain for universal urban domain application reasoning. The combination of three meta domains: time, space and physical unit were also applied to reasoning on relationships between local schools, houses and regions [30]. Most, if not all, urban infrastructural environment has notions of time and space. In order to answer whether a person (a car, an intersection etc.) is safe in an urban environment, we need to answer if it appears at the right place (spatial) and at the right time (temporal). Therefore, modeling safety of urban systems must include

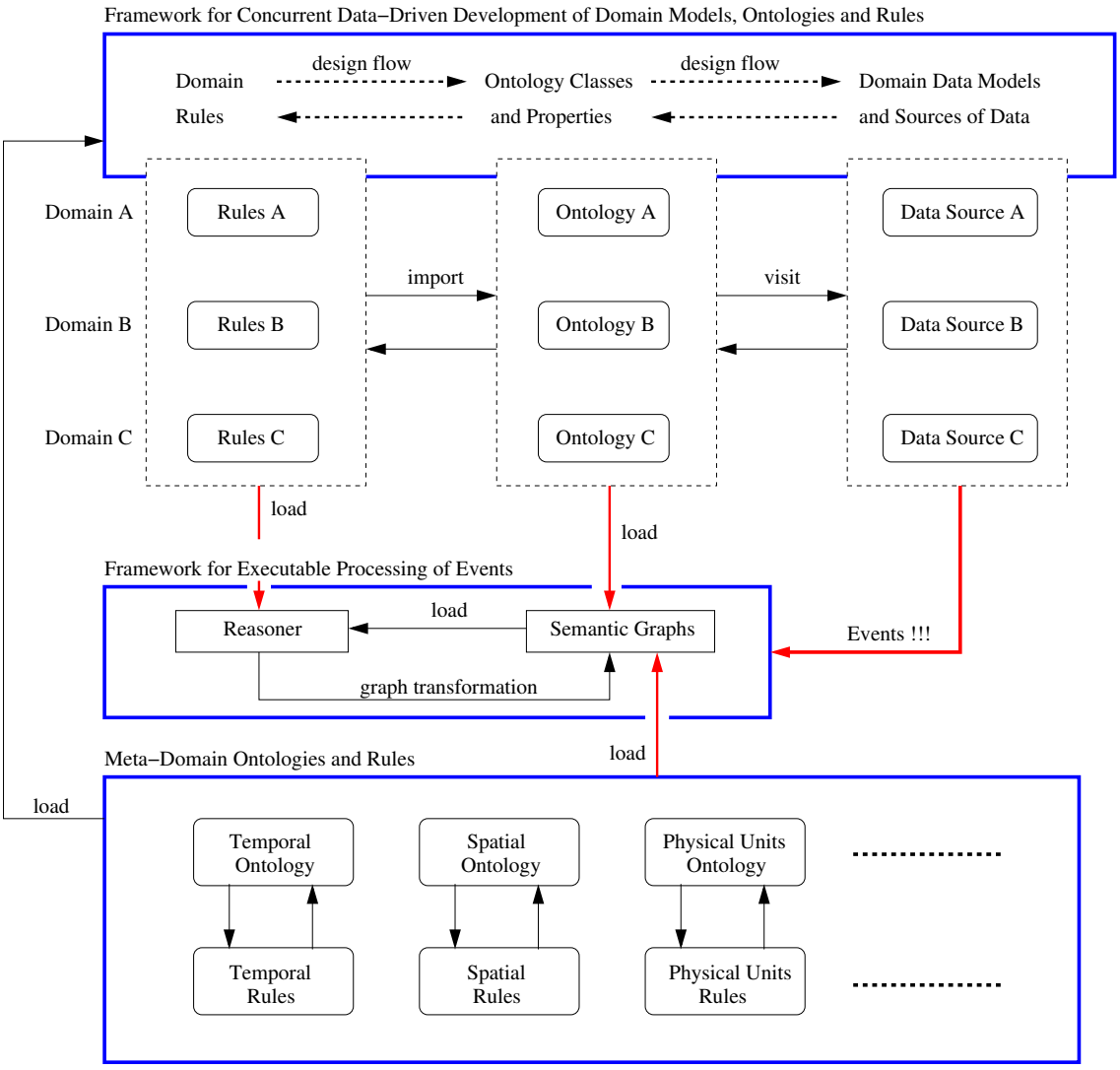


Figure 3.1: Architectural template for multi-domain semantic modeling and reasoning.

spatio-temporal domain ontologies as well as the corresponding jena reasoning rules that relates to space and time.

3.1.2 Concurrent Development of Data Models, Ontologies, Rules

Figure 3.1 shows the architectural template for the multi-domain semantic modeling framework. On the top section of the Figure, lists of rules, ontologies and data sources are inserted to demonstrate that various domains of related information shall be inserted side-by-side as part of the reasoning framework. The design flows indicated the relationships among domain rules, ontologies and data models. The reasoner of the framework, shown within blue box in the middle section, is constructed with the set of domain rules; the initial semantic graphs are constructed by the domain ontologies (classes and properties); events that can trigger semantic graphs transformations are inserted with data imported from physical models. The combination of these three can help to promote semantic graphs transformation based on the jena rules embedded within the reasoner and the events derived from physical models. In addition, the meta-domain ontologies and rules can also be inserted to the semantic graphs and reasoner respectively. This is important since urban applications almost always require temporal, spatial and physical domains of knowledge, and set of rules that monitor meta-domain aspects of information.

3.1.3 Data-Driven Synthesis of Behavior

Figure 3.2 shows a simplified version of data-drive synthesis of system behavior and structure. With what mentioned in previous section, data models, ontologies and

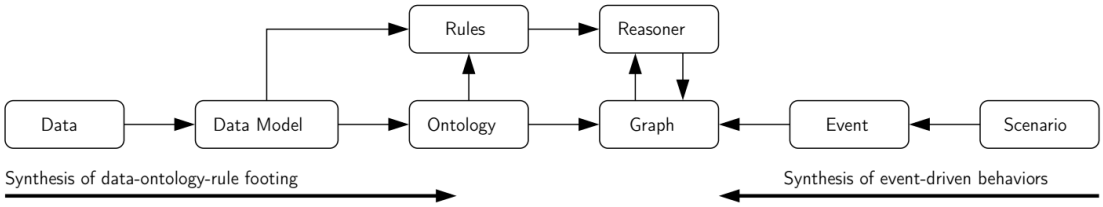


Figure 3.2: Data-driven synthesis of system behavior and structure (simplified).

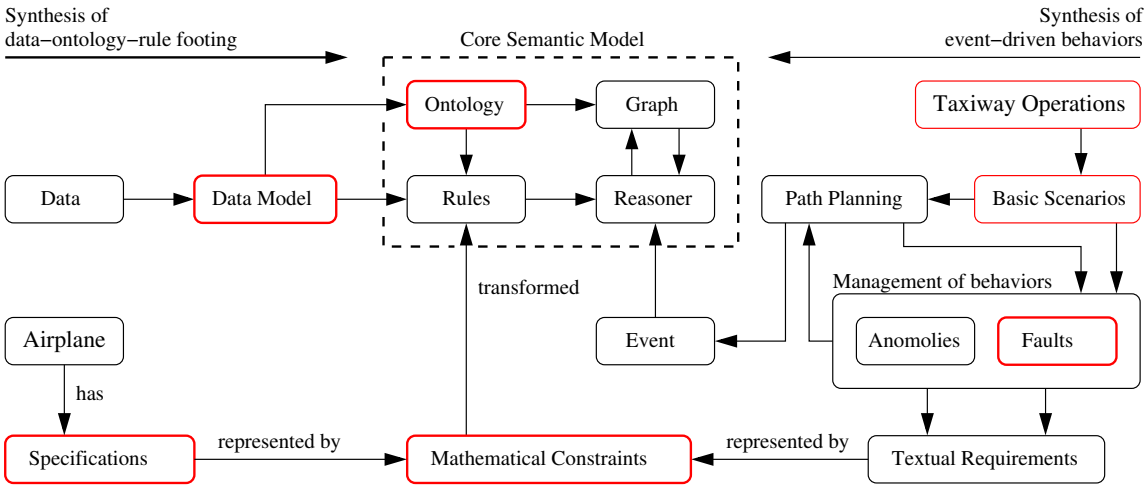


Figure 3.3: Data-driven synthesis of system behavior and structure (detailed). Adapted from [31–33].

rules are the basics of the multi-domain semantic modeling framework. Because of the possibility of insertion of multiple domains of information, reasoning through this framework can bring deterministic results which have considered multiple areas of concerns. On the left-hand-side, it shows the combination of data which constructs data model, ontology and rules, as well as their relationships and design flows. Rules will be used as the reasoner and ontologies will be the semantic graph shown in the middle of the Figure. The transformation of semantic graphs are based on events and rules in which the former uses scenarios to represent events and the latter are the basis of reasoner. Similarly, Figure 3.3 a more detailed version of data-driven synthesis of system behavior and structure. The essence of two Figures is the same, however, detailed descriptions of rules and events are further described. Additional details that indirectly connecting rules and events can also be found in this Figure, where mathematical constraints can be transformed and used as jena rules. The mathematical constraints can also be represented by textual requirements which are derived from behavior management relating to scenarios and events.

3.2 Proposed Methodology

This section briefly introduces the overall methodologies that are used for the proposed conceptual framework.

The benefit of using multi-domain semantic modeling approach makes it capable to describe a problem domain with the knowledge support from multiple participating domains, and new knowledge insertions can be derived, respond to events and transform semantic graphs through the reasoning driven by rule-based mechanisms. Figure 3.3 shows the synthesis of system behavior and structure, both on data-ontology-rule footing and event-driven behaviors. It shows that the co-development of ontologies, rules and data and insertations of individuals to ontologies through visiting data models. The combination of data, data model and onntology enhance the power of rules embedded with back-end functions that act as reasoners and let the semantic graph dynamically respond to the incoming events derived from various scenarios.

Semantic modeling studies have been widely conducted in various fields of studies over the past decades. The idea of using the ontological approach for improving efficiency of planning and processes of a real-world engineering model is not new. Semantic modeling consists of generating graphs of individuals (instances), and reasoning based rules in the form of *if* <*conditions*> then <*consequences*>. These together can jointly form a knowledge basis of a specific domain. In real world urban systems, development of ontologies can represent specific domains of knowledge (i.e., wildfire evacuation, airport operation, traffic intersection).

Figure 3.1 shows the framework of using multiple domains of knowledge including rules, ontologies and data sources from each participating domain and having them exchange information to construct the initial semantic graphs. Events are detected based on the external data sources (right-hand column) which will affect the semantic graphs (center of the Figure) based on the relationships assigned by the semantic ontologies as well as domain rules, which act as reasoners to initiate graph transformations (shown within the blue box in the middle section of the Figure). At the same time, the meta-domain ontologies and rules, which are completely separate from the domain-specific knowledge (top), are loaded to the semantic graphs. These meta-domain knowledge can help to capture changes across various domains, for instance, to identify changes occur in time, space, physical units, and so on. As an integrated framework, various sections mentioned here can work hand-in-hand to provide as an integrated knowledge support framework which inserts semantic meanings to data and infer additional knowledge based on the external events, and make further decisions based on the new and old knowledge.

3.2.1 Integration of Multi-Domain Semantic Modeling and Planning

In this section, detailed view of integrated framework of multi-domain semantic modeling and real-time planning are shown in Figure 3.4.

It indicates that the essence of this framework is constructed with three parts: (a) domain data models, ontologies and rules on top section, (b) interactions between semantic reasoner and semantic graph in the middle section, and (c) the real-time planning framework shown at the bottom. The semantic graphs can be transformed based on the

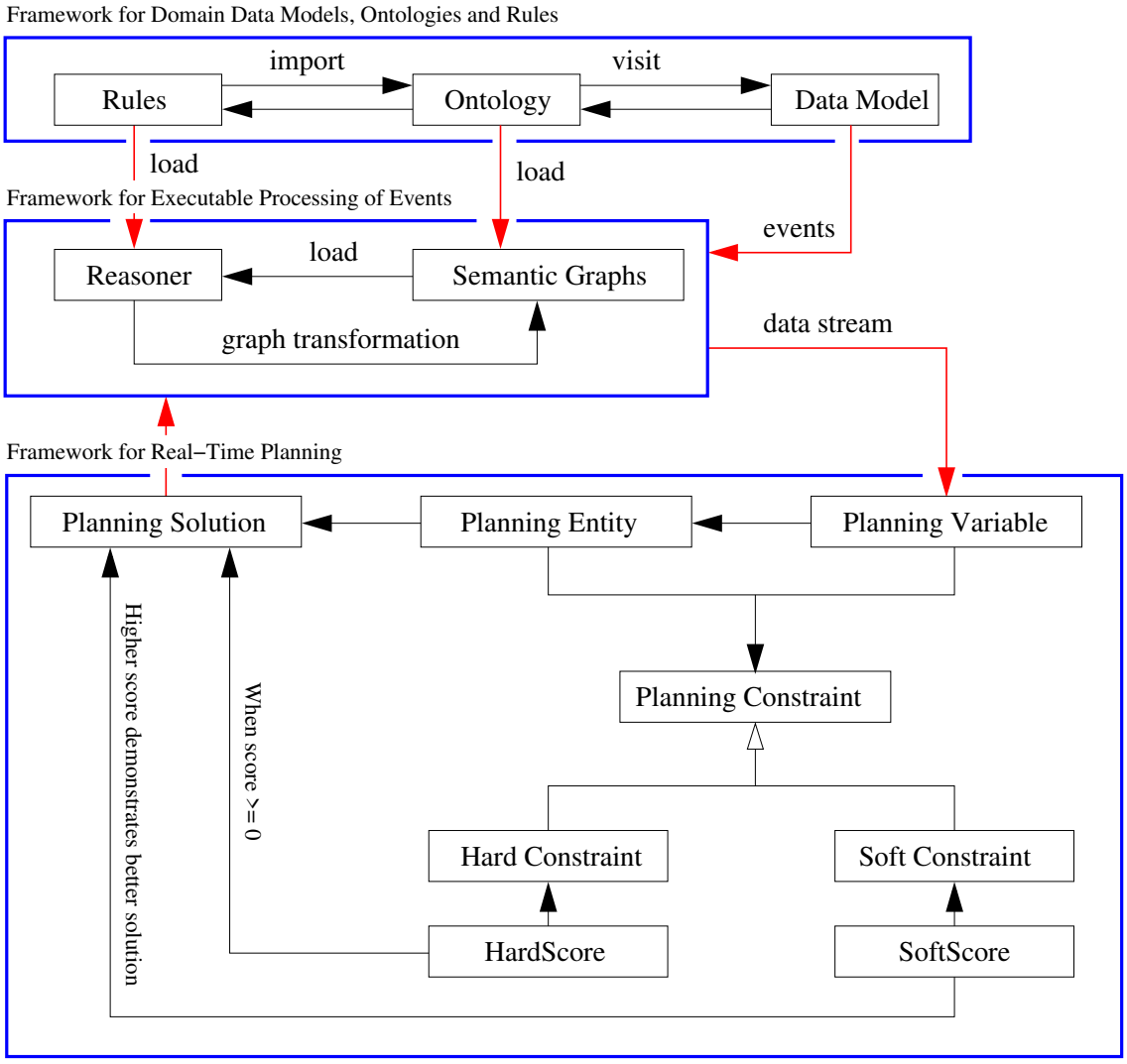


Figure 3.4: Architectural template for integration of multi-domain semantic modeling and reasoning with computational support for real-time planning.

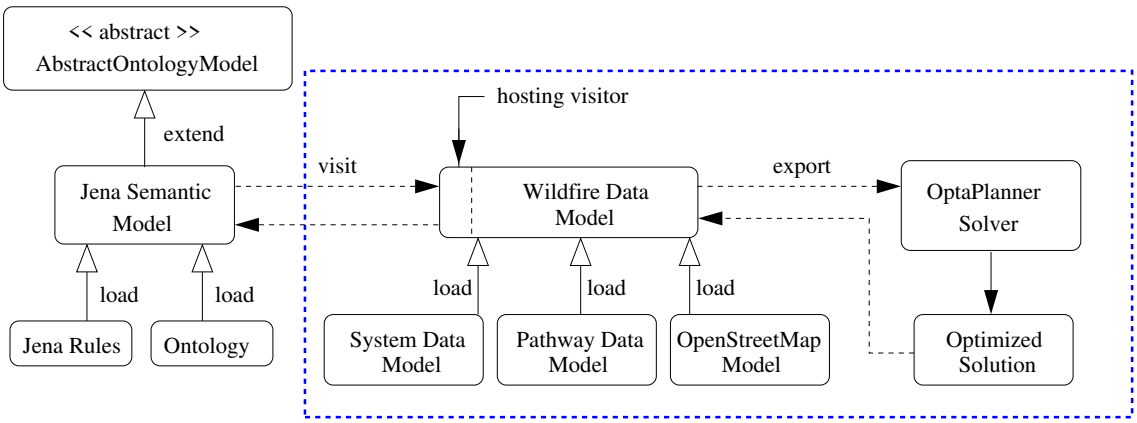


Figure 3.5: Preliminary framework for integration of multi-domain semantic modeling with OptaPlanner.

execution of external events (from data models) and predetermined Jena rules. Then, the semantically identified data is sent to the OptaPlanner and stored under the planning variables where all low-level data is implemented. The planning entities and variables can be used to implement the planning constraints where the latter is constructed with two-level of constraints: hard and soft constraints. From each level of constraints, corresponding score can be computed and used to distinguish between feasible versus unfeasible solutions as well as a better solution among all feasible solutions. Eventually, the adopted planning solution is returned to the semantic graphs. With the new events from the planning solutions, new streams of data may be sent to real-time planning again for dynamic and real-time planning.

3.2.2 Customized Data Models and Planning

The importance of this work is to integrate the planning aspect with the knowledge-based framework previously discussed.

Figure 3.5's left-hand-side shows the summary of Jena Semantic Model and the right-hand-side (framed within the dashed box) indicates the main contribution of the integrated knowledge representation and planning support. The Wildfire Data Model (as an example) placed in the center illustrates how framework adopts the data model and how it act as an significant integral part, which is required to connect with the Jena Semantic Model (left) and the OptaPlanner Constraint Solver (right). The domain-specific domain also contains a hosting visitor (shown in the middle) in which it provides a temporary memory space to store semantic knowledge, insert with data model, then leave the data

model once reasoning and inference are complete. The various data sources (System Data Model, Pathway Data Model, and Open Street Map Data Model) are loaded into the wildfire data model. The major reason for this is because the original data formats of these sources are incompatible for further computation. Proposing a domain-oriented data model provides a space to load various data that has various formats, assimilate them into a workable format, and then use them with semantic graphs to infer new knowledge that contributes to decision making. OptaPlanner, on the other hand, grabs the data that are useful for planning from the Wildfire Data Model, computes the optimized solution based on the imported data, then return the feasible solution sets back to the wildfire (domain) data model. Lastly, the optimized solution from the constraint solver can work with the hosting visitor and provide additional derived knowledge to cope with systems safeties and efficiency.

Chapter 4: Semantic Modeling and Reasoning for Time and Space

This chapter examines methods for the semantic modeling and reasoning for time and space; see Sections 4.1 and 4.2, respectively. Section 4.3 provides some working examples of how temporal and spatial reasoning and semantic modeling can be embedded into the real-world urban problems.

4.1 Time Ontologies, Rules and Reasoning

In order for decisions involving time to be reliable, underlying models of time and theories of reasoning need to be formal. Time can be illustrated either by series of individual time points (instants) or an interval of time. The left-hand side of Figure 4.1 shows a time ontology for time instants and time intervals. The right-hand side of Figure 4.1 shows a possible relations of a pair of time intervals.

Reasoning with Time. Procedures for reasoning with time to capture time instants, time intervals as well as their relations. To this end, Allen's temporal interval calculus (ATIC) logic is capable of manipulating time intervals, expressing temporal properties, and tracking evolution of those time intervals [7]. There are thirteen possible relations between a pair of time intervals. Allen's temporal interval algebra is identified as the most

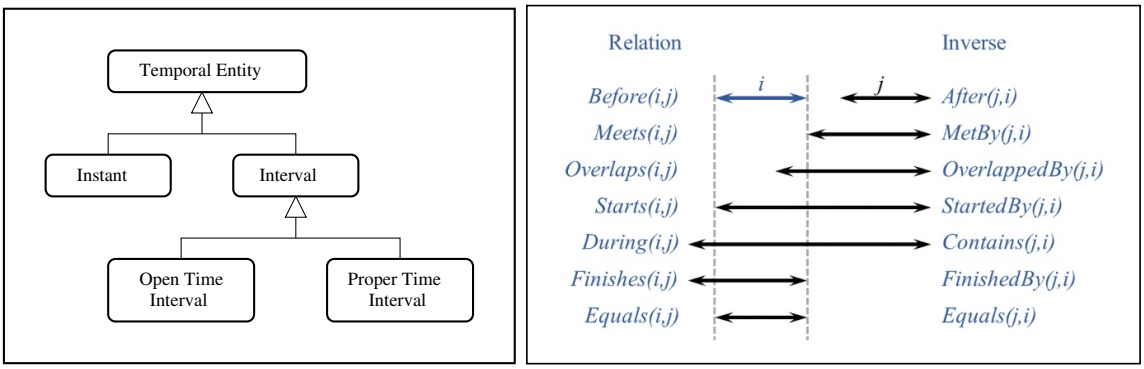


Figure 4.1: Left: Ontology for modeling time instances and time intervals, Right: Allen's algebra for relationships among time intervals.

qualified temporal interval [120].

Reasoning with Time Intervals. Given two time intervals I_1 and I_2, a time-point t and a proposition ϕ, there are plenty of questions can be answered with these information:

- Does time-point t occur within time interval I_1?
- Are two intervals (I_1 and I_2) equal to each other?
- What interval represents the temporal intersection of I_1 and I_2?
- Does I_1 contains all of I_2?
- Are intervals I_1 and I_2 continuous of each other?
- Does I_1 happen before or after I_2?
- Do I_1 and I_2 have the same start or end time instant?

Logical questions may include:

- Does the proposition ϕ hold within I_1?

- And if ϕ holds during I_1, does it hold during I_2?
- Or will the proposition I_1 hold before or after I_1?

Time Ontology and Rules. Figure 4.2 shows a typical time ontology including its classes, data properties as well as object properties. Here, the temporal ontology is defined by four classes and those are TemporalEntity, OpenTimeInterval, Instant and ProperTimeInterval. The OpenTimeInterval and ProperTimeInterval are sub-classes of TemporalEntity. The TemporalEntity class has three data properties: hasDuration (type: duration), beginsAt (type: dateTime) and endsAt (type: dateTime). OpenTimeInterval has another two data properties: hasTimeAnchor (type: dateTime) and hasDuration (type: string). The time Instant class includes hasTime (type: dateTime) and happensAfter and happensBefore as its object properties with other time Instant classes. The ProperTimeInterval class supports the computations between different time intervals and also evaluates their relationships (e.g., intBefore, intAfter, intContains, intMeets, intFinishedBy).

Reasoning with temporal ontology mainly involves time instants and time intervals as part of its reasoning process. The common temporal rules include but not limited to and these temporal rules can be written in Jena (rules), shown in Figure 4.3:

- If a time instant (?x) occurs before another time instant (?y);
- If a time instant (?x) lies inside a time interval (?I1).

Coelho et al. [30] used the temporal domain knowledge for reasoning on ages of local residents to classify levels of education that they can receive accordingly. Temporal-domain can be easily adapted with ontologies and rules since notion of time is supported

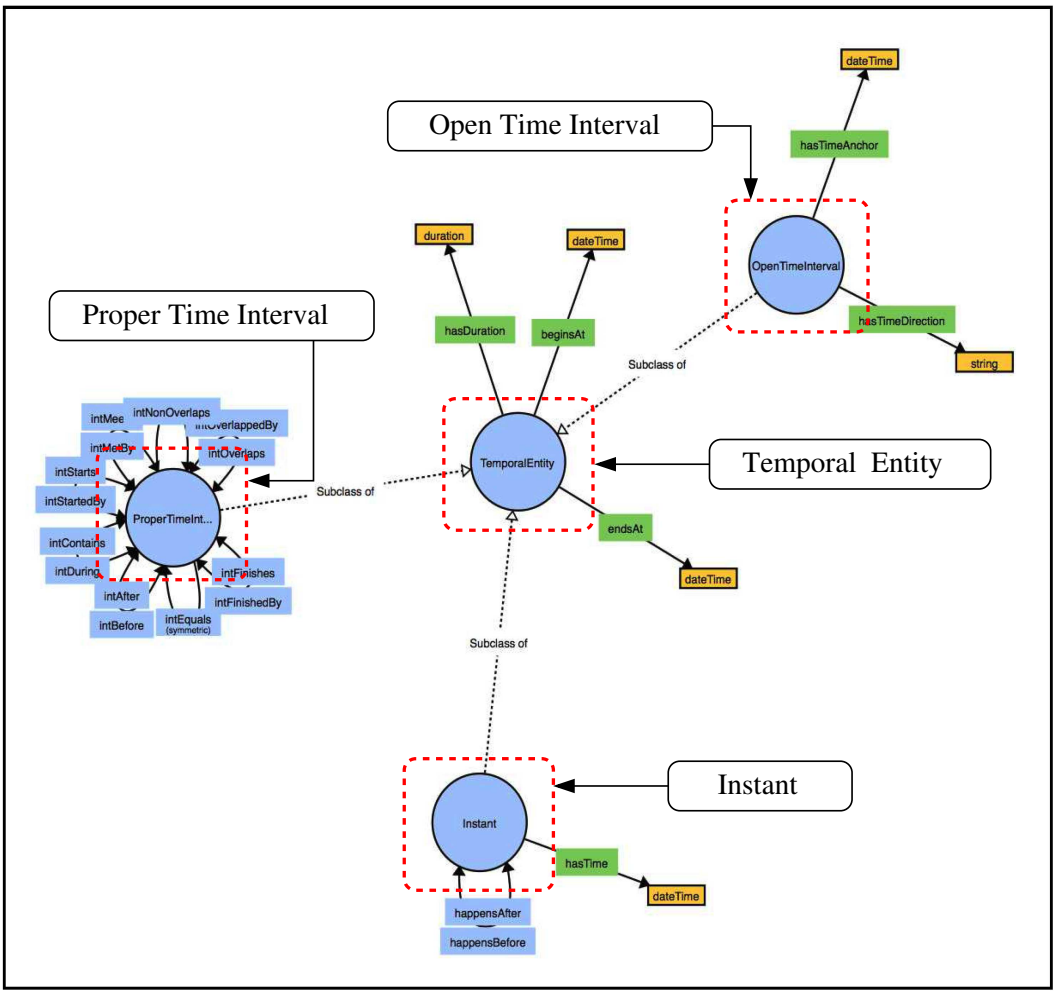


Figure 4.2: Time ontology and its data and object properties.

Jena Rules

```
// Time Rule 1: Deduction of happensBefore time instants....

[ TimeRule01: (?x rdf:type te:Instant) (?y rdf:type te:Instant) (?x te:hasTime ?t1)
              (?y te:hasTime ?t2) lessThan(?t1,?t2) -> (?x te:happensBefore ?y) ]

// Time Rule 2: Deduce if a time instant is inside a time interval

[ TimeRule02: (?x rdf:type te:TemporalEntity) (?y rdf:type te:Instant)
              (?x te:beginsAt ?t1) (?x te:endsAt ?t2) lessThan (?t1, ?t2)
              (?y te:hasTime ?t3) lessThan(?t1,?t3) greaterThan(?t2,?t3) ->
              (?y te:isInInterval ?x) ]
```

Figure 4.3: Jena rules reasoning with time ontology.

in Jena. Temporal ontology can be on a discrete time-space point or interval model to store user paths in their real-time audio museum application [75]. Time-point axiom [71] can support the Process Specification Language (PSL) where time is defined and used as ordered time points which may or may not contain branches of other time points.

4.2 Space Ontologies, Rules and Reasoning

Formal theories for reasoning with space - points, lines, and regions (or polygons) - are covered by region connected calculus [127]. A robust implementation of two-dimensional spatial entities and the associated reasoning procedures are provided by the Java Topology Suite [86]. Spatial relation can be expressed simply by using the combination of point, line and polygons. Spatial reasoning for detecting the geometry relations of sensors and rooms were previously conducted [40], which indicated an abbreviated representation of an experimental spatial (geometry) ontology and the associated data and object properties. Spatial relation is based on the physical region and boundaries of a specific physical component. Spatial ontology was used to identify relations between schools, houses and districts in local neighborhood [30].

Theories of Space. Besides modeling with time, there is a strong need for formal definition of space to support semantic modeling in this domain and systems which spatial domain plays a critical role. This is especially important when it comes to urban safeties. In this thesis, Vieu's views [154] of spatial theories of a mereotopological categorization is adopted. Unlike time and temporal modeling, space is not oriented and nor cyclic. This leads to the below spatial theory classification (see Figure 4.4).

- **Space: Point.** Space is perceived as arrangement of points with focus on orientation and distance concepts. Lines, polygons, regions, and other spatial entities are defined as series of points. This is defined in mathematical spatial theories [77, 136, 153].

- **Space: Interval.** Tuples of intervals resulting from the projection of regular regions on the axes of a reference frame are the primitive spatial entities in this class of spatial theories. These spatial theories [72, 106] are established based on Allen's temporal calculus [7]. Furthermore, they illustrate beyond mereo-topological information which accounts for things related to orientation.

- **Space: Array.** In the theory of arrays, space is treated as a collection of arrays (i.e., a discrete coordinate system). It is extremely beneficial in terms of concurrent capture topological, orientation, and distance information simultaneously. The theory has been widely used to promote machine visualization and spatial database applications, and even links between visual and linguistic spaces [62, 73, 93].

- **Space: Region.** A region of any geometrical shape with dimension more than one is considered the primitive in these theories. However, regions should be of the same dimension because the entire space as well as their interior should not be empty. Theories based on one mereological part and one topological relation falls under this category [23].

- **Space: Multi-dimension.** In these theories, the dimensionality of spatial primitives and one of the whole spaces is not restricted. They do not assume nor define a hierarchy between their primitives, but introduce incidence relationship in lieu of

ontological dependency. Some of these theories concentrate on rendering multiple dimensions [58, 66] while others support notion of boundary [143].

Spatial Modeling Review. Spatial modeling has been implemented in various research areas, such as in construction, where notion of space contributes to various safety related issues. A real-time construction site spatial modeling was conducted through processing range point data that were captured by 3D imaging sensors [65]. A construction safety management study used spatial modeling for construction workers' crowdedness monitoring [165]. A research optimizing geometric data in the semantic representation was conducted to enhance the Industry Foundation Classes (IFC) ontology (ifcOWL), which helps to understand status of IFC [119]. Using IFC is to maximize the ability of basic concepts representation [118]. An analysis on the preliminary error impact was presented for spatial safety of earth moving and surface mining activities, and additionally, a tested model has been included in this research to simulate safety violations, apply object identification and track errors on the collected and processed data [28]. In order to facilitate infrastructure modeling, Zhu and co-workers [169] have studied a variety of optical spatial data collection techniques and selected an appropriate technique from four data collection techniques (photogrammetry, videogrammetry, laser scanning, and 3D camera ranging) based on the infrastructure application requirements. [18] integrated geographic information systems (GIS) and CAD capabilities to conduct a geospatial analysis on a virtual campus using the information of existing and planned facilities (i.e., CAD 2D drawings, 3D solid models, virtual models). [81] conducted a spatial analysis in Building Information Modeling (BIM) to optimize formwork design process using the automatically extracted

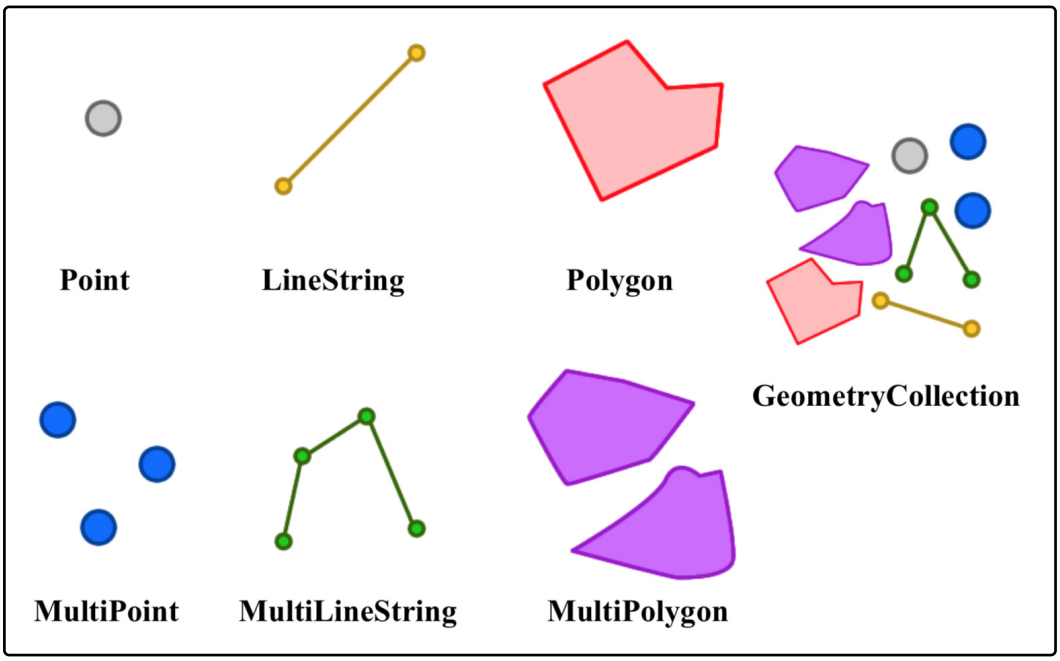


Figure 4.4: Types of geometry supported in Java Topology Suite.

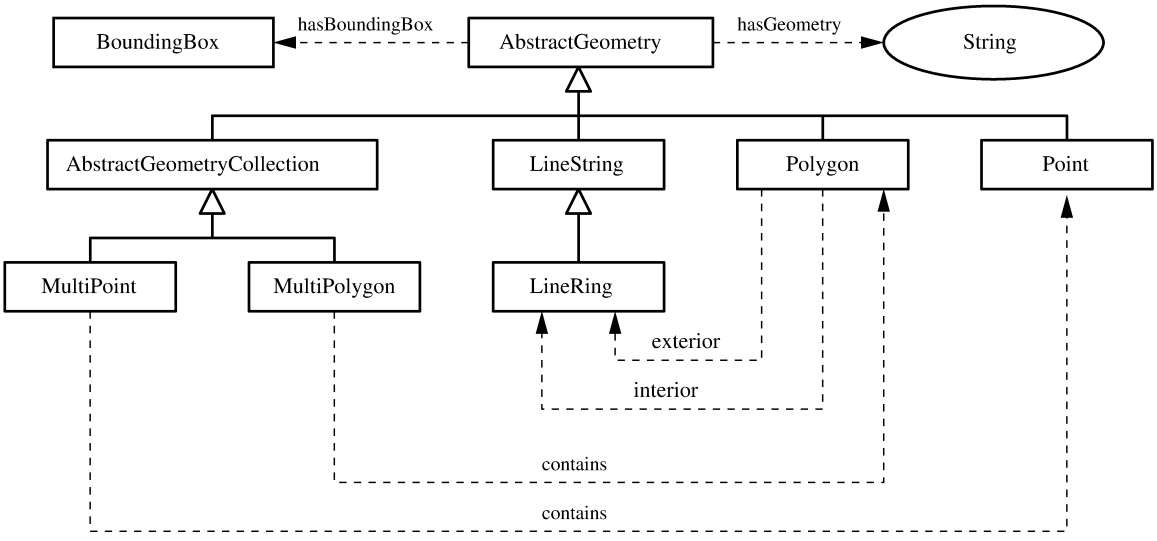


Figure 4.5: Abbreviated representation of spatial (geometry) ontology and associated data and object properties.

data from a BIM model. An implementation of 3D building models was presented by using metric operators in spatial query language [24]. Two ways of implementing these operators are proposed: octree and axis aligned bounding boxes (AABB) approaches. A three-dimensional singularity functions was used to derive stationary and directional activities so as to minimize total project duration [101], which has successfully developed a scheduling algorithm tool for project managers to formalize spatial-temporal constraints of the activities as well as to handle spatial conflicts. An automated integration hazard warning system, GEOWARNS, which reaches user communities directly was generated by using geo-spatial data [61]. Virtual Reality (VR) technology has been used to generate and implement virtual space for simulation of smart home service configuration [97]. Spatial modeling was also conducted for indoors routing problems and requirements of people with disabilities (PWD) during emergency evacuation [74]. A study on e-hailing services of uber demands in New York City (NYC) also utilized spatiotemporal models. Three models were used: one temporal model, vector autoregressive (VAR), and two spatiotemporal models, spatial-temporal autoregressive (STAR) and least absolute shrinkage and selection operator applied on STAR (LASSO-STAR) [46].

Spatial Relationships Among Objects. An abbreviated spatial ontology and the associated data and object properties are shown in Figure 4.5. Realizing the data and object properties of a physical entity helps to understand possible relations between a pair of geometrical objects. The geometric relations between two geometry objects are shown as in Figure 4.6. Each geometry object can represent various shapes, such as a point, line, triangle, rectangle, polygons or any two-dimensional geometry shapes.

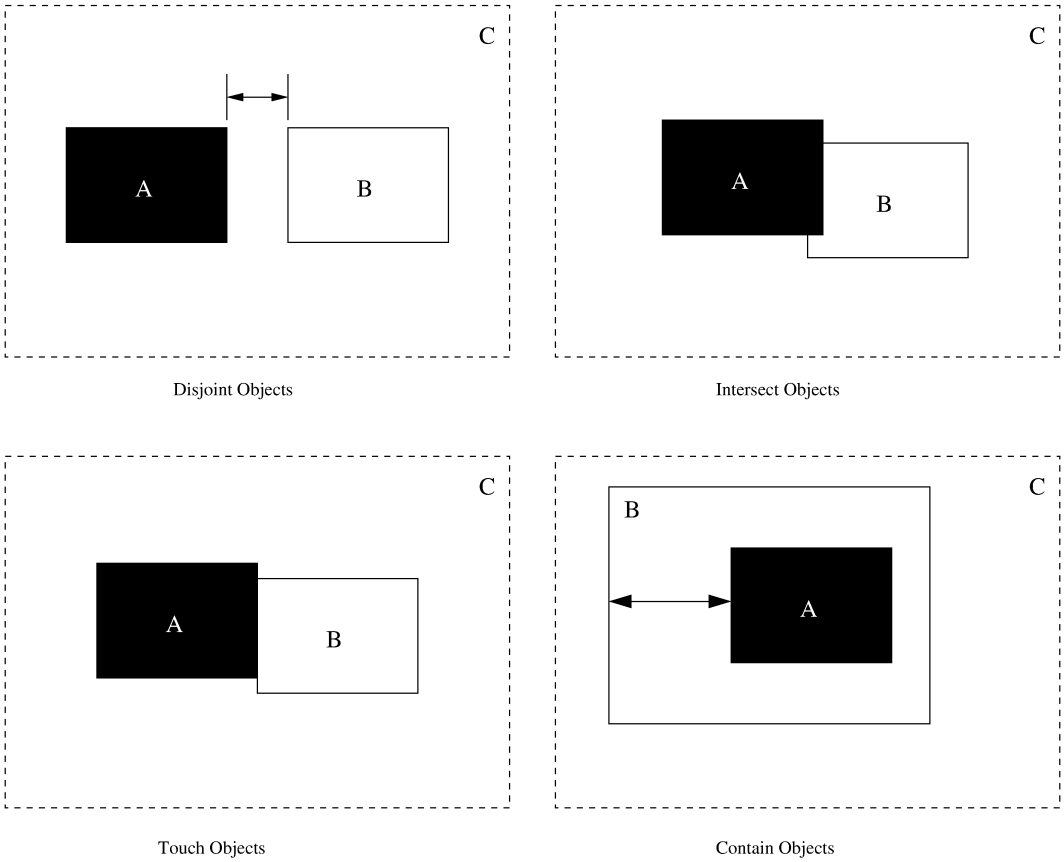


Figure 4.6: Summary of spatial relations between two geometric objects.

4.3 Working Examples

This section uses simple airport taxiway scheduling and obstruction operation examples to explain the insertion of time, space, temporal and spatial ontologies and rules, and time- and space-based reasoning in real-world settings.

4.3.1 Semantic Models of Time and Time-based Reasoning

Step 1: Define Tiny Airport Network Ontology and Semantic Graph with Time

Figure 4.7 shows a simplified airport network ontology of its classes, object properties and data properties. There are 11 classes defined here - *AirportNetwork, AbstractComponent and AbstractGeometry*, and *Runway, Building, Terminal, Obstruction, Airplane, HoldingPosition, FlightSegment and TemporalEntity*. *Runway, Building, Obstruction, Airplane, HoldingPosition* are subclasses of *AbstractComponent*, *FlightSegment* is a subclass of *Airplane* and *Terminal* is a subclass of *Building*. *AbstractComponent* has two data properties - *hasStatus* and *hasName* and two object properties *hasComponent* and *hasGeometry* which connects it with *AirportNetwork* and *AbstractGeometry*. The temporal aspect of this simplified ontology will be illustrated with three object properties defining between *Airplane* and *TemporalEntity* - that are *hasArrivalTime, hasDepartureTime*, and *hasTravelDuration*. The *Airplane* also has another object property that points to itself (to another airplane) which is *takesOffBefore*. This sets the object property relationship between a pair of planes to distinguish which plane takes off earlier or later. The *TemporalEntity* has two data properties to distinguish if it is a time instant or interval.

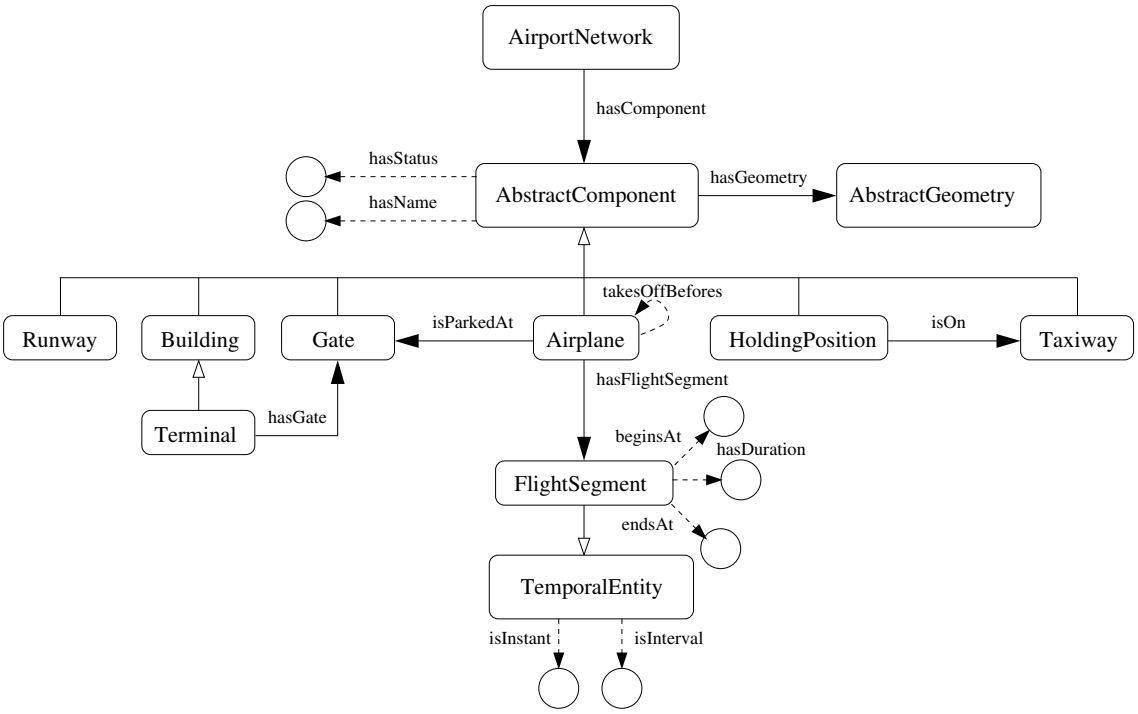


Figure 4.7: Simplified airport network ontology with its classes, object and data properties (reasoning with time).

Figures 4.8 and 4.9 indicate that the assembly of simplified airport network ontology classes and properties by defining temporal relationships and properties. Object datatypes among these 11 classes are settled with the second portion of the code where subclasses are added through *addSubClass()*. Furthermore, the data properties are also assigned where *createDatatypeProperty()* is used to create a data property, and *setDomain()* and *setRange()* are used to define the domain it belongs to and property type (i.e., XSD.boolean). Because of the defined ontology structure, the subclasses of *Component* will automatically inherit a data property of *hasStatus* and an object property of *hasGeometry* which points to *AbstractGeometry* class.

Step 2: Add Individuals to Semantic Graph

Following with the assembly of ontology graph (see Figures 4.10 and 4.11), the

Abbreviated Fragment of Software Code:

```
// Define Classes

   airportnetwork = model.createClass( ns + "AirportNetwork" );
   component      = model.createClass( ns + "AbstractComponent" );
   runway         = model.createClass( ns + "Runway" );
   building       = model.createClass( ns + "Building" );
   gate           = model.createClass( ns + "Gate" );
   terminal       = model.createClass( ns + "Terminal" );
   obstruction    = model.createClass( ns + "Obstruction" );
   holdingPosition= model.createClass( ns + "HoldingPosition" );
   taxiway        = model.createClass( ns + "Taxiway" );
   flightSegment  = model.createClass( ns + "FlightSegment" );
   airplane       = model.createClass( ns + "Airplane" );
   temporalentity = model.createClass( ns + "TemporalEntity" );

// Define relationships among classes ...

   component.addSubClass ( runway );
   component.addSubClass ( building );
   building. addSubClass ( terminal );
   component.addSubClass ( gate );
   component.addSubClass ( airplane );
   airplane.addSubClass ( flightSegment );
   component.addSubClass ( obstruction );
   component.addSubClass ( holdingPosition );
   component.addSubClass ( taxiway );

// Create object properties for the class AirportNetwork and AbstractComponent ...

   hasComponent = model.createObjectProperty( ns + "hasComponent" );
   hasComponent.setDomain( airportnetwork );
   hasComponent.setRange( component );
```

Figure 4.8: Fragment of code to manually set up airport network semantic model with temporal aspects (part a).

Abbreviated Fragment of Software Code:

```
// Create object properties for the class Airplane and FlightSegment...

   hasFlightSegment = model.createObjectProperty( ns + "hasFlightSegment" );
   hasFlightSegment.setDomain( airplane );
   hasFlightSegment.setRange( flightSegment );

// Create object properties for the class Airplane ...

   takesOffBefores = model.createObjectProperty( ns + "takesOffBefores" );
   takesOffBefores.setDomain( airplane );
   takesOffBefores.setRange( airplane );

// Create data properties for the class TemporalEntity ...

   isInstant = model.createDatatypeProperty( ns + "isInstant" );
   isInstant.setDomain( temporalentity );
   isInstant.setRange( XSD.boolean );

   ... other details removed ...

// Create data properties for the class FlightSegment ...

   beginsAt = model.createDatatypeProperty( ns + "beginsAt" );
   beginsAt.setDomain( flightSegment );
   beginsAt.setRange( XSD.string );

   ... other details removed ...
```

Figure 4.9: Fragment of code to manually set up airport network semantic model with temporal aspects (part b).

airport network ontology individuals, associated data properties of each individual, and object properties between various individuals shall be defined with the airport network ontology. The code below starts with establishing a name space for the airport network ontology, constructing an ontology model (graph) which stores individuals and their data and object properties. In the code below, two airplane individuals and two temporal entity individuals are created and added to the semantic graph. The object property relationship between flight segments and airplanes is set to be the *hasFlightSegment* (shown in bottom portion of the code). The data values of the two flight segments are assgined using the *beginsAt* property. Similarly, *hasDuration* and *endsAt* for each flight segment can be filled in. Once these properties are filled, the semantic graph can query and compare between a pair of flight segments or query all three data properties of a flight segment in order to see if any property needs to be updated.

