ABSTRACT

Title of thesis: **Semantic Modeling and Control of Urban Water Supply Networks**

Zebo Peng, Master of Science, 2020

Thesis directed by: Associate Professor Mark Austin
Department of Civil and Environmental Engineering and ISR

Water resources play a central role in the operation of urban systems. Present-day challenges in the use of water resources include over reliance on human-centered system management, and lack of formal approaches to decision-making support. In a step toward mitigating these challenges, the goals of this project are two-fold: first, we explore use of semantic modeling and rule-based reasoning a means to control operations in the water network system operation. The second purpose of this project is to explore opportunities for extending the logic of EPANET software simulation operations to include reasoning that takes into account factors beyond the water network. The case studies integrate time-history simulation and semantic modeling.

Last Modified: June 30, 2020

Semantic Modeling and Control of Urban Water Supply Networks

by

Zebo Peng

Thesis 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
Master of Science
2020

Advisory Committee:
Associate Professor Mark Austin, Chair/Advisor,
Assistant Professor Allison Reilly,
Assistant Professor Michelle Bensi.

Acknowledgments

I would like to express the deepest appreciation to my committee chair, Dr.Mark Austin, who has the attitude and the substance of a genius: he continually and convincingly conveyed a spirit of adventure in regard to research and scholarship, and an excitement in regard to teaching. During the two years' master study, Dr.Mark Austin is the professor I have been in contact with for the longest time. Although sometimes I cannot keep up well with him, I can always feel his rigorous scholarship toward academy and passion for the creative ideas. He can always pop up certain bright opinions during our talking. At the same time, he is also a kind teacher, who is pretty gentle and patient to me, a student with such not good performance. I really cherish this learning experience and enjoy the process of getting along with my advisor Dr.Mark Austin. Thank you!

I would like to thank my committee members, Dr.Allison Reilly and Dr.Michelle Bensi, who provide suggestions and support to my thesis writing and defense, and also my fellow graduate student Maria Coelho and Sachraa Borijigin who help me a lot during my study.

Finally, I must express my very profound gratitude to my parents Junxia Zhang and Dong Peng. They never push pressure on me and support me unconditionally. Don't need to say much. Thank you and love you forever.

Table of Contents

List of Abbreviations

ANSI	American National Standards Institute
API	Application Programming Interface
DL	Description Logic
DOM	Document Object Model
FOL	First-Order Logic
GIS	Geographic Information System
GML	Geography Markup Language
GPM	Gallons Per Minute
GUI	Graphical User Interfaces
IRI	Internationalized Resource Identifiers
ISO	International Organization of Standardization
JAXB	XML Binding for Java
JPL	Jet Propulsion Laboratory
JTS	Java Topology Suite
MBSE	Model-Based Systems Engineering
OOD	Object Oriented Design
OSM	Open Street Map
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
XML	Extensible Mark-up Language
WNTR	Water Network Tool for Resilience

Glossary of Terms

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

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.

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

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

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

Controller: (Mediator) A component of MVC design pattern, that acts as a communication channel between the model and the view.

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.

EPANET: Public domain software package for water distribution system (hydraulic) modeling.

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.

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.

JAXB: XML binding for Java.

Jena: Apache Jena is an open source Java framework for building Semantic Web and linked data applications.

Jena Rules: Jena Rules is an inference (reasoning) engine that plugs into Jena.

Listener: (Observer) A class that registers its interest to be notified for changes in other classes (Observable) in observer design pattern.

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

Observer Pattern: The observer pattern is applicable to problems where a message sender (observable) needs to broadcast a message to one or more receivers (or

observers), but is not interested in a response or feedback from the observers.

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.

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

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

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

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

Resource Description Framework (RDF): a model for encoding semantic relationships between items of data so that these relationships can be interpreted computationally.

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.

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.

List of Figures

Chapter 1: **Introduction**

1.1 Problem Statement

The modern way of life is enabled by remarkable advances in technology (e.g., the Internet, smart mobile devices, cloud computing) and the development of urban systems (e.g., transportation, electric power, wastewater facilities and water supply networks, among others) whose operations and interactions have superior levels of performance, extended functionality and good economics [8, 9]. Because water resources are necessary for sustainability of life (e.g., for consumption, bathing and cooking), and social (e.g., recreation, landscaping) and industrial development, water supply network systems play a central role in the operation of urban systems. From an operations standpoint, we need to understand what strategies of day-to-day operation lead to high levels of efficiency? We believe that high levels of situational awareness are a prerequisite to improvement of day-to-day operations. From a long-term planning perspective, accurate estimation of the future demand and availability of water resources is essential for achieving healthy and sustainable urban behavior [40]. As such, water resource concerns are important both within and outside the boundaries of an urban system. For example, the Public Policy Institute of California reports

that, statewide, average water usage in California is roughly 50% environmental (e.g., habitants, wetlands), 40% agricultural (e.g., for irrigation of farmland) and 10% urban [33]. Large urban areas such as Los Angeles and the San Francisco Bay Area now rely heavily on water imported from other parts of the state, as illustrated in Figure 1.1. The California scenario is typical of many highly populated regions of the World.

Challenges associated with the demand side of the water usage problem include unnecessary waste, improper management (regulations) of water as a limited resource, and lack of formal decision making support for the real-time control of water supply network elements (e.g., tanks, reservoirs, pumps). In a step toward mitigation of the latter concern, modern water supply network systems are designed from a cyber-physical systems perspective [39], with sensing systems and software embedded with the physical network system. Together, sensors and software need to transform streams of sensed data into information to support decision making and implementation of control actions. The long-term appeal of this approach to system design is that it enables management staff to deal with a wide variety of conditions precisely. In practice, however, the overall problem is large and complicated – management staff may have inadequate resources to focus on basic operations and simultaneously deal with broader system-level problems when they arise. Looking forward, it is evident that if human-centered system management could be partially replaced by automation/rule-based control many of these concerns could be resolved. This opportunity leads to a number of new questions:

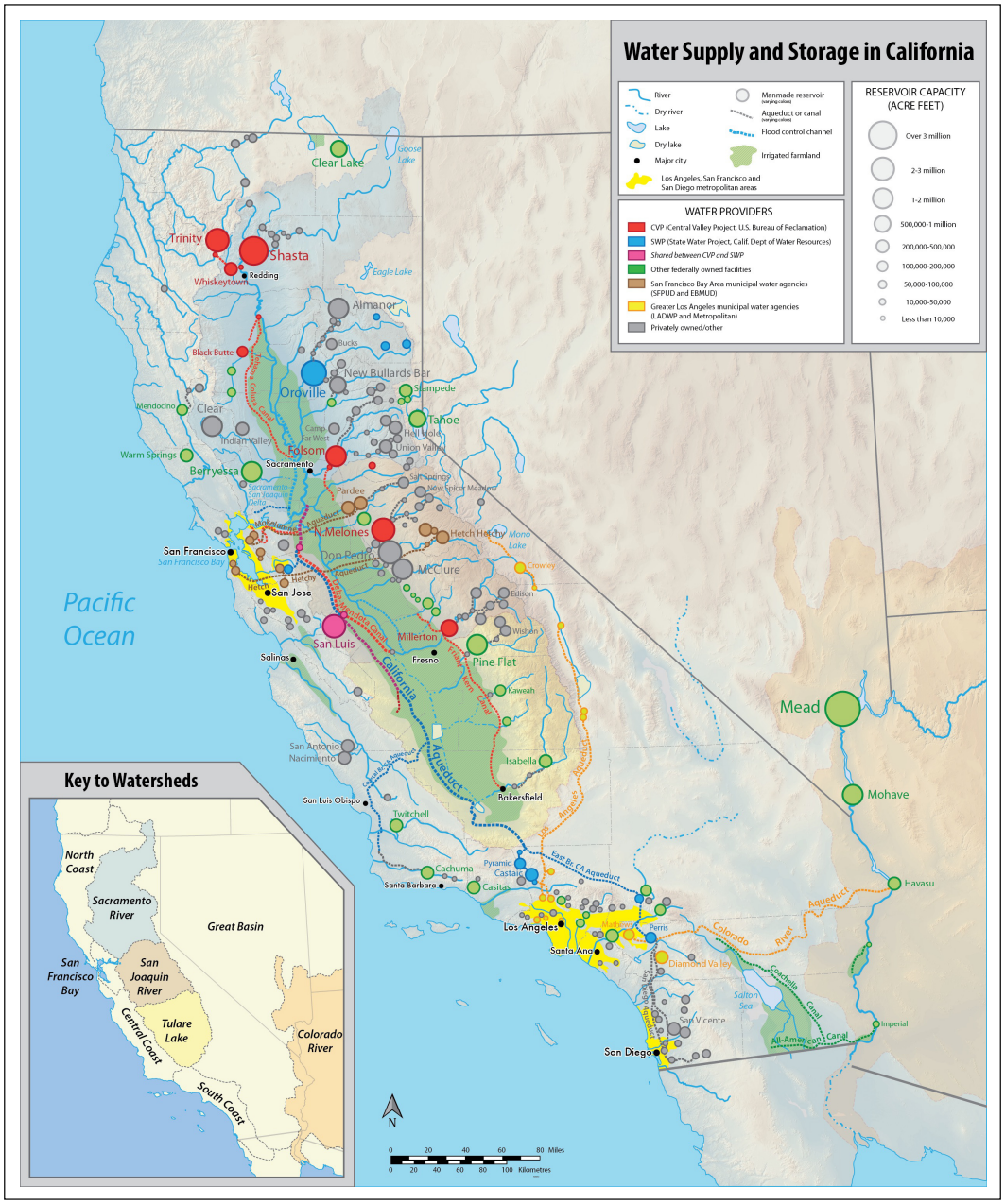

Figure 1.1: Watersheds and water supply networks in California (Source: Wikimedia Commons, https://commons.wikimedia.org/w/index.php?title=File: Water in Californianew.png&oldid=213297625, November 12, 2016).

- How to successfully recognize the operations of systems and elements within it?
- What basic rules should we have for the urban water supply networks?
- How to make the rules more generalized to be employed on more aspects?
- How to find an appropriate way to simulate the whole process of water supply networks?

State-of-the-art water network simulation packages such as EPANET [41] and WNTR (Water Network Tool for Resilience) [29] focus on the time-history simulation of water network attributes (e.g., pressures and velocities). Water network models comprise assemblies of nodes, pipes, valves, pumps, and tank/reservoir components for storage of water. The logic for the control of time-stepping algorithms is deeply embedded within the software and is primarily concerned with making sure hard constraints (e.g., water tanks cannot store negative volumes of water or store more than the maximum tank capacity) are not violated. No attempt is made to connect the logic of the water network system operation to the broader context of decision making within which the water system operate (e.g., details of regional geography and weather, time histories of water demand, planning of future operations). Thus, by themselves, state-of-the-art water network simulation packages provide an inadequate platform for moving toward the resolution of these questions.