Step 3: Query Semantic Graphs

In this section, querying for the semantic graph is shown in Figure 4.12. First, the *SystemDataModel* object is created and assigned as `sdm02`. Then, the *getData()* method in *SystemDataModel* class is called in order to extract the data file of *data/umd-airport-network01.xml* and save it under the `sdm02` object. The next line of code shows the printing method to print the data in the object. The second portion of the code shows the querying of various components that are saved under the `sdm02` object. To accomplish that, the *getComponents(), getStaffs(), getBuildings(), getObstructions(), getTemporalEntities(), and getAirplanes()* methods are used to extract the corresponding list of objects.

Abbreviated Fragment of Software Code:

```
// Namespace for tiny urban network ...

   String ns = "http://cee.umd.edu/taxiway-time#";

// Create ontology model (a graph) ...

   OntModel model = ModelFactory.createOntologyModel();

// Retrieve classes and associated properties from semantic model ...

   OntClass airportnetwork = jsm.findNamedClass("AirportNetwork");

// Add "airplane01" and "airplane02" to the airport network graph model ...

   Individual airplane01 = airplane.createIndividual( ns + "Airplane#01" );
   Individual airplane02 = airplane.createIndividual( ns + "Airplane#02" );
   model.add( airplane01 );
   model.add( airplane02 );

// Add "fsegment01" and "fsegment02" to the airport network graph model ...

   Individual fsegment01 = flightSegment.createIndividual(ns+"FlightSegment#01");
   Individual fsegment02 = flightSegment.createIndividual(ns+"FlightSegment#02");
   model.add( fsegment01 );
   model.add( fsegment02 );
```

Figure 4.10: Fragment of code to add individuals and properties (related to time) to the airport network semantic model (part a).

Abbreviated Fragment of Software Code:

```
// Add "time01" and "time02" to the airport network graph model ...

   Individual time01 = temporalentity.createIndividual( ns + "Time#01" );
   Individual time02 = temporalentity.createIndividual( ns + "Time#02" );
   model.add( time01 );
   model.add( time02 );

// Create statements: Airplane#01/02 hasFlightSegment FlightSegment#01/02 ...

   hasFlightSegment = model.createObjectProperty(ns + "hasFlightSegment");
   Statement plane01stmt = model.createStatement( airplane01,
                                                  hasFlightSegment, fsegment01 );
   Statement plane02stmt = model.createStatement( airplane02,
                                                  hasFlightSegment, fsegment02 );
   model.add ( plane01stmt )
   model.add ( plane02stmt )

// Create statements: FlightSegment#01 beginsAt "2021-12-28T09:00:00Z" ...

   Literal t01 = model.createTypedLiteral("2021-12-28T09:00:00Z",
                                  XSDDatatype.XSDdateTime );
   beginsAt = model.createObjectProperty(ns + "beginsAt");
   Statement fss01 = model.createStatement(fsegment01, beginsAt, t01 );
   model.add ( fss01 );

// Create statement: FlightSegment#02 beginsAt "2021-12-28T09:15:00Z" ...

   Literal t02 = model.createTypedLiteral("2021-12-28T09:15:00Z",
                                  XSDDatatype.XSDdateTime );
   beginsAt = model.createObjectProperty(ns + "beginsAt");
   Statement fss02 = model.createStatement(fsegment02, beginsAt, t02 );
   model.add ( fss02 );
```

Figure 4.11: Fragment of code to add individuals and properties (related to time) to the airport network semantic model (part a).

Abbreviated Fragment of Software Code:

```
// Import and print sdm02 Airport Network Data

   SystemDataModel sdm02 = new SystemDataModel();
   sdm02.getData ( "data/umd-airport-network01.xml" );
   System.out.println( sdm02.toString() );

// Query sdm02 for a list of components, runways, buildings, and so on ...

   List components       = sdm02.getComponents();
   List runways          = sdm02.getRunways();
   List buildings        = sdm02.getBuildings);
   List obstructions     = sdm02.getObstructions();
   List airplanes        = sdm02.getAirplanes();
   List holdingpositions = sdm02.getHoldingpositions();
   List flightsegments   = sdm02.getFlightsegments();
   List temporalentities = sdm02.getTemporalEntities();
```

Figure 4.12: Load and query the airport network data model related to time.

Step 4: Add Rules

Once the System Data Model is assembled correctly, the semantic graph, using *JenaSemanticModel* class, is created and the *loadOntology()* method is used to feed the ontology file into the semantic graph (see Figure 4.13 below). Then the corresponding visitor class is created in order to let the Jena Semantic Model (object) to visit the individuals in the System Data Model (sdm01), and merge the individuals with the semantic correlations. Once the individuals and the corresponding semantic relationships are populated, the Jena Semantic Model can use the *addRules()* method to add the corresponding rules to the semantic model, then use the *executeRules()* method to accomplish the semantic graph transformation.

Abbreviated Fragment of Software Code:

```
// Create Semantic Model for Airport Network System

   JenaSemanticModel jsm01 =
                     new JenaSemanticModel("Airport Network Semantic Model");

   jsm01.loadOntology ("file:ontology/umd-airport-network.owl");

// Create System Data Model Airport Network Ontology Visitor

   SystemDataModelAirportNetworkOntologyVisitor visitor01 =
                     new SystemDataModelAirportNetworkOntologyVisitor();
   visitor01.add( jsm01 );

// Visit System Data Model Accepts Airport Network Ontology Visitor

   sdm01.accept ( visitor01 );

// Add and execute rules

   jsm01.addRules ( "rules/umd-airport-network.rules" );
   jsm01.executeRules();
```

Figure 4.13: Software code to import airport network ontology, add and execute Jena rules.

In order to explain the rule-based control mechanism, a list of rules are added for temporal aspects of the airport network semantic model. The following rules are:

Rule 1: Filling the object property relationships among the subclasses and superclasses.

Rule 2: Comparing two flight segments for a common airplane and assigning sequence between flight segments.

Rule 3: Computing duration of a flight segment when begin and end times are given.

Rule 4: Computing begin time of a flight segment when end and duration are given.

Rule 5: Computing end time of a flight segment when begin time and duration are given.

Step 5: Data and Event-Driven Graph Transformation (Jena Rules)

The semantic graph transformation can be initiated with the implementation of above four rules in the previous section via rule-based control mechanism supported by Apache Jena. Given the airplane individuals' schedules (departure times), an object property relationship of *takesOffBefore* can be assigned between two airplanes in order to know which plane leaves first. This object relationship may change if an airplane's schedules are delayed or canceled which is based on the actual departure time of each airplane. Figure 4.14 show the corresponding software code of the Jena rules described above.

The rules above include propagating class hierarchy relationships, updating an airplane's estimated arrival time based on its departure time and travel duration, as well as updating departure time and travel duration based on the other two temporal attributes of an airplane.

Abbreviated Fragment of Software Code:

```
@prefix air: <http://cee.umd.edu/airport#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

// Rule 01: Propagate class hierarchy relationships ....

[ rdfs01: (?x rdfs:subClassOf ?y), notEqual(?x,?y) ->
          (?a rdf:type ?y) <- (?a rdf:type ?x)] ]

// Rule 02: Compare Flight Segments of an Airplane...

[ AirTime01: (?a rdf:type air:airplane) (?fs1 rdf:type air:flightSegment)
             (?fs2 rdf:type air:flightSegment) notEqual(?fs1, ?fs2)
             (?fs1 air:beginsAt, ?fst1) (?fs2 air:endsAt, ?fst2)
              greaterThan(?fst1,?fst2) -> (?fs2 air:happensBefore, ?fs1) ]

// Rule 03: Computes duration of a Flight Segment ...

[ AirTime02: (?a rdf:type air:airplane) (?fs rdf:type air:flightSegment)
             (?fs air:beginsAt, ?fsb) (?fs air:endsAt, ?fse)
             getDuration(?fsb,?fse,?d) -> (?fs air:hasDuration, ?d) ]

// Rule 04: Computes begin time of a Flight Segment ...

[ AirTime03: (?a rdf:type air:airplane) (?fs rdf:type air:flightSegment)
             (?fs air:endsAt, ?fse) (?fs air:hasDuration, ?fsd)
              getBeginTime(?fse,?fsd,?b) -> (?fs air:beginsAt, ?b) ]

// Rule 05: Computes end time of a Flight Segment ...

[ AirTime05: (?a rdf:type air:airplane) (?fs rdf:type air:flightSegment)
             (?fs air:beginsAt, ?fsb) (?fs air:hasDuration, ?fsd)
              getEndTime(?fsb,?fsd,?e) -> (?fs air:endsAt, ?e) ]
```

Figure 4.14: Simplified Jena rules for the airport network operations (time).

4.3.2 Semantic Models of Space and Space-based Reasoning

Step 1: Define Tiny Airport Network Ontology and Semantic Graph with Space

Figure 4.15 shows a simplified airport network ontology of its classes, object properties and data properties with the assertion of spatial relationships among classes. Here, several spatial object relationships are assigned between classes. For instance, *hasAirplane, hasRunway, hasTerminal, hasHoldingPosition* are assigned among the *Airplane, Runway, Terminal* and *HoldingPosition* classes. Some data properties for spatial reasoning are also declared - *hasWeight, length, width* and other details that are not shown in the simplified network ontology.

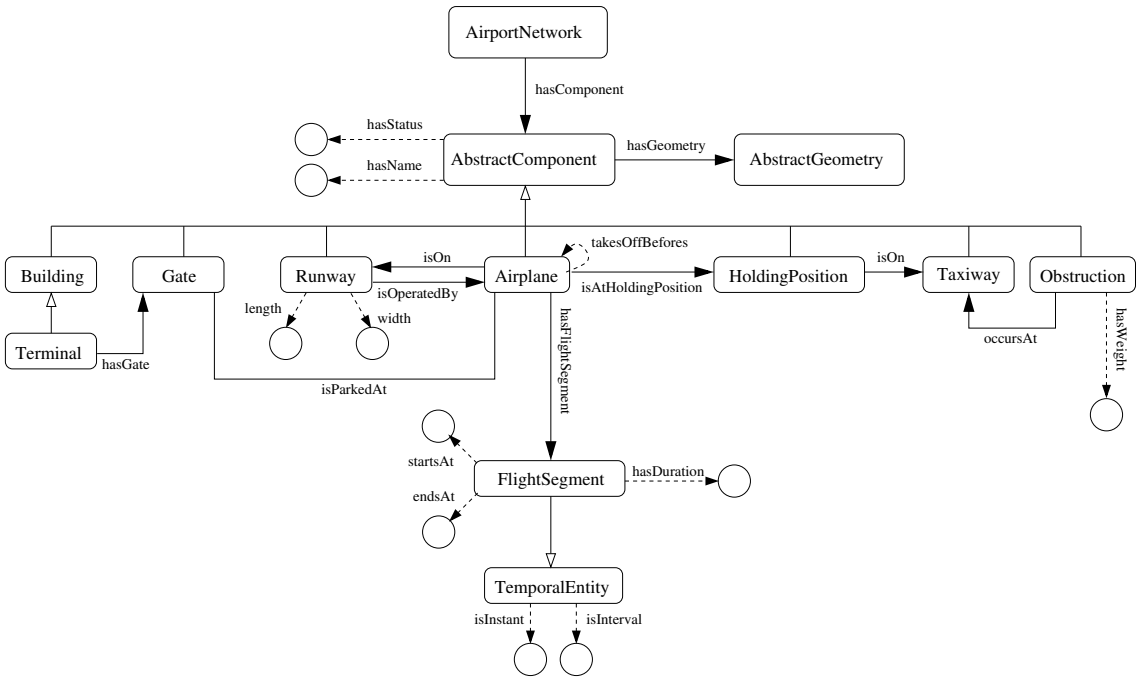


Figure 4.15: Simplified airport network ontology with its classes, object and data properties (reasoning with time + space).

The software code in Figures 4.16 and 4.17 show the assembly of simplified airport network ontology classes and properties which results into a hierarchy for the c

lass:AirportNetwork. Relationships between these seven classes are settle by the second portion of the code where subclasses are added with the addSubClass() method. Furthermore, the data properties are also assigned where the createDatatypeProperty() is used to create a data property, and setDomain() and setRange()methods are used to define the domain it belongs to and property type (i.e., XSD.double). Because of the defined ontology, the subclasses of *Component* will automatically inherit a data property of *hasStatus* because of the relationship between subclass and superclass.

Abbreviated Fragment of Software Code:

```
// Define Classes

   airportnetwork = model.createClass( ns + "AirportNetwork" );
   component      = model.createClass( ns + "AbstractComponent" );
   runway         = model.createClass( ns + "Runway" );
   building       = model.createClass( ns + "Building" );
   terminal       = model.createClass( ns + "Terminal" );
   obstruction    = model.createClass( ns + "Obstruction" );
   holdingPosition= model.createClass( ns + "HoldingPosition" );
   flightSegment  = model.createClass( ns + "FlightSegment" );
   airplane       = model.createClass( ns + "Airplane" );
   temporalentity = model.createClass( ns + "TemporalEntity" );

// Define relationships among classes ...

   component.addSubClass ( runway );
   component.addSubClass ( building );
   component.addSubClass ( obstruction );
   component.addSubClass ( airplane );
   component.addSubClass ( holdingPosition );
   building. addSubClass ( terminal );
   airplane.addSubClass ( flightSegment );
```

Figure 4.16: Fragment of code to manually set up airport network semantic graph with spatial aspects (part a).

Step 2: Add Individuals to Semantic Graph

Following with the assembly of ontology graph, airport network ontology individ-

Abbreviated Fragment of Software Code:

```
// Create object properties for class AirportNetwork and AbstractComponent ...

   hasComponent = model.createObjectProperty( ns + "hasComponent" );
   hasComponent.setDomain( airportnetwork );
   hasComponent.setRange( component );

// Create object properties for class Airplane and other classes ...

   hasRunway = model.createObjectProperty( ns + "hasRunway" );
   hasRunway.setDomain( airplane );
   hasRunway.setRange( runway );

   hasTerminal = model.createObjectProperty( ns + "hasTerminal" );
   hasTerminal.setDomain( airplane );
   hasTerminal.setRange( terminal );

   ... other details removed ...

// Create object properties for class Airplane ...

   takesOffBefores = model.createObjectProperty( ns + "takesOffBefores" );
   takesOffBefores.setDomain( airplane );
   takesOffBefores.setRange( airplane );

// Create data properties for the class Runway ...

   length = model.createDatatypeProperty( ns + "length" );
   length.setDomain( runway );
   length.setRange( XSD.double );

   ... other details removed ...
```

Figure 4.17: Fragment of code to manually set up airport network semantic graph with spatial aspects (part b).

uals, associated data properties of each individual, and object properties between various individuals shall be defined with the airport network ontology (see Figure 4.18). The code below starts with establishing a name space for the airport network ontology, constructing an ontology model (graph) which stores individuals and their data and object properties. Here, two airplane individuals, one obstruction individual, and a runway individual are created. Through the class hierarchy relationships, the geometries of obstruction and airplane classes can be extracted with the getJTSGeometryString() method, and same for the runway individual which has a type of spatial entity. Once the data properties of these spatial entities are extracted, the spatial relationships among these entities can be evaluated with built-in functions, which will be explained in the next sections. In addition, new object properties that set the relationships of various spatial entities can also be created and assigned with Jena rules. Here, the hasRunway relationship is shown in the semantic graph, however, it is not declared below with the individuals but should be completed after checking with the semantic rules (graph transformation).

Step 3: Query Semantic Graphs

In this section, querying for the semantic graph is shown in Figure 4.19. First, the *SystemDataModel* object is created and assigned as `sdm03` . Then, the *getData()* method in *SystemDataModel* class is called in order to extract the data file of *data/umd-airport-network01.xml* and save it under the `sdm03` object. The next line of code shows the printing method to print the data in the object. The second portion of the code shows the querying of various components that are saved under the `sdm03` object. To accomplish that, the *getComponents(), getRunways(), getBuildings(), getObstructions(),*

Abbreviated Fragment of Software Code:

```
// Namespace for tiny urban network ...

   String ns = "http://cee.umd.edu/taxiway-time#";

// Create ontology model (a graph) ...

   OntModel model = ModelFactory.createOntologyModel();

// Retrieve classes and associated properties from semantic model ...

   OntClass airportnetwork = jsm.findNamedClass("AirportNetwork");

// Add "airplane01" and "airplane02" to the airport network graph model ...

   Individual airplane01 = airplane.createIndividual( ns + "Airplane#01" );
   Individual airplane02 = airplane.createIndividual( ns + "Airplane#02" );
   model.add( airplane01 );
   model.add( airplane02 );

// Add "obstacle01" to the airport network graph model ...

   Individual obstacle01 = obstruction.createIndividual( ns + "Obstacle#01" );
   model.add( obstacle01 );

// Add "runway01" individual to the airport network graph model ...

   Individual runway01 = spatialentity.createIndividual( ns + "Runway#01" );
   model.add( runway01 );

// Get geometries of airplane, obstruction and runway ...

   String airplaneGeo = airplane.getJTSGeometryString();
   String obstructionGeo = obstruction.getJTSGeometryString();
   String runwayJTS = runway.getJTSGeometryString();

// Create statement:   Runway#01 hasJTSGeometry getJTSGeometryString() ...

  Literal jtsPointr=model.createTypedLiteral(componentJTS,XSDDatatype.XSDstring);
  Statement jtsGeor=model.createStatement(runway01, hasJTSGeometry, jtsPointr  );
  model.add( jtsGeor );

   ... other details removed ...
```

Figure 4.18: Fragment of code to add individuals and properties (related to space) to the airport network semantic model.

getSpatialEntities(), getAbstractGeometries(), and getAirplanes() methods are used to extract the corresponding list of objects.

Abbreviated Fragment of Software Code:

```
// Import and print sdm03 Airport Network Data

   SystemDataModel sdm03 = new SystemDataModel();
   sdm03.getData ( "data/umd-airport-network01.xml" );
   System.out.println( sdm03.toString() );

// Query sdm03 for a list of components, runways, buildings, obstructions ...

   List components         = sdm03.getComponents();
   List runways            = sdm03.getRunways();
   List buildings          = sdm03.getBuildings);
   List obstructions       = sdm03.getObstructions();
   List airplanes          = sdm03.getAirplanes();
   List abstractgeometries = sdm03.getAbstractGeometries();

   ... other details removed ...
```

Figure 4.19: Load and query the airport network semantic graph related to space.

Step 4: Add Rules

In order to explain the ideas of rule-based control mechanism, a list of rules are added for the spatial aspect of the airport network semantic model: The following rules can be declared and added to the Jena rules in the next step:

- **Rule 1:** Comparing spatial relationship between an airplane and a runway. This rule is to check whether an airplane is currently on a runway.
- **Rule 2:** Comparing spatial relationship between an obstruction object and a runway. This rule is to make sure if a runway is currently available for take-off tasks.
- **Rule 3:** Checking safety distance between a pair of airplane. Given the geome-

tries and locations of airplanes, the edge-to-edge distance can be computed. This rule helps to avoid further collision between the airplanes and checks distances of airplanes over time.

Step 5: Data and Event-Driven Graph Transformation (Jena Rules)

The semantic graph transformation can be initiated with the implementation of above four rules in the previous section via rule-based control mechanism supported by Apache Jena. Given the spatial locations and geometries of the physical entities at an airport (airplanes, runways, obstructions, etc.), new object property relationships can be assigned between the spatial individuals in order to assure safety at an airport. The rules below are described previously where the first rule sets the class hierarchy relationships. The second rule uses a built-in function of *getPlaneInRunway()* in order to assess if a plane is on a runway, and if so, *isOnRunway* will be assigned between airplane and runway individuals. Software code of the rules is shown in Figure 4.20 below.

The third rule uses another built-in function of *getObstructionInRunway()* to evaluate the spatial relationship between an obstruction individual and a runway. If the built-in function returns *true*, meaning the obstruction object and runway are intersected, the runway will be assigned a data property of *isNotAvailable* as *true*. The last rule identifies the distance between a pair of airplanes. If the distance is computed to be less than 200 ft, then the two airplanes are assigned an object relation of *isTooCloseTo* between them.

Abbreviated Fragment of Software Code:

```
@prefix air: <http://cee.umd.edu/airport#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

// Rule 01: Propagate class hierarchy relationships ....

[ rdfs01: (?x rdfs:subClassOf ?y), notEqual(?x,?y) ->
          (?a rdf:type ?y) <- (?a rdf:type ?x)] ]

// Rule 02: Compare Spatial Relationship of Airplane and Runway...

[ AirSpace01: (?a rdf:type air:airplane) (?r rdf:type air:Runway)
              (?a air:hasGeometry, ?ag) (?r air:hasGeometry, ?rg)
               getPlaneInRunway(?ag,?rg,?t)
               equal(?t, "true"^^xs:boolean) -> (?a air:isOnRunway ?r, ?t) ]

// Rule 03: Evaluate if an Runway is being obstructed ...

[ AirSpace02: (?o rdf:type air:obstruction) (?r rdf:type air:Runway)
              (?o air:hasGeometry, ?og) (?r air:hasGeometry, ?rg)
               getObstructionInRunway(?og,?rg,?t)
               equal(?t, "true"^^xs:boolean) -> (?r air:isNotAvailable, ?t) ]

// Rule 04: Safety Distance Between Two Airplanes (>200 ft) ...

[ AirSpace03: (?a1 rdf:type air:airplane) (?a1 air:hasGeometry, ?ag1)
              (?a2 rdf:type air:airplane) (?a2 air:hasGeometry, ?ag2)
               getDistance(?ag1,?ag2,?dist) lessThan(?dist,200) ->
              (?a1 air:isTooCloseTo, ?a2) ]
```

Figure 4.20: Simplified Jena rules for the airport network operations (space).

Chapter 5: Real-Time Planning for Urban Operations

This chapter covers real-time planning for dynamic urban operations. We introduce OptapPlanner, an open-source Java software for the solution of constraint satisfaction problems, and explain how to use planning entities, constraints, and heuristic and metaheuristic algorithms to resolve scheduling conflicts, allocate limited resources, and plan sequences of actions to promote urban safety. This capability will be exercised in the airport taxiway (see Chapter 6) and wildfire evacuation (see Chapter 7) case studies.

5.1 Real-World Planning and Replanning

Planning and Replanning. Planning is the process of determining how a problem can be solved. A well-developed plan sheds light on the road ahead, and will contain an organized set of actions that moves the current system state to a future system state (or goal) that does not currently hold. The synthesis of plans requires lookahead capability and the ability to combine predicted steps (i.e., if an action a is applied to a system in state s_0, then system state will transition to a new state s_1). A second closely related problem is that of scheduling, that is determining when the associated actions will be executed.

In real-world urban settings, planning problems usually involve providing services

or products under a limited number of resources (i.e., time, cost, schedule, employees) [116]. A planner must have the capability to reason about time and limited resources, coordinate with other planning processes, and deal with events that may be difficult to predict. To deal with these uncertainties, plans should be viewed as tentative, rather than entities that will blindly executed to their conclusion. Computational support for plan modification and replanning over time horizons should be embedded within the design of urban operations. As already noted in Chapter 3, this study uses multi-domain semantic modeling and reasoning to assess the overall system state and trigger the need for replanning (i.e., lookahead to plan actions for transition to future system states) of forthcoming goals.

State-of-the-Art Behavior Modeling for Planning and Control. Behavior modeling has been implemented in many research areas [4, 78, 131, 149] related to planning. A dynamic mixed discrete-continuous choice approach [78] is proposed to understand process of decision-making on travels and activity options among public facilities. The framework for risk potential-based motion planning [4] is studied on behavior modeling for random driver speed controls at traffic intersections. [149] focused on integrating various decisions for modeling driving behaviors (i.e., acceleration, lane changing gap acceptance etc.). Behavior modeling is also applied on non-urban issues. A study on eye movements behavior modeling [131] is conducted for selective attention on sequential behaviors of eyes. It is important to note that state-of-the-art behavior modeling for planning and control does not have the capability of handling the urban problems comprised of multiple participating domains that are interconnected and interdependent.

Related Literature: Optimal Allocation of Resources in Real-Time. FEMA [49] mentioned a 4-step hazard mitigation planning process which includes organizing planning process and resources, risk assessment, mitigation strategies, and plan adaptation and implementation. This indicates that resource allocation is an important first-step for planning, even in urban domains to avoid any hazardous events. Some research used MILP [59] to solve dynamic parking space (resource) allocation for drivers with a smart parking system and analyzed with free space detection and driver localization; Rahman and Sharman (2020) [126] used a framework with neural networks (for graph generations) and reinforcement learning for urban planning operations and optimization – compared to other methods, this method is more generalizable and scalable. Our approach covers the real-time planning for resource allocation and the constraints can incorporate both spatial and temporal aspects.

5.2 Constraint Satisfaction Problems

Mathematical Formulation. A constraint satisfaction problem (CSP) consists of a set of varibles X = $\{x_1, x_2, \cdots, x_n\}$, each associated with a domain D_i of values, and a set of constraints C = $\{c_1, c_2, \cdots, c_m\}$, which denote the legal combinations for the variables such that $C_i \subseteq D_1 \times D_2 \times \cdots \times D_n$.

A feasible solution corresponds to the assignment to each variable one of its permissible values, in such a way that all of the contstraints are satisfied. The transformation (or resolution) of partial solutions into complete solutions can be cast as an iterative search procedure.

Planning Variables. Each variable (or component) will have a set of values over which it can evolve over time.

Sequence, Temporal, and Resource Constraints. Constraints are restrictions on the values that one or more of the variables can take. A unary constraint is a restriction on a single variables (e.g., $x_1 \neq 1$). A k-ary constraint places a restriction on k different variables (e.g., $x_1 + x_2 < 1$).

A sequence constraint specifies: (1) all of the possible sequences of values a single component may assume over time, and (2) the legal combination of values over the same time period among comonents (i.e., permissible synchronization). Temporal constraints specify the duration (i.e., time interval) over which a single value holds, as well as constraints among time intervals related to different values (i.e., Allen's algebra). Resource constraints describe resources that need to be shared during the execution of a system.

Hard and Soft Constraints. The mathematical constraints are constructed with a scoring system. Each hard constraint gives an overall hard score which is used to determine whether a solution is feasible, and each soft constraint gives an overall soft score that is used to determine which solution is better. Hard constraints are set to be either less than zero or equal to zero. A negative value of hard constraint explains that a solution is not feasible, hence, it will not be returned as the final solutions.

Cost Functions/Scores. CSP turns into an optimization problem when we wish to find the best model among the feasible solutions.

Scores provide an objective way to compare two solutions. Scoring techniques can be used to maximise/minimize a constraints, associate profit with satisfaction of constraints, and prioritize constraints.

Solution Procedures. CSP problems on finite domains are typically solved using some form of search. Search procedures work by guess an operation (action) to perform, possibly with the assistance of a heuristic. A good guess results in an overall score that is closer to the desired goal. However, when a variable is encountered where none of its permisible values are consitent with a partial solution, a search procedure is required. These procedures can employ combinations of schemes for looking ahead, back tracking and constraint propogation.

5.3 Software Library Support for Real-Time Planning

Table 5.1 contains a summary of software libraries that support development of plannijngn applications. Our adoption of OptaPlanner for real-time planning is driven by several reasons: (1) it supports real-time, dynamic and continuous planning, (2) it is compatible with all JVMs, (3) it contains range of urban scheduling, resource allocation and shortest path problems, (4) it is capable of embedding constraints separately from the main software architecture, and (5) the Java library is open source.

Reference	Software Name	Description
OptaPlanner (2016) [116]	OptaPlanner Constraint-Satisfaction Solver	It supports real-time, dynamic and continuous planning; applicable with all JVMs; variety of optimization applications supported with hard and soft constraints; lightweight, embeddable planning engine; OOP and FP friendly; math constraints aren't required; an open source Java constraint solver.
Choco-solver (2016) [124]	Choco-solver	It solves problems by alternating constraint filtering algorithms with a search mechanism; requires Java Oracle JRE 11 or OpenJDK 11, and Maven 3+; comes with various variables, constraints and strategies; an open-source Java library for constraint programming.
Forecast (2022) [55]	Forecast Resource and Project Management Software	An AI platform runs all project operations for project, resources and financials; project schedules and visualizations; provides high-level bird's eye view of project activities; Not an open source.
Global Shop Solutions (2022) [63]	Global Shop Solutions (ERP Software)	An integrated software for production, financials and project management; 30+ applications from quote to cash; Enterprise-oriented paid software; Not an open source.
JaCoP (2022) [91]	Java Constraint Programming (JaCoP) solver	It includes rich set of primitive, conditional, and logical & global constraints; uses front-end with FlatZinc language; available from maven repository; a Java-based open source solver.
Locoia (2022) [100]	Locoia (Low Code Intelligent Automation)	It integrates entire tech stack through the all-in-one iPaaS solution; promotes process automation; a quick integration software and systems via automated workflows; Not an open source.
Sat4j (2010) [21]	Sat4j (SAT-based solvers in Java)	A java library for solving boolean satisfaction and optimization problems; it solves SAT, Pseudo-boolean and MaxSAT problems with generic constraints; an open source project.

Table 5.1: Summary of software libraries that support development of planning applications.

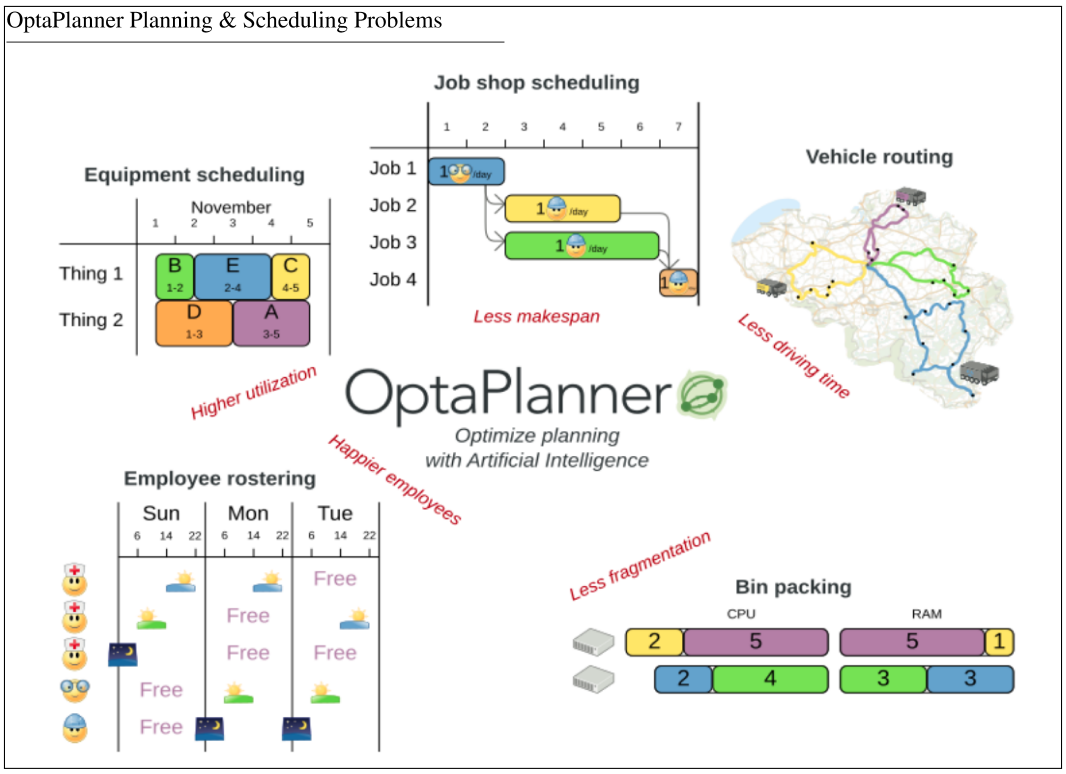


Figure 5.1: Collage of OptaPlanner scheduling and planning problems [116].

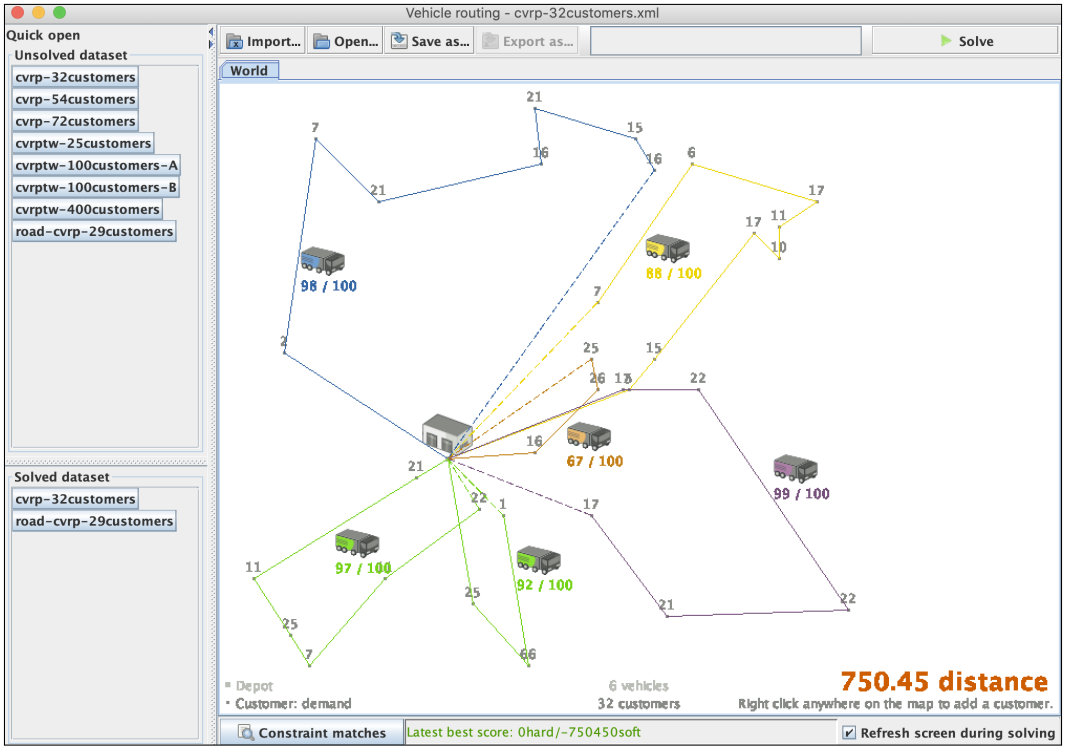


Figure 5.2: Vehicle routing app example (Source: [116]).

5.4 OptaPlanner: A Constraint Satisfaction Solver

OptaPlanner [115] is an open source Java-based constraint solver for the solution of planning problems. It provides a platform to support and design for efficient plans based on limited constraint resources, such as costs, employees, service quality, temporal and spatial constraints, etc. Some applications will demand minimization of resource utilization, others maximization. Most applicatioms will work to minimize the costs associated with scheduling conflicts. The range of planning problems that can be solved – equipment scheduling, job shop scheduling, employee rostering, task assignment, vehicle routing problem – is summarised in Figure 5.1. Figure 5.2 shows an example of the software graphical user interface (GUI) with the vehicle routing problem.

Problem Setup and Solution. Problem setup requires completion of three steps: (1) specification of the planning variables, (2) specification the hard and soft constraints, and (3) specificatiom of the input and output parameters. OptaPlanner provides computational support for the satisfaction of hard and soft constraints (see details below) associated with multiple forms of planning: continuous planning, non-disruptive planning, over-constrained planning and real-time planning. A score calculation system (that uses hard score and soft score) and some complex AI optimization algorithms (e.g., simulated annealing, tabu search, late acceptance) are integrated to help solving NP (nondeterministic polynomial time) problems. An optimal solution is a solution with the highest score.

Hard and Soft Constraint Satisfaction. Constraints are organized into two categories - hard constraints and soft constraints. Hard constraints represent restrictions that must be

satisfied in order for the planning solution to be feasible (e.g., two entities cannot occupy the same space at the same time). Soft constraints represent restrictions that should not be broken. They can be both negative (avoid) and positive (preference).

Score Calculation and DROOLS Rule Engine. The score calculation is driven by the DROOLS rule engine, a business rules management system. Every constraint is written as one or a set of score rules in DRL syntax. The use of Drools comes with the following benefits: (1) incremental score calculation without extra code (DRL uses forward chaining), (2) constraints can be implemented as separate rules, and (3) constraints can be implemented in a variety of formats (e.g., decision tables, and others). Drools also decouples the score level and score weight of each individual constraint and provides flexibility to allow users edit the constraints before problem solving.

At the implementation level, the *ScoreHolder* class has two methods - *reward()* and *penalize()* which further allow users to impact the score calculation by rewarding or penalizing a solution when a constraint is matched. If there is no constraint configuration (a separate class that holds constraint parameters and constraint weights), the class also uses methods, such as *addSoftConstraintMatch()* and *addHardConstraintMatch()* to reward or penalize a matched constraint by assigning a positive or negative value to the addition of overall score respectively.

5.5 Working Example: Path Planning on a Small Network

As a first step toward demonstrating the planning capabilities provided by OptaPlanner, this section follows a step-by-step procedure for setting up the small urban network structure introduced in Chapter 2, and then computing a schedule of evacuation activities that minimize time, and take into account resource constraints. It is important to note that the latter can not be handled by JGraphT.

5.5.1 Problem Description

The small network structure contains a set of nodes and a set of edges in which the latter connects pairs of nodes. The edges can be either directed or undirected. A weight value shall be assigned to each edge in order to identify the cost of using the corresponding set of nodes and edges to traverse from origins to destinations. Figure 5.3 shows the basic structure of the small urban network that has a set of towns (shown on the left) as the origins and a set of safe areas (shown on the right) as the destinations for the optimization problems. The nodes and edges are defined in the Figure and the associated weight value of each edge is also displayed. In addition, the graph is directed which means that the reverse direction of each edge is not accessible. Figure 5.4 shows the same town structure but having an obstruction (i.e., flooding) at the edge of `EG`, increasing the weight (or cost of using that edge) value from 5 to 20.

Planning Entity. The planning entity in this case is the class `Person` which stores the summation of weights of all `Edges` saved under `Pathway`. Furthermore, it also stores

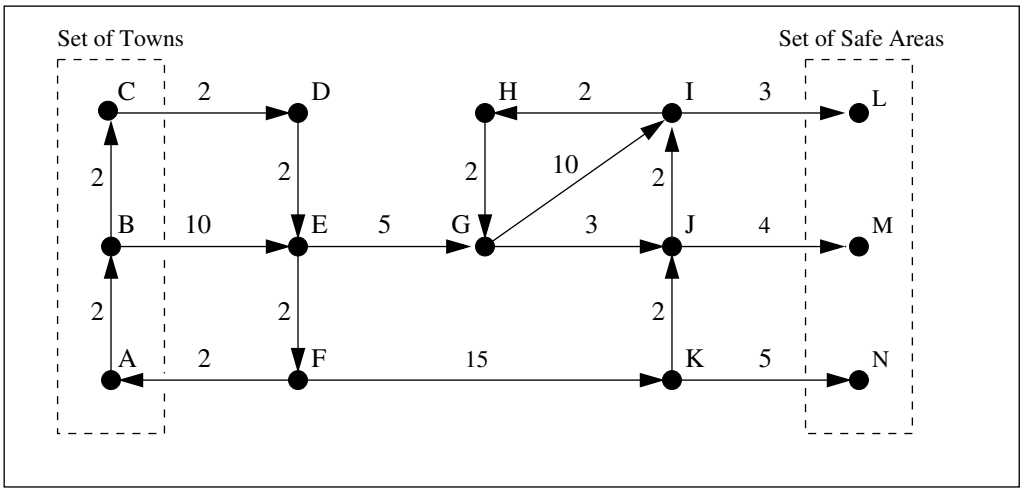


Figure 5.3: Path planning on a tiny town network structure.

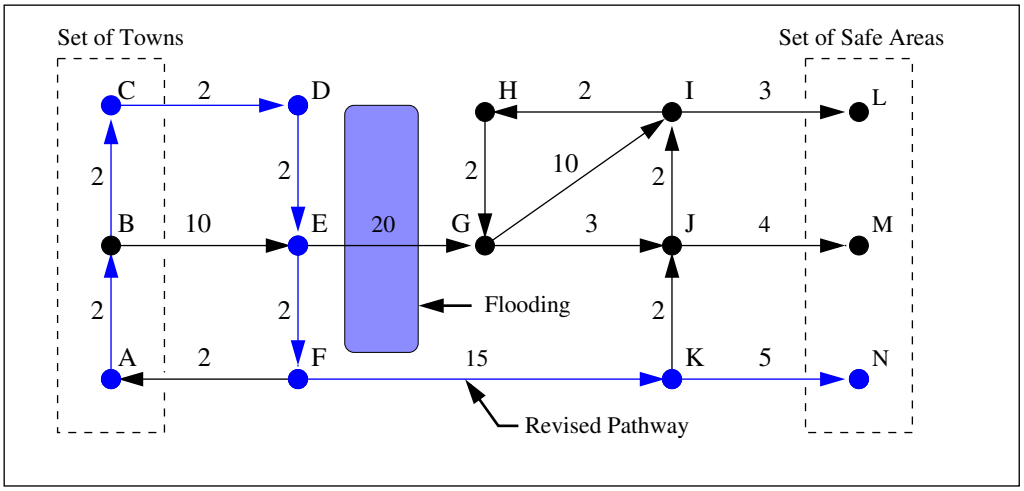


Figure 5.4: Path planning avoids disruptions on a tiny town network structure.

List<Edge> which represents a list of edges, including an Origin and a Destination assigned to each pathway which the person (or any arbitrary physical component of a system) at this location has decided to take as the travel (shortest) path.

Planning Variable. The planning variable is a Java Bean property, representing either a getter or a setter, on a planning entity. This directly points to a planning value which changes over time as the program executes. The main planning variable is the weight attribute stored in Edge, list of edges stored in each Pathway, and locations of associated origins and destinations.

5.5.2 Step-by-Step Solution Procedure

Figure 5.5 shows the class diagram setting up for the small town network and planning solution is set to be finding the shortest path, in this case from <Origin> A to <Destination> N. We assume the entity that is traveling through the nodes and edges has a class Person which has ID and name stored. The planning variables that are based on the planning entity (Person) are also defined. Each Person has a Location and a Pathway. Once the optimization is completed, the optimized path will be saved in Pathway as a List<Edge>. The Location of the person is also a planning variable since the program needs to know the current location of the Person.

The three other classes of List<Edge>, theOrigin and Destination are defined under the class Pathway and they all have an one-to-one relationship with class Pathway. Each Edge contains two nodes that are represented by Location and each Origin and Destination are also denoted by Location, and they have one-to-one relation.

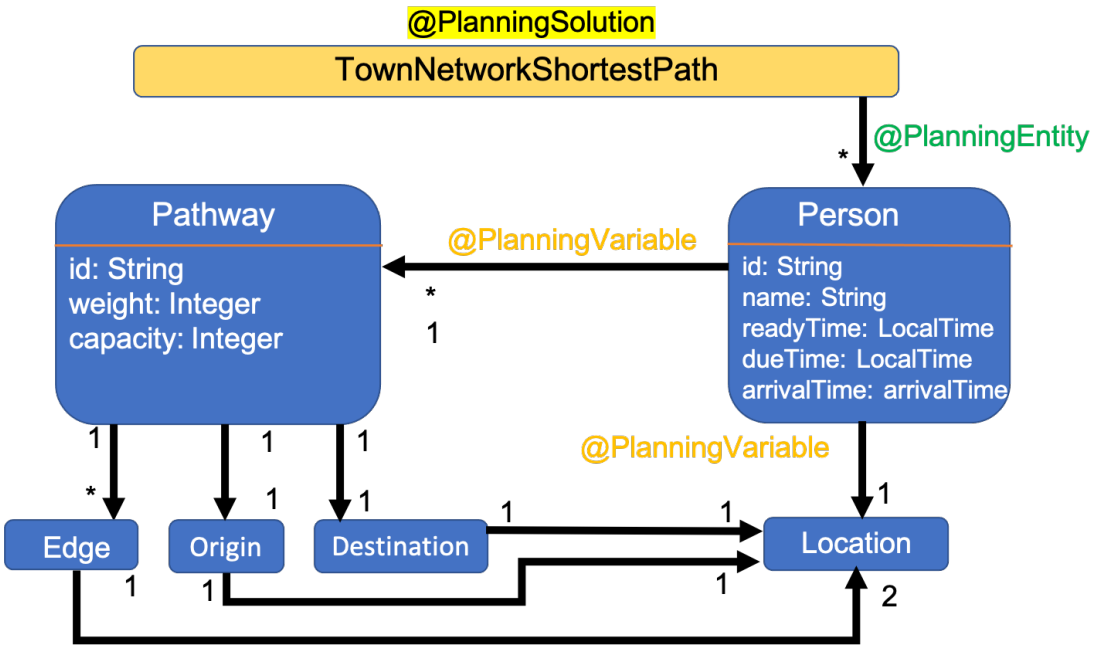


Figure 5.5: Class diagram for the tiny town network structure.

The importance of using OptaPlanner is to clearly define the relationships among various classes and each class can have a set of attributes that are potentially used within optimization. Following this procedure, the constraint has to be defined for the purpose of solving it mathematically. OptaPlanner provides various types of scoring evaluations.. *Hard Score* and *Soft Score* are most commonly used among the problems that are solved with this application. With respect to the scoring systems, it separates the constraints as *Hard Constraints* and *Soft Constraints*. Hard constraints must not be broken and soft constraints may not be broken. Regardless of the order of constraints being defined, hard constraints are always prioritized to be met compared to soft constraints.

Abbreviated Fragment of Software Code:

```
<locationList id="002">
    <Location id="A">
       <id>1</id>
       <latitude>34.0</latitude>
       <longitude>31.0</longitude>
    </Location> ...
    <Location id="B"> ... </Location>
    <Location id="C"> ... </Location>
    <Location id="D"> ... </Location>
</locationList>
<edgeList id="101">
    <Edge id="102">
       <id>1</id>
       <weight>2</weight>
       <location class="Location" reference="A"/>
       <location class="Location" reference="B"/>
    </Edge> ...
</edgeList>
<originList id="201">
    <Origin id="202">
       <id>1</id>
       <location class="Location" reference="A"/>
    </Origin> ...
</originList>
<destinationList id="301">
    <Destination id="302">
       <id>1</id>
       <location class="Location" reference="N"/>
    </Destination> ...
</destinationList>
<pathwayList id="401">
    <Pathway id="402">
        <id>1</id>
       <Pathway class="Origin" reference="202"/>
       <Pathway class="Destination" reference="302"/>
    </Pathway> ...
</pathwayList>
<personList id="501">
    <Person id="502">
        <id>1</id>
       <pathway class="Pathway" reference="402"/>
       <pathway class="Pathway" reference="..."/>
    </Person> ...
</personList>
```

Figure 5.6: Fragment of code for the OptaPlanner Sample Input.

5.5.3 Sample Input and Output

In order to better understand the utilization of OptaPlanner, sample blocks of code for the input (Figure 5.6) and output (Figure 5.7) of the platform are included below. The sample input includes a <locationList> that stores all the associated locations (nodes) shown in figure 5.4. The <edgeList> is added, the weight attribute is defined in each individual edge, and two end nodes of each edge are stored using the node reference id. In addition, the <originList> and <destinationList> are included to specify the specific origins and destinations with respect to the locations. The <pathwayList> is defined in order to collect all possible pathways from the defined origin and destination within the <pathwayList>. After simulation, the Pathway will store the optimized pathway solution..

The OptaPlanner sample output for the small urban network is also shown below. As mentioned in the previous paragraph, the outcomes are stored under the corresponding pathway class. The nested output file structure helps to visualize the order of edges that are taken for this specific pathway. The directed graph solved for this problem is implicitly defined based on the edge individuals stored in the input file. For instance, the <Edge id="102"> has a start location (or node) of <*location class="Location" reference="A"*/> and an end node of <*location class="Location" reference="B"*/>. The optimized solution is based on the summation of weight attributes from all the edges that are being added under the pathway object. The goal is to dynamically run simulation for the optimization until an optimized solution is reached (shown as the blue trajectory in Figure 5.4).