1.2 Project Objectives and Scope

The objectives of this research project are to take initial steps toward development of a platform infrastructure where state-of-the-art simulation of water network systems behavior works alongside semantic representations of water network system knowledge and rules-based reasoning.

Figure 1.2 shows the proposed framework for time-history simulation of water network behaviors coupled with a semantic representation for knowledge representation and processing of rules in response to events. The scope of investigation for this project is highlighted in blue. For the semantic side of the problem, the project objectives are to provide a means whereby decision making is supported by procedures that are both deep and broad in their consideration of knowledge and rules. This project employs a framework for knowledge-based development and event-driven execution of multi-domain systems where the complementary rolls of data, ontologies, and rules are highlighted and have equal importance. The semantic side of the problem is represented by data-ontology-rule triplets.

On the simulation side of the problem, research is need to determine: (1) The extent to which the logic for simulation control can be reconfigured to improve the transparency of the simulator state (i.e., what is the simulator doing and why?), and (2) Ways in which the simulator can send and receive data/information from external entities, such as multi-domain semantic models and software for event-driven processing of semantic graphs. The integration of simulation and semantic modeling

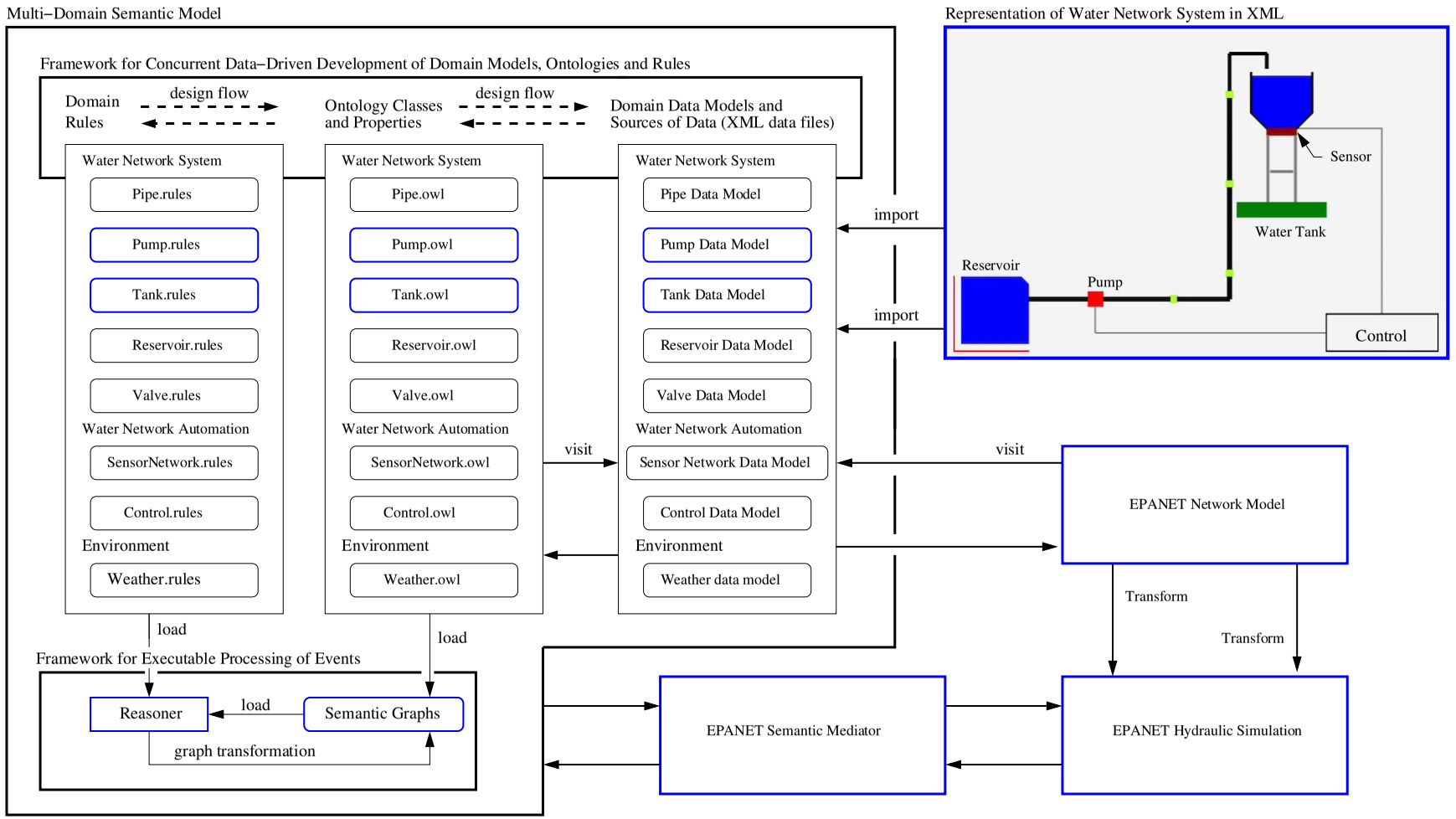

Figure 1.2: Proposed framework for semantic modeling and control of urban water supply networks. The focus areas of investigation for this project are highlighted in blue.

capabilities is handled by Whistle, a Java-enabled scripting language [13, 14, 47] for the definition and assembly of semantic models of water networks, execution of the water network simulations, and graphical display of the network setup and performance. Mechanisms to extend the functionality of Whistle include import of Java classes, and use of wrapper interfaces to external packages such as Jena (for semantic modeling), the Java Topology Suite (for spatial modeling and reasoning), OpenStreetMap (for modeling of urban domains), The Whistle back end software platform also links all the system operation including EPANET simulation, developing the semantic model and rule-based control mechanism as well as the real data retrieving by visitor design pattern.

1.3 Contributions and Organization

This thesis takes a step toward realization of the project objectives and scope. The contributions are as follows:

1. The thesis employs MBSE to create a pathway to the process of building models of water supply networks that includes analysis of requirements and synthesis of rules for rule-based control of system operations.

2. As a starting point, rules are needed for the control of water network elements (e.g., pumps, reservoirs, tanks, valves), and for higher-level supervisory control for planning of operations.

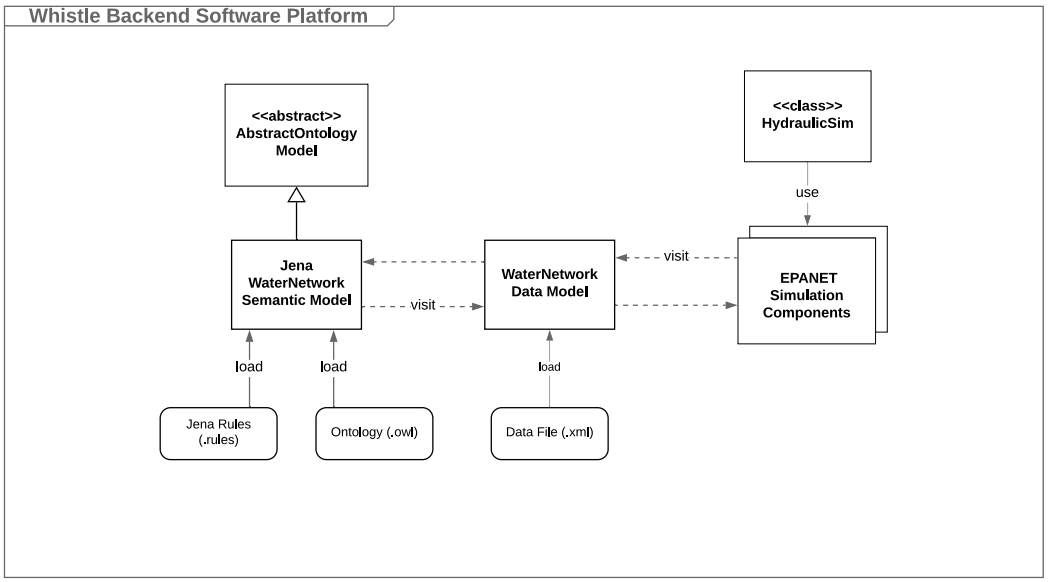

Figure 1.3: Framework for interaction of Jena semantic models and EPAnet simulation packages with System Data Model.

3. A framework is developed for interaction of Jena semantic models and EPAnet simulation models with a central systems data model (SDM). These interactions employ software design patterns (see Figure 1.3).

The thesis is organized as follows: Chapter 2 describes work related to this project. Chapter 3 provides a background on ontologies and rules, explains their relationship to our related work in model-based systems engineering, and explains the concept of Semantic Web. Chapter 4 describes the software architecture of EPAnet. Chapter 5 contains three case study problems: (1) Basic semantic interactions between a pump and a water tank, (2) Simulation modeling and rule-based control of a simple water network system in the Whistle implementation of EPANET, and (3) Semantic modeling for the simple water network system introduced in Case Study 2. Chapter 6 provides a summary and conclusion of the work, and ideas for future work.

Chapter 2: **Related Work**

This project approaches the problem of semantic modeling and control, and simulation of water network behaviors from a model-based systems engineering (MBSE) perspective. The proposed framework for semantic modeling and control is based upon collections of ontologies and rules to define domain-specific knowledge and interactions among domains. The work is preceded by studies at UMD covering semantic modeling and analysis for cyber-physical systems [14, 13, 37, 38, 39], traceability of requirements to component-level behaviors [12], component-based modeling, design and trade-off analysis with RDF graphs [34], validation of connectivity relationships in component-based systems [5], and behavior modeling of distributed urban systems [4, 8, 9].

2.1 Model-based Systems Engineering Perspective

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 [26]. From this standpoint, a model is a simplified

version of a concept, phenomenon, relationship, structure or system. The use of abstraction eliminates details not needed for decision making. As already noted, state-of-the-art approaches to MBSE employ visual modeling abstractions such as SysML [17] to frame the structure and behavior of the engineering problem under consideration. Models of system behavior specify what the system will actually do. Models of system structure specify how the system will accomplish its purpose. Although the models of system structure and behavior are not the same as real items, as least not the ideal representation of that, it can help the engineers to obtain the knowledge which can guide the real system implementation.

State-of-the-art practice in model-based systems engineering (MBSE) is to deal with design complexity through separation of concerns and development along disciplinary lines, followed by procedures for systems integration and validation and verification. While this approach eases work organization, design solutions tend to have loosely coupled system architectures that are limited in levels of achievable performance. Increases in system size and complexity drive the need for: (1) disciplined approaches to system design that involve the application of decomposition, composition, abstraction and use of semi-formal and formal analysis [3, 27], and (2) modeling formalisms that capture cause-and-effect relationships between designer concerns (e.g., correctness of system functionality; adequacy of performance; assurance of safety) and problem solutions.