Abbreviated Fragment of Software Code:

```
<Pathway id="402">
   <id>1</id>
   <totalDistance>30</totalDistance>
   <origin reference="202"/>
   <destination reference="302"/>
     <nextEdge id="102">
       <id>1</id>
       <weight>2</weight> ...
       <nextEdge id="103">
         <id>1</id>
         <weight>2</weight> ...
         <person reference="402"/>
         </nextEdge>
     <person reference="402"/>
     </nextEdge>
   <person reference="402"/>
   </nextEdge>
</Pathway>
```

Figure 5.7: Fragment of code for the OptaPlanner Sample Output.

5.5.4 Tiny Urban Network: Drool Scoring Rules

This section shows the implemented constraint rules for the tiny urban network shortest path problem. OptaPlanner uses two alternatives to calculate scores which is using a simple Java implementation or Drools `DRL` type of rules. The benefits of using drools score calculation includes each constraint as an entirely separate scoring rule in which it makes the problem setup scalable with possibility of having different sets of planning constraints.

There are two types of constraints: hard and soft constraints. If a hard constraint is implemented, there has to be a solution set that meets all of the hard constraints. Otherwise, the problem is deemed infeasible. Soft constraints are more flexible, meaning

that the program will keep simulating until it gets to a level that the solution (evaluation score) cannot be improved. Figure 5.8 shows a shortest path rule for the tiny urban network problem. This rule calculates the distance based on the position of person with respect to the total traveled distance. The implementation of this rule mirrors the constraints and results in `JGraphT` as it does not include other temporal and resource allocation constraints.

Abbreviated Fragment of Software Code:

```
// Using HardSoftScoreHolder class

global HardSoftScoreHolder scoreHolder;

// Rule 01: minimize shortest path ...

rule "Shortest Path Rule"
   when
         $person: Person(previousNode != null, $distanceFromPreviousNode :
                  distanceFromPreviousNode, $totalDistance : totalDistance)
   then
         scoreHolder.addSoftConstraintMatch(kcontext,
                 -1*($totalDistance + $distanceFromPreviousNode));
end
```

Figure 5.8: Fragment of code for the OptaPlanner constraints on shortest path.

Then, `Rule 02` in Figure 5.9 indicates the capacity limit on edges with respect to total number of persons on each edge. When dealing with scheduling problems, such as taxiway operations and wildfire evacuation, this is extremely important as it describes things like traffic congestion, etc. The `Rule 03` shows a temporal constraint for a `Person` where attributes like `dueTime` and `arrivalTime` are used to set the temporal constraints of what is the due time to arrive comparing to the actual arrival time. In this case,

the later an arrival time is, the lower the overall score becomes (by adding `dueTime − arrivalTime` to the total soft score).

Abbreviated Fragment of Software Code:

```
// Rule 02: edge and person capacity rule ...

rule "Edge and Person Capacity"
   when
         $edge : Edge($capacity : capacity)
         $person : Person(edge == $edge, personSize > $capacity,
                                        $personSize : personSize)
   then
         scoreHolder.addHardConstraintMatch(kcontext, ($capacity - $personSize));
end

// Rule 03: temporal rule for a person / a flight ...

rule "Arrival After Due Time"
    when
        Person(dueTime < arrivalTime, $dueTime : dueTime,
                              $arrivalTime : arrivalTime)
    then
        scoreHolder.addSoftConstraintMatch(kcontext, $dueTime -
                                           $arrivalTime.longValue());
end
```

Figure 5.9: Fragment of code for the OptaPlanner constraints on edge capacity and entity schedules.

Some additional rules are added in order to explain the importance of OptaPlanner constraints to resource allocations. `Rule 04` in Figure 5.10 indicates a hard constraint to restrict the capacity of each safe areas to one (in this case, they are nodes `L, M, N` shown in Figure 5.4). Suppose we have three persons traveling from nodes `A, B, C` to nodes `L, M, N`. Having set the capacity of each safe area as one assures three entities will travel to different safe areas. From the resource allocation standpoint, this rule helps to allocate entities to all safe areas without going over each of its capacity.

Abbreviated Fragment of Software Code:

```
// Rule 04: node and person capacity rule ...

rule "Node and Person Capacity"
   when
         $node : Node($capacity : capacity)
         $person : Person(node == $node, personSize > $capacity,
                                        $personSize : personSize)
   then
         scoreHolder.addHardConstraintMatch(kcontext, ($capacity - $personSize));
end
```

Figure 5.10: Fragment of code for the OptaPlanner constraints on capacity of safe areas (nodes).

Additionally, Figure 5.11 shows another soft constraint of adding delay costs to the overall optimization. The `Rule 05` indicates relationships between ready time at edges and arrival time of persons. When a person arrives at an edge, and edge ready time is later than person's arrival time on that edge, then the summation of `person arrival time − edge ready time` will be added to the overall optimization score. A constraint like this can be treated as a hard constraint if delay of any kind for a specific planning problem is not acceptable.

5.5.5 Tiny Urban Network Pathway Results

This section includes the OptaPlanner outcomes for this tiny urban network problem. As previously described, three persons are defined as persons `A, B, C` and their origins are the corresponding nodes of `A, B, C` respectively. The goal is to traverse the urban network and have all three persons to travel to one of the three safe areas,

Abbreviated Fragment of Software Code:

```
// Rule 05: edge and person delay cost rule ...

rule "Edge and Person Delay Cost"
   when
       $edge : Edge($readyTime : readyTime)
       $person : Person(edge == $edge, edgeArrivalTime < $readyTime,
                                   $edgeArrivalTime : edgeArrivalTime)
   then
       scoreHolder.addSoftConstraintMatch(kcontext, $edgeArrivalTime-$readyTime);
end
```

Figure 5.11: Fragment of code for the OptaPlanner constraints on delay costs.

defined as `L, M, N` without going over the capacity of each safe area, which is set to be one because we are assigning three nodes to three destinations (a simplified resource allocation where each destination node only takes one person at a time).

Figures 5.12 and 5.13 and show the pathways of persons `A, B, C` respectively. Person A selected a pathway of $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow K \rightarrow J \rightarrow I \rightarrow L$, with a cost of 32 (`totalDistance`); Person B has $B \rightarrow C \rightarrow D \rightarrow E \rightarrow G \rightarrow J \rightarrow M$ with a cost of 33; and Person C has $C \rightarrow D \rightarrow E \rightarrow F \rightarrow K \rightarrow N$ with a cost of 26. The total costs for the overall pathways of all three persons sums to 91 which gives the smallest overall cost for this tiny urban network optimization problem.

The urban planning problem is more than understanding the spatial relationships. Figures 5.14 and 5.15 show the time and nodal locations for Persons A, B and C with respect to Figures 5.3 and 5.4. In these two charts, the temporal and spatial sequences of each person (differed by color) are shown. This further illustrates the results of this tiny urban network shortest path problem considering both time (horizontal axis) and

Abbreviated Fragment of Software Code:

```
<Pathway id="1">
  <totalDistance>32</totalDistance>
  <origin reference="A"/>
  <destination reference="N"/>
  <nextEdge id="1">
    <location class="Node" reference="B"/>
    <weight>2</weight>
    <nextEdge id="2">
      <location class="Node" reference="C"/>
      <weight>2</weight>
      <nextEdge id="3">
        <location class="Node" reference="D"/>
        <weight>2</weight>
        <nextEdge id="4">
          <location class="Node" reference="E"/>
          <weight>2</weight>
          <nextEdge id="5">
            <location class="Node" reference="F"/>
            <weight>3</weight>
            <nextEdge id="9">
              <location class="Node" reference="K"/>
              <weight>15</weight>
              <nextEdge id="17">
                <location class="Node" reference="J"/>
                <weight>2</weight>
                <nextEdge id="14">
                  <location class="Node" reference="I"/>
                  <weight>2</weight>
                  <nextEdge id="15">
                    <location class="Node" reference="L"/>
                    <weight>3</weight>
                    <person reference="A"/> </nextEdge>
                  <person reference="A"/> </nextEdge>
                <person reference="A"/> </nextEdge>
              <person reference="A"/> </nextEdge>
            <person reference="A"/> </nextEdge>
          <person reference="A"/> </nextEdge>
        <person reference="A"/> </nextEdge>
      <person reference="A"/> </nextEdge>
    <person reference="A"/> </nextEdge>
</Pathway>
```

Figure 5.12: Urban Network Problem: Fragment of OptaPlanner output for Person A.

Abbreviated Fragment of Software Code:

```
<Pathway id="2">
  <totalDistance>33</totalDistance>
  <origin reference="B"/>
  <destination reference="M"/>
  <nextEdge id="2">
    <location class="Node" reference="C"/>
    <weight>2</weight>
    <nextEdge id="3">
      <location class="Node" reference="D"/>
      <weight>2</weight>
      <nextEdge id="4">
        <location class="Node" reference="E"/>
        <weight>2</weight>
        <nextEdge id="8">
          <location class="Node" reference="G"/>
          <weight>20</weight>
          <nextEdge id="12">
            <location class="Node" reference="J"/>
            <weight>3</weight>
            <nextEdge id="16">
              <location class="Node" reference="M"/>
              <weight>4</weight>
              <person reference="B"/> </nextEdge>
            <person reference="B"/> </nextEdge>
          <person reference="B"/> </nextEdge>
        <person reference="B"/> </nextEdge>
      <person reference="B"/> </nextEdge>
    <person reference="B"/> </nextEdge>
</Pathway>
<Pathway id="3">
  <totalDistance>26</totalDistance>
  <origin reference="C"/>
  <destination reference="N"/>
  <nextEdge id="3">
    <location class="Node" reference="D"/>
    <weight>2</weight>
    <nextEdge id="4">
      <location class="Node" reference="E"/>
      <weight>2</weight>
      <nextEdge id="5">
        <location class="Node" reference="F"/>
        <weight>2</weight>
        <nextEdge id="9">
          <location class="Node" reference="K"/>
          <weight>15</weight>
          <nextEdge id="18">
            <location class="Node" reference="N"/>
            <weight>5</weight>
            <person reference="C"/> </nextEdge>
          <person reference="C"/> </nextEdge>
        <person reference="C"/> </nextEdge>
      <person reference="C"/> </nextEdge>
    <person reference="C"/> </nextEdge>
</Pathway>
```

Figure 5.13: Urban Network Problem: Fragment of code OptaPlanner output for Persons B and C.

space (vertical axis). It indicates that OptaPlanner is able to generate a set of results that satisfies hard constraints and optimizes soft constraints that users defined with `Drool` rules. Defining which constraint to be a hard constraint is significantly important as it sets filters to the feasible solutions that only when these constraints are met. For instance, `Rule 03` in Figure 5.9 is a soft constraint that checks if an actual arrival time of a person is later than its scheduled arrival time (or due time). However, this rule can be used as a hard constraint when late arrival can bring a tremendous lost to the optimization (i.e., traveling salesman problem).

Comparing the results of having a `Flood` obstruction in Figure 5.4 to not having an irregular obstruction shown in Figure 5.3, the results are clearly changed because of the obstruction introduced to the network. This means that OptaPlanner is capable of catching the irregular obstruction in the network and assign new optimized route options for persons to reach to safe areas in order to avoid the flood. However, additional delays are also inevitable because of the obstruction (e.g., Person C has a much longer wait at the initial location before leaving to the destination).

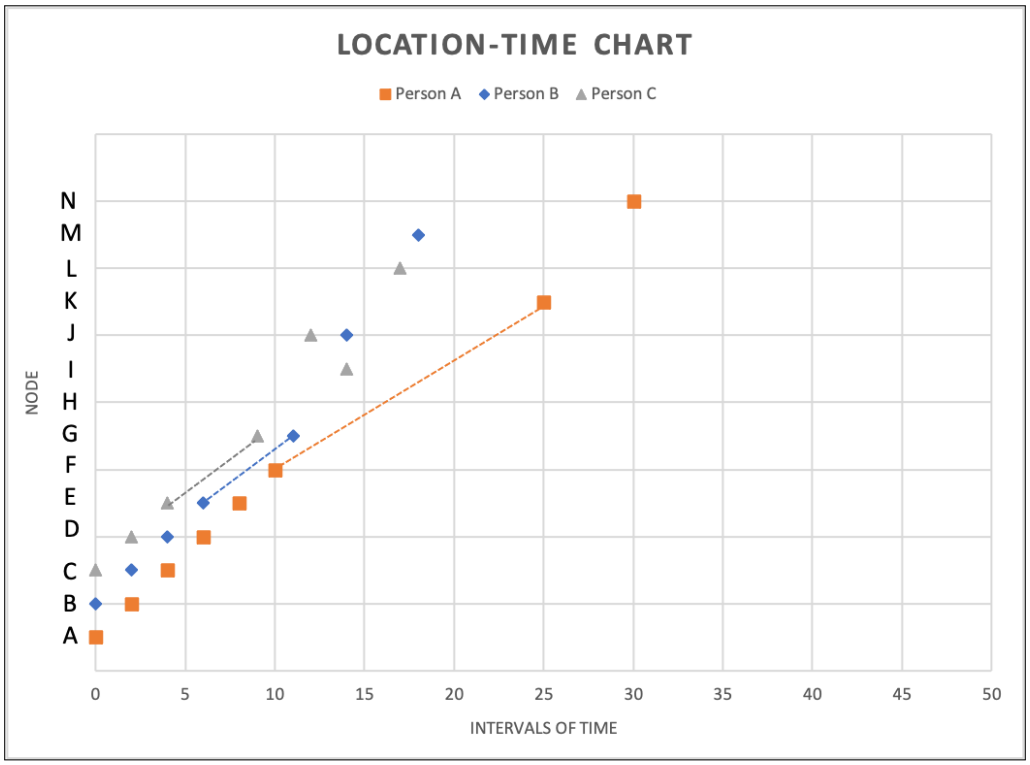


Figure 5.14: Time and location chart for Persons A, B and C (Results for Figure 5.3).

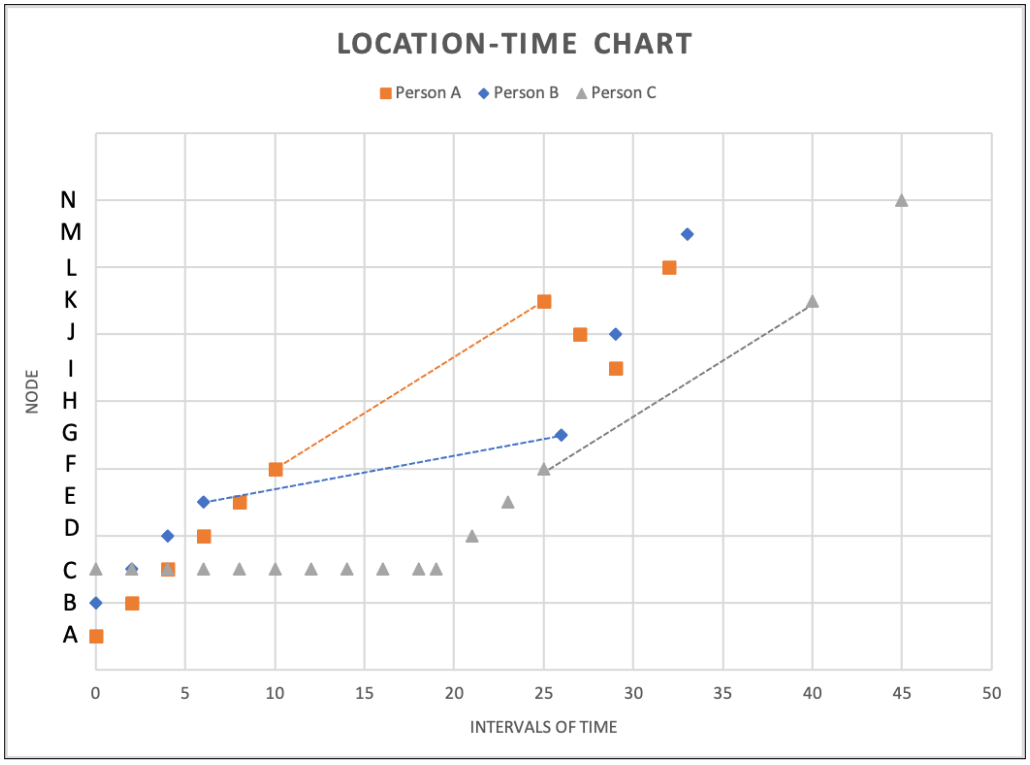


Figure 5.15: Time and location chart for Persons A, B and C (Results for Figure 5.4).

Chapter 6: Case Study A: Multi-Domain Semantic Modeling and Planning for Airport Taxiway Operations

Case Study A looks at multi-domain semantic model and reasoning, coupled with planning algorithm for the Airport Taxiway problem.

6.1 Introduction

6.1.1 Problem Statement

Airport taxiway operations are safety-critical. The management and planning of taxiway operations involves a variety of physical entities in the airport system, including airplanes, taxiways, runways, terminal gates, various forms of obstruction (e.g., due to maintenance) and other physical components at an airport. Decision making for taxiway operations can also be affected by factors beyond the physical environment. For instance, cyber infrastructure systems that assure communication between the control towers and airplanes, weather, cost, sensor systems, building, and traffic control operations. Therefore, a multi-domain semantic modeling and reasoning framework (see Figure 6.1) is needed to cover the range of issues affecting airport taxiway operations and to support decision making procedures during taxiway operations.

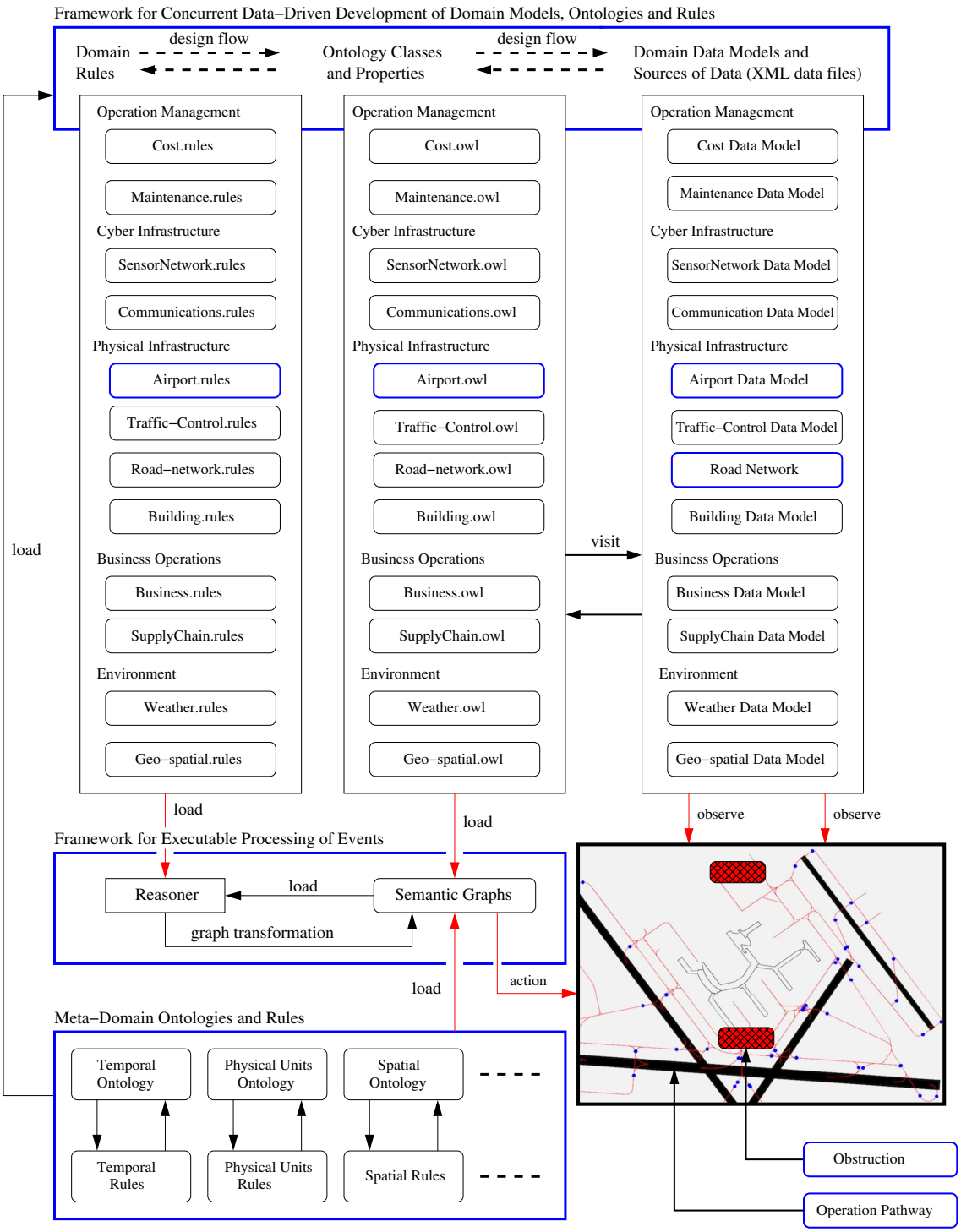


Figure 6.1: Multi-domain semantic modeling for airport taxiway operations.

Working with OptaPlanner. Figure 6.2 shows the input and output view for airport taxiway planning and scheduling with OptaPlanner.

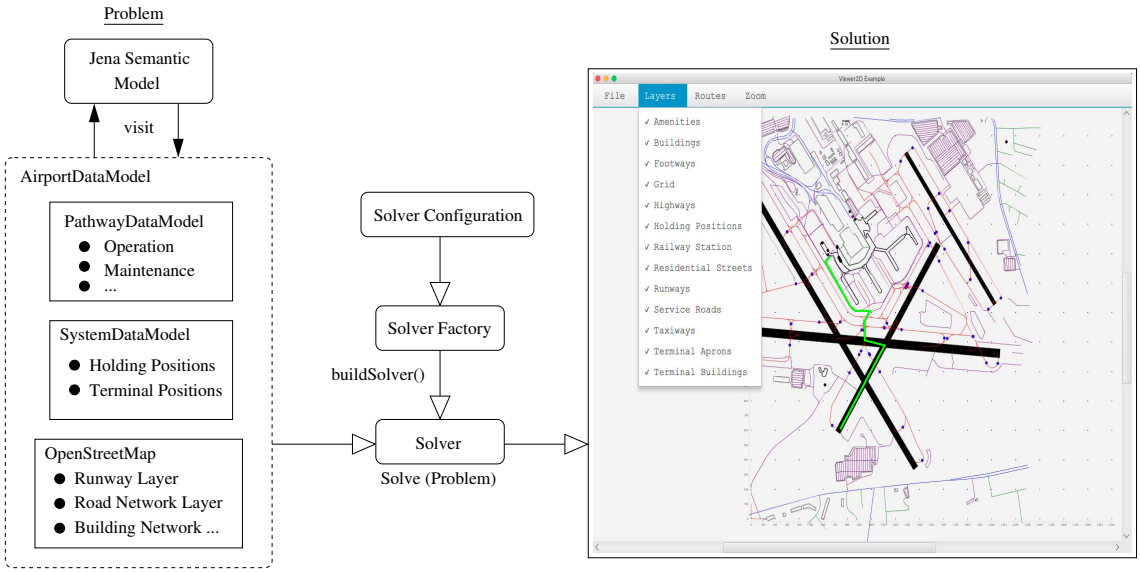


Figure 6.2: Overview of OptaPlanner input/output for airport taxiway operations.

While the types and formulations of data vary from one problem domain to another, the software integration adopts the same technologies. The left-hand side of Figure 6.2 shows the interaction between the Jena Semantic Model and Airport Data Model. The Airport Data Model is a composition of various lower-level data models including: *Pathway Data Model, System Data Model,* and *Open Street Map*. Following the integration and interaction with the semantic models, the filtered data will be passed to OptaPlanner where the middle section of the Figure shows the software configuration on OptaPlanner. After the optimization is complete, corresponding optimized solutions for when a plane shall take-off can be plotted on GUI. The two-dimensional optimized route visualization allow us to see how plane can leave from a terminal to a runway, then take-off safely.

6.1.2 Literature Review

Ground operation activities such as airplane take off, landing and taxiing to/from gates and runways [123] are safety critical. The temporal aspects of these operations require scheduling – each individual flight segment has notions of time instant (e.g., when to land and take off) and time interval (e.g., travel duration, landing duration). Beasley et al. (2000) [19] studied aircraft landing criteria by separating them with landing of a plane and its all successive planes since each airplane has a predetermined time window. Shumsky (1995) [141] also pointed out that the airplane take-off time prediction is one of the major forecast errors for the air traffic systems. And finally, Cheng et al. (2001) [27] strengthens the importance of high precision performance of taxi control to reduce tight time margins needed to clear airplanes on taxiways to cross over active runways.

The complexity of understanding safety issues in an apron operations requires a multi-domain view to visualize and make decisions with a wide range of consideration. And semantic modeling helps to provide meanings and relationships among the physical entities. Cognitive approach is also used to understand airport security situation [51], where situation awareness ontology - situation theory ontology (STO) - is proposed and agent-based architecture is conduced to distribute information among the system. Brassel et al. (2020) [25] used LiDAR semantic segmentation technique to support a simulation-based approach with generated synthetic training data for the apron operations. This can provide labeled point data and capture related semantic information to help decision making in complex situations. In addition, the acronym I-BIM (Infrastructure Building

Information Modeling) is implemented to build a 3-D parametric airport visualization model where it handles airport pavement management and schedules maintenance operations [1]. This is supported by the contextual semantic of BIM models, which features and parameters are defined within objects.

State-of-the-Art Airport Operations Research. Tofail et al. (2020) [148] provided an overview on the APMS for airport pavement maintenance and rehabilitation; Cui et al. (2020) [37] created a three-phase theory for traffic flow management using *Tianjin Binhai International Airport* as a case study which includes: (a) building taxiway flow based on ADS-B historical data through CTM (cell transmission model), (b) simulation of phase change process on MatLab, and (c) conducting error analysis based on simulation results with respect to real data; Salihu et al. (2021) [133] used a discrete event simulation (DES) and cost model to analyze congestion on taxiing operations, and proposed an equivalent annual cost (EAC) estimate for e-tractors which reduces CO_2 emissions during surface operations. Ng et al. (2018) [110] strengthened the importance of using meta-heuristics approach for optimization to assure time and solution quality. In this review study, dynamic or stochastic modeling is also mentioned as emerging methodology among the research community due to its modeling robustness over the years. Sabic et al. (2021) [170] conducted airside optimization measures to assess airport complexity and decision making through through combination of CAST, an airspace simulation software, and prediction models; Brassel et al. (2020) [25] used a 3D simulation-based approach for generating synthetic training data with LiDAR scans and semantic segmentation of apron operations which is helpful to controllers and deal with complex situations; Golda et al. (2021) [67]

Study	Topic	Methodology
Adacher et al. (2018) [3]	Routing and scheduling for ground operations	Alternative graph modeling through formulating two objective functions for minimizing taxiing delays and pollution emission
Balakrishnan and Jung (2012) [15]	Benefits of planning aircraft taxi routes	Integer Programming (IP) to optimize pathway with respect to control points
Beasley et al. (2000) [19]	Schedule landings for single and multiple runway scenarios	Mixed-Integer zero-one formulation and supported with LP relaxation to formulate objective functions
Bertsimas and Patterson (1998) [22]	Enroute capacities	Considering airport enroute runway capacity and Solving large scale IP problems; Controlling aircraft flow by adjusting release time or their speed
Brassel et al. (2020) [25]	Complex situations for apron operations	3D Simulation-based approach; generate synthetic training data with LiDAR and semantic segmentation
Cui et al. (2020) [37]	Traffic flow management	Three-phase theory: taxiway flow, simulation of phase change process, and error analysis
Das et al. (2020) [39]	Gate assignment problem (GAP)	A review paper on analyzing formulation types, objective classification and solution method categorization
Gołda et al. (2021) [67]	Improve airport operation process efficiency	decision-making model supported by simulation tool which the latter is based on genetic algorithm that minimizes delays in take-offs, landings and separation times
Jacquillat and Odoni (2018) [84]	Manage airport demand and capacity	Synthesis of three-major operational and managerial drivers of for airport systems performance: airport capacity, airport operations and flight scheduling
Lei et al. (2020) [96]	Runway capacity evaluation	Distinguished between five runway capacity models: run-slip structure, parallel runway capacity, near-parallel runway, cross-runway capacity, single and parallel runway;

Table 6.1: Summary of literature on airport operations (part 1/2).

Study	Topic	Methodology
Ng et al. (2018) [110]	Optimization of time and solution quality	A review study on the importance of meta-heuristic approach; shown the growing trend and robustness of using dynamic and stochastic modeling.
Salihu et al. (2021) [133]	Congestion of taxiing operations	Discrete Event Simulation (DES).
Šabić et al. (2021) [170]	Airside optimization measures and assessment for airport complexity and decision making	Integrated CAST and prediction models
Scala et al. (2021) [135]	Airport capacity management under uncertainty	A synthesized optimization and simulation framework to reduce conflicts through tuning the key modeling parameters
Shone et al. (2021) [140]	Stochastic modeling applications for air traffic management	A review paper on analyzing demand, capacity and air traffic congestion modeling; current OR approaches include: stochastic optimal control, analytically queuing theory, stochastic integer programming and robust optimization
Tang et al. (2019) [146]	Prediction of collisions and safety-critical events on ramp areas during surface operations	Simulation and spatial-temporal modeling framework using (1) historical data for stochastic modeling and (2) four-scenario analysis for spatial-temporal simulations
Yin et al. (2022) [167]	Effective airport network operations with reconfiguration of O-D problems	Joint apron-runway assignment problem

Table 6.2: Summary of literature on airport operations (part 2/2).

proposed a genetic algorithm-based simulation tool to support the decision making model which evaluates airport processes efficiency. The evaluation is based on minimizing delays from aircraft take-offs, landings and separation times, maximizing airport capacity and plane positions availability, optimizing taxiways and runways as well as guaranteeing operations safety; Jacquillat and Odoni (2018) [84] proposed an integration of three major operational and managerial reasons to better manage airport demand and capacity and those are airport capacity, airport operations and flight scheduling. A tabulated view of the literature can be found in Tables 6.1 and 6.2.

Mitigating Schedule Conflicts in Real-Time: For schedule conflicts mitigation, traditional methods often adopt mathematical approaches to formulate urban problems. For instance, Wang et al. (2017) [156] optimized train schedule and plans with demand analysis and line planning using mixed-integer linear programming (MILP); and Niu et al. (2013) [112] utilized binary integer programming model and nonlinear optimization model for passenger train timetable. In addition, GIS-based approach has been largely used for spatial planning. For instance, Bansal (2011) [17] identified and resolved time-space conflicts for construction management through planning for different activities with GIS approach. Our proposed approach differs from traditional approaches in its use of semantic modeling and reasoning coupled to planning. The real-time planning tool supports constraints solving for optimization where scheduling conflicts can be embedded as part of the temporal constraints within the planning architecture; then, the semantic modeling supports decision making from a high-level situational awareness through the combination of ontologies, rules and data models.

6.1.3 Scope and Objectives

This section introduces the scope and objectives for the airplane taxiway problem. The integrated knowledge-based modeling and the real-time planning provides further insights on taxiway planning decision-makings. Unlike traditional taxiway operations, this conceptual framework is capable of generating possible taxiing options for each airplane from the planning architecture and providing knowledge definitions behind the physical models and entities. The interactions between semantic models and real-time planning can capture failures in a system, take actions based on the failures, and assign new possible plans.

The main objectives of this case study are to understand the interaction between two sides and how they help modeling safety for the taxiway operations. The efficiency of airplane take-off is tremendously related to safety and effective planning. Effective scheduling for airplanes is only possible when safeties are assured. In the extent of semantic modeling, our goal is to understand how will the multi-domain semantic modeling help airplane taxiing operations and how it transforms the semantic graphs to drive new knowledge for further (downstream) decision-making. Additionally, the planning tool can dynamically adjust its current taxiing pathway options based on failures and conflicts detected on the semantic side.

The scope of this study is as follows: Section 6.2 covers the integrated framework and solution procedure; Section 6.3 explains how to model taxiway operations; Section 6.4 discusses several strategies of taxiway operations; Section 6.5 gives a detail view of how to

use the multi-domain semantic modeling for the evacuation problem; Section 6.6 provides the formulation of the planning problem; Section 6.7 illustrates details of the planning problem in OptaPlanner, class diagram, embedded constraints and sample results; Section 6.8 further explains and discusses the results obtained from this framework.

6.2 Integrated Model and Solution Procedure

Solution Procedure. A three-step solution procedure for the airport taxiway operations planning is shown in Figure 6.3. These steps involve knowing the positions and schedules of airplanes, scheduled taxiways of each airplane and obstructions that occur on taxiways (and/or runways). Here, step 1 is proposed to generate a set of reasonable pathways for each airplane based on the holding positions of airplanes and locations and geometries of obstructions. Step 2 focuses on both spatial and temporal aspects of the various physical entities within the operations. This means that the scheduled departure times of each airplane are considered with the determined airplane pathways from the previous steps. In addition, it also considers the time intervals of obstructions with respect to their sizes and locations. The derived information extracted from the previous two steps will help to plan and evaluate alternative options of pathways for airplanes, which is the step 3. This step evaluates the temporal and spatial relationships among airplanes, taxiways and obstructions using their positions, geometries and operating schedules. Based on the evaluation, the planning tool chooses the best solution for each airplane to take-off safely.

Integrated Framework. Figure 6.4 shows the integrated framework that was explained in Figure 6.3. Three steps are matched with the two figures. The integrated framework

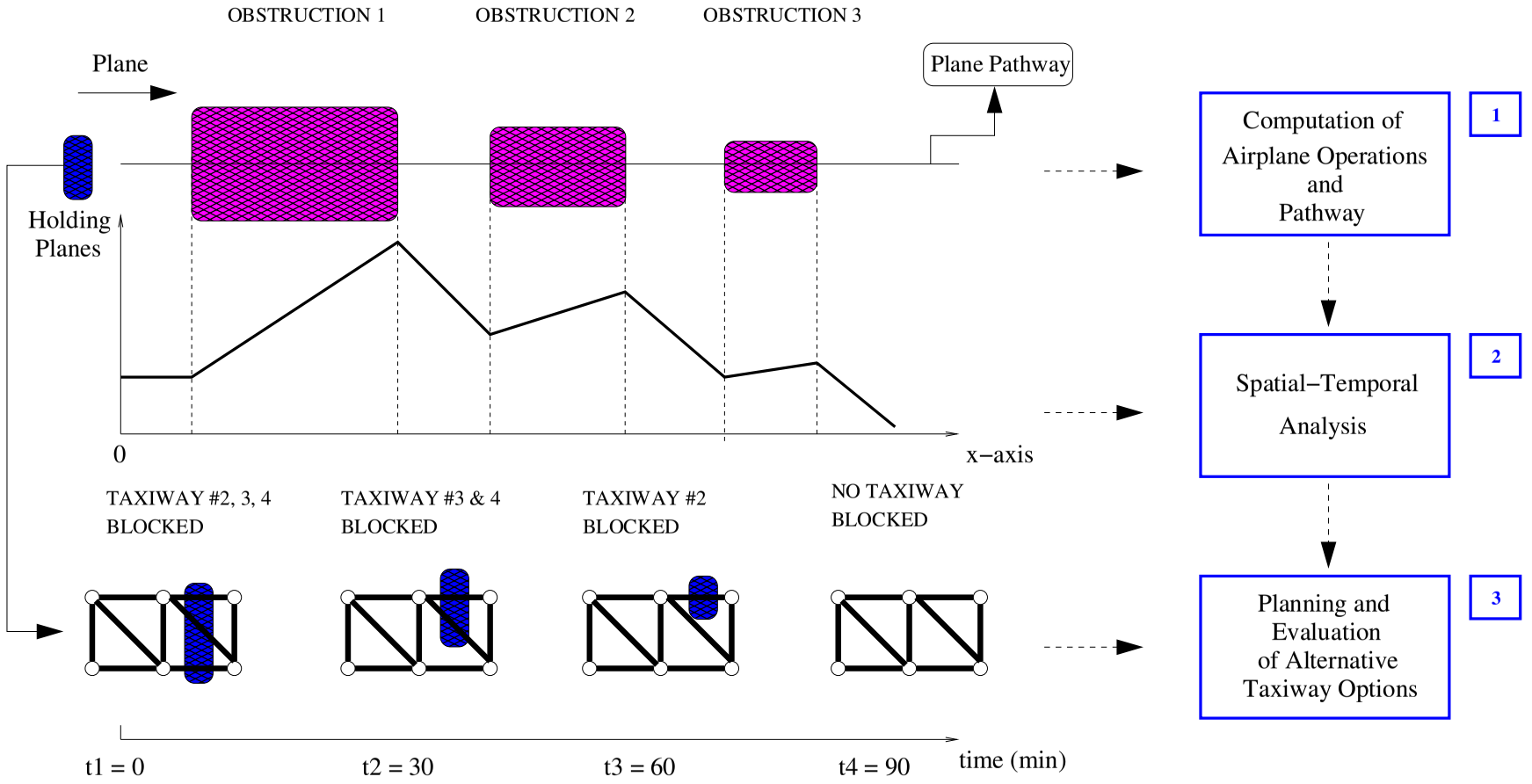


Figure 6.3: Multi-domain modeling: plane, obstruction and taxiways.

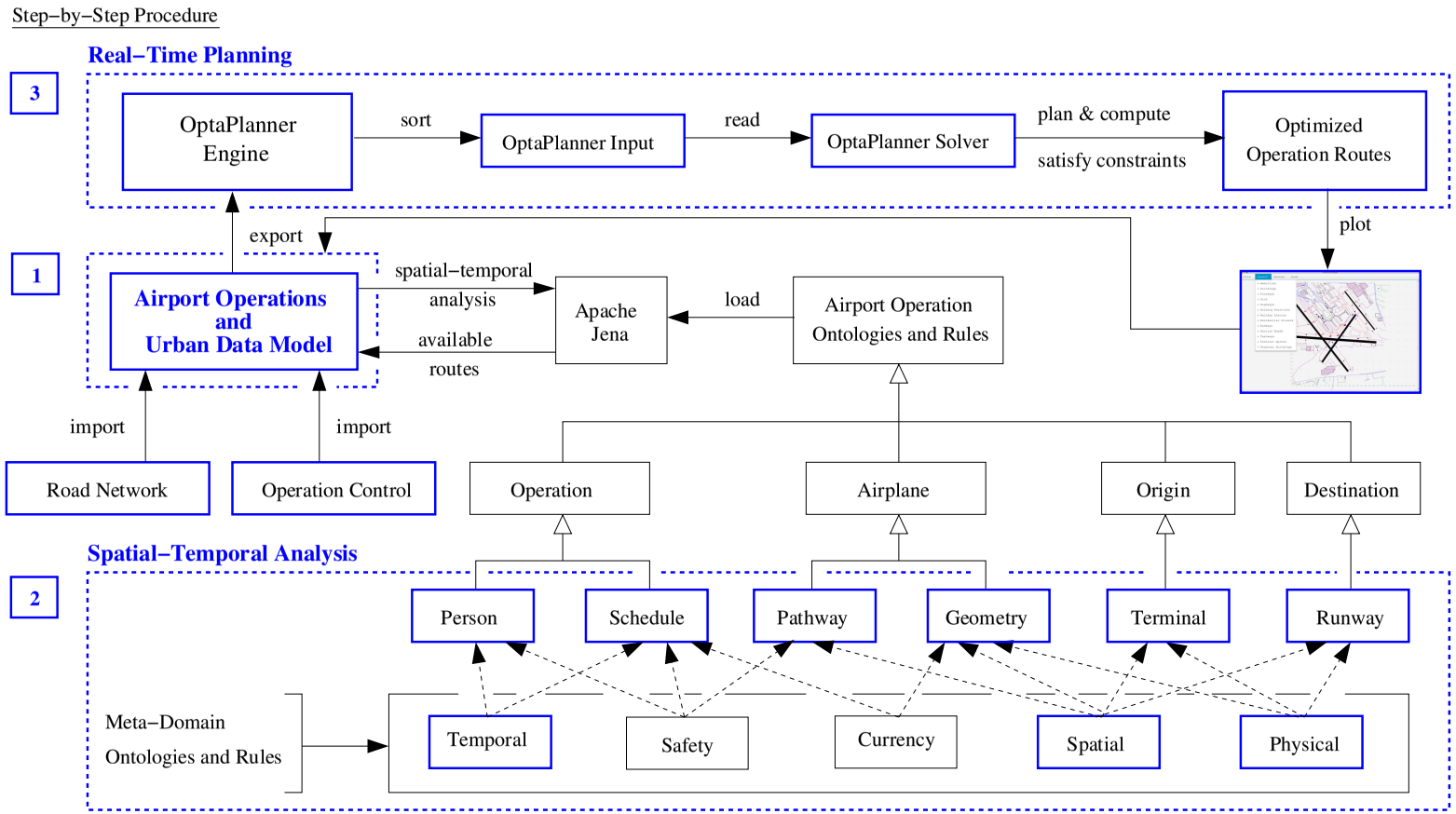


Figure 6.4: Integrated framework and solution procedure.

here shows the details of how does the various data sources are imported, how does the multi-domain semantic knowledge is embedded as part of the spatial-temporal analysis and how will the real-time planning is conducted using the information from the previous two steps. The *Airport Operations and Urban Data Model* shown in the mid-left of the figure indicates a data model that imports the information of road network and operation control. Then, after exchanging information with the *Apache Jena* which stores the knowledge-based semantic graph, the populated individuals and their associated properties will be merged with the domain data model. Following with this step, the information blended with the semantic graph is returned back to the *Airport Operations and Urban Data Model* which filters the useful information for the purposes of planning and exports it to the *OptaPlanner Engine*. Finally, the planning tool takes the input information from the data model, sort them as a readable format in the planner, initiates the constraints solver and computes and outputs the optimized results.

Model Dependencies. Figure 6.5 illustrates the model dependencies for the airport taxiway operations modeling framework. The *Jena Semantic Model* loads the Jena rules and ontologies and save them as the initial version of the semantic graph which has not individuals. The *Airport Data Model* extracts and stores three data sources. Those are the *System Data Model*, the *Pathway Data Model* and the *Open Street Map Data Model*. The robustness of having a domain data model is to take data from various places and use the combination of these data for further analysis. It also helps to blend these data together and reason for new knowledge that can be added to the semantic graphs. The domain data model also hosts a visitor that allows Jena model to visit, grab the data and mix them with

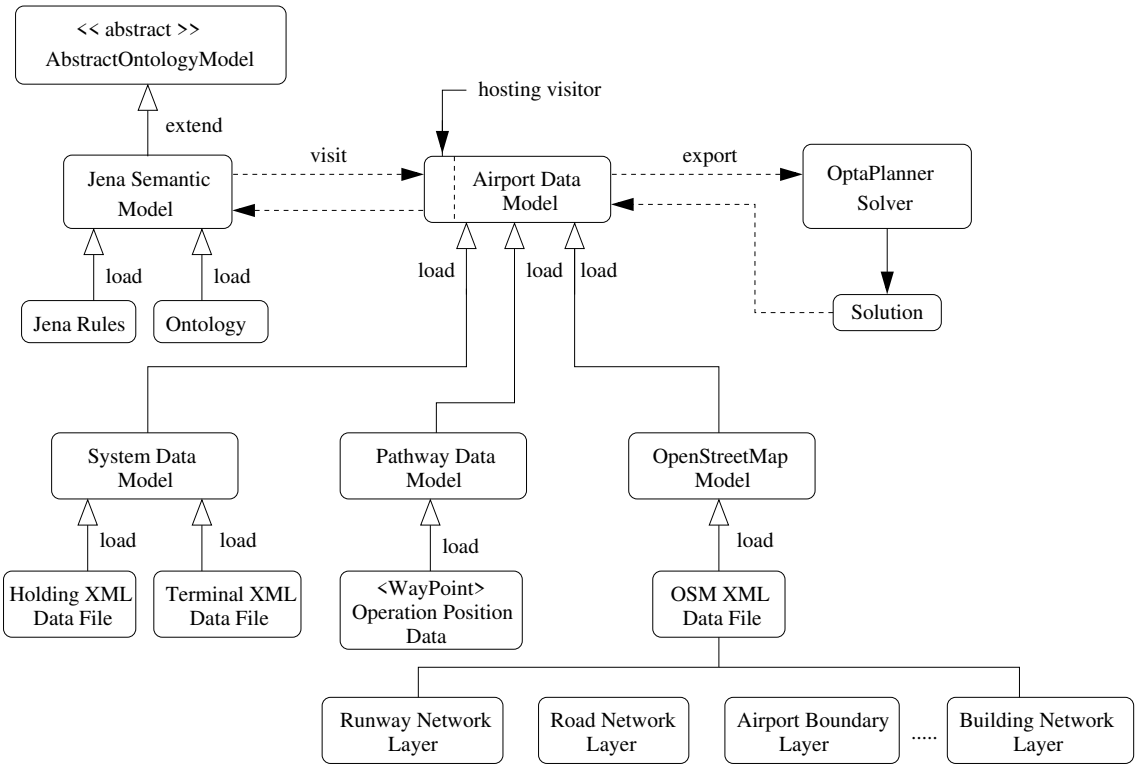


Figure 6.5: Model Dependencies for Airport Taxiway Operations.

the semantic model. On the right hand side, it shows the exported data are sent to the constraint solver and retrieve corresponding planning solutions.

6.3 Modeling Airport Taxiway Operations and Airport Operational Safety

6.3.1 Modeling Airport Taxiway Operations

Modeling airport taxiway operations is challenging because of the complexity raised in temporal, spatial and planning concerns. Different ways of modeling airport taxiway operations are conducted in the scientific community. Surface operations planning for optimizing taxi routes with various sets of control points on the airport surface is conducted with an example for the Dallas-Fort Worth International Airport [16]. The development of an effective safety management system is also important to encounter the coordination and cooperation among various actors at an airport for safety surface operations [161]. Therefore, an integrated risk assessment framework is built to cover the concerns of all relevant stakeholders. Studies also cover issues in fuel consumption and emissions during taxiing with airplane position data [111], improving the communication between taxiway and airborne schedules [134], facing large-scale disruption (heavy snowfall) [85], etc. The importance of modeling taxiway operations mainly focus on scheduling on airplanes and vehicles traffic, and the primary concern of taxiway operations is safety issues at the airport systems.

6.3.2 Modeling Airport Operational Safety

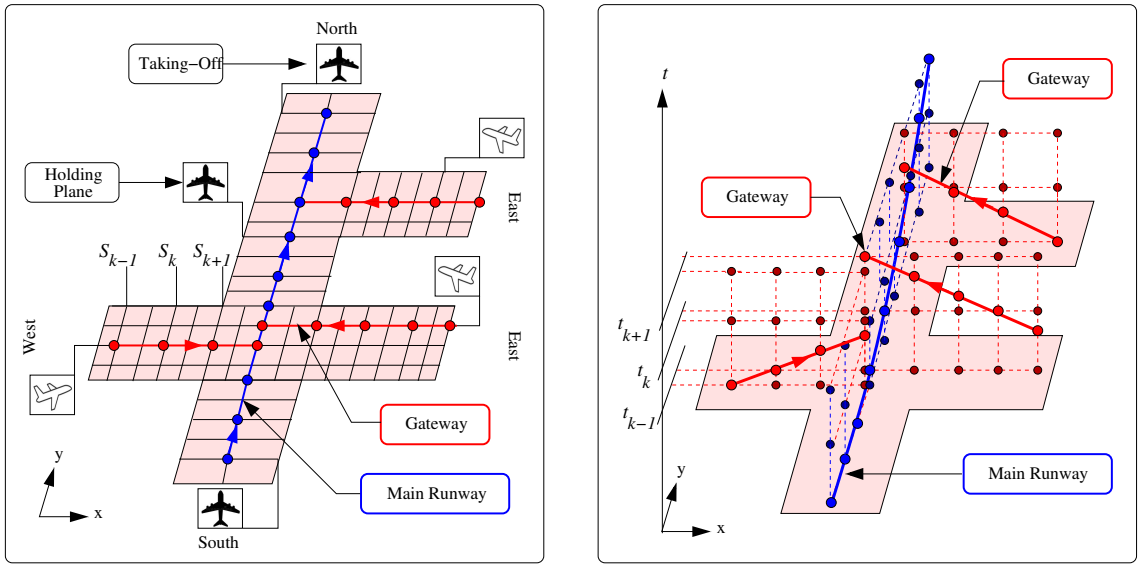
Modeling airport operational safety is a complex procedure since there are many components involved simultaneously and they should communicate efficiently and precisely in order to avoid failures of any kind at any situation. The ultimate goal here is to

minimize the risk and safety related incidents and maximize the system capacity in order to have more airplanes scheduled within a shorter period of time [109]. For taxi safety, the Federal Aviation Administration (FAA) generated a commonly used safe operating practices [47]. The air carrier threat and error management response factors include situational awareness, see and be seen, expectation bias, distraction, haste and fatigue which are across different phases in taxiing - pre-taxi, taxi-out, taxi-in and any-time. The FAA has also been using an airport capacity tool for safety modeling (`runwaySimulator`) which is designed to evaluate capacity planning of an airport and how to improve capacity through new infrastructure and flight procedure [48]. Modeling operational safety involves different categories of safety modeling. This can be caused by airplanes, air traffic controls, collision risk, third-party risk and human errors [109].