In order to address these concerns, a multi-level approach to model-based system design must be taken. Figure 2.1 describes the different levels of development to

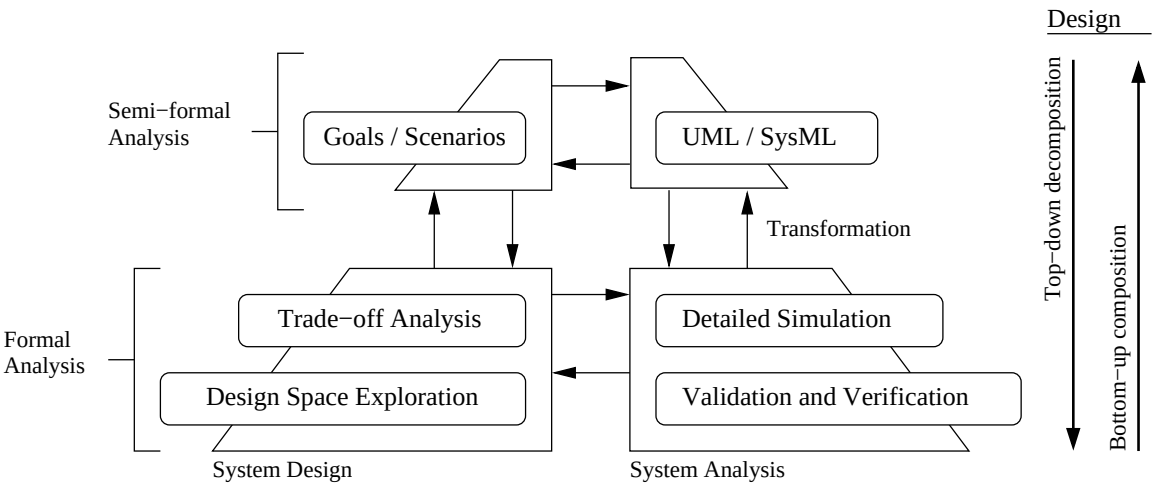

Figure 2.1: Multi-level approach model-based systems engineering. Semi-formal models provide a high-level view of the complete system (efficiency). Formal models provide a detailed view of the actual system (accuracy).

be used. The top level contains semi-formal models expressing ideas (i.e. goals and scenarios) and preliminary designs. At the front end of system development, use of semi-formal languages such as the System Modeling Language (SysML) [17] can help engineers systematically consider scenarios for required system functionality, create visual representations (diagrams) for fragments of behavior, develop requirements (constraints) for system performance and economics, and generate design alternatives that have the potential for delivering good design solutions. Lower level models employ formal languages having precisely defined semantics, and are designed to provide computational support for: (1) Detailed simulation of system behavior to evaluate levels of performance, (2) Validation and verification of the accuracy of functionality and control, (3) Systematic design space exploration, and (4) Trade-off analysis of design features.

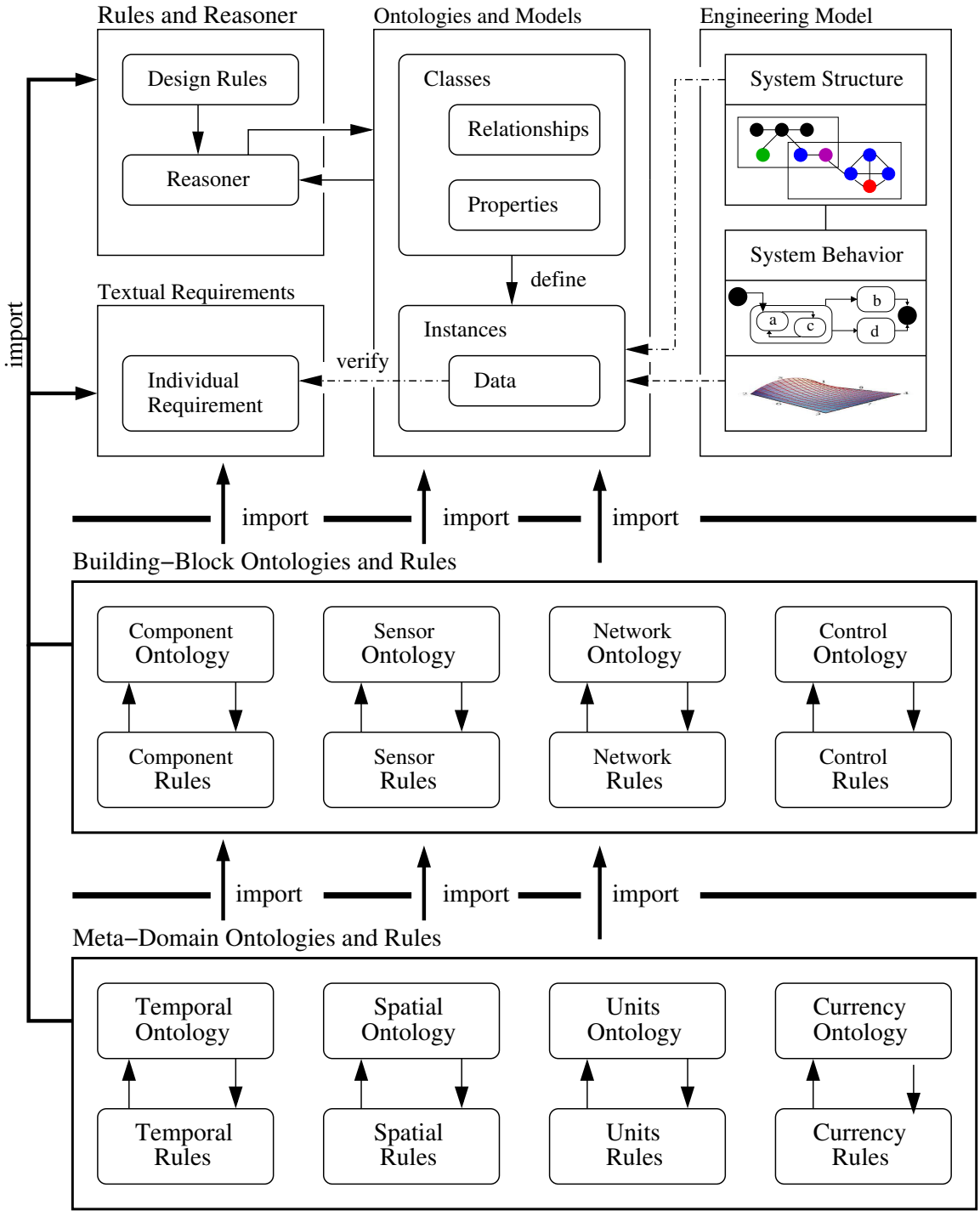

Figure 2.2: Framework for implementation of semantic models using ontologies, rules, and reasoning mechanisms (Adapted from Austin, Delgoshaei and Nguyen [4]).

2.2 Ontologies, Rules, and Reasoning Mechanisms

Figure 2.2 presents a framework for the implementation of semantic models using ontologies, rules, and reasoning mechanisms [13]. An ontology is "a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic [23]." To provide a formal conceptualization within a particular domain, and thereby facilitate communication among people and machines, ontologies need to accomplish three things: (1) Provide a semantic representation of each entity and its relationships to other entities; (2) Provide constraints and rules that permit reasoning within the ontology, and (3) Describe behavior associated with stated or inferred facts.

On the top left-hand side of Figure 2.2, textual requirements are defined in terms of mathematical and logical rule expressions for design rule checking. Engineering models of urban system structure will consist of networks and hierarchies of connected components formally described in terms of geometry (e.g., position, size) and connectivity (e.g., connected, touches, disjoint), possibly organized into layers (e.g., a hierarchy of networks). Engineering models of urban system behavior will be combinations of discrete (e.g., statecharts) and continuous (e.g., differential equations) behaviors. The semantic counterpart of engineering models is ontologies (class hierarchies), individuals (graphs), and rules [14, 13], Semantic graph models will be populated with instances of urban data (i.e., individuals) collected from a wide range of sources. Three examples are Open Street Map (OSM), the geography markup lan-

guage (GML) and CityGML for the population of urban network ontologies [15, 35], online weather servers for the population of weather ontologies, and census data for population demographics.

Rules can be developed for verification of semantic properties (e.g., to verify that a specific data property has been initialized) and for reasoning with data sources and incoming events, possibly from a multiplicity of domains. Implementation of the latter leads to semantic graphs that can dynamically adapt to incoming events (e.g., a weather event).

2.2.1 Rule-based Computations

Computation with rules provides several advantages [32, 42]:

1. Rules that represent policies are easily communicated and understood,

2. Rules retain a higher level of independence than logic embedded in systems,

3. Rules separate knowledge from its implementation logic, and

4. Rules can be changed without changing source code or underlying model.

A rule-based approach to problem solving is particularly beneficial when the application logic is dynamic, and where rules are imposed on the system by external entities. Both of these conditions apply to the design and management of urban water supply systems.

2.3 Data-Ontology-Rule Footprint Model

2.3.1 State-of-the-Art Semantic Modeling

State-of-the-art approaches [39, 46] to semantic modeling of engineering systems focus on the capture and representation of knowledge within one or more domains. A common objective is development of ontologies for the comprehensive representation of knowledge within a domain (e.g., pumps, pipes, valves), with far less effort going to the development of rules for the validation, use, and interaction of the ontology with other ontologies. Two further problems include: (1) a lack of discipline in the development of ontologies for system development, and (2) a lack of computational support for evolution of semantic graphs in response to events. The first factor is one of the reasons why formal representations of ontologies have a reputation of being difficult to develop and use. As a case in point, the integrated model-centric engineering ontologies (IMCE) developed at JPL (Jet Propulsion Laboratory) during the 2000-2010 era [46], the electrical engineering ontology (i.e., electrical.owl) imports the mechanical engineering ontology (i.e., mechanical.owl). Both the electrical and mechanical engineering ontologies import a multitude of foundation ontologies (e.g., analysis.owl, mission.owl, base.owl, project.owl, time.owl) and make extensive use of multiple inheritance mechanisms in the development of new classes. The result is ontologies containing more than several hundred classes, with some classes containing three or four dozen data and object properties. Notions of simplicity in system design through modularity of semantic models (e.g., bundling of ontologies and rules) do not

seem to exist.

2.3.2 Data-Ontology-Rule Footprint Model

Figure 2.3 shows the architectural template for multi-domain semantic modeling used in this project. Instead of creating a small number of all-encompassing ontologies and associated rules, this project puts the development of data, ontologies and rules on an equal footing, and create architectural templates for a specific domain or design concern (a convenient name is the data-ontology-rule footing). Concretely, each row of the first two-block domain items will contain the ontology and rule (e.g., Tank.rules, Tank.owl), which are used for building the semantic graph and its reasoner. These two parts establish the core of rule-based control mechanism that is the framework for executable processing of events.