6.4 Simplified Strategies for Taxiway Operations

6.4.1 Framework for Spatio-Temporal Analysis.

Figure 6.6 shows the spatial and temporal analysis of airport taxiway operations.



(a). Taxiway Operations Problem. **(b). Spatial–Temporal Analysis**

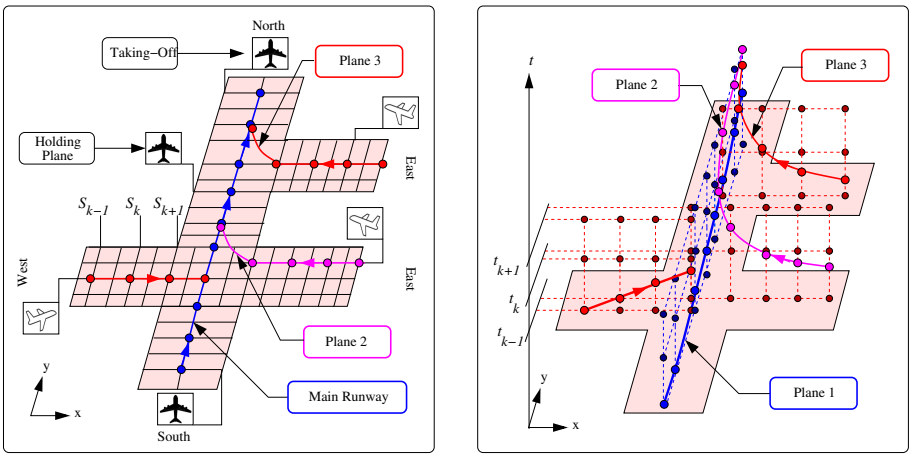
Figure 6.6: Framework for spatio-temporal analysis.

Assuming several planes are waiting at the terminal and/or holding positions, the relationship of the common runway of these planes, positions of these planes, and schedules of them are significantly important to the safety operations in an airport. The right-hand-side of the image indicates a plane on the main runway and it's taking off using the blue trajectory. In addition, three red trajectories show where the other planes are waiting and when they will be getting on the main runway for taking off afterwards.

6.4.2 Three Airport Taxiway Strategies.

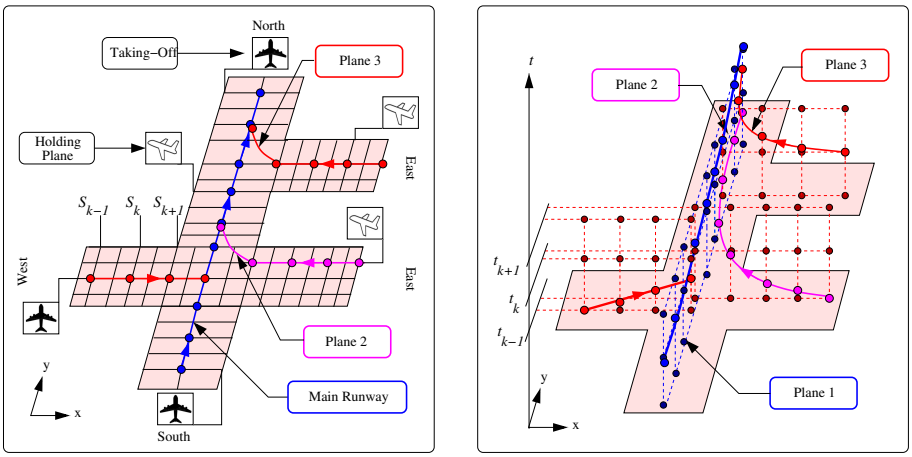
State-of-the-art strategies for taxiway operations at airport are based on controlling accesses to separate between airplanes or between airplanes and vehicular traffic [142]. This is relevant to the airplane damage, delay costs to passengers, and other financial loses. Some prevention strategies are also mentioned including training and testing for operators, staffs and pilots, and controllers which the latter assure safe and efficient operations for vehicle and airplane on the aprons and taxiways.

Figures 6.7 through 6.9 show three possible strategies of multiple planes that is sharing a common runway and how various scheduling can assure safety operations at an airport. The essence of assuring safety is to make sure that the trajectories on three-dimensional images in these figures are not intersected. Three strategies differed by the order for when each plane will leave and how their holding positions matter when it comes to optimized scheduling for plane take-offs. The first strategy (see Figure 6.7) shows that the plane 1 which is on the main runway currently but location is much further away to the end of the runway compare to other planes shall take-off first, then following plane 3 and plane 2. The second strategy (see Figure 6.8) is that if plane 1 is further away from the end of runway take-off, then plane 2 can leave before plane 3 and plane 1; Similarly, strategy three (Figure 6.9) shows an order of plane 3, plane 2, then plane 1 based on its locations, and maybe even their scheduled departure times.



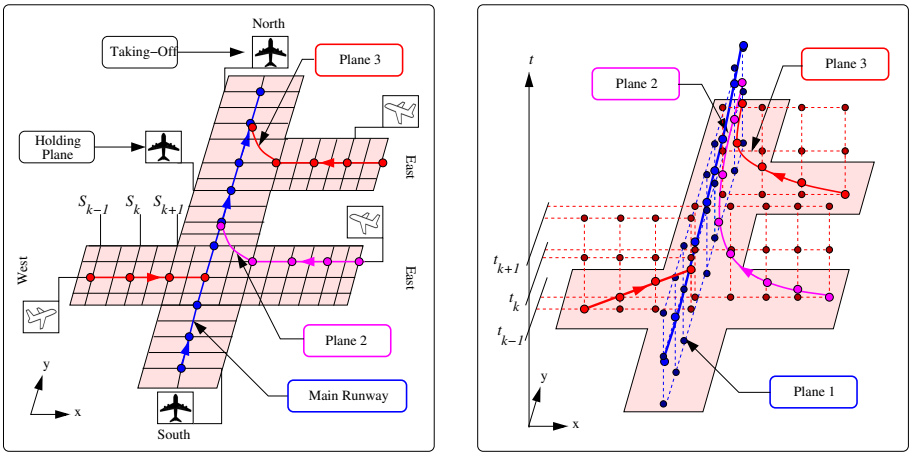
(a). Taxiway Operations Problem. **(b). Spatial–Temporal Analysis**

Figure 6.7: Taxiway strategy 1: plane 1, plane 3, and plane 2.



(a). Taxiway Operations Problem. **(b). Spatial–Temporal Analysis**

Figure 6.8: Taxiway strategy 2: plane 2, plane 3, and plane 1



(a). Taxiway Operations Problem. **(b). Spatial–Temporal Analysis**

Figure 6.9: Taxiway strategy 3: plane 3, plane 2, and plane 1

6.5 Multi-domain Semantic Model and Event-based Reasoning

Multi-domain semantic modeling for airport taxiway operations requires to use a range of knowledge that covers schedules, holding positions, terminal locations, taxiway operations, maintenance, runway conditions and other meta-domain knowledge, such as space, time, graph, etc. This section explains the framework of airport taxiway operations with semantic models. Figure 6.1 shows the concurrent data driven development of multiple domain models, ontologies and their associated rule sets. In the figure, domain-specific ontologies are placed on top of the figure where domain specific rules, ontologies, and data sources are inserted jointly to capture changes in the semantic graphs shown in the middle the figure. Semantic graphs also captures the transformation with Jena reasoners and take actions to the problem domain using the imported rules, ontology files, and data models. At the same time, the meta-domain ontologies and rules are imported side-by-side with the domain-specific knowledge to assure that the knowledge representation of temporal, spatial, and physical unit are consistently used throughout the entire modeling framework.

6.5.1 Generation of Airport Operation Individuals in Semantic Graphs

Figure 6.5 shows the data-driven approach for generating the urban domain individuals in semantic graphs. Firstly, the various types of data (System Data Model, Pathway Data Model, and Open Street Map Data Model) are imported through the Java Object data models with JAXB. Table 6.3 contains a list of data that are used in this approach as well

Data	Description	Format
Holding Position	A list of nodes defined with System Data Model	XML
Terminal Position	A list of nodes defined with System Data Model	XML
Building Network	A list of nodes defined with Open Street Map	OSM
Runway Network Layer	Lists of nodes, ways, and relations defined with Open Street Map	OSM
Road Network Layer	Lists of nodes, ways, and relations defined with Open Street Map	OSM
Operation Pathway Simulation	A list of <WayPoint> are defined with the Pathway Data Model	WKT
Airplane Information	An airplane has a departure time and a set-off location that is based on *Runway* location	OptaPlanner

Table 6.3: Data types used in the Airport Case Study.

as the explanations and formats of data sources. Secondly, the ontologies and Jena rules are loaded into the Jena Semantic Model which creates instances of the relevant ontologies (OWL) through visiting data models and obtaining information on individuals of a specific urban domain (i.e., wildfire evacuation, taxiway operations). Then, after the individuals and their information being transferred to Jena Semantic Model from Airport Data Model, corresponding ontology instances are created, and the Jena Rules are applied for semantic graph transformation (both object and data properties are assigned and updated).

6.5.2 Airport Taxiway Operations Ontology and Jena Rules

A simplified airport taxiway operation ontology with its classes, object properties and data properties is shown in Figure 6.10. The class include airplane, taxiway, runway, maintenance and terminal which have object properties of, for example, hasPlane and hasRunway connecting these classes. Also, the data properties and data types of these

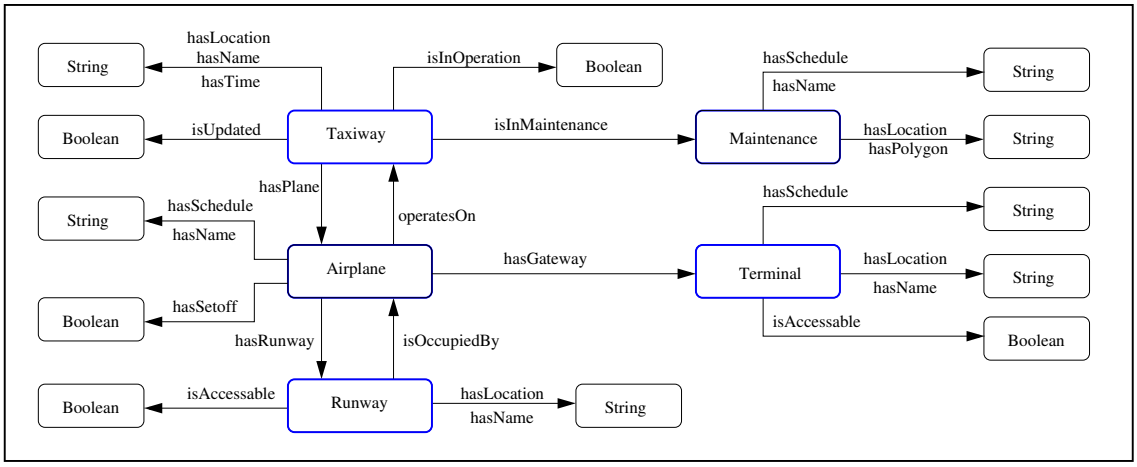


Figure 6.10: Simplified schematic for airplane, taxiway, runway, gateway, maintenance ontology classes and properties.

properties are shown around each class where these are used as triggers to indicate changes within the semantic graphs. For example, boolean dataProperty: isAccessable of class:Runway indicates whether this specific runway is operated by other planes. Change of this property can dynamically change other data properties or object properties in the semantic graph. Therefore, the updated and populated semantic graph with individuals can help with making further decisions for safety operations.

Jena rule sets of the airport taxiway operations are displayed in Figure 6.11. Rule 01 is for propagating class hierarchy relationships among all classes in the ontology. This helps to fill gaps of the class types when one is the subclass of another, the class type of the super class will be assigned to the child class; Rule 02 identifies the location and take-off time of two planes, by comparing their take-off times, the objectProperty: leavesBefore will be assigned to these plane classes. This can help the planning processes since order of plane take-off is predetermined with the semantic relationships; Rule 03 identifies the relationship of a class:Runway and a class:Airplane. If an airplane is on a runway,

then a new objectProperty: isOccupiedBy will be assigned to the runway. This fills the knowledge relationship between the airplane and runway classes; Rule 04 promotes a further calculation of the airplane and runway classes with their dataProperty: hasTakeoffTime and hasSchedule involved. With the backend function of getPlaneSchedule, an objectProperty: isCurrentlyOn can be initially assigned between a airplane and a runway class.

```
@prefix airop: <http://www.isr.umd.edu/airplane#>.
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix geom:  <http://www.isr.umd.edu/geometry#>.
@prefix xs:    <http://www.w3.org/2001/XMLSchema#>.

// Rule 01: Propagate class hierarchy relationships ....

[ rdfs01: (?x rdfs:subClassOf ?y), notEqual(?x,?y),
      (?a rdf:type ?x) -> (?a rdf:type ?y)]

// Rule 02: AirplaneX leavesBefore AirplaneY

[ Plane02: (?p1 rdf:type airop:Airplane) (?p1 airop:hasLocation ?p1l)
    (?p1 airop:hasTakeoffTime ?p1t) (?p2 rdf:type airop:Airplane)
    (?p2 airop:hasLocation ?p2l) (?p2 airop:hasTakeoffTime ?p2t)
    lessthan(?p1t, ?p2t, "true"^^ xs:boolean) -> (?p1 airop:leavesBefore ?p2)
    print(?p1 airop:leavesBefore ?p2) ]

// Rule 03: Runway isOccupiedBy Airplane and RunwayX isOccupied?

[ Plane03: (?p1 rdf:type airop:Airplane) (?r1 rdf:type airop:Runway)
    (?p1 airop:isCurrentlyOn ?r1) -> (?r1 airop:isOccupiedBy ?p1)
    (?r1 airop:isOccupied "true"^^ xs:boolean) ]

// Rule 04: AirplaneX isOn RunwayY at TimeT

[ Plane04: (?p1 rdf:type airop:Airplane) (?r1 rdf:type airop:Runway)
    (?p1 airop:hasTakeoffTime ?pt1) (?r1 airop:hasSchedule ?rs1)
    getPlaneSchedule(?pt1,?rs1,?sch) equal(?sch, "true"^^xs:boolean)
    -> (?p1 airop:isCurrentlyOn ?r1)(?p1 airop:isOnTime "true"^^ xs:boolean)]
```

Figure 6.11: Abbreviated list of Jena Rules for airport taxiway operations.

6.5.3 Software Problem Setup

Figure 6.4 shows the integrated framework of multi-domain semantic modeling and the constraint solver - OptaPlanner. The Figure shows the framework in a three-step process labeled on the left-side of the figure. Step 1 - airport operations simulation and data model through extracting various data sources into the current model, step 2 - Real-Time planning with OptaPlanner, and step 3 - Spatial-Temporal analysis with the multi-domain semantic modeling approach. In this section, the software setup of the modeling framework is added and explained in details.

```
String SDMinputFile01 = "data/umd-airport-terminals.xml" ;
String SDMinputFile02 = "data/umd-airplane-holdings.xml" ;

SystemDataModel sdm01 = new SystemDataModel();
sdm01.getData ( SDMinputFile01 );

SystemDataModel sdm02 = new SystemDataModel();
sdm02.getData ( SDMinputFile02 );

List<Node> terminalPositions   = sdm01.getNodes();
List<Node> holdingPositions = sdm02.getNodes();

AirportOperationDataModel aodm01 = new AirportOperationDataModel();
aodm01.setName("Airport Operation Demo");
aodm01.setTerminals(terminalPositions);
aodm01.setHoldings(holdingPositions);
```

Figure 6.12: Fragment of code to set up the terminal and holding positions with the airport operation data model.

Figure 6.12 shows the previously obtained house and shelter location data, stored in *XML* format, are being added to the SystemDataModel. Then the getNodes() function is used to extract the actual <List> of <Node> individuals that store the coordinates of

terminal and holding positions. The AirportOperationDataModel class is then created and the corresponding List<Node> are stored through setTerminals() and setHoldings() classes.

Figure 6.13 indicates how an Open Street Map file is being handled in Java and how various layers of data are separated and used. Once the OpenStreetMapModel class is created, the corresponding data can be passed via using getData() method. In order to separate layers of information, the CompositeHierarchy class is used to define and obtain different information from OSM. The getWorkspace() method in OpenStreetMapModel can extract the corresponding information using its keyword (i.e., "runway") and then save it under the CompositeHierarchy class.

```
String OSMinputFile01 = "data/umd-bwiairport.osm";
OpenStreetMapModel osm = new OpenStreetMapModel();
osm.getData( OSMinputFile01 );
osm.shrinkFactor( new Quantity( 10.0 ) );
osm.smallDotSize( new Quantity( 30.0 ) );

CompositeHierarchy highwayLayer  = osm.getWorkspace( "highway"     );
CompositeHierarchy runwayLayer   = osm.getWorkspace( "runway"      );
CompositeHierarchy taxiwayLayer  = osm.getWorkspace( "taxiway"     );
CompositeHierarchy serviceLayer  = osm.getWorkspace( "service"     );
CompositeHierarchy streetLayer   = osm.getWorkspace( "residential" );
CompositeHierarchy buildingLayer = osm.getWorkspace( "building"    );
CompositeHierarchy amenityLayer  = osm.getWorkspace( "amenity"     );
CompositeHierarchy apronLayer    = osm.getWorkspace( "apron"       );
```

Figure 6.13: Fragment of code to extract useful information from OSM (OpenStreetMap).

Figure 6.14 shows how the semantic model of airport operation is developed as well as how the Jena rules are added and executed to transform the semantic graph. Here, JenaSemanticModel class is developed to store the domain ontology (including classes,

object and data properties) and then a corresponding visitor class is created to temporarily store the semantic graph. The AirportOpDataModel class (with object "aodm01") then accepts the corresponding visitor and populate its ontology. Finally, the rule class is fed through addRules() method in the JenaSemanticModel and using executeRules() method to complete the semantic graph transformation.

```
// Create semantic model in Apache Jena ...

JenaSemanticModel airportSM = new
             JenaSemanticModel("Airport Operation Semantic Model");
airportSM.loadOntology ("file:ontology/umd-airport-operation.owl");

// Populate semantic graph by visiting airport data model ...

AirportOprDataModelJenaVisitor airopSM_visitor =
                                new AirportOprDataModelJenaVisitor();
airopSM_visitor.setPassword("airopSMV");
airopSM_visitor.addSemanticModel( airportSM );
aodm01.accept ( airopSM_visitor );

// Add rules to reasoner for airport operations ...

airportSM.addRules ( "rules/umd-airop.rules" );
airportSM.executeRules();
```

Figure 6.14: Assemble the airport operations semantic model, populate ontologies, and execute graph transformation.

After all the previous steps of storing data and transforming the semantic graphs, Figure 6.15 shows how the adjusted data are exported to the OptaPlanner software. After the corresponding output from the Airport Data Model and Simulation are transferred, the OptaPlanner starts optimizing using these data and generate plans that satisfy corresponding safety constraints.

```
aodm01.exportToOptaPlanner();
aodm01.exportOptaTerminalPositions();
aodm01.exportOptaHoldingPositions();
aodm01.convertCSVtoXML();
```

Figure 6.15: Assemble the airport operation semantic model, populate ontologies, and execute graph transformation.

6.6 Formulation of Planning Problem

6.6.1 Use Cases and Scenarios

In this section, the use cases and corresponding scenarios of each use case is described for scheduling simple airport taxiway operations. Table 6.4 shows three use cases and corresponding scenarios for each use case. Use cases have used planes to describe the various use cases and scenarios. The real-world airplane operations on taxiways and runways can be much more complex and lengthy than the ones described below. The purpose of this section is to show the possibility of recognizing airport taxiway operations with systems engineering approach. Later, selected use cases and scenarios are adopted into OptaPlanner.

Three use cases shown in table 6.4 include planning for same departure time for planes, planning for different departure times for planes and planning when runway is under maintenance.

Use Case 1 shows when planes are scheduled to have the same departure time, how will the various scenarios can affect the outcomes of planning. Various scenarios may include

Use Case 1. Plan for Same Departure Time.
Scenario 1. The planes operate on the same pathway.
Scenario 2. The planes operate on different pathways with partially overlapping trajectories.
Scenario 3. The planes operate on different pathways without overlapping trajectories.
Use Case 2. Plan for Different Departure Time.
Scenario 1. The planes operate on the same pathway.
Scenario 2. The planes operate on different pathways with partially overlapping trajectories.
Scenario 3. The planes operate on different pathways without overlapping trajectories.
Use Case 3. Plan during Runway Maintenance.
Scenario 1. The planes take off on runways before maintenance occur.
Scenario 2. The planes take off on runways after maintenance occur.
Scenario 3. Some planes take off before maintenance occur on runways.

Table 6.4: Use Cases and Scenarios of Airport Taxiway Operations

when planes share the exact same pathway, when planes operate on different pathways but still partially overlapped, and when planes operate on completely different pathways. Clearly, scenario 3 does not need planning involved since there are no spatial intersections between the trajectories of planes, even the departure time is scheduled to be the same. Scenario 2 may or may not require planning depending on if the partially overlapped trajectories will be used at the same time. Scenario 1, however, requires planning as planes share exact same pathways and scheduled to depart at the same time.

Use Case 2 indicates a use case of planning for planes when they have different departure times. Scenario 1 says the planes will share the same pathway. This could require further planning but it depends on the difference between the departure times of planes. Scenario

2 indicates a partial overlap of the pathways. This may also need further planning as the partial overlap of pathways can result the planes to share closed intervals of times to use the pathways. Scenario 3 will not require any plannings involved.

Use Case 3 shows a different use case compare to the first two use cases. In real-world airport operations, maintenance of runways and taxiways can always happen. Certainly, considering the maintenance of airport can influence the purpose of operational plannings. Three scenarios include that when both planes take off before the maintenance occur, one plane leaves before the maintenance and one leaves after, and both takes off after the maintenance. All three scenarios will require planning as the context of maintenance always requires both spatial and temporal aspects.

6.6.2 Requirements and Constraints

The requirements and constraints are derived based on the defined use cases as well as the various scenarios. Textual requirements are written by the requirement engineers to describe the functionality of a system in an unambiguous way. Once the requirements are written, the corresponding constraints of each requirement can be extracted. This is because the mathematical constraints can be embedded into the planning software as part of the optimization process. For planning airport taxiway operations, both spatial and temporal aspects of constraints shall be defined and implemented. Spatial constraints can describe things like safety distances between airplanes, safety distances between moving objects (airplanes) with other non-moving objects (buildings, terminals, etc.), measure airplane's altitude for landing or setting-off, and so on. Temporal constraints in airplane

scheduling is significantly important as it plans out schedules of all planes and helps to adjust schedules when, for instance, delay occurs. Temporal constraints in planning problems can also be treated as resource allocations, instead the resource is the temporal instants or intervals.

6.6.3 Planning Objectives

Planning problems need planning objectives. Like any other OR problems, the objective function and mathematical constraints need to be defined ahead. For the airport taxiway operations, the objectives include efficient planning for plane departure times and safety planning based on plane schedules and terminal allocations. Instead of a mathematical problem that contains objective function, airport taxiway operations with *OptaPlanner* do not have fixed objective functions since the purpose of planning is dynamic planning. *OptaPlanner* defines planning entities and planning variables which are associated with its DRL rules, implemented as hard and soft constraints. Then the variables that are defined under these planning entities can be used to formulate set of constraints for the planning problem. In the software architecture, the planning objectives are set to fulfill all, if possible, constraints as much as possible. This is mainly because that it uses hard and soft constraints where a scoring system is provided to support and compare between various sets of solutions. Details of the problem formulation is explained in Section 6.7.

6.7 Planning Airport Taxiway Operations

6.7.1 Problem Description

The airport taxiway system contains a set of holding positions, terminals, runways, taxiways and airplanes as shown in Figure 6.16.

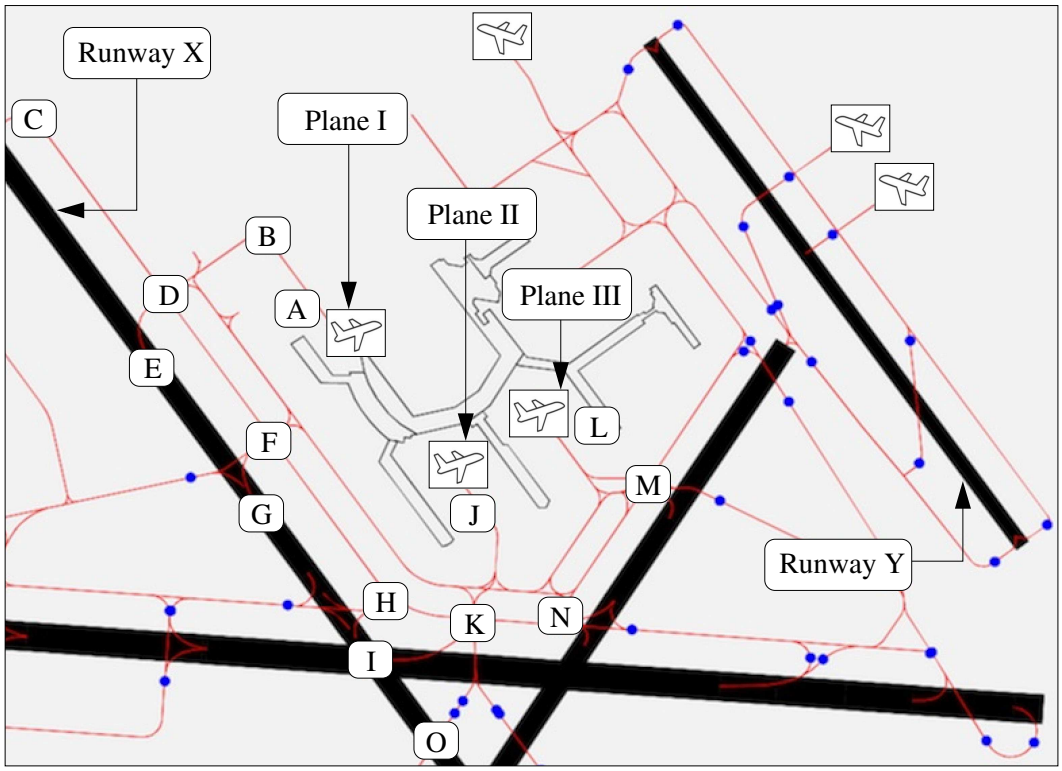


Figure 6.16: Taxiway planning on an airport network structure.

Unlike the small urban network and shortest path problem, the edges between the nodes are undirected, indicating that two-way traffic is allowed. Taxiways are shown in light red lines, runways are thick black lines, terminals are black thin lines, and the holding positions are annotated as blue dots. In order to test the feasibility of the usage of OSM data shown in the Figure with the real-time planning tool, we simplified the problem to a

fixed amount of nodes, runways, airplanes and taxiways. This helps us to understand how the integrated framework will cope with layers of information in OSM data.

6.7.2 Class Diagram

In order to design a planning model for the taxiway operations, the planning entities that represent the input data need to be defined. In this case, these are airplanes, locations, routes and time slots. The domain model helps to determine which classes are planning entities as well as what properties need to be planning variables. This process improves planning performance, simplifies constraints, and increases planning flexibility. Figure 6.17 shows details of the class diagram for the taxiway operation problem. Here, the planning entities are defined as airplanes and planning variables contain a several properties of the airplane. Those are time slot for departure time, location which stores current location of an airplane and then a route which has a list of locations defining the path that an airplane will use for taking-off.

Table 6.5 addresses hard and soft constraints for the airport taxiway operation problem. The hard constraints include a). avoid schedule conflict between two airplanes which avoids any two airplanes appear at the same place and same time, and b). airplane departure assignment which assigns online one departure time for each plane. The hard constraints must be satisfied in the solution set, otherwise the solution is deemed to be infeasible. The soft constraints include a). maximum and b) minimum capacity assignment for runways. These constraint can fluctuate at different times of a day or at different days. Soft constraint c) indicates that when a runway is under maintenance, if an airplane is

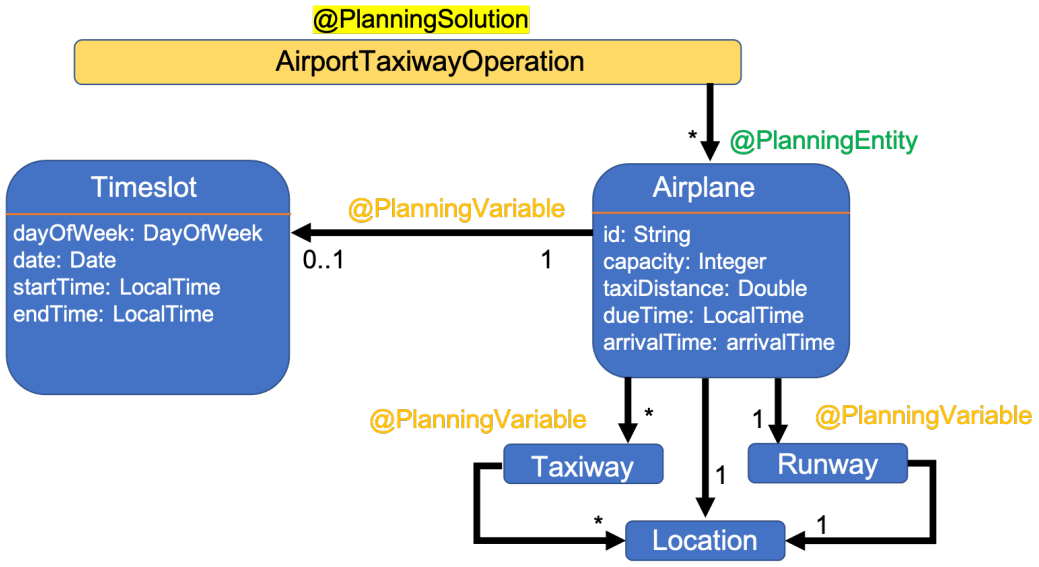


Figure 6.17: OptaPlanner class diagram for airport taxiway operations.

scheduled to use this runway, the airplane may be scheduled to take-off on a different runway (or it can wait until the maintenance is complete). The flexible constraints are treated as soft constraints here. Different set of constraints used in a planning problem can change the planning results drastically. Because of the dynamic nature of airport operations, the best solution is not always the only solution. This also means the results of an exact same setting of a problem can be very different between simulations.

6.7.3 Drool scoring rules for taxiway operations.

Some of the hard and soft constraints from Table 6.5 are explained in details in this section, which *OptaPlanner* uses for planning purposes.

Here, the hard and soft constraints are implemented as part of the score calculation which is embedded with the planning software. Equation 6.1 shows the hard constraint

Hard constraints.
a). Avoid schedule conflict: two airplanes can not be assigned to the same place (location) at the same time (time slot).
b). Airplane departure assignment: each airplane can only have one departure time (timeslot) assigned.
Soft constraints.
c). Minimize traversed pathway during airplanes take-off: total distance between gates and runways.
d). Airplane scheduled delay penalty: Comparing airplane scheduled departure time and actual departure time.
e). Runway and airplane coordination: Comparing runway ready time and airplane arrival time to runway.

Table 6.5: Hard and soft constraints for the airport taxiway operations problem.

score, denoted as $HS_{overall}$, where it demonstrate the feasible solutions from the planning solver. The overall score has to be greater than or equal to zero in order to demonstrate that the result is a feasible solution of the planning problem. Equation 6.2 shows the soft constraint score, denoted as $SS_{overall}$, where it demonstrate the score of a feasible solution. This is implemented to show the difference between various feasible solutions and to demonstrate which is better.

$$HS_{overall} = \sum_{i=1}^{n}(LocationCapacity_i - LocationAirplaneNum_i)+ \sum_{j=1}^{m}(TimeslotStartTime_j - AirplaneReadyTime_j)+ \sum_{k=1}^{m}(TimeslotEndTime_k - AirplaneDepartTime_k) \quad (6.1)$$

$$SS_{overall} = \sum_{i=1}^{n}(AirplaneDueTime_i - AirplaneDepartTime_i)+$$
$$(-1) * \sum_{j=1}^{m}(AirplaneTotalDistance_j + AirplanePreviousDistance_j)+$$
$$\sum_{k=1}^{k}(AirplaneRunwayArrivalTime_k - RunwayReadyTime_k) \quad (6.2)$$

Constraint and Planning View for Taxiway Operations. Figure 6.18 shows the constraint and planning view for the taxiway operation problem (hard constraints). As mentioned above, the hard constraints are set up to consider holding position capacity where it is limited as only one airplane appears at a holding position at a time. This way, the planning problem can focus on allocating airplanes to runways and avoid schedule conflicts since this rule mitigates collision happens at holding positions. Figure 6.19 shows the sample soft constraints, which it considers the early departure rule and shortest path rule over the airplane operations. These rules are designed to check the overall performances of the modeling system and to compare feasible solutions through soft scores via identifying the best solution.

Planning Constraint Details. Rule 01 in Figure 6.20 shows the first hard constraint where no more than one airplane can occupy a location at the same time. In order to implement this, the hard rule has defined that each location cannot hold more airplanes than its capacity, which is set to be one. This rule needs to be strictly implemented as a hard constraint because of the safety operations at airport do not allow two airplanes to appear at the same holding position, for example, at the same time. This won't be true

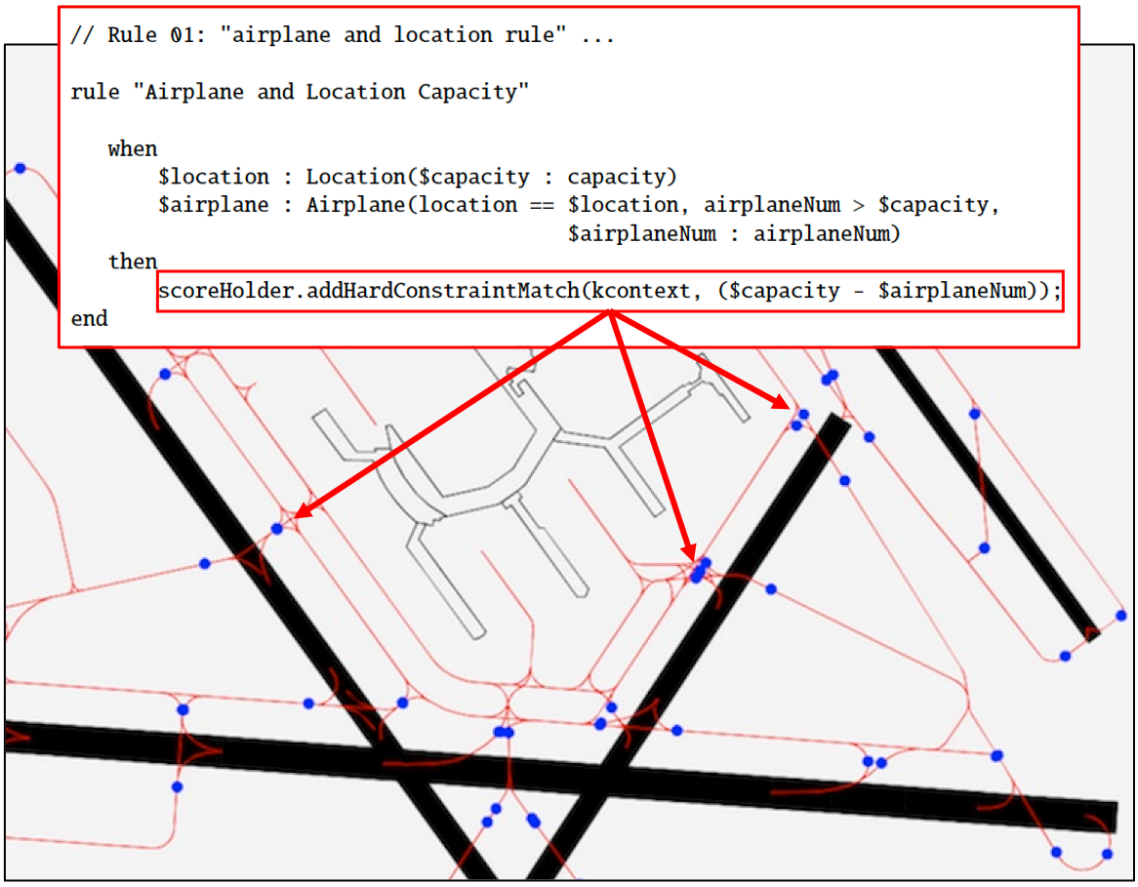


Figure 6.18: Taxiway operations with planning hard constraints view and results.

when it comes to the urban network problem (see Chapter 5), where two persons can appear at the same location and same time as long as one lets the other go first.

Rule 02 in Figure 6.20 forces the start time and end time of a time slot that is compatible with the corresponding airplane's ready time and depart time. This promotes the planning problem by assigning one unique timeslot to each airplane which reduces the chance for time conflicts between schedules of different airplanes. In aircraft operations, different temporal flight segments are assigned in order to indicate the status of an airplane. Definition of `Timeslots` can even be used to compare and distinguish between different time segments of an airplane during landing and take-off, which can surely help to

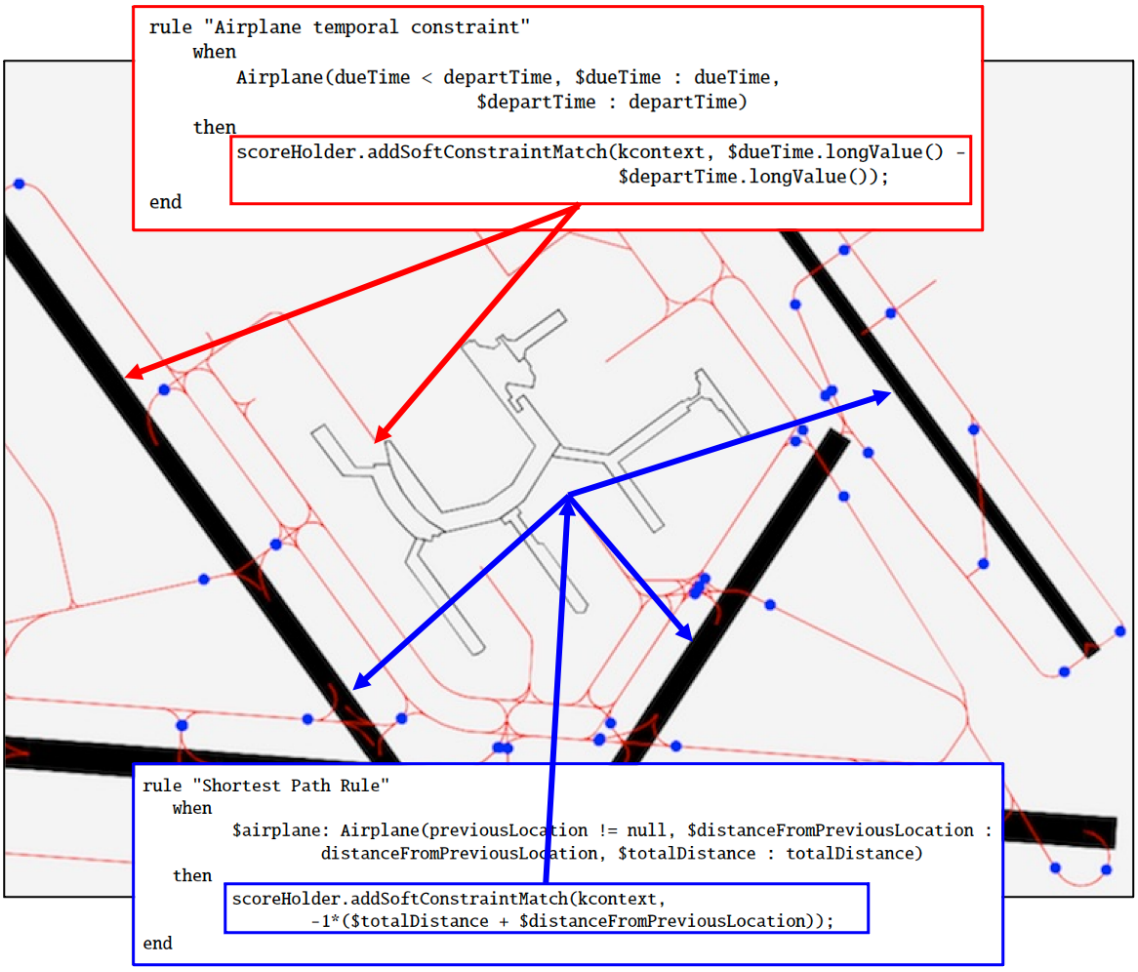


Figure 6.19: Taxiway operations with planning soft constraints view and results.

understand how long and when delays might occur.

Soft Constraints. Rule 03 in Figure 6.21 is the minimum pathway rule which is implemented as a soft constraint since distance minimization can compromise to other safety related rules and should not be the top priority among safety scheduling for airplanes. To the taxiway operations, the priority is always on safety. Reducing the amount of distances between locations can definitely help for better planning purposes, however, it should be used with time assignment as well. Delay happens constantly during airport operations.

Rule 04 in Figure 6.21 shows a penalty rule pf when airplane delay happens. This

Abbreviated Fragment of Software Code:

```
// Using HardSoftScoreHolder class

global HardSoftScoreHolder scoreHolder;

// Rule 01: "airplane and location rule" ...

rule "Airplane and Location Capacity"

   when
       $location: Location($capacity : capacity)
       $airplane: Airplane(location == $location, airplaneNum > $capacity,
                                        $airplaneNum : airplaneNum)
   then
       scoreHolder.addHardConstraintMatch(kcontext, ($capacity - $airplaneNum));
end

// Rule 02: airplane and timeslot...

rule "Airplane and Timeslot rule"
    when
      $timeslot:Timeslot($date : date, $startTime: startTime, $endTime : endTime)
      $airplane: Airplane(timeslot==$timeslot, startTime > $readyTime, endTime >
                    $departTime, $readyTime: readyTime, $departTime: departTime)
    then
      scoreHolder.addHardConstraintMatch(kcontext, ($startTime.longValue() -
         $readyTime.longValue())+($endTime.longValue()-$departTime.longValue()));
end
```

Figure 6.20: Fragment of code for airport taxiway operations hard constraints.

rule compares the due time which is the scheduled departure time of an airplane with its actual departure time (noted as `departTime`). This also has been implemented as a soft constraint, however, the larger delays can jeopardize the overall soft score which deems that solution to be a worse solution comparing

Rule 05 in Figure 6.21 shows the relationship between a runway and an airplane. It checks the delay time of when a runway is assigned to an airplane. This computes the time difference between the runway ready to be used time and the actual arrival time of an airplane on that runway. This significantly reduces the amount of wait time while runway is assigned for certain airplane schedules. It is been implemented as a soft constraint as delay may be acceptable during taxiway operations.

Abbreviated Fragment of Software Code:

```
// Rule 03: minimize shortest path ...

rule "Shortest Path Rule"
   when
       $airplane: Airplane(previousLocation != null,$distanceFromPreviousLocation:
                  distanceFromPreviousLocation, $totalDistance : totalDistance)
   then
       scoreHolder.addSoftConstraintMatch(kcontext,
                 -1*($totalDistance + $distanceFromPreviousLocation));
end

// Rule 04: Airplane delay cost ...

rule "Airplane temporal constraint"
   when
       Airplane(dueTime < departTime, $dueTime : dueTime,
                              $departTime : departTime)
   then
       scoreHolder.addSoftConstraintMatch(kcontext, $dueTime.longValue() -
                                          $departTime.longValue());
end

// Rule 05: runway and airplane delay cost rule ...

rule "Runway and Airplane Delay Cost"
   when
       $runway : Runway($readyTime : readyTime)
       $airplane : Airplane(runway == $runway, runwayArrivalTime < $readyTime,
                                  $runwayArrivalTime : runwayArrivalTime)
   then
     scoreHolder.addSoftConstraintMatch(kcontext, $runwayArrivalTime-$readyTime);
end
```

Figure 6.21: Fragment of code for airport taxiway operations soft constraints.

6.8 Results and Discussion

Results. This section describes the OptaPlanner outcomes for the airport operations problem shown in Figure 6.16. The problem setup comprises three airplanes (i.e., Airplane I, II, and III), a list of nodes that are used in the analysis, and runways X and Y.

Figures 6.22 and 6.23 show the outputs of three taxiways for airplane I to III respectively. The outcomes of the taxiways indicate that `Airplane I` uses $A \rightarrow B \rightarrow D \rightarrow E \rightarrow G \rightarrow I \rightarrow O$ taxiway with a cost of 20 (`taxiDistance`); `Airplane II` uses $J \rightarrow K \rightarrow H \rightarrow I \rightarrow O$ taxiway with a cost of 11; `Airplane III` uses $L \rightarrow M \rightarrow N \rightarrow K \rightarrow H \rightarrow I \rightarrow O$ taxiway with a cost of 16. In this planning problem, all three airplanes are scheduled to use `Runway X` to take-off where the last node of the runway is node `O`. From the taxiway results (Figures 6.22 and 6.23) and the location-time chart (Figure 6.25), we can summarize that all three planes need to get to node `I` and then uses node `N` to leave. Since the rules have limited the number of airplanes appear at a location and a runway is one (or less), the `Airplane I` (in orange) is scheduled to get to node `E` after the `Airplane III` (in blgreyue) left from node `E`. In order to minimize wait times at holding positions, we can see that the constraints let the airplanes wait at their beginning locations (terminals or gates) so that more holding positions can be available. In this case, airplanes `I and III` waited at their beginning locations at nodes `A and L` respectively. We can also see that `Airplane I` did not get to node `E` until `Airplane III` left from node `O`, meaning the latter has taken-off. This is restricted by the runway and airplane constraint where it says no more than one airplane shall be on any runways.

Abbreviated Fragment of Software Code:

```
<Taxiway id="1">
  <taxiDistance>20</taxiDistance>
  <terminal reference="A"/>
  <runway reference="X"/>
  <nextLocation id="1">
    <location class="Node" reference="B"/>
    <weight>2</weight>
    <nextLocation id="2">
      <location class="Node" reference="D"/>
      <weight>2</weight>
      <nextLocation id="3">
        <location class="Node" reference="E"/>
        <weight>2</weight>
        <nextLocation id="4">
          <location class="Node" reference="G"/>
          <weight>5</weight>
          <nextLocation id="5">
            <location class="Node" reference="I"/>
            <weight>6</weight>
            <nextLocation id="6">
              <location class="Node" reference="O"/>
              <weight>3</weight>
              <airplane reference="I"/>
            </nextLocation>
            <airplane reference="I"/>
          </nextLocation>
          <airplane reference="I"/>
        </nextLocation>
        <airplane reference="I"/>
      </nextLocation>
      <airplane reference=I"/>
    </nextLocation>
  <airplane reference="I"/>
 </nextLocation>
</Taxiway>
```

Figure 6.22: Fragment of OptaPlanner output for Airplane I.

Abbreviated Fragment of Software Code:

```
<Taxiway id="2">
  <taxiDistance>11</taxiDistance>
  <terminal reference="J"/>
  <runway reference="X"/>
  <nextLocation id="1">
    <location class="Node" reference="K"/>
    <weight>4</weight>
    <nextLocation id="2">
      <location class="Node" reference="H"/>
      <weight>2</weight>
        <nextLocation id="2">
          <location class="Node" reference="I"/>
          <weight>2</weight>
          <nextLocation id="2">
            <location class="Node" reference="O"/>
            <weight>3</weight>
          <airplane reference=II"/> </nextLocation>
        <airplane reference=II"/> </nextLocation>
      <airplane reference=II"/> </nextLocation>
    <airplane reference="II"/> </nextLocation>
</Taxiway>
<Taxiway id="3">
  <taxiDistance>16</taxiDistance>
  <terminal reference="L"/>
  <runway reference="X"/>
  <nextLocation id="1">
    <location class="Node" reference="M"/>
    <weight>2</weight>
    <nextLocation id="2">
      <location class="Node" reference="N"/>
      <weight>5</weight>
      <nextLocation id="3">
        <location class="Node" reference="K"/>
        <weight>2</weight>
        <nextLocation id="4">
          <location class="Node" reference="H"/>
          <weight>2</weight>
          <nextLocation id="5">
            <location class="Node" reference="I"/>
            <weight>2</weight>
            <nextLocation id="6">
              <location class="Node" reference="O"/>
              <weight>3</weight>
              <airplane reference=III"/> </nextLocation>
            <airplane reference=III"/> </nextLocation>
          <airplane reference=III"/> </nextLocation>
        <airplane reference=III"/> </nextLocation>
      <airplane reference=III"/> </nextLocation>
    <airplane reference="III"/> </nextLocation>
</Taxiway>
```

Figure 6.23: Fragment of OptaPlanner outputs for Airplane II and III.

Details of the results can be viewed in Table 6.6.

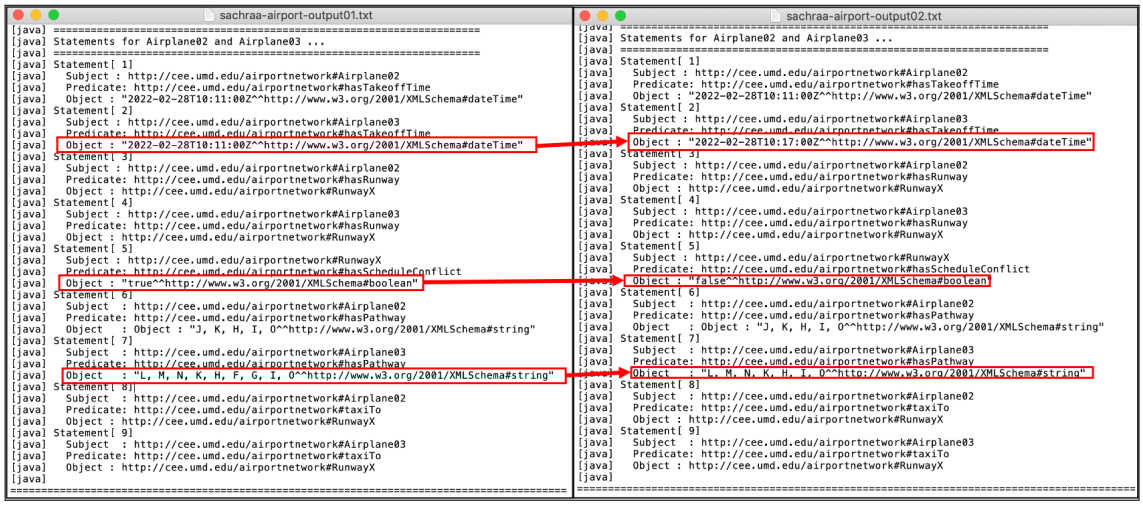
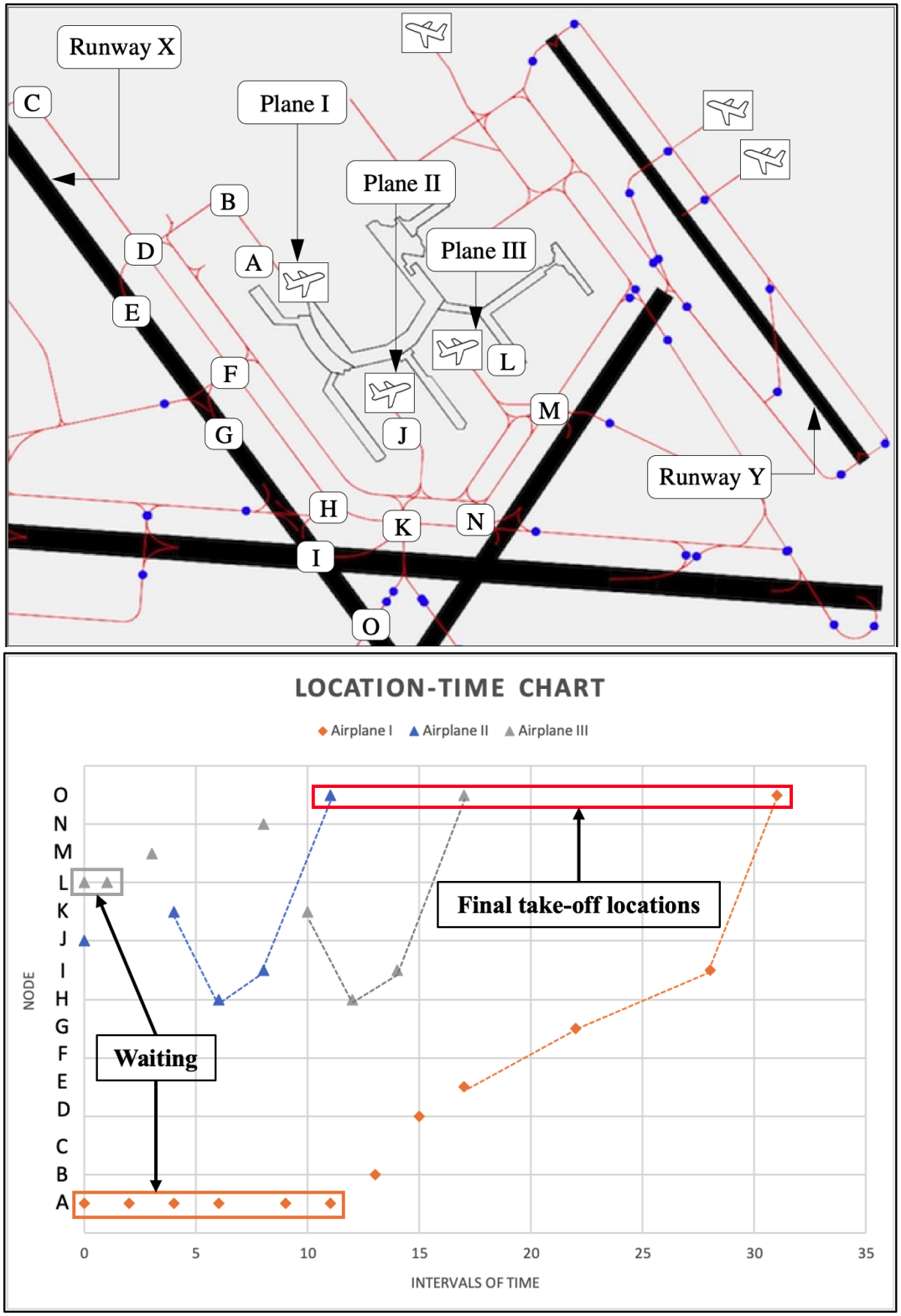


Figure 6.24: Changes in semantic representation for planning and replanning airplanes.

Change in Semantics Triggers Replanning for Taxiway Pathways. Figure 6.24 indicates some of the changes occur in the populated semantics where the data properties of *Airplane03*'s take off time and pathway are changed. This is also caused by the new object property of *hasScheduleConflictWith* assigned between *Airplane02* and *Airplane03*, to avoid schedule conflicts. The populated semantics tells that if two airplane's schedules have conflict (when they are sharing same runway for take-off), the framework is able to identify the relationship semantically, update corresponding information in the semantics, the allow replanning with the real-time planner.

Semantic Graph Transformation. Figure 6.26 shows an example of the semantic graph transformation for the planning of two airplanes. Here, two airplanes - *Airplane02* and *Airplane03* sharing the same runway *RunwayX* are indicated. The left-hand-side shows the initial schedules (with *hasTakeoffTime*) and planned pathways (with *hasPathway*) for the two airplanes, however, semantic captures that both airplanes have a boolean data property



Scenario: Planes I, II and III coordinate their taxiway operations from the gates A, J and L to Runway X.

Figure 6.25: Summary of taxiway operations.

needsNewPlan is *true*. In addition, a data property of *hasScheduleConflict* belongs to *RunwayX* which means that this runway is experiencing schedule conflicts which requires the associated airplanes to replan. The replan is shown on the right-hand-side as the updated plans and relationships among the two airplanes. The green text-highlights show the updated data properties where *Airplane03* has a new pathway and take-off time to avoid the previous conflict with *Airplane02*. The *needsNewPlan* is also switched to *false* indicating no additional plans are required at this time. Also, the *RunwayX*'s data attribute of *hasScheduleConflict* is turned to *false* indicating that this runway is clear to be used with the current plans for these airplanes.

Discussion. The example problem used a simplified version of Baltimore Washington International (BWI) airport landscape as it's geographical modeling environment. In order to practice the framework, initiate the real-time planning and demonstrate the coordination in the airplane taxiway operations, three strategies are illustrated in Figures 6.7, 6.8 and 6.9. Strategy 1 indicates that the airplane on the main runway shall takes off first regardless of other planes' locations. Strategy 2 and 3 evaluate distances between airplanes and allows other planes that are not on the runway take-off first, which is supported with the optimized planning results. As shown in Figure 6.8, for instance, if the safety distance between plane 1 and plane 3 are assured and plane 3 can still have enough time and space to take-off before plane 1 does. This requires real-time planning in measuring safety measures allowing to evaluate both spatial and temporal constraints.

Three strategies can be treated as from the most conservative strategy (scenario

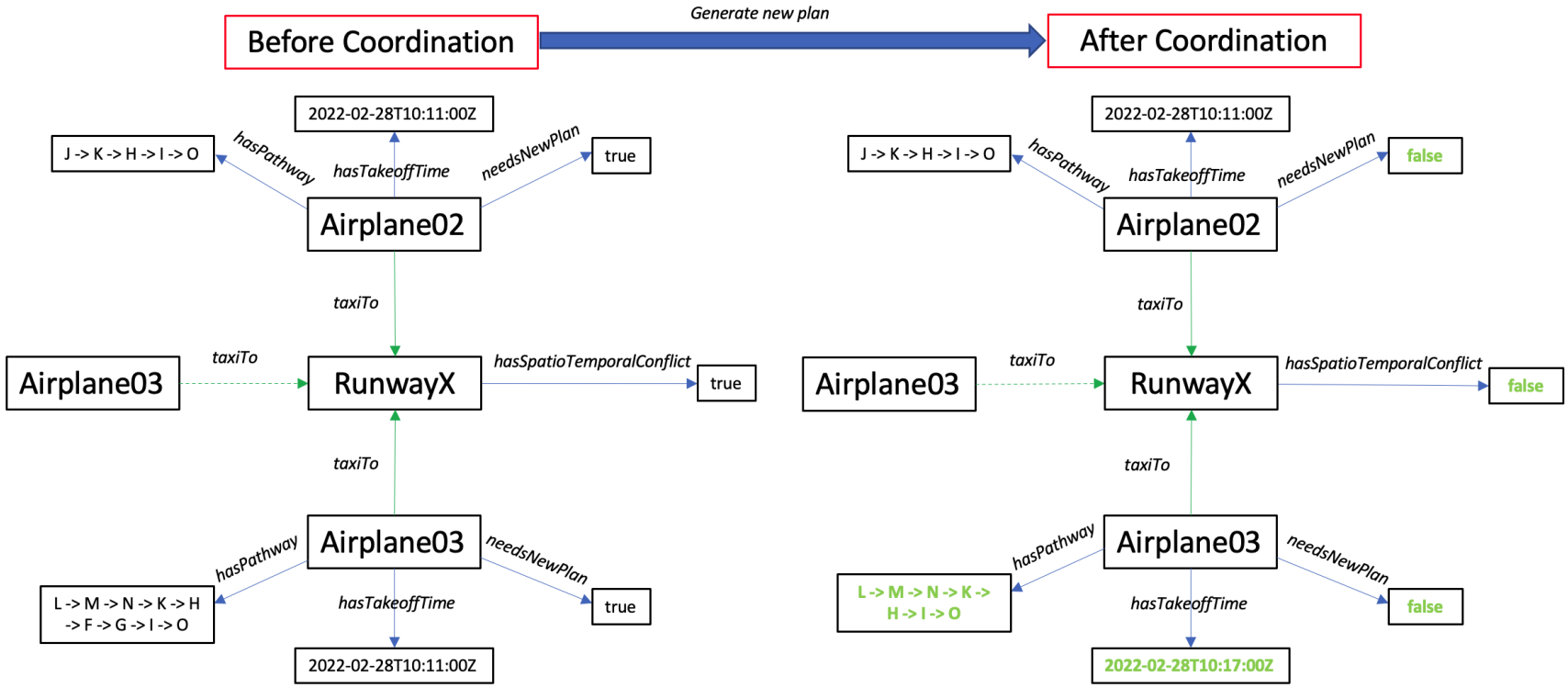


Figure 6.26: Semantic graph transformation for taxiway operations.

Airplane	Pathway (before)	Runway (before)	Score (before)	Pathway (after)	Runway (after)	Score (after)
Airplane01	A, B, D, F, H, I, O	X	11	A, B, D, E, G, I, O	X	13
Airplane02	J, K, H, F, G, I, O	X	13	J, K, H, I, O	X	15
Airplane03	L, M, N, K, H, F, G, I, O	X	5	L, M, N, K, H, I, O	X	9

Table 6.6: Summary of results for taxiway operations (before and after coordination).

1) to the most effective strategy (scenario 3) which requires real-time planning to assure the safety operations for airplane schedules. Scenario 1 is the safest approach to schedule airplanes when they share a ommon runway, however, it can cause unnecessary delays. Scenario 3, on the other hand, is the most efficient strategy among the three as it evaluates the spatial relationships among airplanes and runways dynamically, and assign take-off schedules based on both spatial and temporal constraints. The constraints embedded in OptaPlanner can help to distinguish and evaluate between various scenarios under a same problem setting. The flexibility of altering both the hard and soft constraints allow us to look at the taxiway operations and decide what strategies can be the best to use in the real-world, and what constraints make the most sense to embed for taxiiway operations. Additionally, OptaPlanner results shown in the previous section describes the importance of having real-time planning support for scheduling taxiway problems. In order to reduce the amount of wait time at holding positions, the integrated framework is able to tell how long a plane shall wait at its initial location (gate or terminal) so that further confusions and delays during airplane schedules can be avoided.

Chapter 7: Case Study B: Multi-Domain Semantic Modeling and Planning for Wildfire Evacuation

This case study looks at multi-domain semantic model and reasoning (see Figure 7.1), coupled with planning algorithms for the wildfire evacuation problem.

7.1 Introduction

7.1.1 Problem Statement

The wildfire evacuation domain design employs semantic descriptions of application domains. This approach allows ontologies and rule-based reasoning to enable validation of requirements and communication between multiple domains [12]. Figure 7.1 shows our proposed semantic architecture for distributed system behavior modeling with ontologies, rules and message-passing mechanisms - a multi- domain semantic modeling framework. The textual requirements are defined in order to enable the rule-checking mechanisms that are defined in the forms of mathematical and logical constraints (or Jena rules). The ontology models and rules bring the textual requirements and engineering models together to provide a platform for simulation, adjustment and viewing development of system structures and system behaviors over time. The framework also supports the

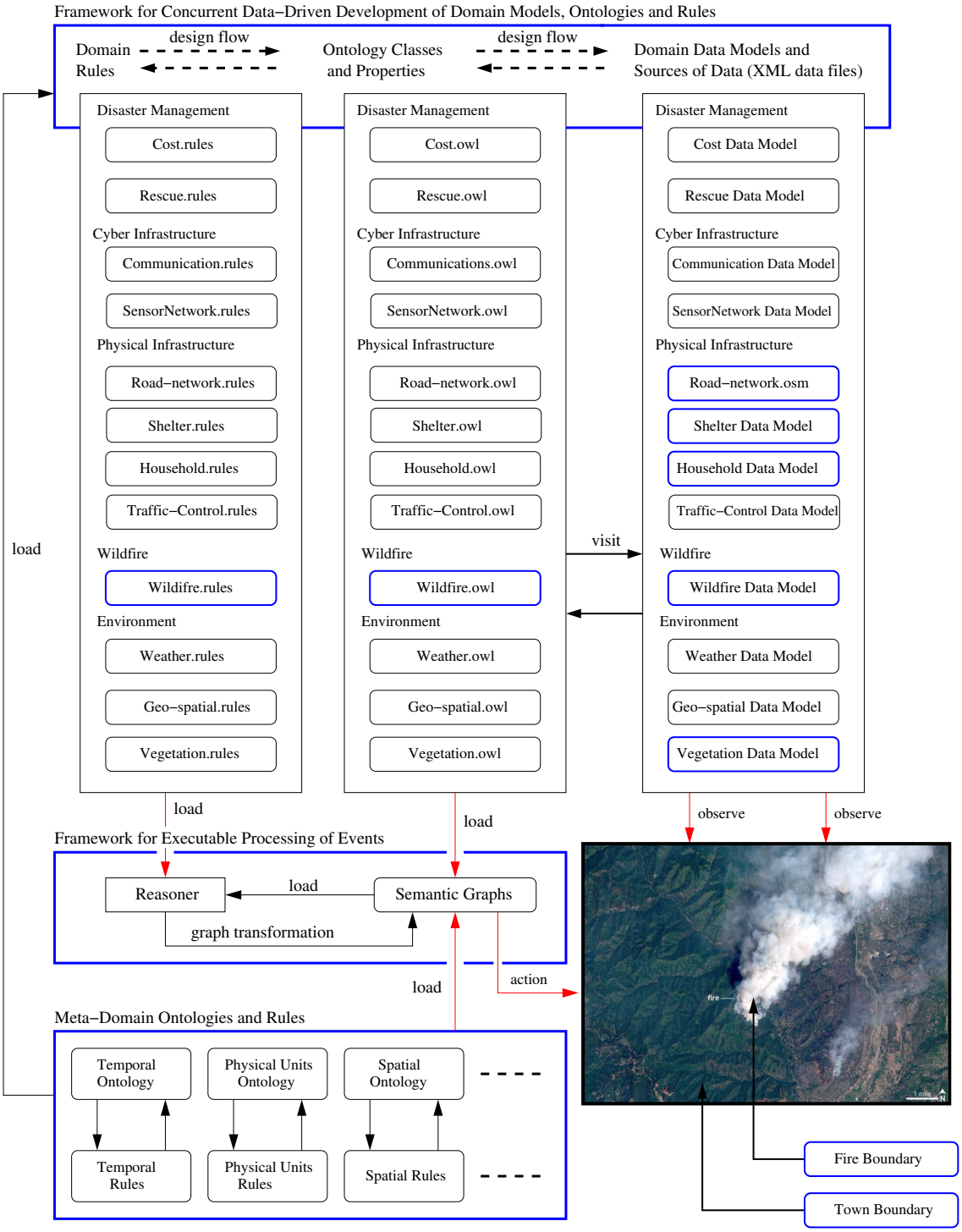


Figure 7.1: Multi-domain semantic model and reasoning infrastructure in wildfire evacuation modeling.

use of domain knowledge from various domains related to the solving problem domain. For example, disaster management, information of shelters and households, supply, environment and transportation, and other information related to rescue plan can be embedded under the framework and supported for reasoning mechanisms through considering multiple participating domains (top). The bottom section of the framework indicates how meta-domain ontologies and rules (e.g., temporal, spatial, physical units, graph) are imported and used in the framework, how they are able to work hand-in-hand with other knowledge from domain-specific domains.

Working with OptaPlanner. Figure 7.2 illustrates a simplified version of input and output view of the proposed framework.

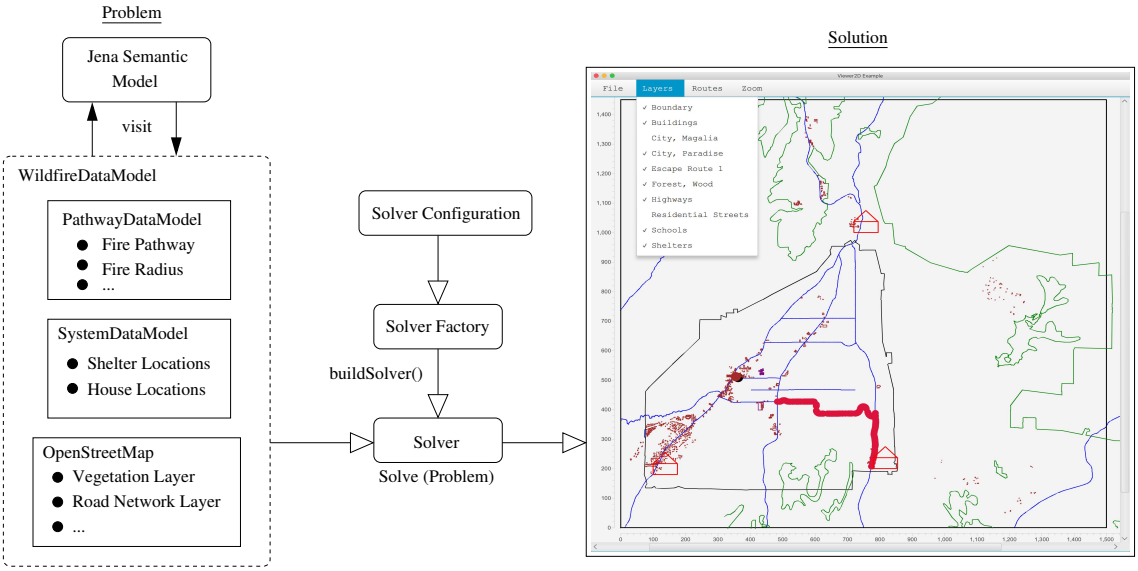


Figure 7.2: Overview of OptaPlanner input/output for wildfire evacuation.

The left-hand-side of Figure 7.2 shows the various data that are extracted and imported to the semantic model and the right-hand-side shows the optimized results based on the information that is passed through the semantic graphs transformation. The middle section

shows how OptaPlanner handles the input data sources, system configuration, then return optimized solution sets back to domain-specific data model. Plotting of the optimized route for a specific evacuee is also feasible in Java, showing as in the Figure.

Modeling wildfire evacuation is challenging, however, the combination of semantic modeling and concurrent data-driven development of domain models, ontologies and rules can accurately capture the knowledge in the wildfire domain which can alter any levels of decision makings. Figure 7.1 also shows how different parts in the semantic framework can work together and promote a knowledge-based framework to help an urban problem like evacuation. In the Figure, the domain-specific ontologies are shown on the top part of the Figure in which the domain specific rules, owls, and data models are inserted jointly together to capture changes in the semantic graphs and respond to them, which is described in the middle section of the Figure. The semantic graphs also capture the transformations with Jena reasoners and take actions to the problem domain using the imported rules, owl files, and data models. Simultaneously, the meta-domain ontologies and rules are imported side-by-side with the domain-specific knowledge's to make sure that the temporal, spatial, and physical unit reasonings are consistent through the entire modeling framework.

7.1.2 Literature Review

The domain of wildfire evacuation requires constant updates of live evacuation information about people, houses, evacuation routes along with the fire dynamics. The wildfire semantic model offers a uniform conceptual abstraction representing related con-

cepts, relationships between these concepts, and their attributes. Concepts for the wildfire evacuation semantic modeling may include wildfire, person, house, shelter, vegetation, and other concepts that are required for the purposes of modeling. Semantic modeling also offers rule-based reasoning mechanisms to query semantic graphs which adds and derives new information assertions to the graphs and results in event-based graph transformations. This also supports roles in the data collection and processing, identification of events, and provides automated decision-making. The dynamic nature of semantic modeling keeps the semantic graphs updated (driven-by Jena rules) based on the semantic relationships among classes and their attributes, which are defined in the ontologies of the participating problem domains.

The development of Semantic Web technologies [20] over the past two decades has made machines capable of accessing and sharing information through automated discovery of new knowledge. In the wildfire domain, a few studies are supported by the semantic modeling framework. An ontology OntoFire [88] was proposed for managing spatial data in the wildfire domain. This enriches its geographical data through knowledge-based semantic graphs and it also allows to identify information easily with the support of spatial navigation mechanisms; SemiCityMap [6], an ontology-based spatial reasoning framework for disaster monitoring, was conducted with satellite image information. This supports querying to find specific locations and paths with the semantic support; Another research work used the combination of Earth Observation (EO) data, ontologies, and linked geospatial data to provide wildfire monitoring services where it is developed with an European project - TELEIOS and a Greek project SWeFS [92]. Additionally, this

work promotes automatic comparison between EO data with auxiliary data, and derives information that satellite images can not capture (e.g., fire in sea).

State-of-the-Art Research in Wildfire Evacuation. Conventional planning methods have difficulties to solve the environmental and social problems in the modern complex urban areas [166]. This opens up the door for researchers to use knowledge-based approach to better describe and model evacuation process in the support of disaster management and decision making. For disaster management (DM), a meta-modeling based tool called Disaster Management Knowledge Repository (DMKR) is proposed to store, reuse and structure DM knowledge, and support knowledge sharing [117]; The Integrated Wildfire Evacuation Decision Support system (IWEDSS) was introduced as a decision support system for wildfire which integrates data from social media, survey, GIS and remote sensing data, and volunteer suggestions. The framework uses semantic model to support its social perception analysis and can also be adopted with other disasters (i.e., hurricane, tsunami) [107]; Another data-driven method of using geo-topics from social media platforms (i.e., Twitter) is discovered to monitor information during events in spatial, temporal and semantic dimensions. This proposed geo-topic tracking system considers three disaster cases (two wildfires and one hurricane), improving situational awareness for disaster planning and policy-making [166]; Semantic embedding models are also used with empirical data to detect whether source domain of antagonists has attributes that are associated with danger in natural disasters [76]. Tabulated literature can be found in Tables 7.1 and 7.2.

Study	Focus	Methodology
Alirezaie et al. (2017) [6]	SemiCityMap, a disaster monitoring framework	An ontology-based spatial reasoning framework for disaster monitoring, was conducted with satellite image information, which can be queried to find specific locations and paths with semantics support.
Cova et al. (2013) [36] Murray-Tuite et al. (2013) [104] Wolshon and Marchive (2007) [163]	Fire simulation and departure time	Predetermined distributions and S-curves
Grajdura et al. (2020, 2021, 2022) [68–70]	Wildfire evacuation simulation	Simulate wildfire evacuation with the support of survey data, describing social characteristics and behaviors of agents.
Hauser and Fleming (2021) [76]	Detect danger in natural disaster	Semantic embedding models are also used with empirical data to detect whether source domain of antagonists has attributes that are associated with danger in natural disasters.
Kalabokidis et al. (2011) [88]	Manage spatial data in the wildfire domain	OntoFire - an ontology was proposed for managing spatial data in the wildfire domain and enriches its geographical data through knowledge-based semantic graphs; convenience of identifying information with the support of spatial navigation mechanisms
Kyzirakos et al. (2013) [92]	Provide wildfire monitoring services	It uses the combination of Earth Observation (EO) data, ontologies, and linked geospatial data; automatic comparison between EO data with auxiliary data, and derives additional information where satellite images can not capture

Table 7.1: Summary of literature on wildfire (disaster) evacuation modeling (part 1/2).

Study	Focus	Methodology
Murray-Tuite et al. (2012) [105]	Determine evacuation route decisions	Integrates survey and GPS-based methods.
Nara et al. (2017) [107]	Wildfire evacuation decision support system (IWEDSS) framework	Integrates data from social media, survey, GIS and remote sensing data, and volunteer suggestions; uses semantic model to support its social perception analysis and can also be adopted with other disasters (i.e., hurricane, tsunami)
Othman and Beydoun (2016) [117]	Disaster management (DM)	A meta-modeling based tool called Disaster Management Knowledge Repository (DMKR) is proposed to store, reuse and structure DM knowledge, and support knowledge sharing
Reilly et al. (2021) [130]	Avoid ambiguous coordination and interaction among researchers from various disciplines	A common knowledge (boundary object) is preferred in hazards research.
Trainor et al. (2013) [150]	Improve transportation modeling assumptions with improved real world mapping	A merged approach of empirical and theoretical insights into transportation evacuation modeling.
Wilmot and Mei (2004) [162]	Simulate (hurricane) evacuation	Perception and behavioral surveys through computer-based telephone interviews for alternative trip generations in hurricane evacuations.
Yao and Wang (2020) [166]	Monitor information during events in spatial, temporal and semantic dimensions	Data-driven method of using geo-topics from social media platforms (i.e., Twitter); this proposed geo-topic tracking system considers three disaster cases (two wildfires and one hurricane), improving situational awareness for disaster planning and policy-making.

Table 7.2: Summary of literature on wildfire (disaster) evacuation modeling (part 2/2).

7.1.3 Scope and Objectives

The scope of this case study is to understand the feasibility of the integrated framework with a disaster planning problem, and how it can be used as an initial step forward to promote disaster planning supported with a knowledge-based reasoning framework followed with real-time dynamic planning capability. It is important to note that unlike the airport taxiway problem, where planning actions will be followed, scheduling for wildfire evacuation is complicated by the range of ways people decide to respond to wildfire attack.

The objectives of this study focus on the how does the knowledge-based framework supports decision makings during evacuation; how will the real-time planning tool extracts filtered data through semantic reasoning from data models, and feed to the planning software. To address these goals, this study covers a range of topics. Section 7.2 covers the integrated framework and solution procedure; Section 7.3 introduces a simplified model for wildfire spread (simulation); Section 7.4 discusses several strategies of evacuation; Section 7.5 gives a detail view of how to use the multi-domain semantic modeling for the evacuation problem; Section 7.6 provides the formulation of the planning problem; Section 7.7 illustrates details of the planning aspects in OptaPlanner, class diagram, embedded constraints and sample results; Section 7.8 further explains and discusses the results obtained from this framework.

7.2 Integrated Framework and Solution Procedure

Solution Procedure. A three-step solution procedure for the wildfire evacuation planning is shown in Figure 7.3. These steps involve knowing the fire geometry and pathway based on its initial location and vegetation location, affected road segments, positions of evacuees and locations of shelters. Step 1 shows the computation of fire pathway and geometry. From the vegetation data extracted through `Open Street Map`, a simple fire simulation with respect to the positions and sizes of vegetation layer can be accomplished. This process helps to generate a pathway for the fire trajectory and geometry. Step 2 conducts the spatial-temporal analysis with respect to fire and road segments. The relationship between road network (also from OSM) and fire information can be computed, and this brings the spatial and temporal information of the road segments (i.e., when and what segments of road network are blocked). Step 3 takes the additional locations of houses and shelters as origins and destinations for the planning problem. It avoids the conflict schedules as well as the blocked road segments, and optimizes evacuation route options for people to evacuate successfully.

Integrated Framework. Figure 7.4 shows the integrated framework that was explained in Figure 7.3. Three steps are matched with the two figures. The integrated framework shows the details of how does the various data sources are imported, how does the multi-domain semantic knowledge is embedded as part of the spatial-temporal analysis and how will the real-time planning is conducted using the information from the previous two steps. The *Wildfire Simulation and Urban Data Model* loads the road network

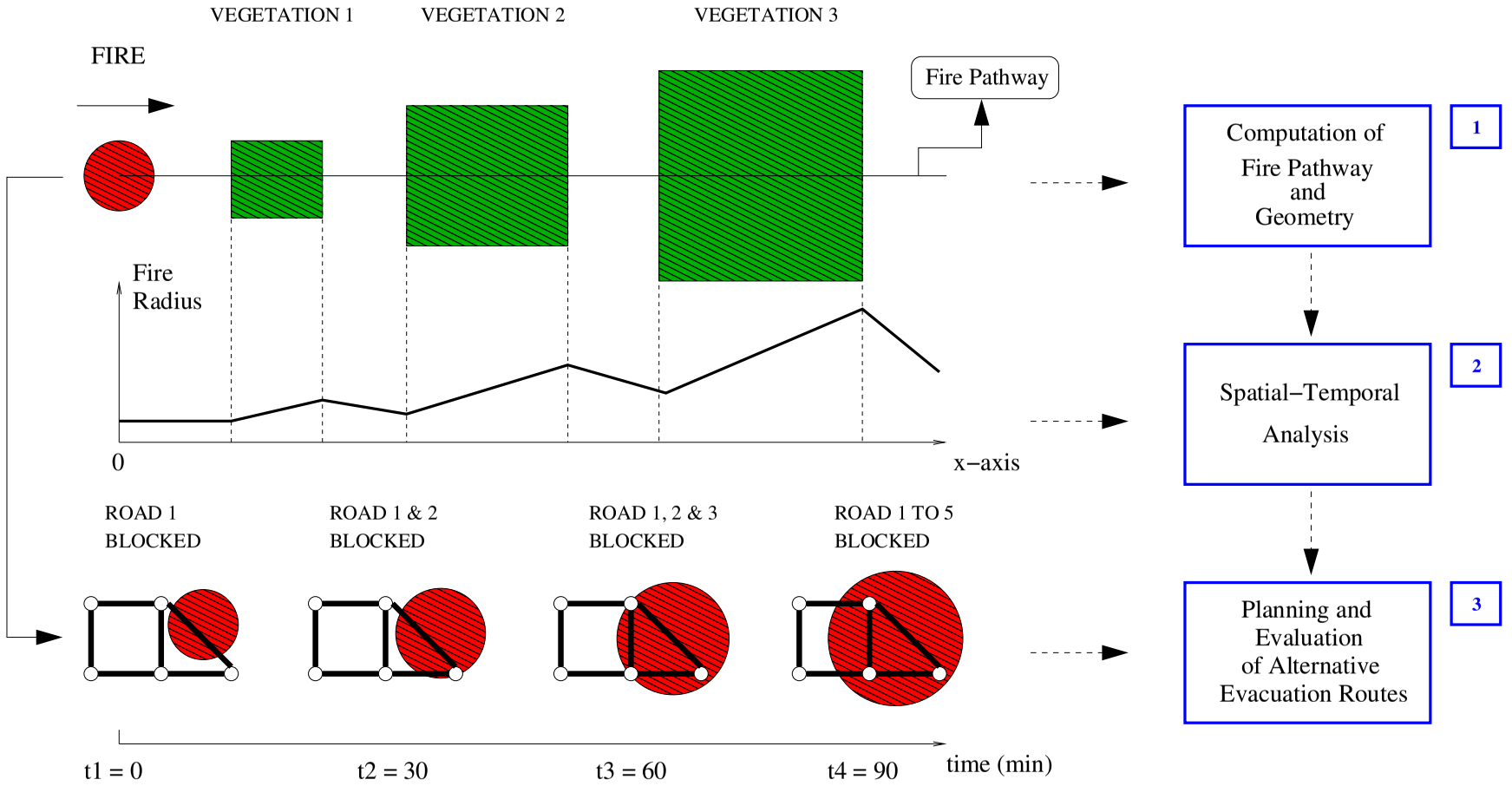


Figure 7.3: Multi-domain modeling: fire, vegetation and road segments.

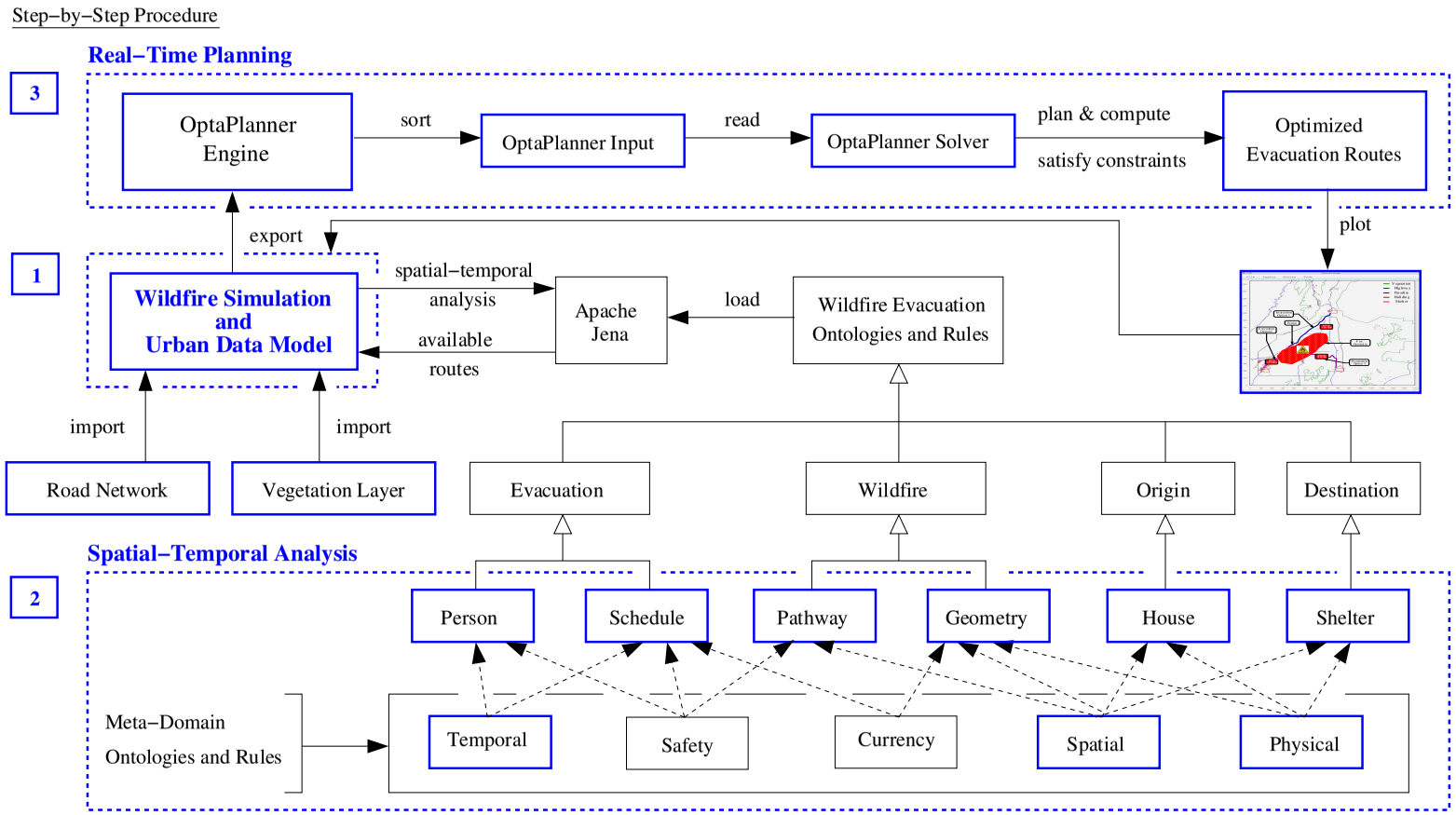


Figure 7.4: Integrated framework and solution procedure.

and vegetation layer information, extracted through OSM, to set up the planning problem environment. It exchanges information with the semantic graph, *Apache Jena*, and initiates graph transformation based on the data (events). The *Wildfire Evacuation Ontologies and Rules* is loaded to the *Apache Jena*. Various domain-specific and meta-domain ontologies and rules are stored here. Spatial-temporal analysis is being checked with the meta-domain ontologies and rules, shown in the bottom of the figure. This portion of the graph shows how does the domain-specific data (person, house, etc.) are sent to the associated ontologies and rules, and how does the temporal and spatial information are attached with the ontologies and rules. The Jena rules help to relate individuals and update their associated data properties. Then the planning engine takes the populated semantic graphs and extracts information that is needed for the purposes of planning. Through a series of steps in OptaPlanner as described previously, the optimized evacuation routes are computed and suggested to each evacuee. Eventually, software helps to visualize the evacuation route via plotting on the GUI.

Model Dependencies. Figure 7.5 shows the model dependencies for the wildfire evacuation domain. Similar to the model dependencies for the airport operation domain (shown in Figure 6.5, it contains three major parts and those are the *Jena Semantic Model*, the *OptaPlanner Solver* and the *Wildfire Data Model*. The semantic model loads the ontologies and Jena rules in order to create an initial version of the semantic graph without any individuals inserted. The *Wildfire Data Model* loads data from three different sources - the *System Data Model*, the `Pathway Data Model` and the *Open Street Map Model*. This process brings a mixture of events based on various data as well as their spatial and

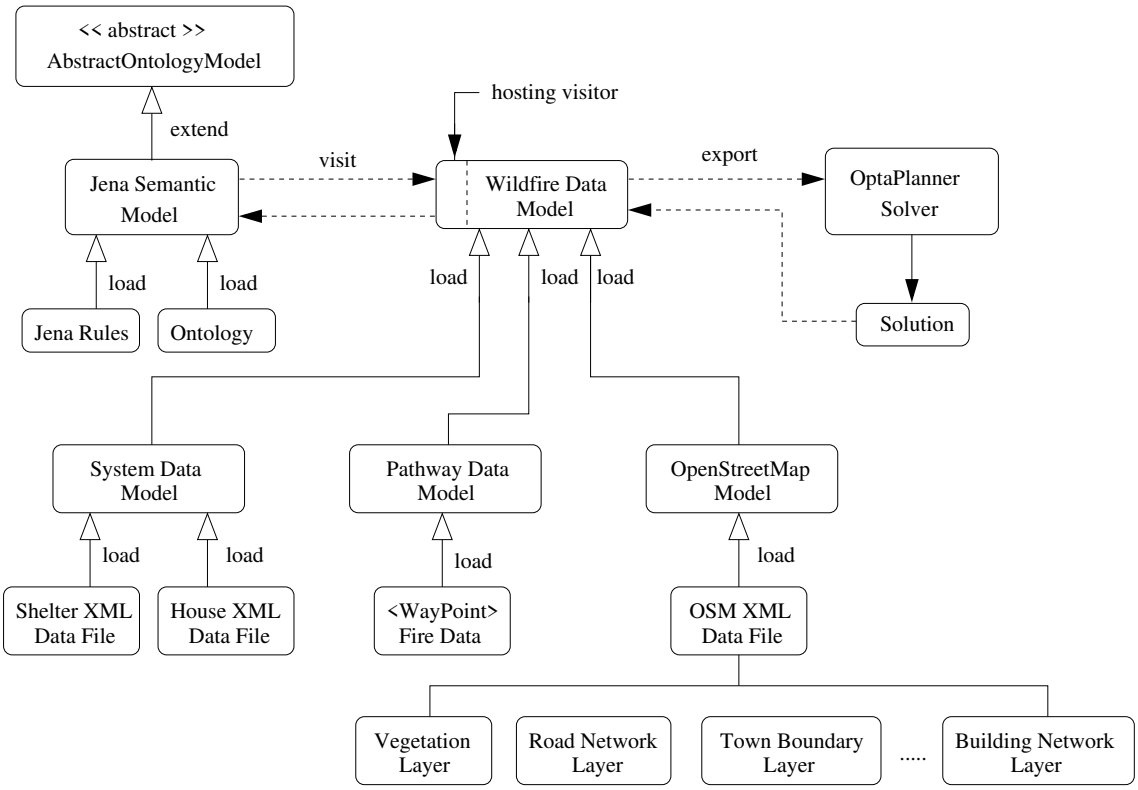


Figure 7.5: Model dependencies for wildfire evacuation modeling.

temporal correlations among them. The data model also hosts a visitor which lets the semantic model grabs the data and starts graph transformations based on the events that are added and reasoned with the ontologies and rules. Lastly, the constraint solver takes necessary information, including the one from both before and after graph transformation, and starts the optimization process based on a set of predefined constraints embedded in the planner.

7.3 Modeling Wildfire Spread and Wildfire/Infrastructure Impact

The purposes of this section are to introduce a simplified model for wild fire spread (simulation) and then discuss various strategies of evacuation.

7.3.1 Simplified Wildfire Spread Model

In order to understand fire behavior for an evacuation modeling, we need to know how fire spreads and the associated parameters that affect the spreads. In [89], authors proposed a simplified mathematical spread model to extract expected fire spread rate (FSR) with parameters of both weather and topography. These factors include wind speed, wind direction, fuel moisture content (FMC) and slope coefficient. The authors also stated that the simplified model is proposed because the traditional complex mathematical models contains nonlinear equations that cannot be easily applied.

The FMC in the model is chosen to be 4, 8, 12, and 16, and it is a percentage value [9]. In [54], the author defined the FMC as the "weight of water per unit weight of dry fuel." The range of fuel moisture content included in this study is 4.0 to 15.0. For cases when moisture is sufficiently high, the combustion can be completely prevented. In other words, the greater the moisture content needs more heat to bring the fuel to ignition point. A simplified statistical results from [54], conducted a series of 198 test fires [89], is tabulated in Table 7.3, where FSR values are presented with respect to FMC and wind speed.

Wind Speed (mi/h)	*FSR (feet/min) Fmc=4*	*FSR (feet/min) Fmc=8*	*FSR (feet/min) Fmc=12*	*FSR (feet/min) Fmc=16*
0	0,76	0,55	0,36	0,18
2	1,78	1,28	0,89	0,51
4	3,34	2,42	1,7	1,03
6	5,38	4,01	2,85	1,78
8	8,1	6,15	4,37	2,83
10	11,75	8,97	6,46	4,24
12	16,7	12,52	9,15	6,36

Table 7.3: Fire spread rate as a function of FMC and wind speed (Source: [9]).

Using the regression model to retreat the FSR curve with respect to FMC and wind speed (V), [89] presented a second degree polynomial equation (see below),

where it shows:

$$FSR = (0.0002 * FMC^2 - 0.0008 * FMC + 0.1225) * V^2 + (-0.0008 * FMC^2 + 0.0005 * FMC + 0.1823) * V + (-0.0019 * FMC^2 - 0.0924 * FMC + 1.2675). \quad (7.1)$$

Equation 7.1 is adopted in our model to compute for the corresponding fire spread rate.

7.3.2 Modeling Impact of Wildfires on Urban Infrastructure

Wildfires have severe impacts on our urban infrastructures. [132] stated that in the Australian fire season between 2019 to 2020, 18 million hectares of vegetation

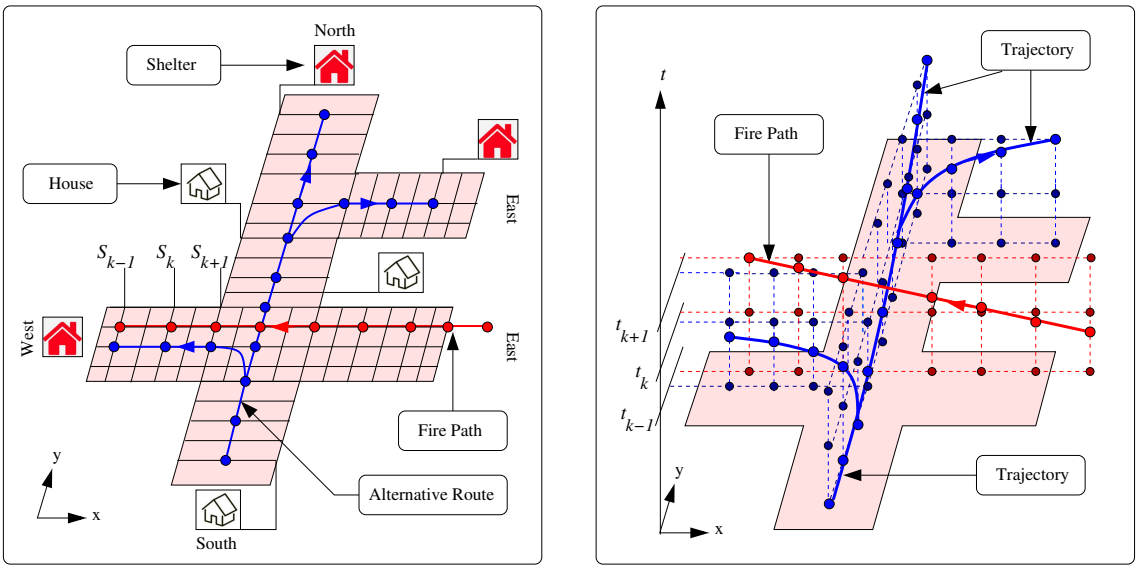
burned leaving fatality of 33 people and 2500 homes destroyed. They also strengthened that the influence of wildfires extend to damage water supplies in the burned areas and possibly to cause watershed hazards which poses a higher risk in long-term drinking water production in the affected regions. Wildfire also impacts our schools and hospitals [138] tremendously which are constantly located in densely populated areas. Wildfires destroys our cyber communication infrastructures easily [8]. The imminent impacts affect the first responders to efficiently communicate with people for evacuations, and long-term impacts require to repair and replace damaged components for the communication/signal towers. Besides its impacts on our physical and cyber infrastructures, it also poses threats to climate change which the latter may cause warmer and windier conditions that make fires in those areas burn and spread even faster [57].