A key benefit of this approach to semantic modeling is that it forces developers to think about the chain of dependency relationships between the data, ontologies and rules, and provide data needed to support decision making – rules require data and object properties from the ontologies, which in turn require data from the data models shown along the right-hand side of Figure 2.3. Semantic graph models will be populated with individuals (i.e., instances of real-world data) by visiting (a software design pattern) the relevant data models and gathering the data and object properties relevant to the application at hand. Rules can be developed for verification of semantic properties (e.g., to verify that a specific data property has been initialized) and for reasoning with data sources and incoming events, possibly from a multiplicity of

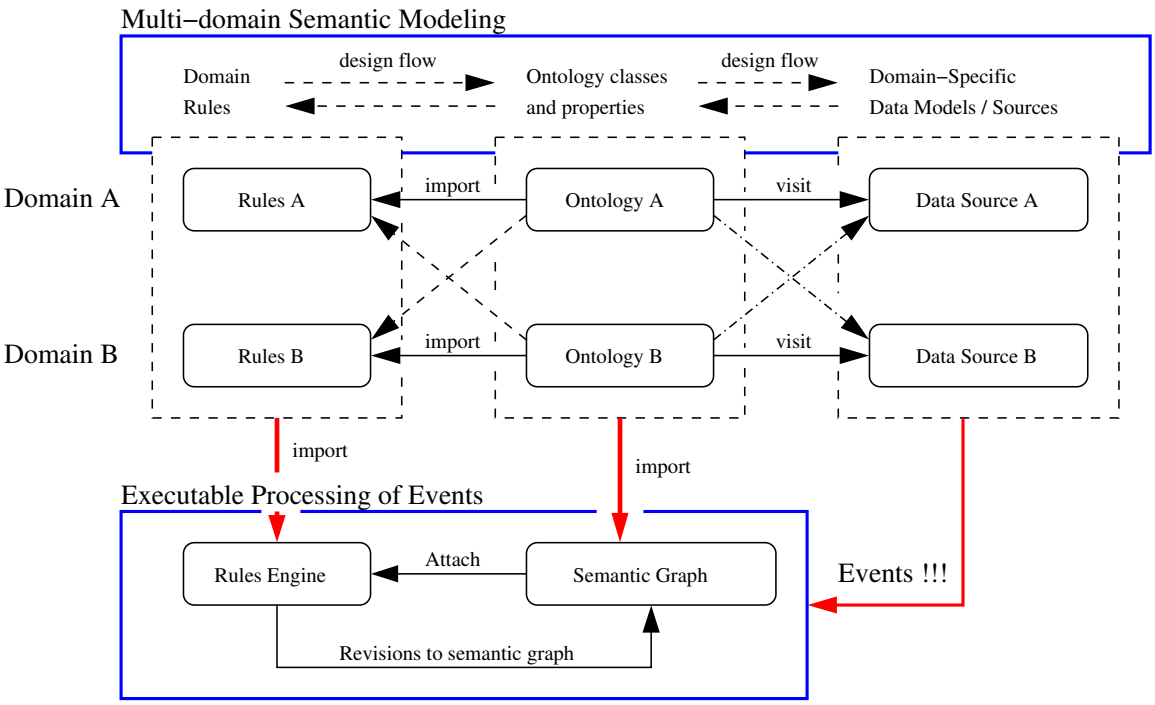

Figure 2.3: Template for semantic modeling with data-ontology-rule footprint (Adapted from Coelho et al. [10]).

domains. Implementation of the latter leads to semantic graphs that can dynamically adapt to incoming events (e.g., a tank is full (or empty), a weather event). A second key benefit of this approach to semantic modeling is that it reduces the complexity of domain models contributing to the semantic graph. This, in turn, strengthens the functionality of the rule-based control methodology. This framework is called data-ontology-rule footprint model.

2.4 System Data Model

2.4.1 Motivation and Approach

As illustrated in Figures 1.2 and 2.3, 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:

1. Build upon Open Street Map [35] with sets of tags to describe components and networks, their attributes and parameters, specifications and constraints, and statechart behaviors.

2. Explore the use of JAXB (as opposed to SAX or DOM in OpenStreetMap) for parsing and processing component and network data models into Whistle [47].

2.4.2 Open Street Map Primary Tags

Open Street Map (OSM) is remarkable. 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 2.4 shows

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 2.4: Fragments of XML in Open Street Map and System Data Model.

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; usually they 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 behaviors.

2.4.3 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 2.5). 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 statecharts (see the upper part of Figure 2.2). Behaviors can also include patterns of loading that will be applied to a component and/or systems. Component performance can be evaluated with respect to mathematical constraints.

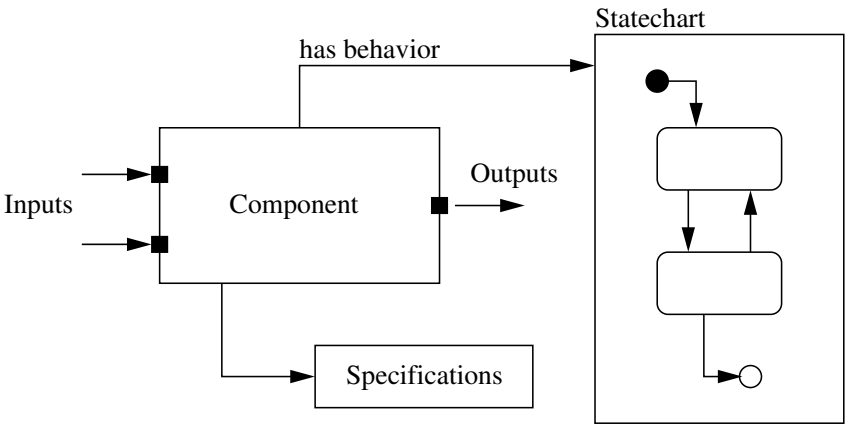

Figure 2.5: Abstract representation of a component model.

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 2.4) 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.

2.5 Software Design Patterns

A design pattern is a general repeatable solution to a commonly occurring problem. Design patterns initially became popular in the 1970s as a means for describing solutions to problems in urban (or city) planning [1]. Then, as the goals of software development increased in ambition and complexity, software design patterns provided template solutions to the structure, behavior and integration of software solutions [19]. These templates define domain agnostic arrangements of abstract methods and logical relationships found in good software solutions.

2.5.1 Accessing Data with the Visitor Software Design Pattern

The purpose of the visitor software design pattern is to separate an algorithm (i.e. system functionality) from an object structure on which it operates. The benefit of this separation is the capabilities to add new functions to the class structure without changing its original structure. In other words, based on the help of the visitor design pattern, engineers can easily transfer the functional logic from one class to various other classes.

The semantic and simulation aspects of this project employ the visitor software design pattern to access data within the system data model. Generally, the visitor design pattern comprises two methods: (1) A method called `visit()` which is implemented by the visitor and is called for every element in the data structure, and (2) Visitable classes providing `accept()` methods that accept a visitor.

Figure 2.6 shows the pathway of development for generation of semantic models, consisting of ontologies, graphs of individuals (specific instances), and rules derived from engineering models.

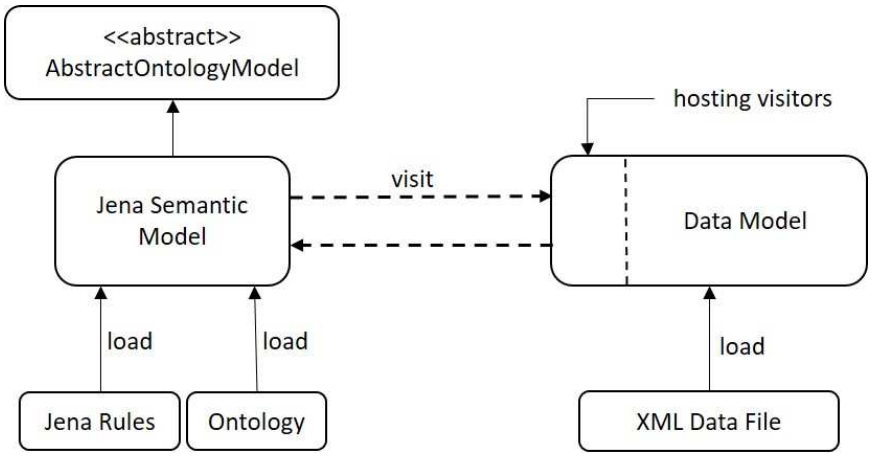

Figure 2.6: Pathway of development for generation of semantic models.

The process begins with the development of ontological descriptions of problem domains in OWL (the Web Ontology Language). Each ontology consists of a creating hierarchy of classes, and data and object properties. Next, we use the Jena Rules formalism to describe rules and represent domain-specific constraints. The data necessary to complete the model can be retrieved from an XML data file through a Data Model. The Data Model reads the XML data file and imports the data. Ontology, rules and data are all combined in the Jena Semantic Model. This semantic model creates an instance of the OWL ontology. Note that the data in the data model may or may not pertain to the ontology instance in its entirety. Through the implementation of a visitor design pattern, the data that does pertain to the ontology instance is transferred to the Jena Semantic Model, where the ontology and rules are applied to it.

Chapter 3: **Semantic Foundations**

3.1 Introduction to Semantic Web

This chapter introduces the Semantic Web vision, and the range of technologies found in its implementation. Basic capabilities of the resource description framework (RDF) and Web Ontology Language (OWL) are described. A simple case study problem involving behavior modeling of water supply network elements with ontologies (Jena) and rules (Jena Rules) is presented. Once the water network model has been manually assembled, the graph of family individuals and relationships will evolve in response to events.

3.1.1 Semantic Web Vision

The World Wide Web was invented in 1989 by Tim Berners-Lee, with the initial purpose to meet the demand for automatic information-sharing among members of scientific communities [7]. The Semantic Web is an extension of the World Wide Web that aims to produce a semantic data structure which allows machines to access and share information, thus constituting a communication knowledge between machines, and automated discovery of new knowledge [21, 23, 44]. This goal

is accomplished through the use of markup languages that enable the introduction, coordination, and sharing of the formal semantics of data, as well as an ability to reason and draw conclusions (i.e., inference) from semantic data obtained by following hyperlinks to definitions of problem domains (i.e., so-called ontologies).

3.1.2 Technical Infrastructure

Figure 3.1 illustrates the technical infrastructure that supports the Semantic Web vision, and the foundation upon which we hope to build our system-behavior models. Each layer exploits and uses capabilities of the layers below.

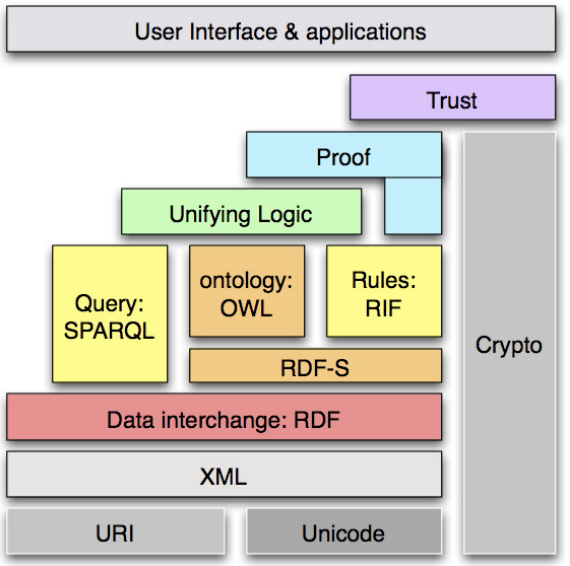

Figure 3.1: Technologies in Semantic Web Layer Cake [16].