7.4 Simplified Strategies of Evacuation

7.4.1 Framework for Spatio-Temporal Analysis

Figure 7.6 shows the framework for spatio-temporal analysis for wildfire evacuation problem. The left-hand-side of the image shows the two-dimensional (or spatial) distribution of various physical entities for this specific domain.

To simplify this as a case study problem, we have included house, shelter, fire path and evacuation route as variables for conducting the spatial-temporal analysis. For urban safety plannings, we tend to only visualize the spatial distribution of the domain, however the temporal aspect can easily be neglected. The same spatial understanding from the



(a). House−to−Shelter Evacuation Problem. **(b). Spatial−Temporal Analysis**

Figure 7.6: Framework for Spatio-Temporal Analysis

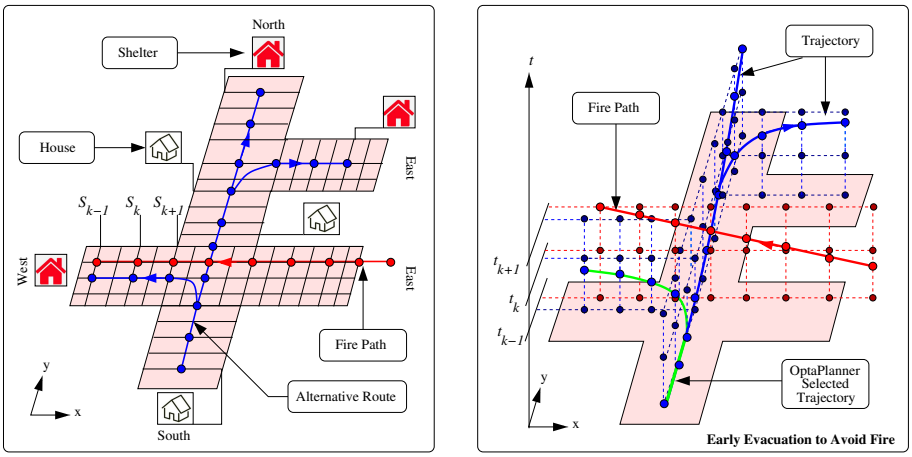
left-hand-side image can be plotted as the one on right-hand-side, where the vertical axis is plotted for time. In order to keep things safe, we need to assure things shall happen at the right place and also at the right time.

7.4.2 Three Wildfire Evacuation Strategies

State-of-the-art strategies for wildfire evacuation plannings are based on preparation and announcements from local officials and governments, specially at the WUI (wildland-urban interface) regions. In most regions in the U.S., the community has emergency response plans to alert the public the actual plans in-evacuation, for instance, where and when to evacuate, and best routes for departing from you locations [26]. The OSHA states that having an evacuation plan before hand will prevent further injuries and avoid confusion during evacuation [113]. A thorough evacuation plan includes aspects of conditions that activate the plan, chain of command, emergency functions, detailed evacuation

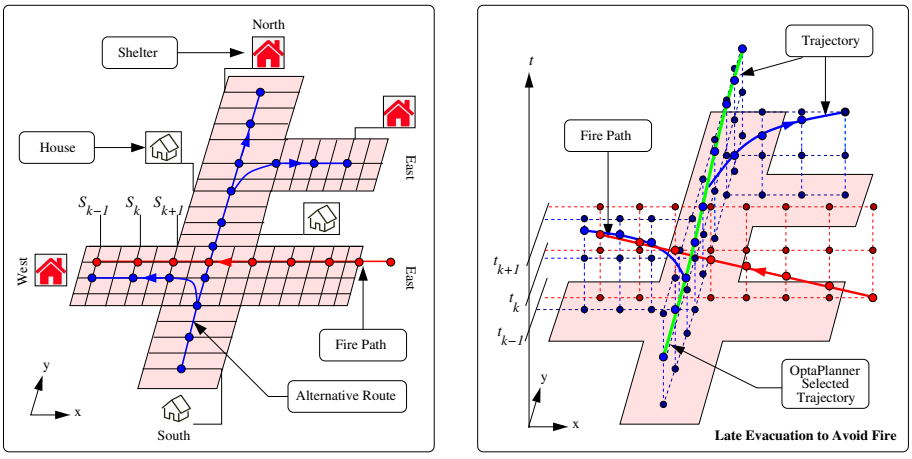
procedures, count personnel, equipment for personnel, and review the plan with workers. FEMA also provides evacuation tips for people to evacuation safely [113]. However, it is impossible to apply a uniform strategy to all people during evacuation since different circumstances and situations can be faced when wildfires are in place.

In this section, we shows a three simple evacuation strategies. Figures 7.7 through 7.9 show three evacuation strategies for evacuating safely. Strategy one (see Figure 7.7) plots the spatial and temporal relationships of a scenario people taking evacuation routes before the arrival of fire. Figure shows that the red line represents fire and blue lines represent various evacuation options. The two dimensional image (left) only shows the spatial relationships between fire and various evacuation routes, but the three dimensional image (right), plotting the temporal aspect in vertical axis, clearly identifies the relationships between when and where the actual evacuations have taken place with respect to fire; The second strategy (see Figure 7.8) is so-called stay as long as possible. Again the relationship between fire and evacuation routes are difficult to identify on the two-dimensional image, but the right-hand-side image indicates that the fire pathway happened before all the other evacuation routes. This means that people waited after fire has passed assuming the fire has finished burning; Third strategy (see Figure 7.9) indicates that the start time of fire and people evacuation are the same (at time t_k). If fire goes relatively slow from its initial position, which it does in this case, then people should have enough time and space to evacuate before fire comes to block the road. Likewise, there are many wildfire evacuation strategies that are used and through emphasizing the importance of understanding the temporal domain, it will significantly increase the chance of safety evacuation.



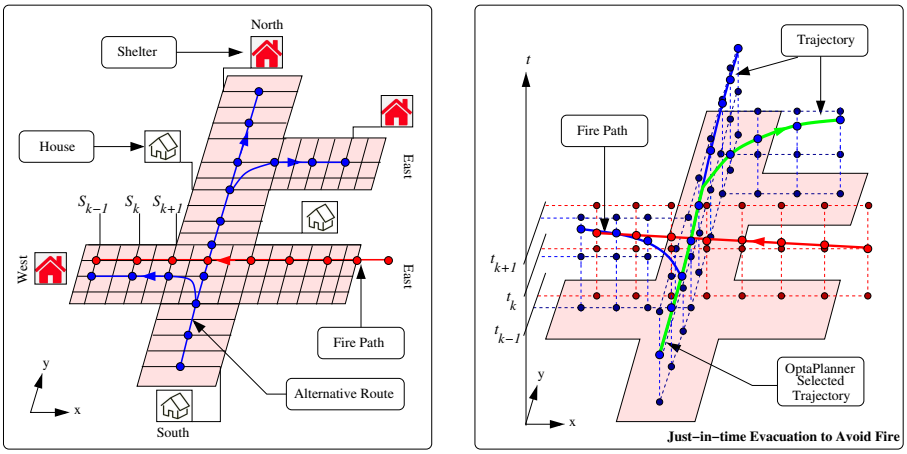
(a). House–to–Shelter Evacuation Problem. **(b). Spatial–Temporal Analysis**

Figure 7.7: Evacuation strategy 1: Play it safe (evacuate before fire arrives).



(a). House–to–Shelter Evacuation Problem. **(b). Spatial–Temporal Analysis**

Figure 7.8: Evacuation strategy 2: Stay as long as possible (evacuate just-in-time).



(a). House–to–Shelter Evacuation Problem. **(b). Spatial–Temporal Analysis**

Figure 7.9: Evacuation strategy 3: Stay and fight (evacuate after the fire is extinguished).

7.5 Multi-Domain Semantic Model and Event-based Reasoning

Multi-Domain Semantic Model. Multi-domain semantic modeling for wildfire evacuation planning requires to use a range of knowledge that covers geographical information of the area that is affected, fire condition, shelter positions, road condition, public evacuation notices and and other meta-domain knowledge, such as space, time, graph, etc. The multi-domain semantic modeling framework is shown in Figure 7.1 where it indicates a concurrent data driven development of multiple domain models, ontologies and rule sets. First column on the left of the figure gives a set of domain rules that are required, second shows the various ontology files that keeps the classes and properties information, and third column appears the various data models that are needed for each domain. Three columns work side-by-side to fill the reasoner, semantic graphs and observe events which are shown in the middle section of the figure. The bottom section gives the meta-domain ontologies and rules which are also loaded to the semantic graph. This is separated from the domain-specific domains (top) because space, time and physical units can be used across all domain-specific domains. Extracting various rules, ontologies and data models from various perspectives can maximize the reality of modeling a domain problem through considering all necessary knowledge that is involved.

Rule-based Reasoning Approach. Rule-based reasoning is particularly beneficial for a dynamic application logic (i.e., change of a data value that needs to be immediately reflected to a larger system). The rules are also applied externally meaning the rule files can be defined separately from the data sources and semantic graphs. Rules are also

used to resolve situations of conflicts and competing objectives. The eXtensible Markup Language (XML), Open Street Map (OSM), resource description framework (RDF), and Web Ontology Language (OWL) all can be implemented with Jena rules through semantic graph reasoning to start graph transformations. Additionally functionality of rules include rules can represent predefined guidelines or requirements that are being easily communicated and understood; b). rules are defined independently and can be imported to use whenever needed; c). rules can distinguish from knowledge and its implemented logic; d). rules can be easily changed without making changes in the model nor in the implemented source code.

7.5.1 Generation of Wildfire Simulation Individuals

Figure 7.5 shows the data-driven approach for generating the urban domain individuals in semantic graphs. Firstly, the various types of data (System Data Model, Pathway Data Model, and Open Street Map Data Model) are imported through the Java Object data models with JAXB.

Table 7.4 shows the list of data that are used in this approach as well as the explanations and formats of data sources. Secondly, the ontologies and Jena rules are loaded into the Jena Semantic Model which creates instances of the relevant ontologies (OWL) through visiting data models and obtaining information on individuals of a specific urban domain (i.e., wildfire evacuation, taxiway operations). Then, after the individuals and their information being transferred to Jena Semantic Model from Wildfire Data Model, corresponding ontology instances are created, and the Jena Rules are applied for semantic

Data	Description	Format
Shelter Location	A list of nodes defined with System Data Model	XML
House Location	A list of nodes defined with System Data Model	XML
Vegetation Layer	Lists of nodes, ways, and relations defined with Open Street Map	OSM
Road Network Layer	Lists of nodes, ways, and relations defined with Open Street Map	OSM
Wildfire Pathway Simulation	A list of <WayPoint> are defined with the Pathway Data Model	WKT
Personal Information	An evacuee (person) has a departure time and a set-off location that is based on *House* location	OptaPlanner

Table 7.4: Data types used in the Wildfire Case Study.

graph transformation (both object and data properties are assigned and updated).

7.5.2 Simplified Wildfire Evacuation Ontology and Jena Rules

Simplified Wildfire Evacuation Ontology. Figure 7.10 shows a simplified wildfire evacuation ontology, its associated classes as well as object and data properties. The classes here include wildfire, person, house, vegetation, and shelter, which are highlighted in blue boxes and the object properties are to represent the relationships between pairs of classes. For instance, an object property of *isInVegetation* is assigned from the wildfire class to the vegetation class in order to identify whether a wildfire is intersecting a vegetation area at a certain time.

Jena Rules. Figure 7.11 shows a set of Jena Rules associated with the wildfire ontology shown in Figure 7.10. Rule 01 here is to propagate class hierarchy relationships; Rule 02 is to locate the wildfire location and vegetation information and findout if the wildfire

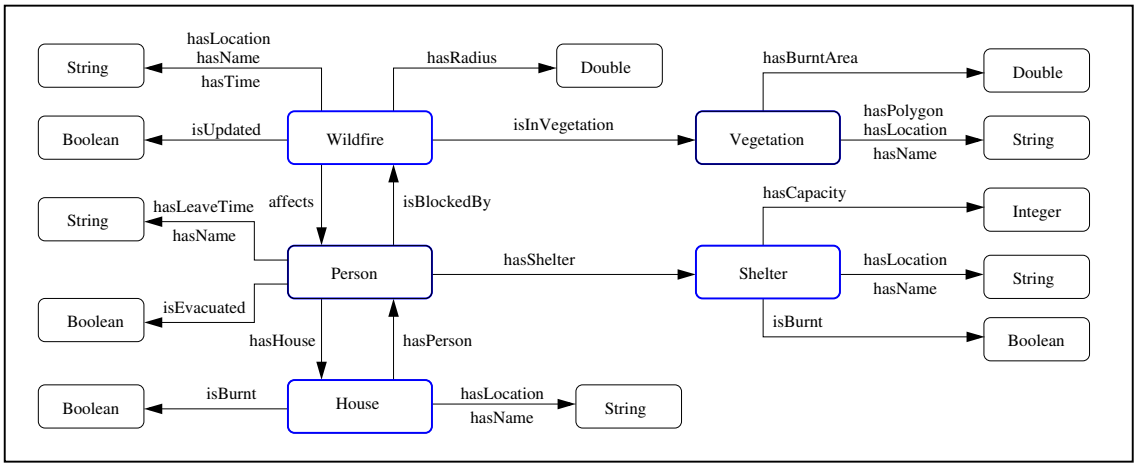


Figure 7.10: Simplified schematic for wildfire, person, house, vegetation and shelter ontology classes and properties.

```
@prefix wf:   <http://www.isr.umd.edu/wildfire#>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix geom: <http://www.isr.umd.edu/geometry#>.
@prefix xs:   <http://www.w3.org/2001/XMLSchema#>.

// Rule 01: Propagate class hierarchy relationships ...

[ rdfs01: (?x rdfs:subClassOf ?y), notEqual(?x,?y),
          (?a rdf:type ?x) -> (?a rdf:type ?y)]

// Rule 02:Evaluate relationship of wildfire and vegetation ...

[ Fire02: (?w rdf:type wf:Wildfire) (?w wf:hasLocation ?wl)
          (?w wf:hasRadius ?wr)(?v rdf:type wf:Vegetation)(?v wf:hasPolygon ?vp)
           getFireInVegetation(?wl,?vp,?t), equal(?t, "true"^^xs:boolean) ->
          (?w wf:isInVegetation ?v) print(?w wf:isInVegetation ?v ?t) ]

// Rule 03: Compute new fire radius based on wildfire and vegetation ...

[ Fire03: (?w rdf:type wf:Wildfire) (?w wf:hasLocation ?wl)
          (?w wf:hasRadius ?wr) (?w wf:radiusIsUpdated ?rupd) (?v rdf:type
           wf:Vegetation)(?v wf:hasPolygon ?vp)(?w wf:isInVegetation ?v)
           getNewFireRadius(?wl,?vp,?wr, ?newWR), notEqual(?wr, ?newWR) ->
          (?w wf:hasRadius ?newWR)] (?w wf:radiusIsUpdated "true"^^xs:boolean) ]
```

Figure 7.11: Abbreviated list of Jena Rules for wildfire evacuation.

intersects with any vegetation; Rule 03 is used to update the fire information as the fire needs to be updated in size and location when it intersects with a vegetation. A built-in function called *getNewFireRadius()* here is called for computing the new fire radius based on the current fire size and location as well as vegetation size and location.

7.5.3 Software Problem Setup

Figure 7.12 shows the previously obtained house and shelter location data, stored in *XML* format, are being added to the SystemDataModel. Then the getNodes() function is used to extract the actual <List> of <Node> individuals that store the coordinates of houses and shelters. The WildfireDataModel class is then created and the corresponding List<Node>s are stored through setShelters() and setHouses() classes.

```
String SDMinputFile01 = "data/umd-wildfire-houses.xml" ;
String SDMinputFile02 = "data/umd-wildfire-shelters.xml" ;

SystemDataModel sdm01 = new SystemDataModel();
sdm01.getData ( SDMinputFile01 );
SystemDataModel sdm02 = new SystemDataModel();

sdm02.getData ( SDMinputFile02 );

List<Node> houseLocations   = sdm02.getNodes();
List<Node> shelterLocations = sdm03.getNodes();

WildfireDataModel wfdm01 = new WildfireDataModel();

wfdm01.setName("Wildfire Evacuation Demo");
wfdm01.setShelters(shelterLocations);
wfdm01.setHouses(houseLocations);
```

Figure 7.12: Portion of code to set up the house and shelter locations with the wildfire data model.

Figure 7.13 indicates how an Open Street Map file is being handled in Java and how various layers of data are separated and used. Once the OpenStreetMapModel class is created, the corresponding data can be passed via using getData() method. In order to separate layers of information, the CompositeHierarchy class is used to define and obtain different information from OSM. The getWorkspace() method in OpenStreetMapModel can extract the corresponding information using its keyword (i.e., "highway") and then save it under the CompositeHierarchy class.

```
String OSMinputFile01 = "data/umd-town-information.osm" ;

// Create openstreetmap data model ...

OpenStreetMapModel osm = new OpenStreetMapModel();
osm.getData( OSMinputFile01 );
osm.shrinkFactor( new Quantity( 10.0 ) );
osm.smallDotSize( new Quantity( 30.0 ) );


// Extract domain-specific composite hierarchies  ...

CompositeHierarchy highwayLayer  = osm.getWorkspace( "highway"     );
CompositeHierarchy schoolLayer   = osm.getWorkspace( "school"      );
CompositeHierarchy streetLayer   = osm.getWorkspace( "residential" );
CompositeHierarchy buildingLayer = osm.getWorkspace( "building"    );
CompositeHierarchy paradiseLayer = osm.getWorkspace( "Town"        );
CompositeHierarchy amenityLayer  = osm.getWorkspace( "amenity"     );
CompositeHierarchy forestLayer   = osm.getWorkspace( "Forest"      );
```

Figure 7.13: Portion of code to extract useful information from OSM.

Figure 7.14 shows how the semantic model of wildfire evacuation is developed as well as how the Jena Rules are added and executed to transform the semantic graph. Here, JenaSemanticModel class is developed to store the domain ontology (including classes, object and data properties) and then a corresponding visitor class is created to temporarily

store the semantic graph. The WildfireDataModel class (with object "wfdm01") then accepts the corresponding visitor and populate its ontology. Finally, the rule class is fed through addRules() method in the JenaSemanticModel and using executeRules() method to complete the semantic graph transformation.

```
JenaSemanticModel wildfireSM = new JenaSemanticModel("Wildfire Semantic Model");
wildfireSM.loadOntology ("file:ontology/umd-wildfire.owl");
WildfireDataModelJenaVisitor wildfireSM_visitor =
                                                new WildfireDataModelJenaVisitor();
wildfireSM_visitor.setPassword("WildfireSMV");
wildfireSM_visitor.addSemanticModel( wildfireSM );
wfdm01.accept ( wildfireSM_visitor );

wildfireSM.addRules ( "rules/umd-wildfire.rules" );

wildfireSM.executeRules();
```

Figure 7.14: Assemble the wildfire semantic model, populate ontologies, and execute graph transformation.

After all the previous steps of storing data and transforming the semantic graphs, Figure 7.15 shows how the adjusted data are exported to the OptaPlanner software. After the corresponding output from the Wildfire Data Model and Simulation are transferred, the OptaPlanner starts optimizing using these data and generate plans that satisfy corresponding safety constraints.

```
wfdm01.exportToOptaPlanner();
wfdm01.exportOptaHouses();
wfdm01.exportOptaShelters();
wfdm01.convertCSVtoXML();
```

Figure 7.15: Assemble the wildfire semantic model, populate ontologies, and execute graph transformation.

7.6 Formulation of Planning Problem

7.6.1 Use Cases and Scenarios

In this section, the use cases and corresponding scenarios of each use case is described for the formulation of wildfire evacuations.

Use Case 1. Plan to leave before fire arrives.	
Scenario 1.	The evacuees escape on the same pathway.
Scenario 2.	The evacuees escape on different pathways with partially overlapping trajectories.
Scenario 3.	The evacuees escape on different pathways with no overlapping trajectories.
Use Case 2. Plan to leave just-in-time.	
Scenario 1.	The evacuees escape on the same pathway.
Scenario 2.	The evacuees escape on different pathways with partially overlapping trajectories.
Scenario 3.	The evacuees escape on different pathways with no overlapping trajectories.
Use Case 3. Plan to leave after fire is extinguished.	
Scenario 1.	The evacuees escape on the same pathway.
Scenario 2.	The evacuees escape on different pathways with partially overlapping trajectories.
Scenario 3.	The evacuees escape on different pathways with no overlapping trajectories.

Table 7.5: Use cases and scenarios for wildfire evacuation operations.

Table 7.5 shows the use cases and corresponding scenarios. The complexity of wildfire evacuation is well known and these use cases are used to describe general strategies that people take during evacuations. Planning for wildfire evacuation can be fundamentally

different to the airport taxiway operations because of the uncertainties, randomness, and social aspects of driving people to make different decisions.

Three use cases shown in Table 7.5 include planning to leave before fire arrives, planning to leave just-in-time and planning to leave after fire is extinguished. Other strategies can be taken as people make all kinds of decisions during emergency evacuations.

Use Case 1 shows the case that planning share occur before fire arrives. Three scenarios here include that when two evacuees escape on the same route, when two evacuees escape on different route options but part of the route is overlapped, and when two evacuees escape on different route and no overlapped road segments during evacuation. Planning to leave before fire arrives is definitely the safest way to avoid wildfire damage in any kind, however, it is not widely taken as an evacuation option for a fire prone area. This is mostly because false fire alerts were sent and people do not choose to evacuate until they see it themselves or hear to from someone they trust [68, 69].

Use Case 2 indicates the case that proper evacuation is adopted just in time. This is to say that people choose to evacuate when they visually detect fire and decide to evacuate to a nearest shelter location. Three scenarios under this use case are the same as before. Because of the undermined nature of human-beings, some people choose to leave when they see fire themselves.

Use Case 3 shows a completely different strategy where people choose to evacuate after the fire is extinguished. This is oddly an option for most people but some choose to evacuate afterwards because of late notice of fire or no access to vehicles or evacuation transportation [69]. Some even choose to stay without evacuating. Some late notices for

residents in fire prone area can occur because of the cell tower is down, or elderly people do not use smart phones to receive government notification messages, as well as many other unpredictable reasons [68].

For a real-world wildfire evacuation problem, the use cases can be many and that is because many factors in a natural disaster evacuation are unpredictable (or they change all the time). Fire use cases and scenarios can differ in many different ways even just to look at the different social-demographics that represent the people in town. However, common strategies can be treated as use cases and scenarios test the capability of evacuation route assignment with the constraint solver.

7.6.2 Requirements and Constraints

The requirements and constraints are derived based on the defined use cases as well as the various scenarios. Textual requirements are written by the requirement engineers to describe the functionality of a system in an unambiguous way. Once the requirements are written, the corresponding constraints of each requirement can be extracted. This is because the mathematical constraints can be embedded into the planning software as part of the optimization process.

7.6.3 Planning Objectives

Planning problems need planning objectives. Like any other OR problems, the objective function and mathematical constraints need to be defined ahead. For the disaster evacuation problem, the objectives include efficient planning for safe evacuation with

minimizing evacuation time, fatality and resource allocation (transportation, shelters, etc.) Instead of a mathematical problem that contains objective function, disaster evacuation with *OptaPlanner* do not have fixed objective functions since the purpose of planning is dynamic planning. *OptaPlanner* defines planning entities and planning variables which are associated with its DRL rules, implemented as hard and soft constraints. Then the variables that are defined under these planning entities can be used to formulate set of constraints for the planning problem. In the software architecture, the planning objectives are set to fulfill all, if possible, constraints as much as possible. This is mainly because that it uses hard and soft constraints where a scoring system is provided to support and compare between various sets of solutions.

7.7 Planning for Wildfire Evacuation

7.7.1 Problem Description

Figure 7.16 is a plan view of the planning problem for wildfire evacuation over an urban road network structure. The problem setup has 19 evacuees (and their house locations) and three shelter locations.

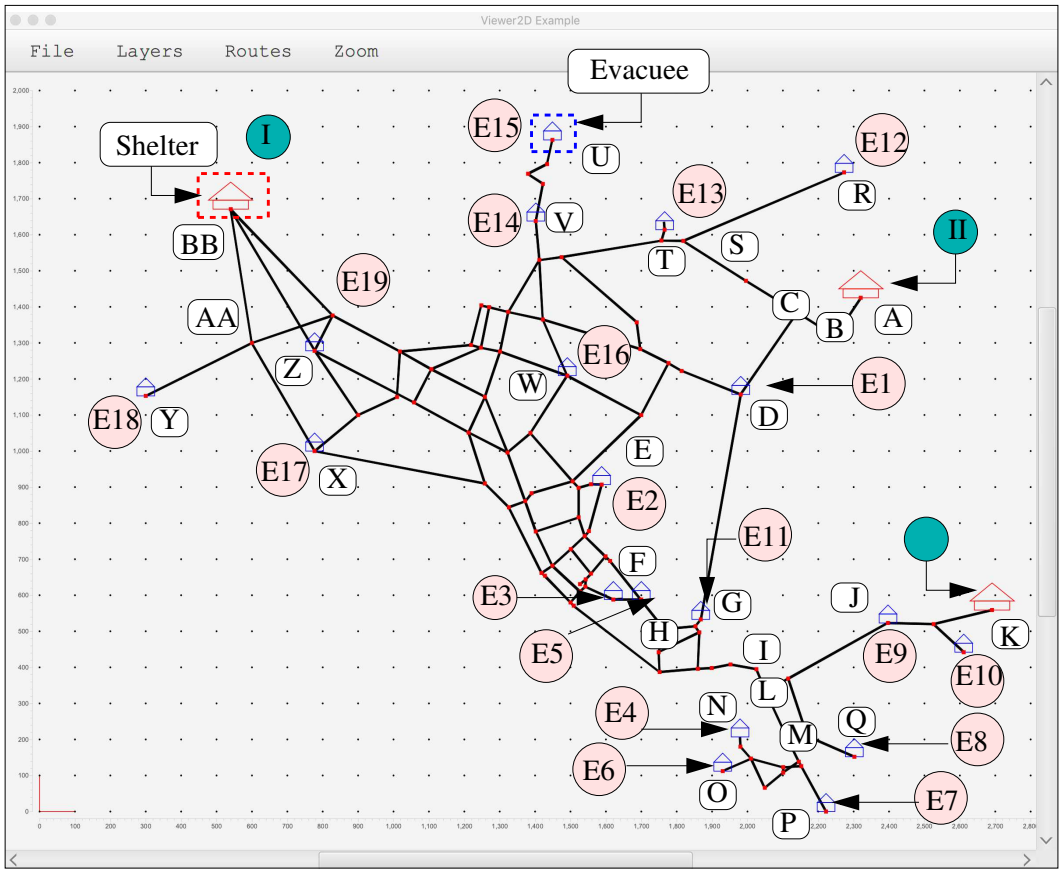


Figure 7.16: Evacuation planning on a urban road network structure.

In a departure from the small urban network problem (Chapter 2), the edges between the nodes are undirected which means that it allows two-way traffic. The road segments are shown as black edges, locations are red dots, house and shelter locations are drawn in

the map as well as a list of annotated nodes. In order to test the feasibility of the usage of OSM data with the real-time planning, we simplified the problem to a fixed amount of nodes, road segments, houses, shelters and evacuees. This helps to understand how the integrated framework can cope with OSM data (can be provided as separate layers of information) and how it handles an evacuation problem from a system's point of view.

7.7.2 Class Diagram

In order to design a planning mode for the wildfire evacuation problem, the planning entities that represent the input data need to be defined. In this case, these are evacuees, vehicles, houses, shelters and available roads. The domain model also helps to determine which classes are planning entities and what properties should be the planning variables. This process promotes planning performance, simplify constraints, and increase planning flexibility.

Figure 7.17 shows details of the class diagram for the wildfire evacuation problem. Here, the planning entities are defined as the evacuees and planning variables contain a several properties related to evacuee. Those are houses, shelters, vehicles, evacuation route options and locations which stores locations of a house that belongs to an evacuee and a vehicle which is assigned to an evacuee. Then, the vehicle has an evacuation route to take the evacuee to a shelter location. Some data properties of the classes `Evacuee` and `Vehicle` are shown in the diagram.

Table 7.6 addresses hard and soft constraints for the wildfire evacuation problem. Hard constraints include: a). the vehicle capacity limit which assures number of evacuee

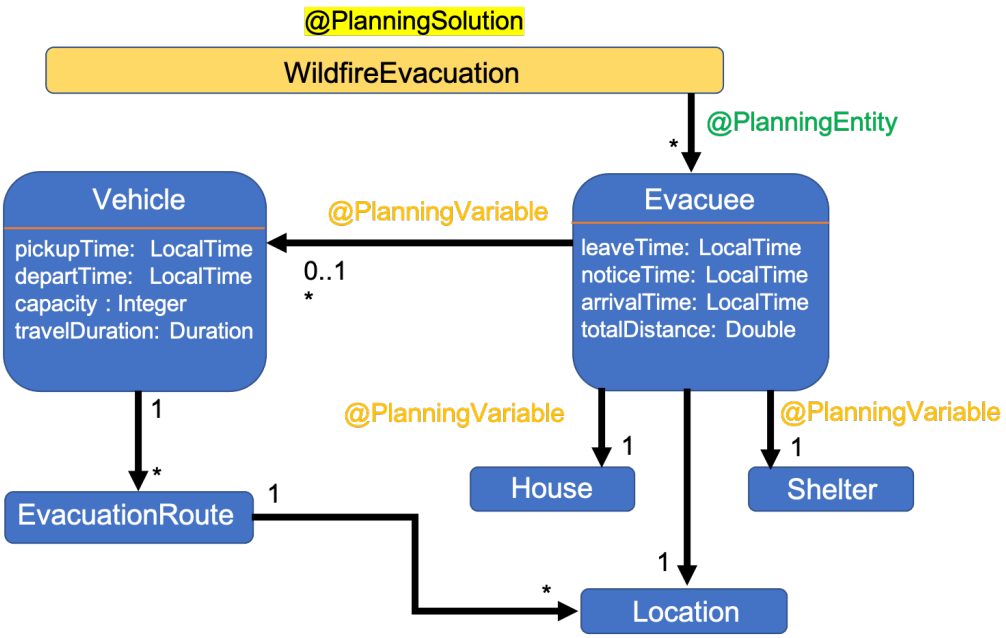


Figure 7.17: OptaPlanner Class Diagram: Wildfire Evacuation

in a vehicle cannot be more than its capacity limit; b). shelter capacity limit that checks the capacity of a shelter should not be exceeded; c). a road segment that is blocked by fire is treated to be an unavailable road; d). one vehicle is assigned to have one shortest path. This helps the algorithm to constantly update its shortest path and removes the duplicated pathways; and e). the path that a vehicle selected must be the least time consuming pathway. This helps the algorithm to change the shortest path to a pathway that takes less time for evacuation. Some soft constraints are also added. These are a). vehicle sharing allows an evacuee to pickup someone that needed help who does not have vehicle access or is not able to drive; b). early departure constraint reflects that an evacuee should evacuate as soon as they find out the fire; c). no evacuation shall be adopted if the circumstances do not allow one to evacuate. The soft constraints will affect the results but the hard constraints must be satisfied in order to have a feasible solution. The complication of a real-world wildfire evacuation scenario can have lots of hard and soft constraints to fulfill

therefore a wide range of differentiation can be represented.

Hard constraints.
a). Vehicle capacity limit: a vehicle cannot carry more evacuees than its capacity.
b). Shelter capacity limit: a shelter cannot take evacuees more than its capacity.
Soft constraints.
c). Early departure time: an evacuee shall leave the house as soon as the fire is noticed.
d). Shortest path: an evacuee shall take a shortest pathway that has the least congestion time.
e). Minimum travel duration: vehicle taken path takes the least time to evacuate.

Table 7.6: Hard and soft constraints for the Wildfire Evacuation Problem.

7.7.3 Evacuation Scoring Rules Modeled in Drools

This section explains detail of some hard and soft constraints from Table 7.6, which textitOptaPlanner uses for planning.

Figure 7.20 shows a hard constraint to set that the number of evacuees in a vehicle cannot exceed the capacity of the vehicle. For evacuation cases, this rule helps to understand by comparing two scenarios such as when no one wants to share vehicle versus everyone will share vehicle during evacuation. We can also use this to understand human behaviors during a disaster evacuation , for instance, how many people are willing to share and help others when they have empty seats in their cars. If everyone is willing to help and all empty seats are well allocated to the surrounding neighbors or people who need help, how efficient the evacuation process can be accelerated. Although, the constraint

rules are implemented for resource allocation and optimization purposes, these can also help to understand human behaviors through understanding the relationships between the meaning of constraints and the end populated results.

$$HS_{overall} = \sum_{i=1}^{n}(ShelterCapacity_i - ShelterEvacueeNum_i) + \sum_{j=1}^{m}(VehicleCapacity_j - VehicleEvacueeNum_j) \quad (7.2)$$

Here, the hard and soft constraints are implemented as part of the score calculation which is embedded with the planning software. Equation 7.2 shows the hard constraint score, denoted as $HS_{overall}$, where it demonstrate the feasible solutions from the planning solver. The overall score has to be greater than or equal to zero in order to demonstrate that the result is a feasible solution of the planning problem.

Equation 7.3 shows the soft constraint score, denoted as $SS_{overall}$, where it demonstrate the score of a feasible solution. This is implemented to show the difference between various feasible solutions and to demonstrate which is better.

$$SS_{overall} = \sum_{i=1}^{n}(EvacueeNoticeTime_i - EvacueeLeaveTime_i) + (-1) * \sum_{j=1}^{m}(EvacueeTotalDistance_j + EvacueePreviousDistance_j) + \sum_{k=1}^{m}(EvacueePlannedDuration_k - EvacueeTravelDuration_k) \quad (7.3)$$

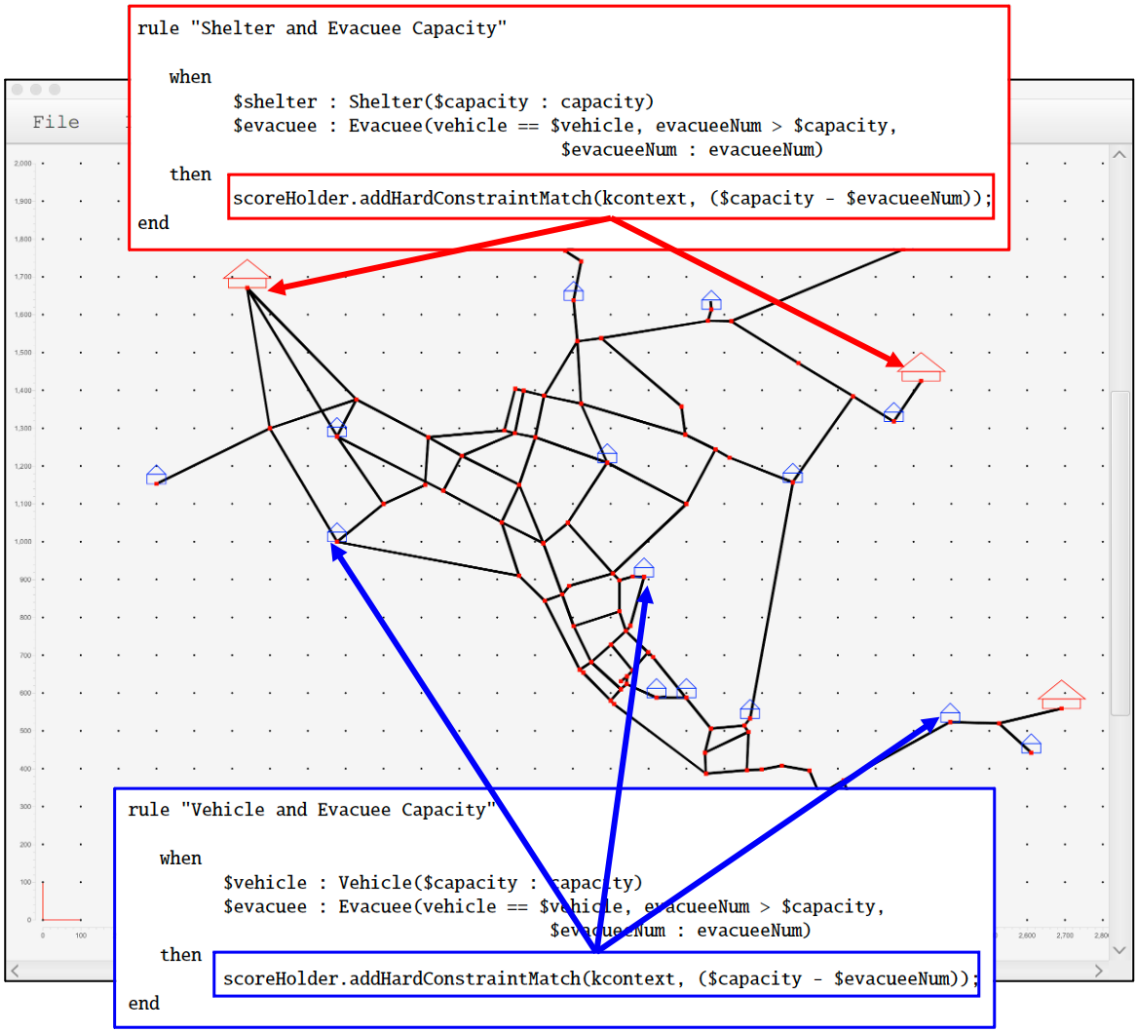


Figure 7.18: Small town with planning hard constraints view and results.

Constraint and Planning View for Wildfire Evacuation. Figure 7.18 shows the constraint and planning view for the wildfire evacuation problem (hard constraints). As mentioned above, the hard constraints are set up to consider shelter capacity and vehicle capacity for the evacuation problem. This way, the planning problem can focus on allocating people for evacuation as well as sheltering in order to make sure people are being taken care of.

Figure 7.19 shows the sample soft constraints for the evacuation problem, which

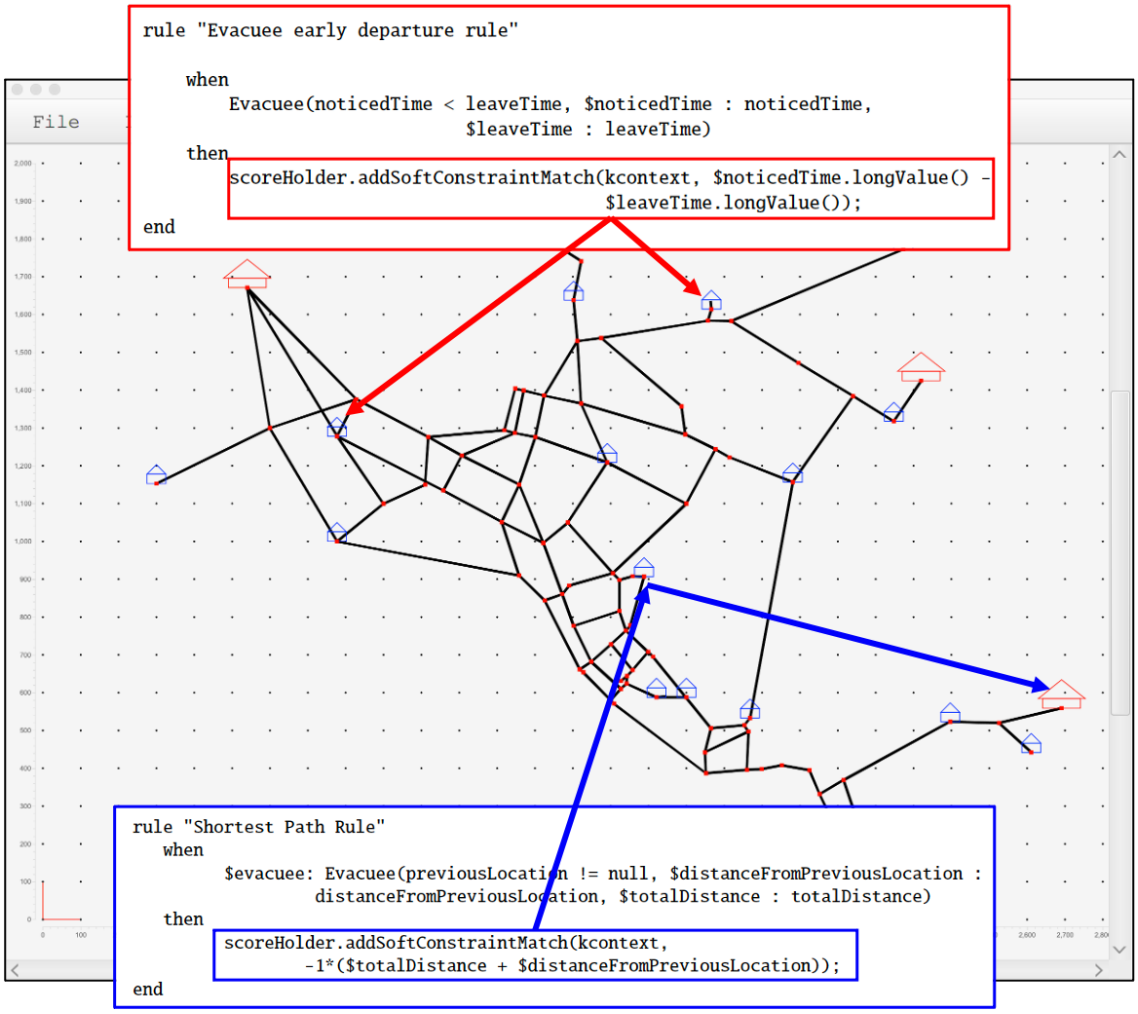


Figure 7.19: Small town with planning soft constraints view and results.

considers the early departure rule and shortest path rule over the course of evacuation. These rules are meant to check the overall performances of the system and to compare through the soft score calculation identifying best solutions out of all the feasible solutions.

Planning Constraint Details. Similarly as `Rule 01`, Figure 7.20 shows another hard constraint to limit the number of evacuees at a shelter location, where the capacity of shelter shall not be exceeded. This promotes people are being allocated equally to available shelter locations so that they can get help as soon and much as possible. It further helps to reduce congestion on road network while evacuation is on-going if shelter resources are

Abbreviated Fragment of Software Code:

```
// Using HardSoftScoreHolder class

global HardSoftScoreHolder scoreHolder;

// Rule 01: "vehicle and evacuee rule" ...

rule "Vehicle and Evacuee Capacity"

   when
         $vehicle : Vehicle($capacity : capacity)
         $evacuee : Evacuee(vehicle == $vehicle, evacueeNum > $capacity,
                                    $evacueeNum : evacueeNum)
   then
         scoreHolder.addHardConstraintMatch(kcontext, ($capacity - $evacueeNum));
end

// Rule 02: "shelter and evacuee rule" ...

rule "Shelter and Evacuee Capacity"

   when
         $shelter : Shelter($capacity : capacity)
         $evacuee : Evacuee(vehicle == $vehicle, evacueeNum > $capacity,
                                    $evacueeNum : evacueeNum)
   then
         scoreHolder.addHardConstraintMatch(kcontext, ($capacity - $evacueeNum));
end
```

Figure 7.20: Fragment of code for wildfire evacuation hard constraints.

being evenly distributed.

For the soft constraints, Figure 7.21 shows three soft constraints implemented for the wildfire evacuation problem. Rule 03 minimizes the difference between noticed time and actual leave time for evacuees. Evacuees should consider evacuating as soon as they found out about the fire. And this constraint adds higher penalties if evacuees make longer decision for whether to evacuate; Rule 04 sets a shortest path constraint based on the pathway an evacuee takes which contains a list of locations. It is set to be a soft constraint

Abbreviated Fragment of Software Code:

```
// Rule 03: "evacuee early departure" ...

rule "Evacuee early departure rule"

    when
        Evacuee(noticedTime < leaveTime, $noticedTime : noticedTime,
                              $leaveTime : leaveTime)
    then
        scoreHolder.addSoftConstraintMatch(kcontext, $noticedTime.longValue() -
                                           $leaveTime.longValue());
end

// Rule 04: minimize shortest path ...

rule "Shortest Path Rule"
   when
         $evacuee:Evacuee(previousLocation != null,$distanceFromPreviousLocation:
                  distanceFromPreviousLocation, $totalDistance : totalDistance)
   then
         scoreHolder.addSoftConstraintMatch(kcontext,
                 -1*($totalDistance + $distanceFromPreviousLocation));
end

// Rule 05: minimize travel duration ...

rule "Smallest travel duration rule"
   when
         $vehicle : Vehicle(plannedDuration < travelDuration, $plannedDuration :
                            plannedDuration, $travelDuration : travelDuration )
   then
         scoreHolder.addSoftConstraintMatch(kcontext,
                 $plannedDuration - $travelDuration);
end
```

Figure 7.21: Fragment of code for wildfire evacuation soft constraints.

to help make the travel distances shorter for all evacuees; Rule 05 compares the planned evacuation duration and actual travel duration of a vehicle, and it adds penalties if the actual travel duration is more than the planned travel duration.

7.7.4 Fire Sequences

Figure 7.22 shows the sequence of fire for the small urban town with respect to the nodes and edges. We can see that the fire moves and expands over the course of the four Figures (from left to right) with the time moving from t_k to t_{k+3}. To model the dynamic of wildfires, understanding its location and size, and its relationships with surrounding road network is important because the corresponding road information (blocked or available) can be identified for this urban town evacuation problem.

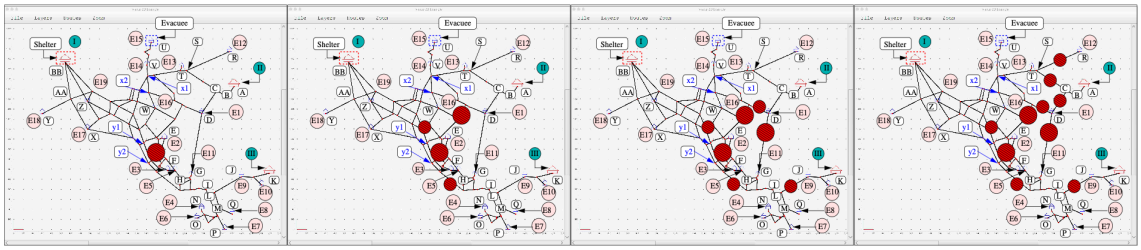


Figure 7.22: Small town evacuation with sequence of fire at various times.

7.8 Results and Discussion

As previously mentioned, the integrated framework consists of three major sections – computation of fire pathway and geometry, spatial-temporal analysis, and planning and evaluation for alternative evacuation routes (see in Figure 7.3). This section further discusses the preliminary results exported from the constraint solver where the evacuation routes and schedules are plotted for the nineteen evacuees (initial locations are assumed with the house locations).

Results. The examples of these the evacuation problem contain a list of house locations and terminal positions in order to test the connectivity of the framework (between two sides - Whistle and OptaPlanner). The additional spatial-temporal strategies are also discussed in this section such to illustrate the framework is capable of dynamically adjust its, evacuation and/or taxiway operational, plans when the optimizing simulation is running.

This section inserts the planning outcomes for the evacuation problem shown in Figure 7.16. Here, we are using four Evacuees *1, 2, 3 and 4* as well as a list of discrete nodes that are used in this analysis for illustration purposes. In addition, the house locations of these evacuees and three shelter locations are used as shown.