Briefly, the bottom layer is constructed of Uniform Resource Identifier (URI) and Unicode. URI and Unicode provide capability for identifying resources on the Web, linking documents, and providing 16-bit representation for multi-lingual languages.

The extensible Markup Language (XML) is an open standard which describes how to declare and use simple tree-based data structures within a plain text file (human readable format). XML is a meta-language (or set of rules) for defining domain- or industry-specific markup languages. XML can also be used to filter, sort and re-purpose the data for different devices using the Extensible Stylesheet Language Transformation (XSLT) [45, 48]. XML data is organized into tree hierarchies. As already noted, Semantic Web applications can gather information from a variety of sources, and in the context of our application, merge and organize these sources for decision making. Unfortunately, there is no easy way for tree structures to be merged. The resource description framework (RDF) solves this problem by allowing for the representation of graphs of data on the web – graphs can always be merged. The web ontology language (OWL) provides for semantic descriptions of the underlying data. Together, XML, RDF and OWL allow for the implementation of reasoning that can prove whether or not assertions are true or false.

3.2 Working with RDF and OWL

3.2.1 Resource Description Framework (RDF)

While XML provides support for the portable encoding of data, it is limited to information that can be organized within hierarchical relationships. This can be a problematic situation for XML as a synthesized object may or may not fit into a hierarchical (tree) model. A graph, however, can, and thus we introduce the Resource Description Framework (RDF).

RDF is a graph-based assertional data model for describing the relationships between objects and classes (i.e., data and metadata) in a general but simple way, and for designating at least one understanding of a schema that is sharable and understandable. The graph-based nature of RDF means that it can resolve circular references, an inherent problem of the hierarchical structure of XML. An assertion is the smallest expression of useful information. RDF captures assertions made in simple sentences by connecting a subject to an object and a verb, as shown in Figure 3.2.

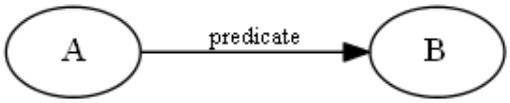

Figure 3.2: Example of RDF triple where node A is a subject, "predicate" is a verb, and node B is an object.

In practical terms, English statements are transformed into RDF triples consisting of a subject (this is the entity the statement is about), a predicate (this is the named attribute, or property, of the subject) and an object (the value of the named attribute). Subjects are denoted by a URI. Each property will have a specific meaning and may define its permitted values, the types of resources it can describe, and its relationship with other properties. Objects are denoted by a "string" or URI. The latter can be web resources such as requirements documents, other Web pages or, more generally, any resource that can be referenced using a URI (e.g., an application program or service program).

A set of related statements constitute an RDF graph. RDF graphs can be used

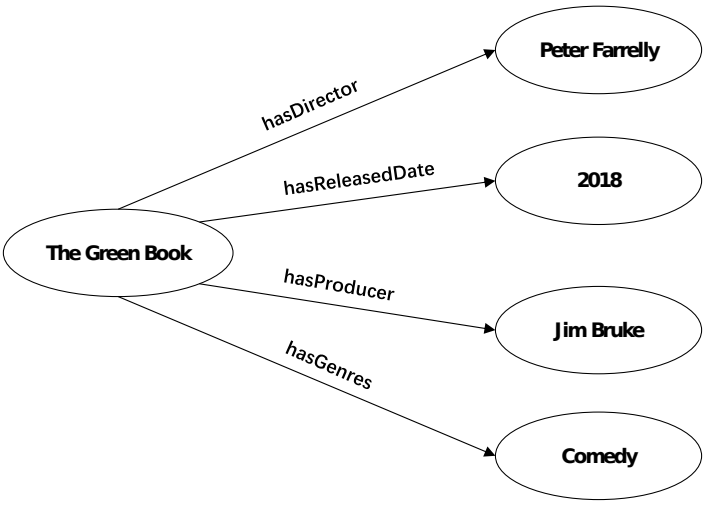

Figure 3.3: An RDF graph of relationships important to The Green Book.

to model a wide variety of relationships, including those among friends, location data, business data, and show information about a restaurant and a movie [44]. Figure 3.3 illustrates, for example, a graph model of relationships relevant to The Green Book.

Limitations of RDF. Unfortunately, RDF is unable to capture vital knowledge attributes such as existence and cardinality or localized range and domain constraints as well as richer properties such as transitivity, inverse or symmetrical properties [24]. This makes it weaker to describe resources in sufficient detail and difficult in use to support reasoning. The Web Ontology Language (OWL) was developed to address the weaknesses of RDF [25].

3.2.2 Web Ontology Language (OWL)

The Web Ontology Language (OWL) is a DL-based knowledge representation language for constructing ontologies. OWL is based on the basic features of RDF

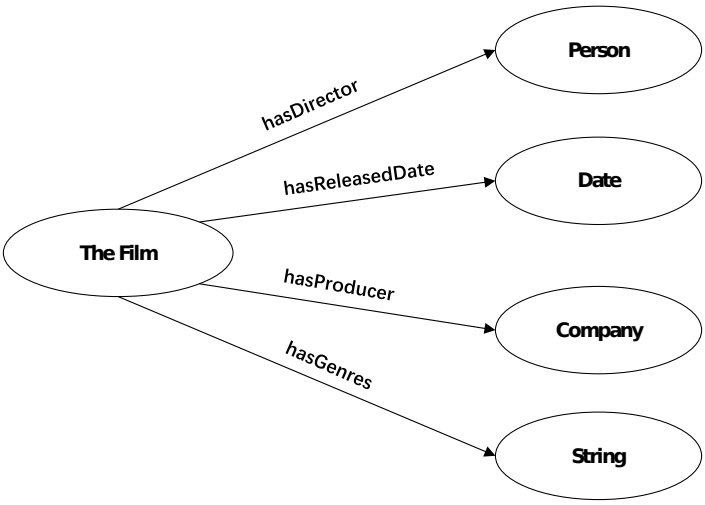

Figure 3.4: An OWL graph of relationships important to The Film.

introduced above but it strengthens it by adding structure and vocabulary for describing properties and classes. They enable richer property definitions(e.g.: transitivity), class property restrictions(e.g.: allValuesFrom), and relationship between classes(e.g.: subClassOf). The additional capabilities allow ontological systems to use reasoning structures and infrastructure to infer new facts (triples) from existing ones with FOL as baseline mathematical, formal foundation. Below is an example of how The Green Book example presented above can be translated into OWL. See Figure 3.4 and Figure 3.5.

In the example, the class Film, Person and Company are defined. OWL can also define two types of properties: object properties and datatype properties. Object properties specify relationships between pairs of resources. Datatype properties, on the other hand, specify relation between a resource and a data type value; they are equivalent to the notion of attributes in some formalisms. In the example above,

```
// Define Classes ...

<owl:Class rdf:about="http://example.org/monaLisa#Film">
</owl:Class>

<owl:Class rdf:about="http://example.org/monaLisa#Person">
</owl:Class>

<owl:Class rdf:about="http://example.org/monaLisa#Company">
</owl:Class>

// Define Datatype Properties ...

<owl:DatatypeProperty rdf:about="http://example.org/monaLisa#hasGenres">
   <rdfs:domain rdf:resource="http://example.org/monaLisa#Film"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="http://example.org/monaLisa#hasReleasedDate">
   <rdfs:domain rdf:resource="http://example.org/monaLisa#Film"/>
   <rdfs:range rdf:resource="&xsd;date"/>
</owl:DatatypeProperty>

// Define Object Properties ...

<owl:ObjectProperty rdf:about="http://example.org/monaLisa#hasDirectorr">
   <rdfs:domain rdf:resource="http://example.org/monaLisa#Film"/>
   <rdfs:range rdf:resource="http://example.org/monaLisa#Person"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="http://example.org/monaLisa#hasProducer">
   <rdfs:domain rdf:resource="http://example.org/monaLisa#Film"/>
   <rdfs:range rdf:resource="http://example.org/monaLisa#Company"/>
</owl:ObjectProperty>
```

Figure 3.5: Formal definition of a “Awarded Film” in OWL.

hasGenres and hasReleasedDate are defined as datatype properties, while hasDirector and hasProducer are defines as object properties. The rdfs:domain and rdfs:range properties are used to specify the domain and range of a property. The rdfs:domain of a property specifies that the subject of any statement using the property is a member of the class it specifies. Similarly, the rdfs:range of a property specifies that the object of any statement using the property is a member of the class or datatype it specifies.

The family of OWL encompasses three languages distinguished by their increasing expressiveness. *OWL Lite* allows the expression of simple syntax and constraints but inferencing is more tractable using this version. *OWL DL* has a human-friendly syntax, inferencing is decidable and the language is computationally complete. *OWL Full* ensures full compatibility with RDF and RDFS languages however, the cost is that there is no guarantee in the validity of all computed statements[36].

3.3 Working with Jena and Jena Rules

Not all technologies on the semantic web are standardized. Some are emergent ones that are used mostly for horizontal and vertical integration of multiple layers of the stack. Generally speaking, there are Application Programming Interfaces (API) used to complete integration tasks.

3.3.1 Jena

Apache Jena [2] is an open source Java framework for building Semantic Web and linked data applications. Jena provides APIs (application programming interfaces) for developing code that handles 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 logic, such as the DL, is needed for horizontal integration of top layers of stacks and provide the rigorous, formal support needed by applications.

3.3.2 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, as shown in Figure 3.7.

3.4 Case Study: Simplified Event-Driven Water Network Controls

In this small cast study, the simplified semantic model for water network with ontologies and rules are built manually by Jena and Jena Rules. From the viewpoint of ontology and rule-based control mechanisms, this case study will describe the event-driven semantic modeling operation process. The semantic graph will show logical relationships among the ontology classes, which can be used to further rule-based control mechanism.

3.4.1 Definition of the Water Network Ontology

Figure 3.6 shows a simplified water network ontology, the relationship among classes and properties.

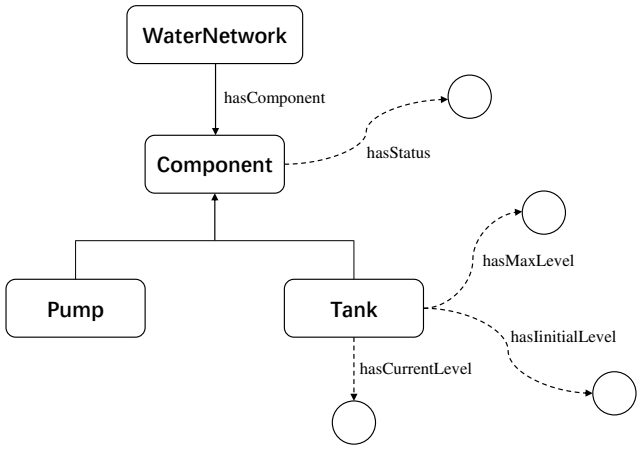

Figure 3.6: Relationship between classes and properties in a water network ontology.

This simplified semantic graph defines four ontology classes, three object properties and a data property. The ontology class WaterNetwork contains ontology component with two ontology subclasses tank and pump. Ontology component has properties: hasStatus modeled as string data type which will be inherited by the two subclasses. The tank ontology has its own specific properties: hasInitialLevel and hasMaxLevel which are modeled as double data type.

3.4.2 Adding Rules

To better explain the ideas of rule-based control mechanism, the following are a list of rules that can be used in a simplified water network semantic model:

The following rules can be declared:

Rule 1: Combining the system initial working time, current time, water flow rate, `getCurrentLevel()` compute's a tank's current water level.

Rule 2:bhy Tank has the full status when its water level attain the max level.

Rule 3: The max level is a water level range between 8 to 10.

Rule 4: If tank is full, then pump is closed.

Figure 3.7 shows the evolution of a graph defining the properties of one pump and tank as a function of time.

As time goes by, the elements' data properties (e.g, tank's current water level, status) will change based on the water network rules.

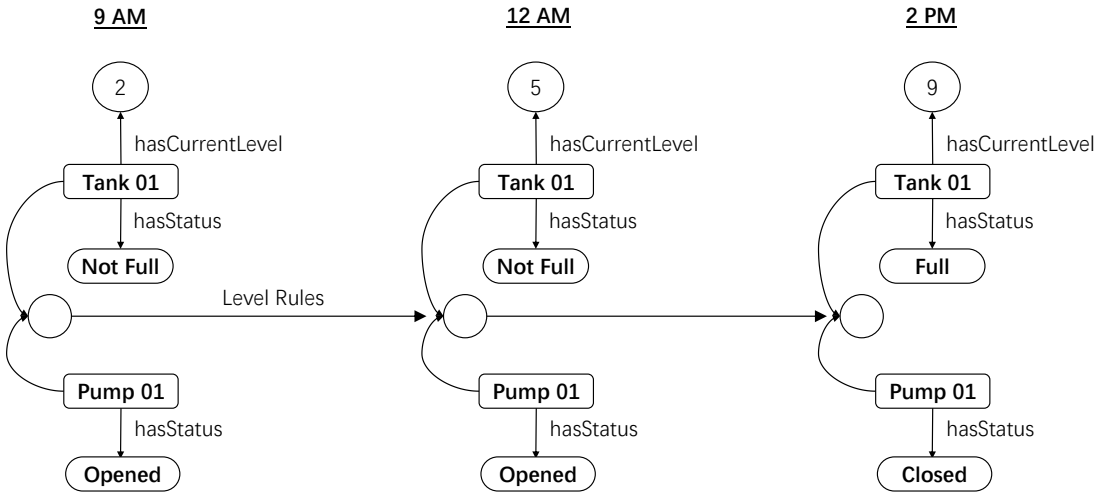

Figure 3.7: Time-based evolution of semantic graph.

3.4.3 Definition and Organization of Ontology Classes

The abbreviated fragment of code below demonstrates the definition of the water network ontology classes, their assembly into a hierarchy, and definition of data and object properties for the class `Component`, the data properties for the class `Tank`.

```
// Define classes ...

waterNetwork = model.createClass( ns + "WaterNetwork");
component    = model.createClass( ns + "Component");
pump         = model.createClass( ns + "Pump");
tank         = model.createClass( ns + "Tank");

// Define relationships among classes ...

waterNetwork.addSubClass (   pump );
waterNetwork.addSubClass (   tank );

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

hasComponent = model.createObjectProperty( ns + "hasComponent");
hasComponent.setDomain( waterNetwork );
```

```
hasComponent.setRange( component);

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

hasStatus = model.createDatatypeProperty( ns + "hasStatus");
hasStatus.setDomain(component);
hasStatus.setRange( XSD.String );

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

hasInitialLelve = model.createDatatypeProperty( ns + "hasInitialLevel");
hasInitialLevel.setDomain(tank);
hasInitialLevel.setRange( XSD.Double );

hasMaxLevel = model.createDatatypeProperty( ns + "hasMaxLevel");
hasMaxLevel.setDomain(tank);
hasMaxLevel.setRange( XSD.Double);
```

The `WaterNetwork` contains `Component` depending on object property `hasComponent`. Regarding to the subclass relationship, the `Component` has the string type property `hasStatus` which hierarchy inherited by the `Tank` and `Pump`. The `Tank` also has its own double type properties `hasCurrentLevel` and `hasMaxLevel`.

3.4.4 Adding Individuals to the Water Network Model

The next procedure will show a process of how to add a component individual with the related data properties to a specific water network. The following section of code below demonstrates the development process of defining name space, creating graph model, adding specific individual for `Tank01` to graph model and its statement of having status.

```
// Namespace for the water network ontology ...

String ns = "http://www.ontologies.org/waterNetwork#";
```

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

OntModel model = ModelFactory.createOntologyModel();

// Add "tank01" to the tank to the water network graph model ...

Individual c01 = boy.createIndividual( ns + "Pump01" );
model.add ( c01 );

// Create statement: Pump01 has status

Literal status = model.createTypedLiteral( "Not Full", XSDDatatype.XSDString );
Statement cbd = model.createStatement( c01, hasStatus, status );
model.add ( cbd );
```

Jena provides very powerful facilities for querying the graph model, subject to a wide range of search criteria.

3.4.5 Event-Driven Rule-Based Control (Jena Rules)

Considering the above four rules, the semantic graph transformation is enabled by the rule-based control mechanism. Given the initial working time and the current time with certain characteristic of tank, the function `getCurrentLevel()` compute the current water level of tank which will be compared with the tank's max water level. Then, based on the rules related to tank state change, comparison result will be presented in text as tank's status which may change later. Figure 3.7 shows the evolution of a graph transformation about the data properties of `tank01` and `pump01` in terms of time.

```
@prefix wn:  <http://www.ontologies.org/waterNetwork#>.
@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: Compute and update the status of tank and pump...

[ UpdateCurrentLevel: (?t rdf:type wn:Tank) (?t wn: hasArea ?ar)
                      (?t wn: hasFlowRate ?f) (?t wn:hasCurrentLevel ?cl)
              getCurrentLevel(?ar,?f,?cl,?d) notEqual(?cl, ?d) ->
             remove(2) (?t wn:hasCurrentlevel ?d) ]

[ UpdateTankStatus: (?t rdf:type wn:Tank) (?t wn: hasCurrentLevel ?cl)
                    greaterThan(?cl, 9) -> (?t wn: hasStatus Full)]