Figures 7.23 and 7.24 show the outputs of four evacuation routes for evacuees 1 to 4 respectively. The outcomes of the taxiways indicate that `Evacuee 1` uses $D \rightarrow C \rightarrow B \rightarrow A$ evacuation route with a cost of 10 (`totalDistance`); `Evacuee 2` uses $E \rightarrow F \rightarrow G \rightarrow I \rightarrow J \rightarrow K$ evacuation route with a cost of 29; `Evacuee 3` uses $F \rightarrow H \rightarrow I \rightarrow J \rightarrow K$ evacuation route with a cost of 23; `Evacuee 4` uses $N \rightarrow M \rightarrow$

$L \rightarrow J \rightarrow K$ evacuation route with a cost of 22. From the populated results for these four vehicles, we can see that `Vehicle 1` selected `Shelter II` to evacuate, and other three vehicles chose `Shelter III` to evacuate. It also contains lists of ordered nodes which are used for evacuation.

Figure 7.28 shows the times and positions for all 19 Evacuees with respect to Figure 7.16. The beginning locations of these evacuees and their house locations are appeared at `time 0`. The location-time results also show where all evacuees are being evacuated eventually and when they arrive at shelter locations. We can also find from the results that the `Evacuee 2` and `Evacuee 3` have waited in the beginning for a few intervals of time . This could be caused by several reasons that are related to constraints embedded in the planner. For example, Rule 05 is defined to minimize travel duration which makes evacuee to evacuate at a later time because of the possible congestion that could occur if one evacuates early. The planning functionality is flexible in a sense that different simulation with the exact same input data sets and constraints may occur to different set of results. Because of the flexibility of satisfying the soft constraints, an earlier termination of the program can provide a result that might not be optimal.

Semantics Trigger Replanning for Evacuation. Figure 7.27 indicates some changes occurred in the populated semantics where the data properties of Person01's evacuation route and the status of whether the original evacuation route is blocked are changed. The populated semantics are capable of telling if an evacuee's original evacuation plan is still feasible and if not, the real-time planning takes this semantic decision from the semantic model and replan to find a new feasible evacuation route considering the current state of

Abbreviated Fragment of Software Code:

```
<Vehicle id="1">
  <capacity>4</capacity>
  <totalDistance>10</totalDistance>
  <house class="House" reference="D"/>
  <shelter class="Shelter" reference="A"/>
  <EvacuationRoute id="1">
    <nextLocation id="1">
      <location class="Node" reference="C"/>
      <weight>6</weight>
      <nextLocation id="2">
        <location class="Node" reference="B"/>
        <weight>2</weight>
        <nextLocation id="3">
          <location class="Node" reference="A"/>
          <weight>2</weight>
          <evacuee reference="1"/> </nextLocation>
        <evacuee reference="1"/> </nextLocation>
      <evacuee reference="1"/> </nextLocation>
 </EvacuationRoute>
</Vehicle>
<Vehicle id="2">
  <capacity>4</capacity>
  <totalDistance>29</totalDistance>
  <house class="House" reference="E"/>
  <shelter class="Shelter" reference="K"/>
  <EvacuationRoute id="1">
    <nextLocation id="1">
      <location class="Node" reference="F"/>
      <weight>6</weight>
      <nextLocation id="2">
        <location class="Node" reference="G"/>
        <weight>5</weight>
        <nextLocation id="3">
          <location class="Node" reference="I"/>
          <weight>5</weight>
          <nextLocation id="4">
            <location class="Node" reference="J"/>
            <weight>6</weight>
            <nextLocation id="5">
              <location class="Node" reference="K"/>
              <weight>7</weight>
              <evacuee reference="2"/> </nextLocation>
            <evacuee reference="2"/> </nextLocation>
          <evacuee reference="2"/> </nextLocation>
        <evacuee reference="2"/> </nextLocation>
      <evacuee reference="2"/> </nextLocation>
 </EvacuationRoute>
</Vehicle>
```

Figure 7.23: Fragment of OptaPlanner output for Evacuees 1 and 2.

Abbreviated Fragment of Software Code:

```
<Vehicle id="3">
  <capacity>4</capacity>
  <totalDistance>23</totalDistance>
  <house class="House" reference="F"/>
  <shelter class="Shelter" reference="K"/>
  <EvacuationRoute id="1">
    <nextLocation id="1">
      <location class="Node" reference="H"/>
      <weight>4</weight>
      <nextLocation id="2">
        <location class="Node" reference="I"/>
        <weight>6</weight>
        <nextLocation id="3">
          <location class="Node" reference="J"/>
          <weight>6</weight>
            <nextLocation id="4">
              <location class="Node" reference="K"/>
              <weight>7</weight>
              <evacuee reference="3"/> </nextLocation>
          <evacuee reference="3"/> </nextLocation>
        <evacuee reference="3"/> </nextLocation>
      <evacuee reference="3"/> </nextLocation>
 </EvacuationRoute>
</Vehicle>
<Vehicle id="4">
  <capacity>4</capacity>
  <totalDistance>22</totalDistance>
  <house class="House" reference="N"/>
  <shelter class="Shelter" reference="K"/>
  <EvacuationRoute id="1">
    <nextLocation id="1">
      <location class="Node" reference="M"/>
      <weight>4</weight>
      <nextLocation id="2">
        <location class="Node" reference="L"/>
        <weight>3</weight>
        <nextLocation id="3">
            <location class="Node" reference="J"/>
            <weight>8</weight>
            <nextLocation id="4">
              <location class="Node" reference="K"/>
              <weight>7</weight>
            <evacuee reference="4"/> </nextLocation>
          <evacuee reference="4"/> </nextLocation>
        <evacuee reference="4"/> </nextLocation>
      <evacuee reference="4"/> </nextLocation>
 </EvacuationRoute>
</Vehicle>
```

Figure 7.24: Fragment of OptaPlanner output for Evacuees 3 and 4.

Abbreviated Fragment of Software Code:

```
<Vehicle id="1">
  <capacity>4</capacity>
  <totalDistance>44</totalDistance>
  <house class="House" reference="D"/>
  <shelter class="Shelter" reference="A"/>
  <EvacuationRoute id="2">
    <nextLocation id="1">
      <location class="Node" reference="X1"/>
      <weight>10</weight>
      <nextLocation id="2">
        <location class="Node" reference="X2"/>
        <weight>10</weight>
        <nextLocation id="3">
          <location class="Node" reference="X3"/>
          <weight>10</weight>
          <nextLocation id="4">
            <location class="Node" reference="C"/>
            <weight>10</weight>
            <nextLocation id="5">
              <location class="Node" reference="B"/>
              <weight>2</weight>
              <nextLocation id="6">
                <location class="Node" reference="A"/>
                <weight>2</weight>
                <evacuee reference="1"/> </nextLocation>
              <evacuee reference="1"/> </nextLocation>
            <evacuee reference="1"/> </nextLocation>
          <evacuee reference="1"/> </nextLocation>
        <evacuee reference="1"/> </nextLocation>
      <evacuee reference="1"/> </nextLocation>
 </EvacuationRoute>
</Vehicle>
```

Figure 7.25: OptaPlanner Output for the New Evacuation Route of Evacuee 1.

Abbreviated Fragment of Software Code:

```
<Vehicle id="2">
  <capacity>4</capacity>
  <totalDistance>53</totalDistance>
  <house class="House" reference="E"/>
  <shelter class="Shelter" reference="K"/>
  <EvacuationRoute id="2">
    <nextLocation id="1">
      <location class="Node" reference="Y1"/>
      <weight>10</weight>
      <nextLocation id="2">
        <location class="Node" reference="Y2"/>
        <weight>10</weight>
        <nextLocation id="3">
          <location class="Node" reference="F"/>
          <weight>10</weight>
          <nextLocation id="4">
            <location class="Node" reference="G"/>
            <weight>5</weight>
            <nextLocation id="5">
              <location class="Node" reference="I"/>
              <weight>5</weight>
              <nextLocation id="6">
                <location class="Node" reference="J"/>
                <weight>6</weight>
                <nextLocation id="7">
                  <location class="Node" reference="K"/>
                  <weight>7</weight>
                  <evacuee reference="2"/> </nextLocation>
                <evacuee reference="2"/> </nextLocation>
               <evacuee reference="2"/> </nextLocation>
              <evacuee reference="2"/> </nextLocation>
            <evacuee reference="2"/> </nextLocation>
          <evacuee reference="2"/> </nextLocation>
        <evacuee reference="2"/> </nextLocation>
      <evacuee reference="2"/> </nextLocation>
 </EvacuationRoute>
</Vehicle>
```

Figure 7.26: OptaPlanner Output for the New Evacuation Route of Evacuee 2.

edges (and whether it is blocked).

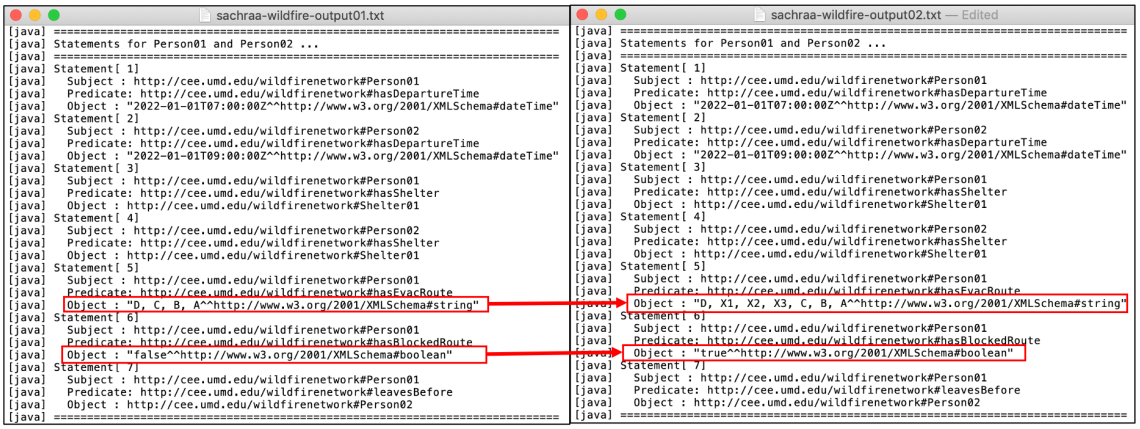
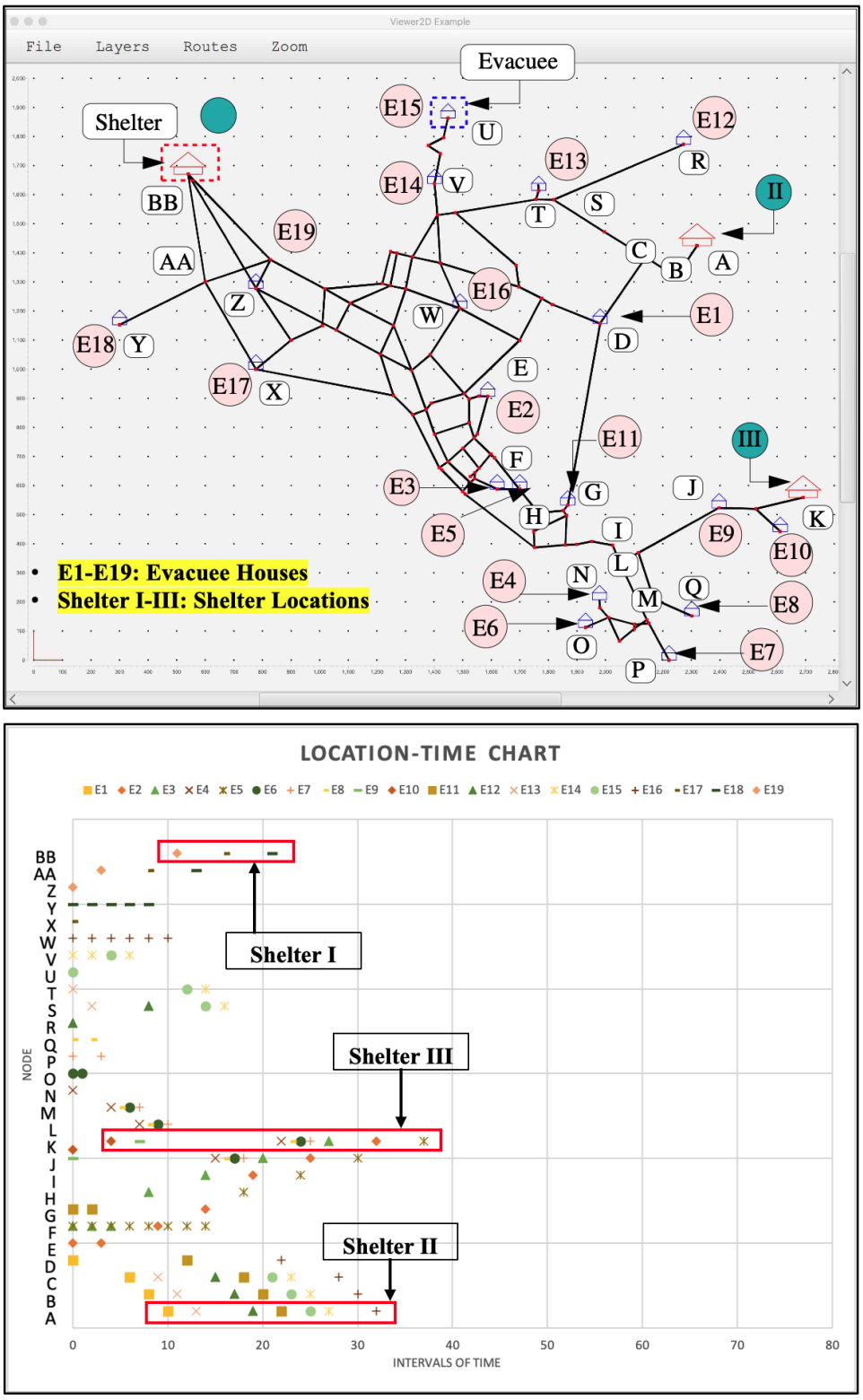


Figure 7.27: Changes in semantic representation for planning and replanning evacuation.

Evacuation Routes Altered for Evacuee I and II. Figure 7.22 shows a sequence of road segments that are blocked by fire (indicated with red circles). The additional nodes of `X1, X2, X3` and `Y1, Y1` are introduced for the illustration purposes. We can see that the evacuation routes for Evacuees `I` and `II` must change because fire has blocked the edges between nodes `C, D` and `E, F`. The new routes for the two evacuees shall be: (1). Evacuee `I` uses $D \rightarrow X1 \rightarrow X2 \rightarrow X3 \rightarrow C \rightarrow B \rightarrow A$; (2). Evacuee `II` uses $E \rightarrow Y1 \rightarrow Y2 \rightarrow F \rightarrow G \rightarrow I \rightarrow J \rightarrow K$. This is also indicated in Figure 7.27 where the semantic is able to capture replanning is required for Evacuee `I`, and the before and after semantic graph transformations are also shown within the red boxes. Also, the corresponding `OptaPlanner` evacuation routes of these two evacuees are populated in Figures 7.25 and 7.26. And the new set of results for all 19 evacuees are shown in Figure 7.29. Because of the fire interruption, the evacuation routes and plans have changed compare to the ones shown in Figure 7.28. Comparing the two sets of results, the significant delays for several evacuees are shown because of the fire interruption.

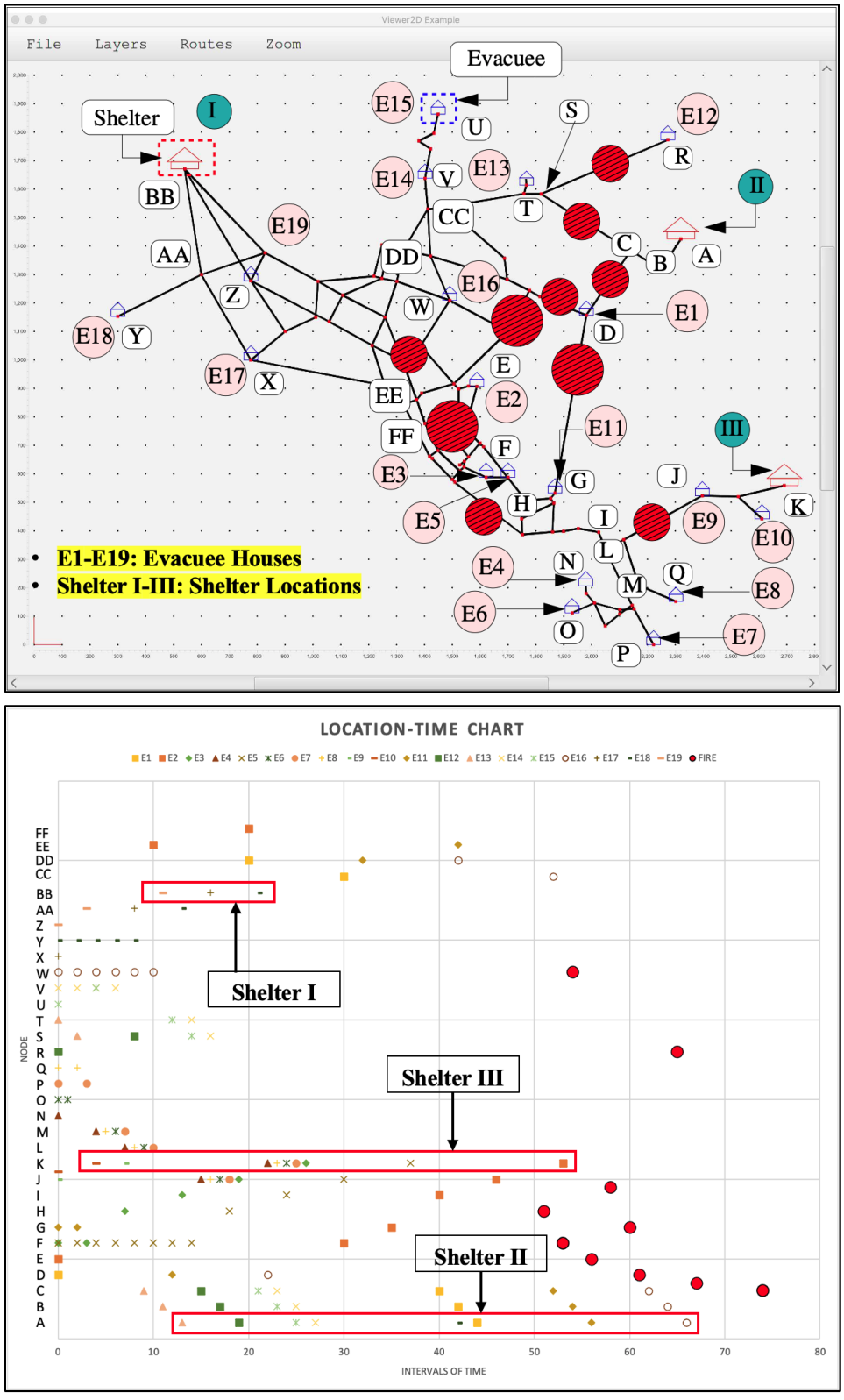


Scenario: Before obstruction assessment - wildfire evacuation of urban residents E1-E19 to Shelters I-III.

Figure 7.28: Small town evacuation planning view and results.

Figure 7.29 shows a small town evacuation problem where there are 19 evacuees are located. The results on the right-hand-side of the image indicates that all evacuees are successfully evacuated to one of the three shelter locations. In the results, the final locations (arrived shelters) of evacuees are boxed in red rectangles. In addition, the fire locations are plotted as red circles where it does not intersect any evacuee's evacuation route. This indicates that the planning is capable of understanding the original planning data, insert with semantic representation, and once the semantic determines whether an evacuation route is blocked, the replanning starts based on the changes found in the semantics. Details of the results can be viewed in Table 7.7.

Semantic Graph Transformation. Figure 7.30 shows an example of the semantic graph transformation for the relationships of evacuees. Here, two evacuees - *Evacuee01* and *Evacuee02* evacuating to the same shelter *Shelter01* are indicated. The left-hand-side shows the initial scheduled arrival times (with *arrivesAt*) and planned pathways (with *hasPathway*) for the two evacuees, however, semantic captures that both evacuees have a boolean data property *isBlocked* is *true*. In addition, an object property of *isBlockedBy* is assigned from these evacuees to a *WildfireX*, meaning that the current scheduled evacuation routes are blocked by fire and reschedule is required. Also, another data property of *isEvacuated* shows as *false* which illustrates that the current evacuation statuses of these evacuees. Because of these new knowledge, the modeling architecture decides to replan showing on the right-hand-side, as the updated plans. The green text-highlights show the updated data properties where both evacuees have a new pathway and a scheduled arrival time to avoid the road blockage. Eventually, when the evacuees are being successfully



Scenario: Just-in-time evacuation of urban residents E1-E19 to Shelters I-III.

Figure 7.29: Small town evacuation replanning view and results.

evacuated to a shelter, the *isEvacuated* is switched to *true* and *isBlocked* is *false* showing that these evacuation routes are not being blocked. The previous object property of *isBlockedBy* from evacuees to wildfire individual is removed since their evacuation routes are altered and avoided the blockage.

Discussion. The three strategies (or scenarios) for evacuation are shown in Figures 7.7, 7.8 and 7.9. The tentative analysis for a small town of nineteen house locations are added in order to test the capability of the framework. Three strategies here include evacuation before fire arrives, just in-time and after fire is extinguished. In these three strategy plannings, the house locations of planning evacuees are set to be the same as well as simulation of fire dynamics. The results generally align with the three strategies that early evacuation is shown in the blue line indicated in the Figure, the late evacuation (waiting for hours after fire extinguishes) and the in-time evacuation, which assure people evacuate before fire blocks any roads.

The integrated framework of real-time planning and multi-domain semantic modeling enables planning for a small wildfire evacuation in real-time to avoid schedule conflicts, pathway blockages as well as shelter assignment (considered as resource allocation.) Although, the limited number of evacuees are being used for this analysis, the framework shows the dynamic planning feature for wildfire evacuation supported with the knowledge-based decision-making. This proposed software architecture is scalable for modeling urban problems since the low-level details of the urban network (e.g., data models) are embedded completely separate from the software architecture. In addition, the semantic rules and ontologies as well as the planning problem formulation are also in-

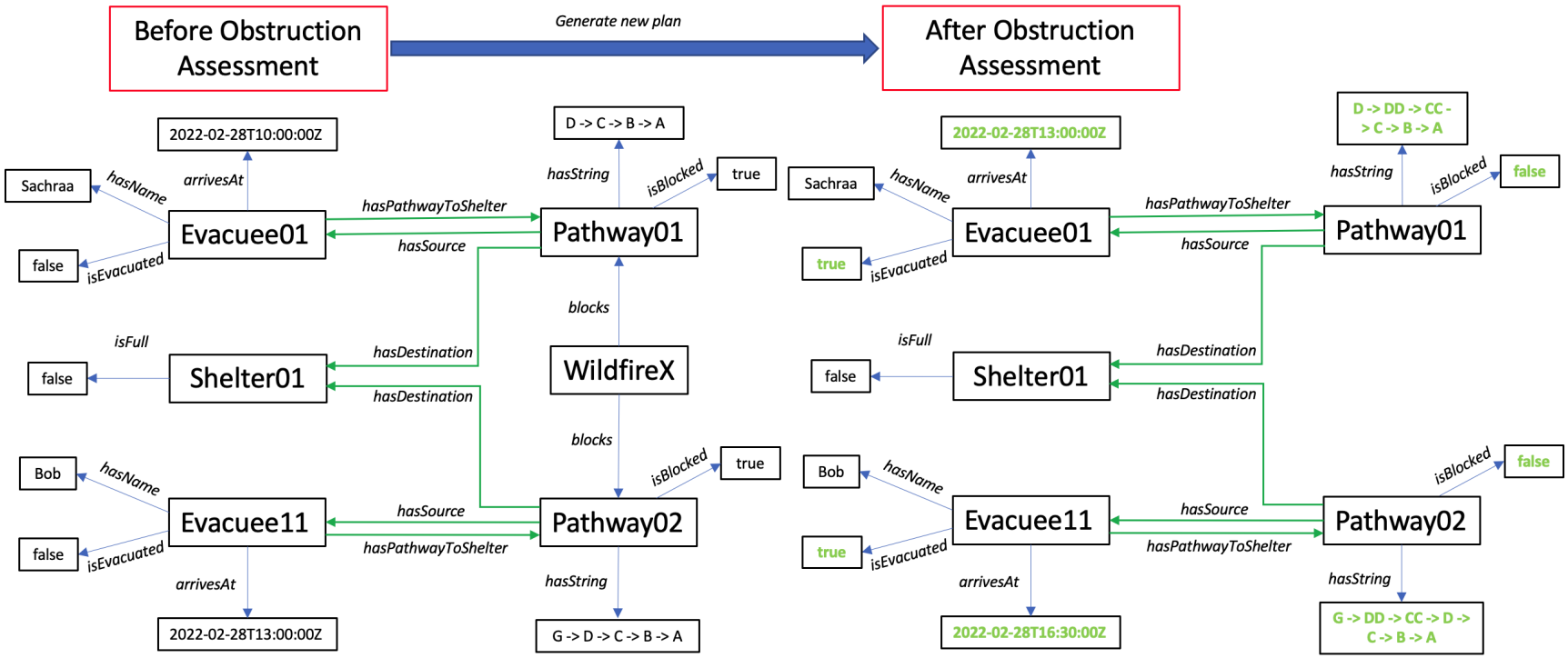


Figure 7.30: Semantic graph transformation for wildfire evacuations

Evacuee	Pathway (before)	Shelter (before)	Score (before)	Pathway (after)	Shelter (after)	Score (after)
Evacuee01	D, C, B, A	II	90	D, DD, CC, C, B, A	II	56
Evacuee02	E, F, G, I, J, K	III	68	E, EE, FF, F, G, I, J, K	III	47
Evacuee03	F, H, I, J, K	III	73	F, H, I, J, K	III	73
Evacuee04	N, M, L, J, K	III	78	N, M, L, J, K	III	78
Evacuee05	F, H, I, J, K	III	63	F, H, I, J, K	III	63
Evacuee06	Q, M, L, J, K	III	76	Q, M, L, J, K	III	76
Evacuee07	P, M, L, J, K	III	77	P, M, L, J, K	III	77
Evacuee08	Q, M, L, J, K	III	93	Q, M, L, J, K	III	93
Evacuee09	J, K	III	96	J, K	III	96
Evacuee10	J, K	III	78	J, K	III	44
Evacuee11	G, D, C, B, A	II	81	G, D, DD, CC, C, B, A	II	81
Evacuee12	R, S, C, B, A	II	87	R, S, C, B, A	II	87
Evacuee13	T, S, C, B, A	II	73	T, S, C, B, A	II	73
Evacuee14	V, T, S, C, B, A	II	75	V, T, S, C, B, A	II	75
Evacuee15	U, V, T, S, C, B, A	II	68	U, V, T, S, C, B, A	II	44
Evacuee16	W, D, C, B, A	II	68	W, D, DD, CC, C, B, A	II	44
Evacuee17	X, AA, BB	I	84	X, AA, BB	I	84
Evacuee18	Y, AA, BB	I	79	Y, AA, BB	I	58
Evacuee19	Z, AA, BB	I	89	Z, AA, BB	I	89

Table 7.7: Summary of results for wildfire evacuation modeling (before and after obstruction assessment).

tegrated separately from the main framework. This promotes the flexibility of researchers analyzing urban problems and customizes problem settings with ones own set of data and knowledge packages. In real-world evacuation problem, it is much more complicated than what we have described here because human behaviors are extremely difficult to predict, especially when it comes with disaster planning. In fact, most people do not follow public guidelines or strategies since disaster evacuation can be a life threatening event. The traditional wildfire evacuation approaches are lack of using systematic view to understand systems dependencies as well as decision-making from a high-level awareness while majority of the studies focus on certain aspect for evacuation (e.g., departure timing, location modeling, social-demographic influence.) The essence of this framework is to support real-time planning, coupled with real-time planning, which is very much needed for an evacuation problem with the knowledge support framework. A current limitation of the framework is lack of real-world big data against which evacuation strategies can be evaluated and guided.

Chapter 8: Conclusion and Future Research

8.1 Conclusion

Summary of Work. This dissertation promotes a new integrated software architecture that enhances the urban domain safety, through the use of semantic web technologies (ontologies, rules and reasoning) coupled to OptaPlanner. This combination of technologies supports event-driven system simulation. The participating domain-neutral (e.g., spatial, temporal, physical-unit, graph) ontologies and rules and domain-specific (e.g., airport taxiway operations) ontologies and rules represent knowledge essential for decision making associated with planning and replanning operations. The reasoning rules are developed with Apache Jena; the data-driven semantic graph transformations derive new knowledge representations.

Summary of the Contributions. The contributions of this dissertation are as follows:

1. Constructed a knowledge-based multi-domain semantic modeling framework for safety urban operations where domain-specific and domain-neutral ontologies and Jena rule sets are created with Jena Semantic Model and external data are stored and populated with System Data Model and OSM Model. This framework is capable of merging urban information and data with semantic representation, which promotes

decision-making via events capturing mechanisms.

2. The integration of semantic modeling and real-time planning alters decision-makings for a urban domain both supported with understanding the relationships behind the physical models and plans are being generated real time. The interactions between the two sides allow to use semantic modeling and reasoning mechanisms to make decisions at certain moments (e.g., if replanning is required), and adopt planner to replan if semantic models capture any failures or losses that require the replanning.

3. Investigated and explored the conceptual framework through three urban applications: small urban network shortest path problem (see Chapters 2 and 5), airport taxiway operations (see Chapter 6) and wildfire evacuation (see Chapter 7). It integrates the various participating domains of data into the domain specific data model (e.g., airport operations data model) in which they are helped by optimized planning and supported for safety assurance.

8.2 Future Research

The future work can be extended to indicate various forms of urban uncertainties for these three urban applications. The wildfire evacuation and airport taxiway operations include uncertainties that can be related to humanbehaviors, weather conditions, and many other unpredictable factors. Plus, knowledge comes with deficiencies commonly constructed by humans where uncertainty can also be explained with knowledge incompleteness caused by deficiencies among acquired knowledge [14]. The uncertainties can

also be embedded and implemented using constraints in the planning tool which controls overall outcomes for the constraint solver. The score calculation system can be adopted further as the evaluation of various forms of uncertainty, even if it is only used to compare solutions. Overall, the aspect of knowledge deficiencies with uncertainty can be further studied.

Furthermore, in order to have a better visualization of both spatial and temporal illustrations on urban safety operations, we will project urban problems onto a 3-dimensional visualization plotting with both space and time. This will help researchers to understand the importance of modeling time and space, especially in the context of civil engineering which it regularly involves these domains to secure for system's safety.

Ultimately, the approach in this dissertation will be used in a real-world urban planning problem and application. The challenges include the extraction of Jena rules of the specific urban problems, accessing most up-to-date data which constantly changes over time, and embedding, extraction and implementation of OptaPlanner constraints. Generally speaking, semantic modeling framework with ontologies and Jena rules works with various sources of data perfectly from the Internet of Things (IOT). On the other hand, the web-based resources can send data to the multi-domain semantic framework. For the future of smart urbanization, the integration of web resource data and semantic models will provide more semantically meaningful data that are supported by smart devices in our urban areas.

8.2.1 Path Planning and Trajectory Adjustment

Path Planning and Trajectory Adjustment. Figure 8.1 shows the path planning and trajectory adjustment for airport operations. The thick black lines indicate the airport main runways, thin black lines show the terminal buildings and red lines indicate the taxiway trajectories. The outline of the airport footprint is extracted from the Open Street Map (OSM) where the layers of information can be extracted and plotted separately. This helps to understand the geographical relationships among various physical entities. In addition, the blue dots show the holding positions that are on the taxiways and a dashed blue lines show the pathway from a terminal (source) to the final holding position which is right before taking-off on a runway. Besides airplane taxiing operations, maintenance (or other types of obstructions) can also occur at any time at the airport regular operations. This constantly affects the scheduled pathways for airplanes or associated vehicles) and initiates trajectory adjustment in order to assign new pathways that are not obstructed. The dynamic planning capability of physical systems is supported by the semantic modeling approach which is shown at the bottom of the Figure. The `Semantic Model Interface`, on the right-hand-side, connects the physical world to the semantic graphs (or knowledge representation behind the physical models). Observation from the physical environment sends events to semantic graphs. And after the semantic graphs are inserted with individuals derived from the physical models, the event-driven behavior mechanisms will drive the transformation of the semantic graphs, which can respond to external events and update associated information in the semantic graphs from the physical systems in order to provide further

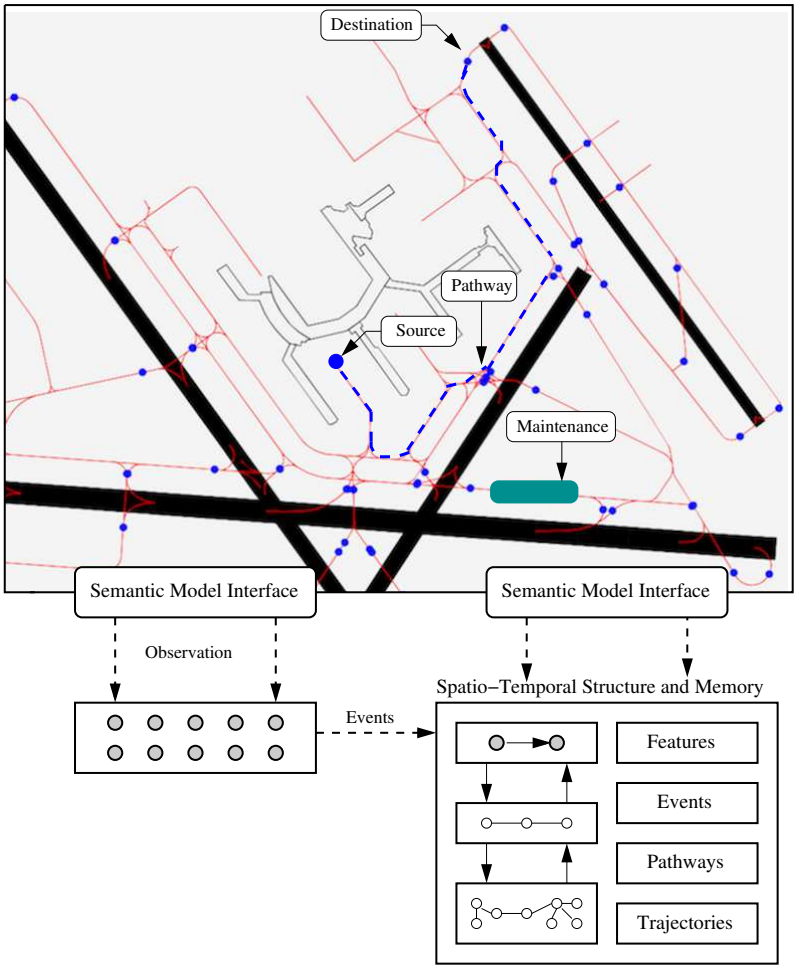


Figure 8.1: Path planning and trajectory adjustment.

decision-making for possible trajectory adjustments.

For potential extensible future work, we recognized that the Figure 8.1 shows the connection between the semantic modeling and reasoning mechanisms with the aircraft taxiing path planning and trajectory dynamic adjustment when it comes to a dynamic physical system. Semantic model interface allows the observation on the changes occur in the systems, using its knowledge-based framework and assigns relationships among physical entities as well as updates their attributes. Events are detected from the physical systems and using the other participating domain knowledge, both from domain-specific

and domain-neutral domains, the interface is capable of detecting and updating changes, and making dynamic decisions based on the changes discovered. These potential features of dynamic path planning coupled with semantic modeling framework shall be further investigated.

8.2.2 Urban Safety System Components and Architectures

The urban system's safety reveals components involving multiple physical and engineering domains, operating across multiple time instants and intervals, and dynamically interacting with surrounding environments which contain physical components that constantly change over time. In the context of urban operations, individual behavior of the physical components and surrounding systems in urban settings can be expressed by `Urban Agents` and `Urban Environments`. Future work will explore the interactions of the integrated framework proposed in this dissertation with urban safety systems supported by behavioral modeling through urban agents and environments. We categorize the components of urban safety systems as the following:

a) *Urban Environments.* These are the environments that are provided as platforms and bases for agents to communicate and interact. Each of these is capable of dynamically adjust based on the changes on the urban agents.

b) *Urban agents.* They act as dynamic entities in all urban settings. Interacting agents can involve interactions of same type of agents, or different types of agents. For instance, a pedestrian agent can interact with another pedestrian agent, as well as vehicle agent, traffic control agent and so on. All agents contain necessary

parameters that are used together to measure safety of the systems.

8.2.3 Urban System Agent Communications and Interactions

Future research can also be extended to secure urban system safeties through effective communications and interactions between different urban system agents. The interacting systems are also required to adjust its status and behavior based on interactions among these communicating urban entities.

Below contains a list of aspects we will focus on the communications and interactions between various urban entities (agents) to extend our work in the future:

- *1). Two-way interactions between the urban agents and the involved environment.* The interactions between individual agents, environments and between agents and environments are significant reasons to make the changes in urban systems dynamic. Understanding their relationships with the additional support of knowledge-based representation can accurately detect events in the urban systems and respond to the individual agents and environments in the systems.

- *2). Distributed system components.* In the current urban settings, distributed system components are constantly involved. The Cyber Physical Systems (or CPS) can be used to handle a multiplicity of physics and distributed components with concurrent behaviors [44, 94, 144]. A small network such as a LAN (local area network) or a much larger network such as the Internet can both be spanned with the distribution of components. Some links among the platforms will not go through the wireless network. This helps to define and connect the urban agents and environments,

and their distributed system components that are required to know to secure urban safeties.

- *3). Embedded computational platforms.* With the increasing complexity in engineering systems, the physical components involved in these systems can have some embedded computational capabilities. With the support of using urban agents and environments, the physical components can be represented by them, and their computational capabilities can realize further interactions among the agents and environments in the systems.

Chapter A: Tiny Small Urban Network System

This appendix contains abbreviated descriptions of the tiny urban network system as modeled in: (1) The System Data Model, and (2) Print Urban Network System Data Model.

A.1 System Data Model Representation (TinyUrbanNetwork.xml)

```
<?xml version="1.0" encoding = "UTF-8"?>
<SystemDataModel author = "Sachraa Borjigin" date = "2022-01"  source = "UMD">

  <!-- ========================================== -->
  <!-- UrbanNetwork modeling parameters ...       -->
  <!-- ========================================== -->

  <attribute key = "Units"              value =    "m" />
  <attribute key = "Pattern"            value =    "1" />
  <attribute key = "Demand Multiplier"  value =  "1.0" />
  <attribute key = "Tolerance"          value = "0.01" />

  <!-- ====================================== -->
  <!-- Network coordinates ...                -->
  <!-- ====================================== -->

  <node ID = "001" x =   "0.0" y =   "0.0" type="Point"/>
  <node ID = "002" x =   "0.0" y = "100.0" type="Point"/>
  <node ID = "003" x =   "0.0" y = "200.0" type="Point"/>
  <node ID = "004" x = "100.0" y = "200.0" type="Point"/>
  <node ID = "005" x = "100.0" y = "100.0" type="Point"/>
  <node ID = "006" x = "100.0" y =   "0.0" type="Point"/>
  <node ID = "007" x = "200.0" y = "100.0" type="Point"/>
  <node ID = "008" x = "200.0" y = "200.0" type="Point"/>
  <node ID = "009" x = "300.0" y = "200.0" type="Point"/>
  <node ID = "010" x = "300.0" y = "100.0" type="Point"/>
```

```
<node ID = "011" x = "300.0" y =   "0.0" type="Point"/>
<node ID = "012" x = "400.0" y = "200.0" type="Point"/>
<node ID = "013" x = "400.0" y = "100.0" type="Point"/>
<node ID = "014" x = "400.0" y =   "0.0" type="Point"/>
<node ID = "015" x = "130.0" y =  "50.0" type="Point"/>
<node ID = "016" x = "160.0" y =  "50.0" type="Point"/>
<node ID = "017" x = "160.0" y = "150.0" type="Point"/>
<node ID = "018" x = "130.0" y = "150.0" type="Point"/>

<!-- ======================================== -->
<!-- Network ways ...                         -->
<!-- ======================================== -->

<way ID="001">
   <attribute key = "type" value = "Edge"/>
   <attribute key = "weight" value = "2.0"/>
   <node ID="001" />
   <node ID="002" />
   <shape type = "LineString">
        <attribute key = "width" value =  "1.5"/>
        <attribute key = "color" value = "black"/>
   </shape>
</way>

<way ID="002">
   <attribute key = "type" value = "Edge"/>
  <attribute key = "weight" value = "2.0"/>
   <node ID="002" />
   <node ID="003" />
   <shape type = "LineString">
        <attribute key = "width" value =  "1.5"/>
        <attribute key = "color" value = "black"/>
   </shape>
</way>

... details of ways removed ...

<way ID="021">
   <attribute key = "type" value = "ObstructionEdge"/>
   <attribute key = "weight" value = "5.0"/>
   <node ID="017" />
   <node ID="018" />
   <shape type = "LineString">
        <attribute key = "width" value =  "1.5"/>
        <attribute key = "color" value = "black"/>
   </shape>
</way>

<way ID="022">
   <attribute key = "type" value = "ObstructionEdge"/>
   <attribute key = "weight" value = "5.0"/>
   <node ID="017" />
   <node ID="018" />
   <node ID="015" />
```

```
      <shape type = "LineString">
           <attribute key = "width" value =  "1.5"/>
           <attribute key = "color" value = "black"/>
      </shape>
  </way>

    <!-- ======================================== -->
    <!-- Relations for urban network segments     -->
    <!-- ======================================== -->

    <relation ID="001">
       <description text="Pathway 01 for the Urban Network." />
       <attribute key = "type" value = "Pathway"/>
       <attribute key = "end1" value =    "001" />
       <attribute key = "end2" value =    "014" />
       <attribute key = "length"    value = "800" />
       <attribute key = "status"    value = "Open" />
       <way ID="001" />
       <way ID="002" />
       <way ID="003" />
       <way ID="004" />
       <way ID="005" />
       <way ID="009" />
       <way ID="018" />
       <shape type = "MultiPolygon">
         <attribute key = "width" value =  "5.0"/>
         <attribute key = "color" value = "Orange"/>
       </shape>
    </relation>

  <relation ID="002">
     <description text="Pathway 02 for the Urban Network." />
     <attribute key = "type" value = "Pathway"/>
     <attribute key = "end1" value =    "001" />
     <attribute key = "end2" value =    "012" />
     <attribute key = "length"    value = "800" />
     <attribute key = "status"    value = "Open" />
     <way ID="001" />
     <way ID="002" />
     <way ID="003" />
     <way ID="004" />
     <way ID="008" />
     <way ID="012" />
     <way ID="014" />
     <way ID="015" />
     <shape type = "MultiPolygon">
         <attribute key = "width" value =  "5.0"/>
         <attribute key = "color" value = "Blue"/>
     </shape>
  </relation>

<relation ID="003">
   <description text="Obstruction 01 at the Urban Network." />
   <attribute key = "type" value = "Obstruction"/>
```

```
   <attribute key = "length"    value = "260" />
   <attribute key = "status"    value = "Open"/>
   <attribute key = "severity"  value = "3.0"/>
   <way ID="015" />
   <way ID="016" />
   <way ID="017" />
   <way ID="018" />
   <way ID="015" />
   <shape type = "MultiPolygon">
       <attribute key = "width" value =  "5.0"/>
       <attribute key = "color" value = "Blue"/>
   </shape>
</relation>
</SystemDataModel>
```

A.2 Print Urban Network System Data Model

```
urban01:
     [java] ------------------------------------------
     [java] SYSTEM DATA MODEL ...
     [java] ==========================================
     [java] --- Argument  1: "date" = "2022-01"
     [java] --- Argument  2: "author" = "Sachraa Borjigin"
     [java] --- Argument  3: "source" = "UMD"
     [java] ------------------------------------------
     [java] System Attribute  1:  [key, value] = [ "Units", "m" ]
     [java] System Attribute  2:  [key, value] = [ "Pattern", "1" ]
     [java] System Attribute  3:  [key, value] = [ "Demand Multiplier", "1.0" ]
     [java] System Attribute  4:  [key, value] = [ "Tolerance", "0.01" ]
     [java] System Attribute  5:  [key, value] = [ "Duration", "24:00" ]
     [java] System Attribute  6:  [key, value] = [ "Quality Timestep", "0:05" ]
     [java] System Attribute  7:  [key, value] = [ "Pattern Timestep", "2:00" ]
     [java] System Attribute  8:  [key, value] = [ "Pattern Start", "0:00" ]
     [java] System Attribute  9:  [key, value] = [ "Report Timestep", "1:00" ]
     [java] System Attribute 10:  [key, value] = [ "Start ClockTime", "12 am" ]
     [java] System Attribute 11:  [key, value] = [ "Statistic", "None" ]
     [java] ------------------------------------------------------------
     [java] No nodes          = [ 14.0 ]
     [java] No ways           = [ 18.0 ]
     [java] No relations      = [ 2.0 ]
     [java] No components     = [ 0.0 ]
     [java] No constraints    = [ 0.0 ]
     [java] No behaviors      = [ 0.0 ]
     [java] ------------------------------------------
     [java]
     [java] LIST OF NODES
     [java] ==========================================
     [java]
     [java] Node: ID = 001, (x,y) = (0.0, 0.0), type = Point
     [java] ------------------------------------------------------
     [java]
     [java] Node: ID = 002, (x,y) = (0.0, 100.0), type = Point
     [java] ------------------------------------------------------

     ... details of nodes removed ...

     [java]
     [java] Node: ID = 013, (x,y) = (400.0, 100.0), type = Point
     [java] ------------------------------------------------------
     [java]
     [java] Node: ID = 012, (x,y) = (400.0, 200.0), type = Point
     [java] ------------------------------------------------------
     [java]
     [java] LIST OF WAYS
     [java] ==========================================
```

```
[java]
[java] Way [ ID = 001 ] ...
[java]     [ nodes(0).ID = 001, nodes(1).ID = 002 ] ...
[java] ------------------------------------------------------
[java] Way Attribute  1:  [key, value] = [ "type", "Edge" ]
[java] ------------------------------------------------------
[java] Node  1: ID = 001
[java] Node  2: ID = 002
[java] ------------------------------------------------------
[java] Way Length =     100.00 ...
[java] ------------------------------------------------------
[java] Shape: type = LineString
[java] ------------------------------------------------------
[java] No attributes = [ 3.0 ]
[java] No nodes      = [ 0.0 ]
[java] ------------------------------------------------------
[java] Attribute 1:  [key, value] = [ "weight", "2.0" ]
[java] Attribute 2:  [key, value] = [ "width", "1.5" ]
[java] Attribute 3:  [key, value] = [ "color", "black" ]
[java] ------------------------------------------------------
[java] ------------------------------------------------------
[java]
[java] Way [ ID = 002 ] ...
[java]     [ nodes(0).ID = 002, nodes(1).ID = 003 ] ...
[java] ------------------------------------------------------
[java] Way Attribute  1:  [key, value] = [ "type", "Edge" ]
[java] ------------------------------------------------------
[java] Node  1: ID = 002
[java] Node  2: ID = 003
[java] ------------------------------------------------------
[java] Way Length =     100.00 ...
[java] ------------------------------------------------------
[java] Shape: type = LineString
[java] ------------------------------------------------------
[java] No attributes = [ 3.0 ]
[java] No nodes      = [ 0.0 ]
[java] ------------------------------------------------------
[java] Attribute 1:  [key, value] = [ "weight", "2.0" ]
[java] Attribute 2:  [key, value] = [ "width", "1.5" ]
[java] Attribute 3:  [key, value] = [ "color", "black" ]
[java] ------------------------------------------------------

... details of ways removed ...

[java]
[java] Way [ ID = 018 ] ...
[java]     [ nodes(0).ID = 011, nodes(1).ID = 014 ] ...
[java] ------------------------------------------------------
[java] Way Attribute  1:  [key, value] = [ "type", "Edge" ]
[java] ------------------------------------------------------
[java] Node  1: ID = 011
[java] Node  2: ID = 014
[java] ------------------------------------------------------
[java] Way Length =     100.00 ...
```

```
[java] -------------------------------------------------------
[java] Shape: type = LineString
[java] -------------------------------------------------------
[java] No attributes = [ 3.0 ]
[java] No nodes      = [ 0.0 ]
[java] -------------------------------------------------------
[java] Attribute 1:  [key, value] = [ "weight", "5.0" ]
[java] Attribute 2:  [key, value] = [ "width", "1.5" ]
[java] Attribute 3:  [key, value] = [ "color", "black" ]
[java] -------------------------------------------------------
[java] -------------------------------------------------------
[java]
[java] LIST OF RELATIONS
[java] ========================================
[java]
[java] Relation: ID = 001 ...
[java] -------------------------------------------------------
[java] Description = [ Pathway 01 for the Urban Network. ]
[java] -------------------------------------------------------
[java] Attribute  1:  [key, value] = [ "type", "Pathway" ]
[java] Attribute  2:  [key, value] = [ "end1", "001" ]
[java] Attribute  3:  [key, value] = [ "end2", "014" ]
[java] Attribute  4:  [key, value] = [ "length", "800" ]
[java] Attribute  5:  [key, value] = [ "status", "Open" ]
[java] -------------------------------------------------------
[java] Way [ ID = 001, nodes(0).ID = 001, nodes(1).ID = 002, type = Edge ] ...
[java] Way [ ID = 002, nodes(0).ID = 002, nodes(1).ID = 003, type = Edge ] ...
[java] Way [ ID = 003, nodes(0).ID = 003, nodes(1).ID = 004, type = Edge ] ...
[java] Way [ ID = 004, nodes(0).ID = 004, nodes(1).ID = 005, type = Edge ] ...
[java] Way [ ID = 005, nodes(0).ID = 005, nodes(1).ID = 006, type = Edge ] ...
[java] Way [ ID = 009, nodes(0).ID = 006, nodes(1).ID = 011, type = Edge ] ...
[java] Way [ ID = 018, nodes(0).ID = 011, nodes(1).ID = 014, type = Edge ] ...
[java] -------------------------------------------------------
[java] Shape: type = MultiPolygon
[java] -------------------------------------------------------
[java] No attributes = [ 2.0 ]
[java] No nodes      = [ 0.0 ]
[java] -------------------------------------------------------
[java] Attribute 1:  [key, value] = [ "width", "5.0" ]
[java] Attribute 2:  [key, value] = [ "color", "Orange" ]
[java] -------------------------------------------------------

... details of relations removed ...

[java] -------------------------------------------------------
```

Chapter B: Tiny Urban Network Semantic Model: Ontology, Rules and Graph Query

B.1 Urban Network Ontology (umd-urbannet.owl)

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY family_ontology "http://cee.umd.edu/urbannetwork#" >
]>

<rdf:RDF xmlns="http://cee.umd.edu/urbannetwork#"
     xml:base="http://cee.umd.edu/urbannetwork"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:family_ontology="http://cee.umd.edu/urbannetwork#">
    <owl:Ontology rdf:about="http://cee.umd.edu/urbannetwork"/>

    <!--
    ///////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////////////////
     -->
```

```
<!-- http://cee.umd.edu/urbannetwork#Person -->

<owl:Class rdf:about="http://cee.umd.edu/urbannetwork#Person">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<!-- http://cee.umd.edu/urbannetwork#UrbanNetwork -->

<owl:Class rdf:about="http://cee.umd.edu/urbannetwork#UrbanNetwork">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<!-- http://cee.umd.edu/urbannetwork#Component -->

<owl:Class rdf:about="http://cee.umd.edu/urbannetwork#Component">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<!-- http://cee.umd.edu/urbannetwork#Node -->

<owl:Class rdf:about="http://cee.umd.edu/urbannetwork#Node">
  <rdfs:subClassOf rdf:resource="http://cee.umd.edu/urbannetwork#Component"/>
</owl:Class>

<!-- http://cee.umd.edu/urbannetwork#Edge -->

<owl:Class rdf:about="http://cee.umd.edu/urbannetwork#Edge">
  <rdfs:subClassOf rdf:resource="http://cee.umd.edu/urbannetwork#Component"/>
</owl:Class>

<!-- http://cee.umd.edu/urbannetwork#Obstruction -->

<owl:Class rdf:about="http://cee.umd.edu/urbannetwork#Obstruction">
  <rdfs:subClassOf rdf:resource="http://cee.umd.edu/urbannetwork#Component"/>
</owl:Class>

<!-- http://cee.umd.edu/urbannetwork#Pathway -->

<owl:Class rdf:about="http://cee.umd.edu/urbannetwork#Pathway">
  <rdfs:subClassOf rdf:resource="http://cee.umd.edu/urbannetwork#Component"/>
</owl:Class>

<!--
///////////////////////////////////////////////////////////////
//
// Object Properties for class UrbanNetwork
//
///////////////////////////////////////////////////////////////
 -->

<!-- http://cee.umd.edu/urbannetwork#hasComponent -->

<owl:ObjectProperty rdf:about="http://cee.umd.edu/urbannetwork#hasComponent">
```

```
    <rdfs:domain rdf:resource="http://cee.umd.edu/urbannetwork#UrbanNetwork"/>
    <rdfs:range  rdf:resource="http://cee.umd.edu/urbannetwork#Component"/>
  </owl:ObjectProperty>

  <!--
  ///////////////////////////////////////////////////////////////
  //
  // Object Properties for class Node
  //
  ///////////////////////////////////////////////////////////////
   -->

  <!-- http://cee.umd.edu/urbannetwork#hasEdges -->

  <owl:ObjectProperty rdf:about="http://cee.umd.edu/urbannetwork#hasEdges">
      <rdfs:domain rdf:resource="http://cee.umd.edu/urbannetwork#Node"/>
      <rdfs:range  rdf:resource="http://cee.umd.edu/urbannetwork#Edge"/>
  </owl:ObjectProperty>

  <!--
  ///////////////////////////////////////////////////////////////
  //
  // Object Properties for class Edge
  //
  ///////////////////////////////////////////////////////////////
   -->

  <!-- http://cee.umd.edu/urbannetwork#hasSource -->

  <owl:ObjectProperty rdf:about="http://cee.umd.edu/urbannetwork#hasSource">
      <rdfs:domain rdf:resource="http://cee.umd.edu/urbannetwork#Edge"/>
      <rdfs:range  rdf:resource="http://cee.umd.edu/urbannetwork#Node"/>
  </owl:ObjectProperty>

 <!-- http://cee.umd.edu/urbannetwork#hasTarget -->

 <owl:ObjectProperty rdf:about="http://cee.umd.edu/urbannetwork#hasTarget">
     <rdfs:domain rdf:resource="http://cee.umd.edu/urbannetwork#Edge"/>
     <rdfs:range  rdf:resource="http://cee.umd.edu/urbannetwork#Node"/>
 </owl:ObjectProperty>

<!-- http://cee.umd.edu/urbannetwork#isBlockedBy -->

<owl:ObjectProperty rdf:about="http://cee.umd.edu/urbannetwork#isBlockedBy">
    <rdfs:domain rdf:resource="http://cee.umd.edu/urbannetwork#Edge"/>
    <rdfs:range  rdf:resource="http://cee.umd.edu/urbannetwork#Obstruction"/>
</owl:ObjectProperty>

  <!--
  ///////////////////////////////////////////////////////////////
  //
  // Data properties for class Edge
  //
  ///////////////////////////////////////////////////////////////
```

```
     -->

    <!-- http://cee.umd.edu/urbannetwork#isDirected -->

    <owl:DatatypeProperty rdf:about="http://cee.umd.edu/urbannetwork#isDirected">
        <rdfs:domain rdf:resource="http://cee.umd.edu/urbannetwork#Edge"/>
        <rdfs:range rdf:resource="&xsd;boolean"/>
    </owl:DatatypeProperty>

   <!-- http://cee.umd.edu/urbannetwork#hasWeight -->

   <owl:DatatypeProperty rdf:about="http://cee.umd.edu/urbannetwork#hasWeight">
       <rdfs:domain rdf:resource="http://cee.umd.edu/urbannetwork#Edge"/>
       <rdfs:range rdf:resource="&xsd;double"/>
   </owl:DatatypeProperty>

   <!--
   ///////////////////////////////////////////////////////////////
   //
   // Data properties for class Obstruction
   //
   ///////////////////////////////////////////////////////////////
    -->

   <!-- http://cee.umd.edu/urbannetwork#hasSeverity -->

   <owl:DatatypeProperty rdf:about="http://cee.umd.edu/urbannetwork#hasSeverity">
       <rdfs:domain rdf:resource="http://cee.umd.edu/urbannetwork#Obstruction"/>
       <rdfs:range rdf:resource="&xsd;integer"/>
   </owl:DatatypeProperty>

     <!--
     ///////////////////////////////////////////////////////////////
     //
     // Data properties for class Obstruction
     //
     ///////////////////////////////////////////////////////////////
      -->

     <!-- http://cee.umd.edu/urbannetwork#hasStatus -->

     <owl:DatatypeProperty rdf:about="http://cee.umd.edu/urbannetwork#hasStatus">
         <rdfs:domain rdf:resource="http://cee.umd.edu/urbannetwork#Component"/>
         <rdfs:range rdf:resource="&xsd;string"/>
     </owl:DatatypeProperty>

</rdf:RDF>

<!-- Generated by the OWL API (version 3.5.0) http://owlapi.sourceforge.net -->
```

B.2 Urban Network Jena Rules (umd.urbannet.rules)

```
 @prefix ur: <http://cee.umd.edu/urbannetwork#>.
 @prefix geom: <http://www.isr.umd.edu/geometry#>.
 @prefix  geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>.
 @prefix  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

 @prefix xs: <http://www.w3.org/2001/XMLSchema#>.

// Rule 01: Propagate class hierarchy relationships ...

[ rdfs01: (?x rdfs:subClassOf ?y), notEqual(?x,?y) ->
          (?a rdf:type ?y) <- (?a rdf:type ?x)] ]

// Rule 02: Obstruction and Edge Rule ...

[ Urban01: (?e rdf:type ur:Edge) (?o rdf:type ur:Obstruction)
           (?e ur:hasEdgeGeometry ?eg) (?o ur:hasJSTGeometry ?og)
            getObsInEdge(?og,?eg,?t) equal(?t, "true"^^xs:boolean) ->
           (?e ur:isBlockedBy ?o) print(?o,'isBlockedBy',?e,?t)]

// Rule 03: Computes and Updates Edge's Weight ...

[ Urban02: (?e rdf:type ur:Edge) (?o rdf:type ur:Obstruction)
           (?e ur:isBlockedBy ?o) (?e ur:hasWeight ?ew)
           (?o ur:hasSeverity ?os) computesEdge(?ew,?os,?newWeight) ->
           remove(3), (?e ur:hasWeight ?newWeight)
           print(?e,'hasWeight',?hasWeight)]

// Rule 04: Edges and Nodes for Directed Graph ...

[ Urban03: (?e rdf:type ur:Edge) (?n1 rdf:type ur:Node) (?n2 rdf:type ur:Node)
           notEqual(?n1,?n2), (?e ur:hasSource ?n1) (?e ur:hasTarget ?n2) ->
           (?e ur:isDirected "true"^^xs:boolean)
           (?n1 ur:hasEdges ?e) (?n2 ur:hasEdges ?e) ]
```

B.3 Print Urban Network Semantic Model

```
[java] ***
[java] *** ================================================ ...
[java] *** PART 04: Create Semantic Model for Urban Network      ...
[java] *** ================================================ ...
[java] ***
[java] *** Jena Semantic Model: Urban Network Semantic Model ...
[java] *** ============================================================
[java] *** Model size = 42 ...
[java] *** ============================================================
[java]
[java] Named Classes:
[java] ============================================================
[java]
[java] Named Class(1): Component
[java] --- Full Name: http://cee.umd.edu/urbannetwork#Component
[java] --- Superclass: Thing ...
[java] --- Subclass: Pathway ...
[java] --- Subclass: Obstruction ...
[java] --- Subclass: Edge ...
[java] --- Subclass: Node ...
[java] --- Data Property Name: hasStatus ...
[java] ---              Domain: Component ...
[java]
[java] Named Class(2): Edge
[java] --- Full Name: http://cee.umd.edu/urbannetwork#Edge
[java] --- Superclass: Component ...
[java] --- Data Property Name: hasWeight ...
[java] ---              Domain: Edge ...
[java] --- Data Property Name: isDirected ...
[java] ---              Domain: Edge ...
[java] --- Data Property Name: hasStatus ...
[java] ---              Domain: Component ...
[java] --- Object Property: hasSource ...
[java] ---            Range: Node ...
[java] --- Object Property: hasTarget ...
[java] ---            Range: Node ...
[java] --- Object Property: isBlockedBy ...
[java] ---            Range: Obstruction ...
[java]
[java] Named Class(3): Node
[java] --- Full Name: http://cee.umd.edu/urbannetwork#Node
[java] --- Superclass: Component ...
[java] --- Data Property Name: hasStatus ...
[java] ---              Domain: Component ...
[java] --- Object Property: hasEdges ...
[java] ---            Range: Edge ...
[java]
[java] Named Class(4): Obstruction
```

```
[java] --- Full Name: http://cee.umd.edu/urbannetwork#Obstruction
[java] --- Superclass: Component ...
[java] --- Data Property Name: hasSeverity ...
[java] ---              Domain: Obstruction ...
[java] --- Data Property Name: hasStatus ...
[java] ---              Domain: Component ...
[java]
[java] Named Class(5): Pathway
[java] --- Full Name: http://cee.umd.edu/urbannetwork#Pathway
[java] --- Superclass: Component ...
[java] --- Data Property Name: hasStatus ...
[java] ---              Domain: Component ...
[java]
[java] Named Class(6): Person
[java] --- Full Name: http://cee.umd.edu/urbannetwork#Person
[java] --- Superclass: Thing ...
[java]
[java] Named Class(7): UrbanNetwork
[java] --- Full Name: http://cee.umd.edu/urbannetwork#UrbanNetwork
[java] --- Superclass: Thing ...
[java] --- Object Property: hasComponent ...
[java] ---            Range: Component ...
[java] ============================================================
[java]
[java] Individuals:
[java] ============================================================
[java] ============================================================
[java]
[java] Statements:
[java] ============================================================
[java] Statement[   1]
[java]   Subject  : http://cee.umd.edu/urbannetwork#hasWeight
[java]   Predicate: http://www.w3.org/2000/01/rdf-schema#range
[java]   Object   : http://www.w3.org/2001/XMLSchema#double
[java] Statement[   2]
[java]   Subject  : http://cee.umd.edu/urbannetwork#hasWeight
[java]   Predicate: http://www.w3.org/2000/01/rdf-schema#domain
[java]   Object   : http://cee.umd.edu/urbannetwork#Edge
[java] Statement[   3]
[java]   Subject  : http://cee.umd.edu/urbannetwork#hasWeight
[java]   Predicate: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
[java]   Object   : http://www.w3.org/2002/07/owl#DatatypeProperty

... details of statements removed ...

[java] Statement[ 233]
[java]   Subject  : http://cee.umd.edu/urbannetwork#isDirected
[java]   Predicate: http://www.w3.org/2000/01/rdf-schema#range
[java]   Object   : http://www.w3.org/2001/XMLSchema#boolean
[java] Statement[ 234]
[java]   Subject  : http://cee.umd.edu/urbannetwork#isDirected
[java]   Predicate: http://www.w3.org/2000/01/rdf-schema#domain
[java]   Object   : http://cee.umd.edu/urbannetwork#Edge
[java] Statement[ 235]
```

```
[java]   Subject  : http://cee.umd.edu/urbannetwork#isDirected
[java]   Predicate: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
[java]   Object   : http://www.w3.org/2002/07/owl#DatatypeProperty
[java] ============================================================
```

Bibliography

[1] Abbondati, F., Biancardo, S. A., Palazzo, S., Capaldo, F. S., and Viscione, N. I-bim for existing airport infrastructures. *Transportation Research Procedia 45* (2020), 596–603. Transport Infrastructure and systems in a changing world. Towards a more sustainable, reliable and smarter mobility.TIS Roma 2019 Conference Proceedings.

[2] Abel D., Andersen T., and Conaboy C. *Jets crash on Logan taxiway*. In *Boston Globe, 10/03/16*.

[3] Adacher, L., Flamini, M., and Romano, E. Airport ground movement problem: Minimization of delay and pollution emission. *IEEE Transactions on Intelligent Transportation Systems 19*, 12 (2018), 3830–3839.

[4] Akagi, Y., and Raksincharoensakand, P. Stochastic driver speed control behavior modeling in urban intersections using risk potential-based motion planning framework. In *2015 IEEE Intelligent Vehicles Symposium (IV)* (2015), pp. 368–373.

[5] Aldred, J. NASA releases satellite imagery showing massive scale of the Camp Fire in California.

[6] Alirezaie, M. and Kiselev, A. and Längkvist, M. and Klügl, F. and Loutfi, A. An Ontology-Based Reasoning Framework for Querying Satellite Images for Disaster Monitoring. *Sensors 17*, 11 (November 2017), 2545.

[7] Allen, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM 26*, 11 (1983), 832–843.

[8] Anderson, S., Barford, C., and Barford, P. Five alarms: Assessing the vulnerability of us cellular communication infrastructure to wildfires. In *Proceedings of the ACM Internet Measurement Conference* (New York, NY, USA, 2020), IMC '20, Association for Computing Machinery, p. 162–175.

[9] Anderson, H.E. Predicting wind-driven wildland fire size and shape. *Research Paper INT-69, USDA Forest Service, Intermountain Forest and Range Experiment Station* (1983).

[10] Apache Jena. An open source Java framework for building semantic web and linked data applications. See https://jena.apache.org/, 2022.

[11] Austin, M., Baras, J., and Kositsyna, N. Combined research and curriculum development in information-centric systems engineering. *In Proceedings of the Twelth Annual International Symposium of The International Council on Systems Engineering (INCOSE), Las Vegas, USA* (2003).

[12] Austin, M.A., Delgoshaei, P., Coelho, M. and Heidarinejad M. Architecting smart city digital twins: combined semantic model and machine learning approach. *Journal of Management in Engineering, ASCE 36*, 4 (2020).

[13] Austin, M.A., Mayank, V. and Shmunis N. PaladinRM: graph-based visualization of requirements organized for team-based design. *Systems Engineering: The Journal of the International Council on Systems Engineering 9*, 2 (2006), 129–145.

[14] Ayyub, B. M. On uncertainty in information and ignorance in knowledge. *International Journal of General Systems 39*, 4 (2010), 415–435.

[15] Balakrishnan, H., and Jung, Y. A framework for coordinated surface operations planning at dallas-fort worth international airport.

[16] Balakrishnan, H., and Jung, Y. *A Framework for Coordinated Surface Operations Planning at Dallas-Fort Worth International Airport*. 2012.

[17] Bansal, V. Use of GIS and topology in the identification and resolution of space conflicts. *Journal of Computing in Civil Engineering 25*, 2 (2011), 159–171.

[18] Bansal, V. K. Use of geographic information systems in spatial planning: A case study of an institute campus. *Journal of Computing in Civil Engineering 28*, 4 (2014), 05014002.

[19] Beasley, J., Krishnamoorthy, M., Sharaiha, Y., and Abramson, D. Scheduling Aircraft Landings—The Static Case. *Transportation Science 34* (2000), 180–197.

[20] Berners-Lee, T., Hendler, J., and Lassa, O. The semantic web. *Scientific American* (May 2001), 35–43.

[21] Berre, D. L., and Parrain, A. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation 7* (2010), 59–64.

[22] Bertsimas, D., and Patterson, S. S. The air traffic flow management problem with enroute capacities. *Operations Research 46*, 3 (1998), 406–422.

[23] Borgo, S. and Guarino, N. and Masolo C. A Pointless Theory of Space Based On Strong Connection and Congruence. *Principles of Knowledge Representation and Reasoning, San Mateo, CA, USA* (1996), 220–229.

[24] Borrmann, A., Schraufstetter, S., and Rank, E. Implementing metric operators of a spatial query language for 3d building models: Octree and b-rep approaches. *Journal of Computing in Civil Engineering 23*, 1 (2014), 34–46.

[25] Brassel, H., Zouhar, A., and Fricke, H. 3d modeling of the airport environment for fast and accurate lidar semantic segmentation of apron operations. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)* (2020), pp. 1–10.

[26] CalFire, 2022. PRE-EVACUATION PREPARATION STEPS.

[27] Cheng, V., Sharma, V., and Foyle, D. A study of aircraft taxi performance for enhancing airport surface traffic control. *IEEE Transactions on Intelligent Transportation Systems 2*, 2 (2001), 39–54.

[28] Chi, S., and H.Caldas, C. Design of a preliminary error impact analysis model for spatial safety assessment of earthmoving operations. *Automation in Construction 22* (2012), 212–222.

[29] Coelho, M. Distributed system behavior modeling of urban systems with ontologies, rules and message passing mechanisms. *M.S. Thesis, University of Maryland, College Park, MD 20742, USA* (2017).

[30] Coelho, M., Austin, M., and Blackburn, M. Semantic behavior modeling and event-driven reasoning for urban system of systems. *International Journal on Advances in Intelligent Systems 10*, 3 and 4 (2017), 365–382.

[31] Coelho, M., Austin, M.A., and Blackburn, M.R. Distributed System Behavior Modeling of Urban Systems with Ontologies, Rules and Many-to-Many Association Relationships. *The Twelth International Conference on Systems (ICONS 2017)* (April 23-27 2017), 10–15.

[32] Coelho, M., Austin, M.A., and Blackburn, M.R. Semantic Behavior Modeling and Event-Driven Reasoning for Urban System of Systems. *International Journal on Advances in Intelligent Systems 10*, 3 and 4 (December 2017), 365–382.

[33] Coelho, M., Austin, M.A., and Blackburn, M.R. *The data-ontology-rule footing: A building block for knowledge-based development and event-driven execution of multi-domain systems*. Chapter 21, Systems Engineering in Context, Springer, 2019, pp. 255–266.

[34] Corbett, D. R. *Graph-Based Representation and Reasoning for Ontologies*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 351–379.

[35] Cova, T.J., L. D. S. L. D. F. Toward simulating dire wildfire scenarios. *Nat. Hazard. Rev. 22*, 3 (2021), 1–4.

[36] Cova, T., Theobald, D., Norman, J., and Siebeneck, L. Mapping wildfire evacuation vulnerability in the western us: The limits of infrastructure. *GeoJournal. 78*, 2 (2013), 273–285.

[37] Cui, Z., Yang, X., Feng, C., and Yang, G. Research on characteristics of aircraft traffic flow in airport surface. In *2020 7th International Conference on Dependable Systems and Their Applications (DSA)* (2020), pp. 429–440.

[38] da Silva, R. M., Junqueira, F., Filho, D. J. S., and Miyagi, P. E. Control architecture and design method of reconfigurable manufacturing systems. *Control Engineering Practice 49* (2016), 87–100.

[39] Daş, G. S., Gzara, F., and Stützle, T. A review on airport gate assignment problems: Single versus multi objective approaches. *Omega 92* (2020), 102146.

[40] Delgoshaei, P., and Austin, M. Framework for knowledge-based fault detection and diagnostics in multi-domain systems: application to heating ventilation and air conditioning systems. *International Journal on Advances in Intelligent Systems 10*, 3 and 4 (2017), 393–409.

[41] Delgoshaei, P., Austin, M., and Pertzborn, A. A semantic framework for modeling and simulation of cyber-physical systems. *International Journal On Advances in Systems and Measurements 7*, 3-4 (December 2014), 223–238.

[42] Delgoshaei, P., Austin, M., and Veronica, D. A semantic platform infrastructure for requirements traceability and system assessment. *The Ninth International Conference on Systems (ICONS 2014)* (February 2014), 215–219.

[43] Dennison, P. E., Brewer, S. C., Arnold, J. D., and Moritz, M. A. Large wildfire trends in the western united states, 1984–2011. *Geophysical Research Letters 41*, 8 (2014), 2928–2933.

[44] Derler, P., Lee, E. A., and Sangiovanni-Vincentelli, A. Addressing modeling challenges in cyber-physical systems. *Technical Report N0 . UCB/EECS-2011-17, Electrical Engineering and Computer Sciences University of California Berkley.* (2011).

[45] Eghtedarpour, N. A synergetic control architecture for the integration of photovoltaic generation and battery energy storage in dc microgrids. *Sustainable Energy, Grids and Networks 20* (2019), 100250.

[46] Faghih, S., Safikhani, A., Moghimi, B., and Kamga, C. N. Predicting short-term uber demand in new york city using spatiotemporal modeling. *Journal of Computing in Civil Engineering 33*, 3 (2019), 05019002.

[47] Federal Aviation Administration (FAA), 2022.

[48] Federal Aviation Administration (FAA), 2022. runwaySimulator Airport Capacity Model – Airports. .

[49] Federal Emergency Management Agency FEMA. Hazards mitigation planning process.

[50] Feigenbaum, L. Semantic web technologies in the enterprise.

[51] Fenza, G., Furno, D., Loia, V., and Veniero, M. Agent-based cognitive approach to airport security situation awareness. In *2010 International Conference on Complex, Intelligent and Software Intensive Systems* (2010), pp. 1057–1062.

[52] Fogarty, K., and Austin, M.A. Systems modeling and traceability applications of the higraph formalism. *Systems Engineering 12*, 2 (2009), 117–140.

[53] Folk, L., Kuligowski, E., Gwynne, S., and Gales, J. A provisional conceptual model of human behavior in response to wildland-urban interface fires. *Fire Technol. 55*, 5 (2019), 1619–1647.

[54] Fons, W. L. Analysis of fire spread in light forest fuels. *Journal of Agricultural Research 72* (1946), 93–121.

[55] Forecast.app. Forecast. accessed from https://www.forecast.app/platform.

[56] Fridenthal, S., Moore, A. and Steiner, R. A practical guide to SysML: the systems modeling language, 2008.

[57] Fried, J., Torn, M., and Mills, E. The impact of climate change on wildfire severity: A regional forecast for northern california. *Climatic Change 64* (2004), 169–191.

[58] Galton, A. Taking dimension seriously in qualitative reasoning. *W. Wahlster (ed), ECAI'96. Chichestern* (1996), 501–505.

[59] Geng, Y., and Cassandras, C. G. Dynamic resource allocation in urban settings: A "smart parking" approach. *2011 IEEE International Symposium on Computer-Aided Control System Design (CACSD)* (2011), 1–6.

[60] Ghaderi, M., Gheitasi, K., and Lucia, W. A novel control architecture for the detection of false data injection attacks in networked control systems. In *2019 American Control Conference (ACC)* (2019), pp. 139–144.

[61] Ghosh, J. K., Bhattacharya, D., Boccardo, P., and Samadhiya, N. K. Automated geo-spatial hazard warning system geowarns: Italian case study. *Journal of Computing in Civil Engineering 29*, 5 (2015), 04014065.

[62] Glasgow, J. The Imagery Debate Revisited: A Computational Perspective. *Computational Intelligence,* (1993), 9:309–333.

[63] Global Shop Solutions. Forecast. accessed from https://www.globalshopsolutions.com/.

[64] Golshani, N., Shabanpour, R., Mohammadian, A. K., Auld, J., and Ley, H. Analysis of evacuation destination and departure time choices for no-notice emergency events. *Transportmetr. A: Transport Sci. 15*, 2 (2019), 896–914.

[65] Gong, J., and H.Caldas, C. Data processing for real-time construction site spatial modeling. *Automation in Construction 17* (2007), 526–535.

[66] Gotts, N. M. Formalizing Commonsense Topology: The INCH Calculus. *Four International Symposium on Artificial Intelligence and Mathematics - AI/MATH'96, Fort Lauderdale(FL)* (1996), 72–75.

[67] Gołda, P., Zawisza, T., and Izdebski, M. Evaluation of efficiency and reliability of airport processes using simulation tools
. *Eksploatacja i Niezawodność 23*, 4 (2021), 659–669.

[68] Grajdura, S., Borjigin, S., and Niemeier, D. Fast-moving dire wildfire evacuation simulation. *Transportation Research Part D: Transport and Environment 104* (2022), 103190.

[69] Grajdura, S. A., Borjigin, S. G., and Niemeier., D. A. Agent-based wildfire evacuation with spatial simulation: A case study. *In3rdACM SIGSPATIAL International Workshop on GeoSpatial Simulation (GeoSim'20), Seattle, WA, USA.ACM, New York, NY, USA* (November 3–6 2020).

[70] Grajdura, S. and Qian, X and Niemeier, D. Awareness, departure, and preparation time in no-notice wildfire evacuations. *Safety Science 139* (2021).

[71] Grunninger, M. *Ontology of the Process Specification Language*. 2004.

[72] Gusgen, H. W. Spatial reasoning based on Allen's temporal logic. *International Computer Science Institute, Berkeley, CA, USA* (1989).

[73] Habel C. Propositional and Depictorial Representations of Spatial Knowledge: The case of Path Concepts. *R. Studer(ed.): Natural Language and Logic, Lecture Notes in Computer Science,Berlin* (1990), 94–117.

[74] Hashemi, M., and Karimi, H. A. Indoor spatial model and accessibility index for emergency evacuation of people with disabilities. *Journal of Computing in Civil Engineering 30*, 4 (2016), 04015056.

[75] Hatala, M., Wakkary, R., and Kalantari, L. Rules and ontologies in support of real-time ubiquitous application. *Journal of Web Semantics* (2005).

[76] Hauser, D. J., and Fleming, M. E. Mother nature's fury: Antagonist metaphors for natural disasters increase forecasts of their severity and encourage evacuation. *Science Communication 43*, 5 (2021), 570–596.

[77] Hernandez, D. and Clementini, E. and Di Felice, P. Qualitative Distances. *A. Frank and W. Huhn (eds.): Spatial Information Theory* (1995), 45–57.

[78] Hoogendoorn, S. P., and Bovy, P. H. L. Pedestrian travel behavior modeling. *Netw Spat Econ 5* (2005), 193–216.

[79] Hou, X., Ong, S., Nee, A., Zhang, X., and Liu, W. Graonto: A graph-based approach for automatic construction of domain ontology. *Expert Systems with Applications 38*, 9 (2011), 11958 – 11975.

[80] Hurwitz D. S. *The "Twilight Zone" of Traffic costs lives at Stoplight Intersections*. Oregon State University, Corvallis, Oregon, USA.

[81] Hyun, C., Jin, C., Shen, Z., and Kim, H. Automated optimization of formwork design through spatial analysis in building information modeling. *Automation in Construction 95* (2018), 193–205.

[82] International Council on Systems Engineering (INCOSE). SE Vision 2025, see details in https://www.incose.org/products-and-publications/se-vision-2025, 2014.

[83] Jackson, D. Dependable software by design. *Scientific American 294*, 6 (June 2006).

[84] Jacquillat, A., and Odoni, A. R. A roadmap toward airport demand and capacity management. *Transportation Research Part A: Policy and Practice 114* (2018), 168–185.

[85] Janic, M. Modeling airport operations affected by a large-scale disruption. *Journal of Transportation Engineering 135*, 4 (2009), 206–216.

[86] Java Topology Suite. http://www.vividsolutions.com/jts/.

[87] JGraphT: a java library of graph theory data structures and algorithms, See http://www.jgrapht.org (Accessed January 15, 2022).

[88] Kalabokidis, K. and Nikos, A. and Vaitis, M. OntoFire: An ontology-based geo-portal for wildfires. *Natural Hazards and Earth System Sciences 11* (2011), 3157–3170.

[89] Karouni, A., Daya, B., Bahlak, S. and Chauvet, P. A simplified mathematical model for fire spread predictions in wildland fires combining between the models of Anderson and Rothermel. *International Journal of Modeling and Optimization 4*, 3 (2014), 197–200.

[90] Kitching, C. *2 planes clip on Metro Airport taxiway*. In *Daily Mail, Available at: http* : $//www.dailymail.co.uk/travel/travel_news/article-3021054/Part-wing-torn-two-Ryanair-planes-collide-taxiing-runway-Dublin-Airport.html$*, accessed 10/03/16* (2016).

[91] Kuchcinski, K., and Szymanek, R. Java constraint programming (jacop) solver. accessed from https://github.com/radsz/jacop.

[92] Kyzirakos, K. and Karpathiotakis, M. and Garbis, G. and Nikolaou, C. and Bereta, K. and Papoutsis, I. and Herekakis, T. and Michail, D. and Koubarakis, M. and Kontoes, C. Wildfire monitoring using satellite images, ontologies and linked geospatial data, Journal of Web Semantic. *Journal of Web Semantics 24* (January 2013), 18–26.

[93] Latecki, L. and Pribbenow, S. On Hybrid Reasoning for Processing Spatial Expressions. *ECAI' 92* (1992), 389–393.

[94] Lee, E. A. Cyber-physical systems : A rehash or a new intellectual challenge? *Invited Talk in the Distinguished Speaker Series, Design Automation Conference (DAC), Austin, TX.* (2013).

[95] Lehmann, F. Semantic networks. *Computers & Mathematics with Applications 23*, 2-5 (1992), 1–50.

[96] Lei, J., Chong, X., Chen, D., Chen, Z., and Chen, Q. A review of airport runway capacity evaluation model. *IOP Conference Series: Materials Science and Engineering 780*, 7 (mar 2020), 072019.

[97] Lertlakkhanakul, J., Choi, J.-W., and Kim, M.-Y. Building data model and simulation platform for spatial interaction management in smart home. *Automation in Construction 17*, 8 (2008), 948–957.

[98] Levin, D.P. *Collision in Detroit; At Least 8 Die in Collision On Detroit Airport Runway*. In *The New York Times, Available at: http://www.nytimes.com/1990/12/04/us/collision-in-detroit-at-least-8-die-in-collision-on-detroit-airport-runway.html, accessed 10/03/16* (1990).

[99] LIBERTO, T. D., June 2018. Wildfires burn through southwestern Colorado in June 2018. See https://www.climate.gov/news-features/event-tracker/wildfires-burn-through-southwestern-colorado-june-2018.

[100] Locoia. Forecast. accessed from https://www.locoia.com/.

[101] Lucko, G., Said, H. M., and Bouferguene, A. Construction spatial modeling and scheduling with three-dimensional singularity functions. *Automation in Construction 43* (2014), 132–143.

[102] Michail D., Kinable J. Naveh B. and Sichi J.V. JGraphT– A Java library for graph data structures and algorithms. *ACM Trans. Math. Softw. 46*, 2 (May 2020).

[103] Mosteller M., Austin M.A., Yang S. and Ghodssi R. Platforms for engineering experimental biomedical systems. *IEEE Systems Journal 9* (September 2006), 1–11.

[104] Murray-Tuite, P., Schweitzer, L., and Morrison, R. Household no-notice evacuation logistics: How well do households optimize? *J. Transp. Saf. Security. 4*, 4 (2013), 336–361.

[105] Murray-Tuite, P., Yin, W., Ukkusuri, S., and Gladwin, H. Changes in evacuation decisions between hurricanes ivan and katrina. *Transportation Research Record 2312, Transportation Research Board, Washington, DC* (2012), 98–107.

[106] Muskerjee, A. and Joe, G. A qualitative Model for Space. *Conference on Artificial Intelligence (AAAI90), Cambridge, MA, USA* (1990), 721–727.

[107] Nara, A., Yang, X., Ghanipoor Machiani, S., and Tsou, M.-H. An integrated evacuation decision support system framework with social perception analysis and dynamic population estimation. *International Journal of Disaster Risk Reduction 25* (2017), 190–201.

[108] Nassar, N., and Austin, M.A. Model-Based Systems Engineering Design and Trade-Off Analysis with RDF Graphs. In *11th Annual Conference on Systems Engineering Research (CSER 2013)* (Georgia Institute of Technology, Atlanta, GA, March 19-22 2013).

[109] Netjasov, F., and Janic, M. A review of research on risk and safety modelling in civil aviation. *Journal of Air Transport Management 14*, 4 (2008), 213–220.

[110] Ng, K., Lee, C., Chan, F. T., and Lv, Y. Review on meta-heuristics approaches for airside operation research. *Applied Soft Computing 66* (2018), 104–133.

[111] Nikoleris, T., Gupta, G., and Kistler, M. Detailed estimation of fuel consumption and emissions during aircraft taxi operations at dallas/fort worth international airport. *Transportation Research Part D: Transport and Environment 16*, 4 (2011), 302–308.

[112] Niu, H., and Zhou, X. Optimizing urban rail timetable under time-dependent demand and oversaturated conditions. *Transportation Research Part C: Emerging Technologies 36* (2013), 212–230.

[113] Occupational Safety and Health Administration (OSHA), 2022. Wildfires: Preparedness. See https://www.osha.gov/wildfires/preparedness.

[114] OpenStreetMap (OSM). An open geographic database of the world. For details, see https://www.openstreetmap.org (Accessed January 15, 2022).

[115] OptaPlanner (2016), A Constraint-Satisfaction Solver. For details, see: https://www.optaplanner.org (Accessed, Jan 4., 2017).

[116] OptaPlanner.org. Optaplanner.

[117] Othman, S. H., and Beydoun, G. A metamodel-based knowledge sharing system for disaster management. *Expert Systems with Applications 63* (2016), 49–65.

[118] Pauwels, P., Deursen, D. V., Verstraeten, R., Roo, J. D., Meyer, R. D., de Walle, R. V., and Campenhout, J. V. A semantic rule checking environment for building performance checking. *Automation in Construction 20*, 5 (2011), 506–518.

[119] Pauwelsa, P., Krijnenb, T., Terkaj, W., and Beetz, J. Enhancing the ifcowl ontology with an alternative representation for geometric data. *Automation in Construction 80* (2017), 77–94.

[120] Petnga, L., and Austin, M. An ontological framework for knowledge modeling and decision support in cyber-physical systems. *Advanced Engineering Informatics 30*, 1 (2016), 77–94.

[121] Petnga, L., and Austin, M.A. Tubes and metrics for solving the dilemma-zone problem. In *The Tenth International Conference on Systems(ICONS 2015), Barcelona, Spain, April 19 - 24* (2015), pp. 119–124.

[122] Preuss, P. Smart moves: California's next-gen infrastructure. *Berkeley Engineer,* Publication of UC Berkeley College of Engineering, University of California, Berkeley, CA 94720 *11* (2017).

[123] Prout, A., Atlee, J. M., Day, N. A., and Shaker, P. Semantically configurable code generation. In *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems* (2008), MoDELS '08, pp. 705–720.

[124] Prud'homme, C., Fages, J.-G., and Lorca, X. Choco solver documentation. *TASC, INRIA Rennes, LINA CNRS UMR 6241* (2016).

[125] Radeloff, V. C., Helmers, D. P., Kramer, H. A., Mockrin, M. H., Alexandre, P. M., Bar-Massada, A., Butsic, V., Hawbaker, T. J., Martinuzzi, S., Syphard, A. D., and Stewart, S. I. Rapid growth of the us wildland-urban interface raises wildfire risk. 3314–3319.

[126] Rahman, M. F., and Sharma, N. Reinforcement learning based approach for urban resource allocation and path planning problems. *2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)* (2020), 115–118.

[127] Randell, D. A., Cui, Z., and Cohn, A. G. A spatial logic based on regions and connectivity. *Division of Artificial Intelligence, School of Computer Studies, Leeds University* (1994).

[128] Rathi, A. K., and Santiago, A. J. Urban network traffic simulations: Traf-netsim program. *Journal of Transportation Engineering 116*, 6 (1990), 734–743.

[129] INSTALLING AND CONFIGURING RED HAT BUSINESS OPTIMIZER. CHAPTER 2. WHAT IS A PLANNING PROBLEM? See https://access.redhat.com (Accessed January 25, 2022).

[130] Reilly, A. C., Dillon, R. L., and Guikema, S. D. Agent-based models as an integrating boundary object for interdisciplinary research. *Risk analysis : an official publication of the Society for Risk Analysis. 41*, 7 (2021), 1087–1092.

[131] Rimey, R. D., and Brown, C. M. Selective attention as sequential behavior: Modeling eye movements with an augmented hidden markov model. *Technical Report* (2008), 193–216.

[132] Robinne, F.-N., Hallema, D. W., Bladon, K. D., Flannigan, M. D., Boisramé, G., Bréthaut, C. M., Doerr, S. H., Di Baldassarre, G., Gallagher, L. A., Hohner, A. K., Khan, S. J., Kinoshita, A. M., Mordecai, R., Nunes, J. P., Nyman, P., Santín, C., Sheridan, G., Stoof, C. R., Thompson, M. P., Waddington, J. M., and Wei, Y. Scientists' warning on extreme wildfire risks to water supply. *Hydrological Processes 35*, 5 (2021), e14086.

[133] Salihu, A. L., Lloyd, S. M., and Akgunduz, A. Electrification of airport taxiway operations: A simulation framework for analyzing congestion and cost. *Transportation Research Part D: Transport and Environment 97* (2021), 102962.

[134] Samà, M., D'Ariano, A., Corman, F., and Pacciarelli, D. Coordination of scheduling decisions in the management of airport airspace and taxiway operations. *Transportation Research Part A: Policy and Practice 114* (2018), 398–411. Transportation and Traffic Theory: Behavior and Planning Applications.

[135] Scala, P., Mota, M. M., Wu, C.-L., and Delahaye, D. An optimization–simulation closed-loop feedback framework for modeling the airport capacity management problem under uncertainty. *Transportation Research Part C: Emerging Technologies 124* (2021), 102937.

[136] Schlieder, C. Reasoning about ordering. *A. Frank and I. Campari (eds.): Spatial Information Theory - A Theoretical Basis for GIS* (1995), 341–349.

[137] Schmidt, A., Schellroth, F., and Riedel, O. Control architecture for embedding reinforcement learning frameworks on industrial control hardware. In *Proceedings of the 3rd International Conference on Applications of Intelligent Systems* (New York, NY, USA, 2020), APPIS 2020, Association for Computing Machinery.

[138] Schulze, S., Fischer, E., Hamideh, S., and Mahmoud, H. Wildfire impacts on schools and hospitals following the 2018 california camp fire. *Natural Hazards 104* (2020), 901–925.

[139] Semantic data model, See https://en.wikipedia.org/wiki/Semantic_data_model (Accessed January 28, 2022).

[140] Shone, R., Glazebrook, K., and Zografos, K. G. Applications of stochastic modeling in air traffic management: Methods, challenges and opportunities for solving air traffic problems under uncertainty. *European Journal of Operational Research 292*, 1 (2021), 1–26.

[141] Shumsky, R. A. *Dynamic statistical models for the prediction of aircraft take-off times*. PhD thesis, Boston, MA, 1995.

[142] SKYbrary, 2022. Taxiway Collisions. See https://skybrary.aero/articles/taxiway-collisions.

[143] Smith, B. Ontology and the Logistic Analysis of Reality. *N. Guarino and R. Poli(eds): International Workshop on Formal Ontology in Conceptual Analysis and Representation* (1993), 51–68.

[144] Sztipanovits, J., Bapty, T., Karsai, G., and Neema, S. Model-integration and cyber physical systems : A semantic perspective. *Institute for Software Integrated Systems, Vanderbilt University, TN.* (2011).

[145] Talmaki, S. A., Dong, S., and Kamat, V. R. *Geospatial Databases and Augmented Reality Visualization for Improving Safety in Urban Excavation Operations*. 2010, pp. 91–101.

[146] Tang, P., Wang, Y., Sun, Z., and Liu, Y. Data-driven spatiotemporal simulation of ground movements of aircraft for preventive airport safety. In *2019 Winter Simulation Conference (WSC)* (2019), pp. 2992–3000.

[147] Tidwell, D. Xslt. *O'Reilly and Associates, Sebastopol, California.* (2001).

[148] Tofail, M. M., Oh, E., Chai, G., and Bell, P. An overview of the airport pavement management systems (APMS). *Int. J. Pavement Res. Technol. 13* (2020), 581–590.

[149] Toledo, T., Koutsopoulos, H. N., and Ben-Akiva, M. Integrated driving behavior modeling. *Transportation Research Part C: Emerging Technologies 15*, 2 (2007), 96 – 112.

[150] Trainor, J. E., Murray-Tuite, P., Edara, P., Fallah-Fini, S., and Triantis, K. Interdisciplinary approach to evacuation modeling. *Natural Hazards Review 14*, 3 (2013), 151–162.

[151] United States Government Accountability Office (US GAO). *Aviation Runway and Ramp Safety: Sustained Efforts to Address Leadership, Technology, and Other Challenges Needed to Reduce Accidents and Incidents*. Report GAO-08-29, United States Government Accountability Office, Report to Congressional Requesters, November.

[152] United States Government Accountability Office (US GAO). *Aviation Safety: FAA has Increased Efforts to Address Runway Incursions*. Report GAO-08- 1169T, United States Government Accountability Office(GAO), Testimony Before the Subcommittee on Aviation, Committee on Transportation and Infrastructure, House of Representatives, September.

[153] Vieu, L. A Logical Framework for Reasoning about Space. *A. Frank and I. Campari (eds.): Spatial Information Theory - A Theoretical Basis for GIS,* (1993), 25–35.

[154] Vieu, L. Spatial representation and reasoning in Artificial intelligence. *Spatial and Temporal Reasoning* (1997), 5–41.

[155] Vinnicombe, G. Frequency domain uncertainty and the graph topology. *IEEE Transactions on Automatic Control 38*, 9 (1993), 1371–1383.

[156] Wang, Y., Tang, T., Ning, B., and Meng, L. Integrated optimization of regular train schedule and train circulation plan for urban rail transit lines. *Transportation Research Part E: Logistics and Transportation Review 105* (2017), 83–104.

[157] Whistle: A Java-enabled scripting language for assembly and execution of multi-domain systems. See http://www.isr.umd.edu/~austin/whistle.html (Accessed January 15, 2022).

[158] Wikipedia. Constraint satisfaction problem. *Wikimedia Foundation* (January 2022).

[159] Wikipedia contributors. 4d-rcs reference model architecture — Wikipedia, the free encyclopedia, 2022. [Online; accessed 19-March-2022].

[160] Wikipedia contributors. Model–view–controller — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=1078624183, 2022. [Online; accessed 27-March-2022].

[161] Wilke, S., Majumdar, A., and Ochieng, W. Y. Airport surface operations: A holistic framework for operations modeling and risk management. *Safety Science 63* (2014), 18–33.

[162] Wilmot, C. G., and Mei, B. Comparison of alternative trip generation models for hurricane evacuation. *Nat. Hazards Rev. 5* (2004), 170–178.

[163] Wolshon, B., and Marchive, E. Emergency planning in the urban-wildland interface: Subdivision-level analysis of wildfire evacuations. *J. Urban Plann. Dev. 133*, 1 (2007), 73–81.

[164] (XSLT)., X. S. T. L. See http://www.w3.org/style/xsl.

[165] Yan, X., Zhang, H., and Li, H. Estimating worker-centric 3d spatial crowdedness for construction safety management using a single 2d camera. *Journal of Computing in Civil Engineering 33*, 5 (2019), 04019030.

[166] Yao, F., and Wang, Y. Tracking urban geo-topics based on dynamic topic model. *Computers, Environment and Urban Systems 79* (2020), 101419.

[167] Yin, S., Han, K., Ochieng, W. Y., and Sanchez, D. R. Joint apron-runway assignment for airport surface operations. *Transportation Research Part B: Methodological 156* (2022), 76–100.

[168] Zhong, S., Xiong, Z., Yao, G., and Zhu, W. A cps-enhanced subway operations safety system based on the short-term prediction of the passenger flow. *In: Hu S., Yu B. (eds) Big Data Analytics for Cyber-Physical Systems.* (2020).

[169] Zhu, Z., and Brilakis, I. Comparison of optical sensor-based spatial data collection techniques for civil infrastructure modeling. *Journal of Computing in Civil Engineering 23*, 3 (2008), 170–177.

[170] Šabić, M., Šimić, E., and Begović, M. Airport modeling software as a tool for assessing airport complexity and decision making. In *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)* (2021), pp. 944–949.