[ UpdatePumpStatus: (?t rdf:type wn:Tank) (?p rdf:type wn:Pump)
                    (?t wn: hasStatus ?st) equals(?st, 'FUll')
                    -> (?p wn: hasStatus Closed)]
```

The first rule propagates class hierarchy relationships. The second list of rules update the tank's current water level, status and pump's status. Note that elements' interaction can be presented in last rule, which the pump's status change based on the tank's state.

Chapter 4: **Water Network Simulation**

This chapter discusses water network simulation with the Java implementation of EPANET, working as a simulation platform within Whistle. Many changes to the code have been made: we modified some bugs in certain water network element's classes and also added toString() method in most of the structure-level classes. The latter help us to check the API operations are correct. The work also includes development of a wrapper class called EpanetMVC to walk an engineer through the multi-step process of defining and simulating behavior for a water network system.

4.1 State-of-the-Art Software for Water Network Simulation

State-of-the-art software for water network simulation is defined by two packages, EPANET and WNTR.

4.1.1 EPA Water Network Simulation (EPANET)

EPANET [41] is a computer program that performs extended period simulation of hydraulic and water quality behavior within pressurized pipe networks. From a physical standpoint, pressurized pipe networks are collections of interconnected

elements such as pipes, pumps, reservoirs and valves. Mathematically, pipe networks are graphs consisting of sets of edges (e.g., pipes and pumps) and nodes (e.g., for reservoirs, tanks, and junctions (i.e., intersections of pipe elements). By pre-setting the water network parameters and initial water flow status, EPANET is able to simulate the movement and the change of hydraulic parameters for a range of discrete points in time. This provides end-users with insight into time behavior of hydraulic networks and the adequacy of network designs.

Background. The development of EPANET dates back to the early 1990s. Lewis Rossman started the development of EPANET in 1991 and released the first version in 1993. At that time, the water network modeling software market was dominated by commercial products. The EPANET software, in contrast, was open source and quickly became the commonly used water network simulation software for researchers or scientists. The earliest versions of EPANET were written in ANSI C [28], language which gained wide acceptance in technology companies in the 1980s. The Java implementation (officially released in 2012) of EPANET provides water network simulation capabilities in an object-oriented style.

The Java implementation of EPANET is a simulation engine which is full Java rewrite of the EPANET software written in standard ANSI C. It makes available a comprehensive water network modeling and simulation based on the .INP input modeling files.

Software Architecture. Figure 4.1 is a high-level view of the EPANET software architecture and flow of computations in a standard water network simulation.

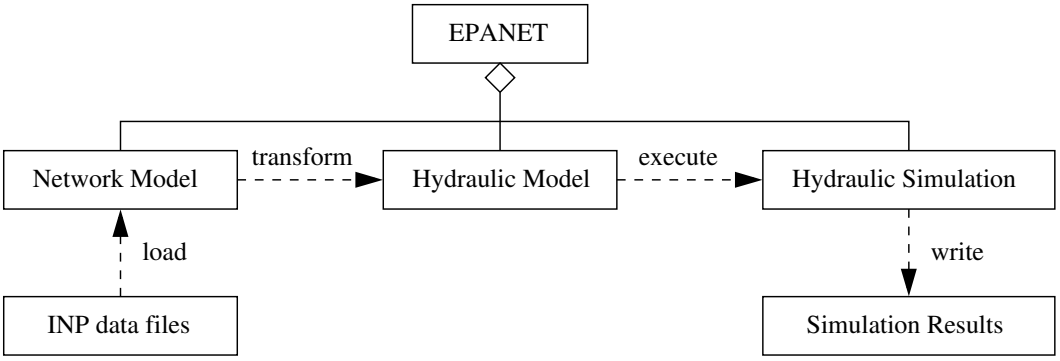

Figure 4.1: EPANET Software architecture and Work Flow.

For the purposes of hydraulic simulation, EPANET is the composition of two packages, a network model, and a hydraulic model. The network model supports representation of standard entities found in a water network system (e.g., pipes, pumps, valves, etc) and support for regulation of behaviors with controls and rules. State-of-the-art simulations read data files having a `.INP` format).

Mathematical Model. EPANET assumes that fluid flows will be steady state and incompressible. The steady state assumption implies zero acceleration of fluid flows. Assuming that fluids are incompressible means that pressure can be expressed in terms of equivalent hydraulic head. Together these assumptions allow for the overall network state to be described by the head at each node and the flow in each element. The behavior of a water network corresponds to a sequence of steady state flows. At each time step, standard numerical procedures such as Newton-Raphson iteration are used to solve the system level hydraulic equations. Validation procedures can check for: (1) conservation of mass, and (2) conservation of energy [31].

Control and Rules. Controls are lists of actions to which a water network systems should respond. Rules are constraints that are designed to prevent undesirable

behavior. For example:

- **Rules for Water Network Operation:** (1) If tank status is full, then pump status is closed, (2) If tank level less than max level, and time is operation, then pump status is opened, and (3) If season is spring, then tank has lower maximum level.

- **Rules for Water Conservation:** (1) Improve the frequency of survey for water pipe leakage, (2) Timely maintenance or replacement of water network infrastructures, and (3) Promptly adjusting the amount of water supply based on the change of water demands.

- **Rules for Water Quality (Contamination Prevention):** Rules to prevent contamination of water supply networks: (1) Selecting construction materials that do not promote microbial growth, (2) Constructions or pipelines with less curvature, dead zones, and (3) Maintaining hot water temperatures above 50 degrees and a cold water temperature below 20 degrees.

Recent Research. Despite being almost 30 years old, EPANET is still actively used in research studies. Recent research investigations have included development of ontologies for support of collaboration in federated water-power simulations [22], procedures for modeling network leakage [18], use of evolutionary algorithms in water distribution network design [6], and scheduling of pumping stations [20]. An EPANET Open Source Initiative was launched in 2018 [43], with on-going research focused on: (1) extensions needed for solving equations for dispersion of chemical contamination,

and (2) development of graphical user interfaces that integrate water network systems modeling with geographic information systems (GIS).

4.1.2 Water Network Tool for Resilience (WNTR)

The Water Network Tool for Resilience (WNTR, pronounced winter) is a Python package designed to simulate and analyze resilience of water distribution networks. In this context, a network refers to the collection of pipes, pumps, nodes, and valves that make up a water distribution networks [29]. The most prominent feature of WNTR is its support for great system resilience analysis, specifically: (1) It contains a multi-functional software platform for modeling different types of hydraulic condition especially the disruptive incidents, and (2) It inherits the original Python advantage of scientific computation including the employment of Python scientific computation packages like numpy and pandas. The choice of Python means WNTR can be easily combined with procedures for simulation data processing and analysis (i.e., analysis of data streams) and studies involving AI or machine learning.

WNTR Software Framework. The WNTR Python package consists of multiple subpackages with object-oriented software design patterns. Every small package is made up of several modules that are .py files containing various classes with fields and methods. Figure 4.2 present the simplified software framework of WNTR. Table 4.1 demonstrates the sub-packages's description of WNTR. The core contents of WNTR is the employment of sub-packages **network** and **sim**. The network subpackage comprises with classes defining the basic structure of water network with various

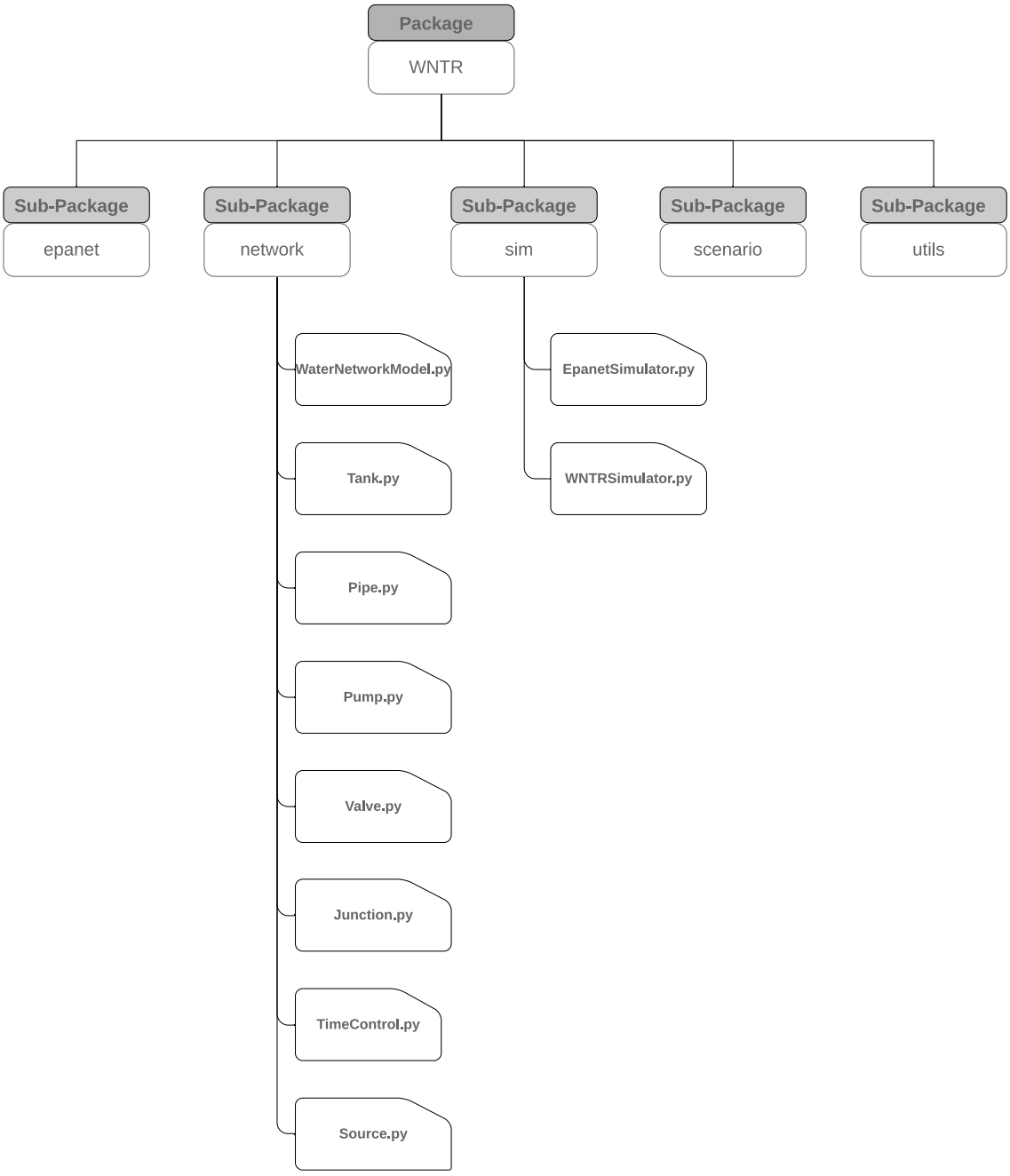

Figure 4.2: Software architecture of WNTR.

Table 4.1: WNTR subpackages.

Subpackage	**Description**
epanet	Contains EPANET 2 compatibility functions for WNTR.
metrics	Contains methods to compute resilience,including hydraulic,water quality, water security,and economic metrics. Methods to compute topographic metrics are included in the wntr.network.graph module.
network	Contains methods to define a water network model, network controls, and graph representation of the network.
scenario	Contains methods to define disaster scenarios and fragility/survival curves.
sim	Contains methods to run hydraulic and water quality simulations using the water network model.
graphics	Contains methods to generate graphics.
utils	Contains helper functions.

Table 4.2: Select classes in the *network* subpackage.

Class	**Description**
WaterNetworkModel	Contains methods to generate water network models,including methods to read and write INP fields,and access/add/remove/modify network components. This class links to additional model classes which define network components,controls,and model options.
Tank	Contains methods to define tanks.Tanks are nodes with storage capacity.
Pipe	Contains methods to define pipes.Pipes are links that transport water.
Pump	Contains methods to define pumps.Pumps are links that increase hydraulic head.

Table 4.3: Select classes in the *sim* Subpackage.

Class	**Description**
EpanetSimulator	The EpanetSimulator uses EPANET 2 Programmer's Toolkit to run demand-driven hydraulic simulations and water quality simulations. When using the EpanetSimulator, the water network model is written to an EPANET INP file which is used to run an EPANET simulation.
WNTRSimulator	The WNTRSimulator uses custom Python solvers to run demand-driven and pressure dependent demand hydraulic simulation and includes models to simulate pipe leaks.

elements and also the control operation of it. And just like EPANET, WNTR water network models can be built from EPANET INP files. WNTR provides round-robin support for generation of INP files directly from water network models. The sim subpackage define the classes to run the hydraulic and water quality simulation by two kinds of simulators: the EpanetSimulator and the WNTRSimulator. The details of classes defined in the subpackages **network** and **sim** are listed Table 4.2 and Table 4.3.

4.2 EPANET Software Architecture

Software architecture is a term that refers to the fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations. The architecture of a software system is a metaphor, analogous to the architecture of a building. It functions as blueprint for the system and the developing project, laying out the tasks necessary to be executed by the design teams.

4.2.1 Network and Hydraulic Model Class Hierarchies

Figures 4.3 and 4.4 show the organization of classes in the EPANET network and hydraulic models, respectively. The network model is primarily responsible for the setup, validation, and storage of water network system models comprising components (e.g., tanks, pumps, junctions and pipes), patterns for demands for water at

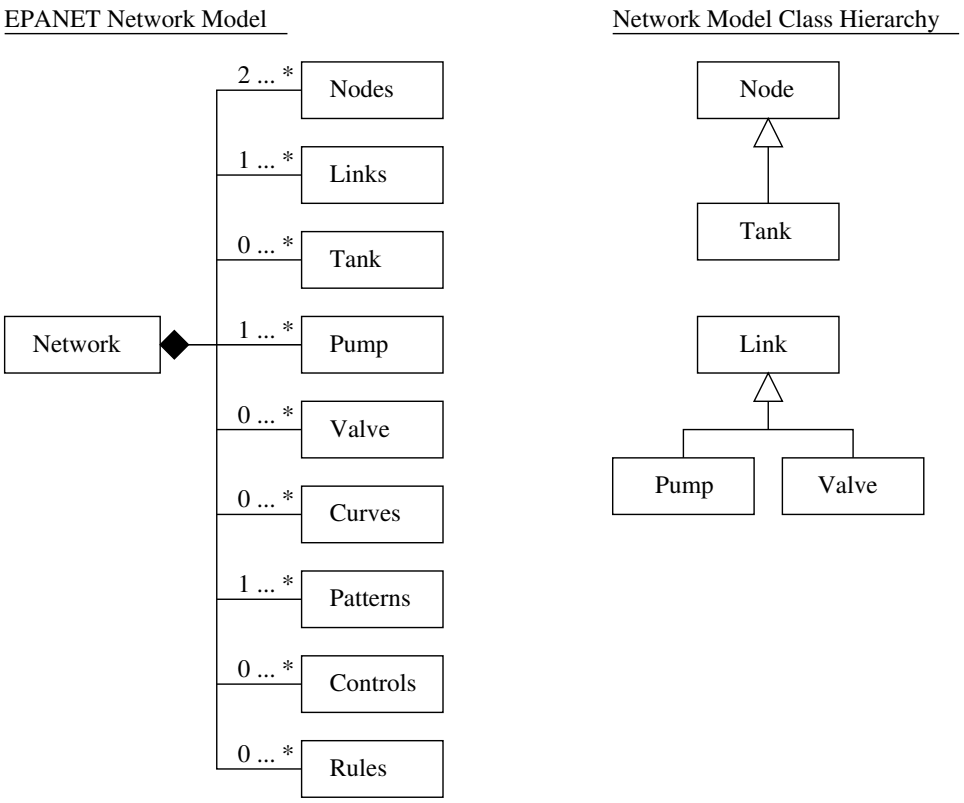

Figure 4.3: EPANET network model. Left: organization of classes. Right: class hierarchies.

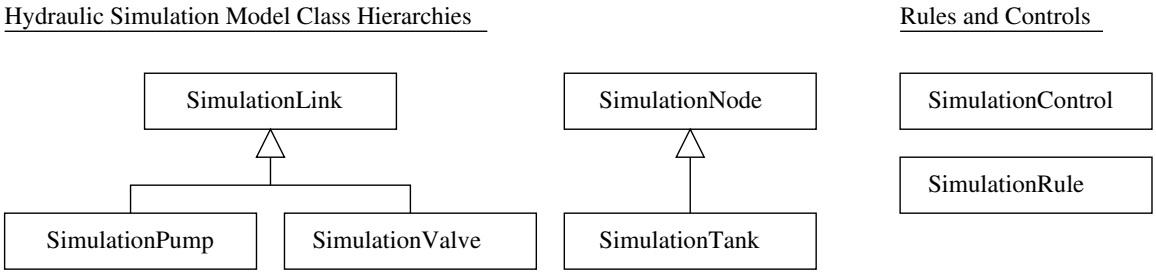

Figure 4.4: EPANET hydraulic model. Left: organization of classes for simulation nodes and links. Right: rule and control classes.

junctions, curves for head-flow relationships in pumps, and controls and rules. The right-hand side of Figures 4.3 shows the organization of classes used in the water network simulation. Networks are modeled as assemblies of nodes and links. Tanks are a specialized form of node. Pumps and valves are a specialized forms of link.

As illustrated in Figure 4.1, network models are transformed into hydraulic models for the purposes of simulation. The latter contain additional methods for the evaluation of: (1) numerical gradients that are part of the numerical simulation, and (2) checks to see if a serious violation of physical behavior has occurred. To keep the book keeping of the program structure tidy, the class Node in network model traces to the class SimulationNode in the hydraulic model. Some of the classes in the hydraulic model (e.g., SimulationLink) contain references to their counterparts in the network model (e.g., Link).

4.2.2 Software Architecture of EPANET in Whistle

The Java implementation of EPANET has been installed in Whistle as an application (i.e., under the pathway src.whistle.application.epanet)

Figure 4.5 shows the architecture of EPANET. The class EpanetMVC defines various kinds of methods used for calling the functions of building, simulation, visualization and control of the water network. Loading the .INP file by InputParse class, The Network class create the basic framework of water network consisting of data structure storing network elements including pump, curve, rule etc, and also some corresponding query functions.

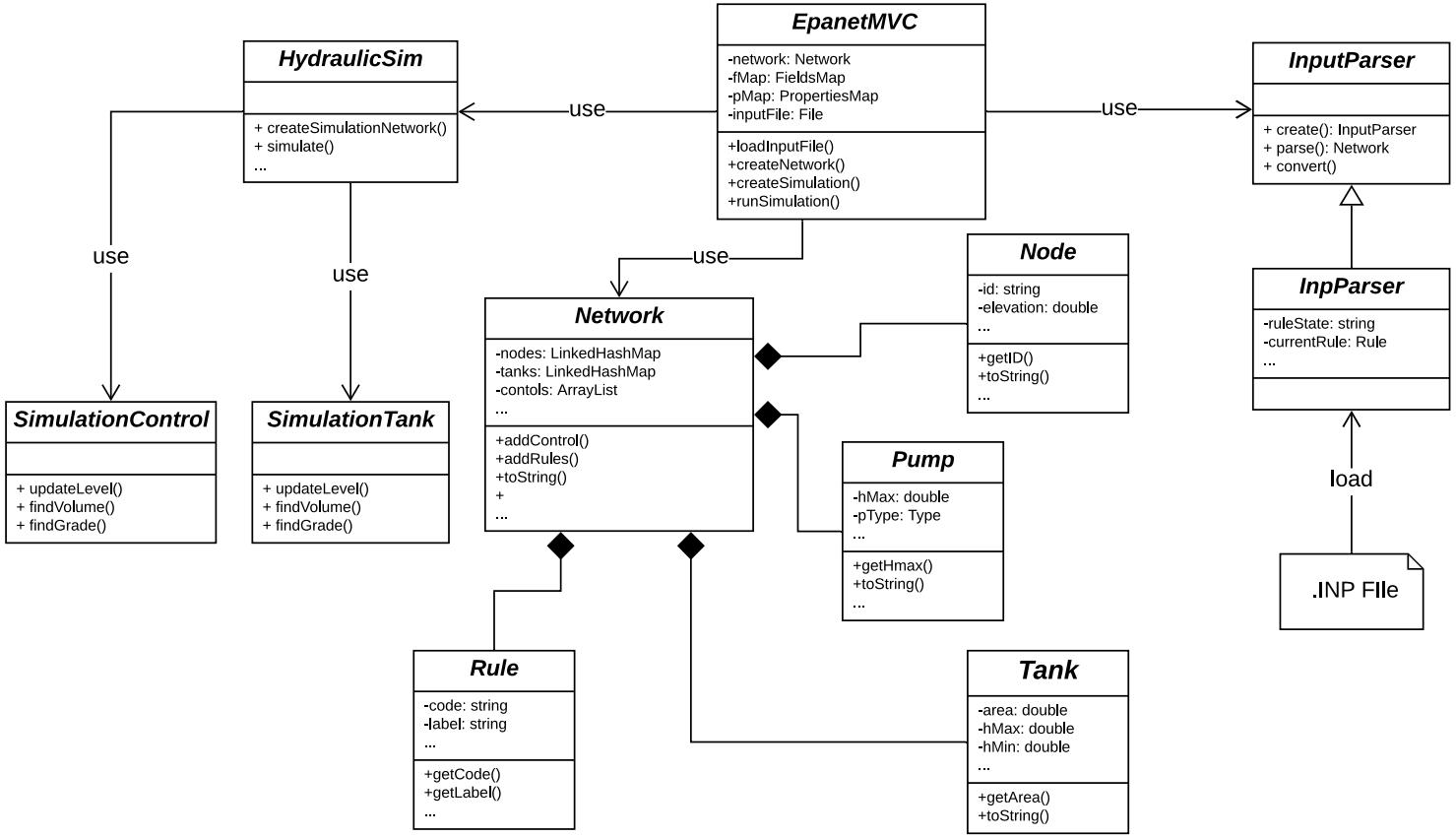

Figure 4.5: Simplified software architecture for EPANET in Whistle.

```
// ------------------------------------------------------------
// Main routine for hydraulic simulation ...
// ------------------------------------------------------------

simulate() {
   for i = 0; i <= no of time steps; i = i + 1 ) {
       runHyd();   <-- Solve network hydraulics in a single time period ...
       timeStep(); <-- Compute duration of next time step ...
   }
}

// ------------------------------------------------------------
// Solve network hydraulics in a single time step ...
// ------------------------------------------------------------

boolean runHyd() {
    computeDemands();  <-- find new demands ....
    computeControls(); <-- find control actions ...

    NetSolveStep nss = netSolve(); <-- Solve network equations
}

// ------------------------------------------------------------
// Compute timeStep to advance simulation ....
// ------------------------------------------------------------

long timeStep() throws ENException {

    // Set default time step ...

    // Revise time step based on smallest time to fill or drain a tank

    tstep = SimulationTank.minimumTimeStep(nTanks, tstep);

    // Compute minimum timestep based on control ...

    tstep = SimulationControl.minimumTimeStep( ... );

    // Compute minimum timestep based on rules ...

    SimulationRule.Result res = SimulationRule.minimumTimeStep( ... );
    tstep = res.step;

    return tstep;
}
```

Figure 4.6: Pseudocode for computing the duration of a time step in hydraulic simulation.

```
   public static Result minimumTimeStep( ...,
       List<SimulationRule> rules, List<SimulationTank> tanks, ... ) {

      // Find interval of time for rule evaluation

      tnow = Htime;
      tmax = tnow + tstep;

      // Step through time, updating tank levels, until either
      // a rule fires or we reach the end of evaluation period.

      do {
         Htime += dt1;                                  // Update simulation clock.
         SimulationTank.stepWaterLevels(tanks,.. dt1); // Find new tank levels.

         // Check rules ...

         int checkInt = check(fMap,pMap,rules,log,Htime,dt1,dsystem);

         // Stop iteration if rules fire

         if (checkInt != 0) break;

         // Update time increment and actual increment

         dt = Math.min(dt, tmax - Htime); dt1 = dt;
   } while (dt > 0);
}

// ------------------------------------------------------------
// Check which rules should fire at current time.
// ------------------------------------------------------------

private static int check( ..., List<SimulationRule> rules,Logger log, ...) {

    // Start of rule evaluation time interval

    long Time1 = Htime - dt + 1;
    List<ActItem> actionList = new ArrayList<ActItem>();
    for(SimulationRule rule : rules) {
        boolean ruleActive = rule.evalPremises(fMap,pMap,Time1,Htime,dsystem);
        updateActionList(rule,actionList, ruleActive );
    }

    int actionResult = takeActions(fMap,pMap,log,actionList,Htime);
    return actionResult;
}
```

Figure 4.7: Pseudocode for computing the minimum permissible time step in SimulationRule.

4.2.3 Step-by-Step Procedure for Hydraulic Simulation

Figure 4.6 contains high-level pseudocode for the procedure and logic required to advance the hydraulic simulation by a time step. The logic for running one time-step is: (1) compute demands (loads), (2) find control actions (if relevant), and (3) compute the duration of the next time step. The `timeStep()` computes the largest time step that can be taken without a tank overflowing (or becoming empty), or violation of a control or rule occurring.

Figure 4.7 shows pseudocode for the lower-level processing of logic within SimulationRule. The method `minimumTimeStep( ... )` in SimulationRule (see Figure 4.4) computes the minimum time step needed to march forward with the simulation without firing any rules. It also updates tank levels. The lower-level details of determining which rules should fire at a specified (current) time are handled by the method `check()`. The main loop of `check()` systematically examines each of the rules to see which of the premises evaluates to true, and needs to be added to an action list. Finally, `takeActions(..)` implements actions (e.g., opening and closing links) on the action list, and returns the number of actions executed. A positive number of actions means that checkInt is non zero, and we have the permissible time step.

The logic for the evaluation of rules is deeply embedded within the EPANET software, opaque and, frankly, flawed. For example, when the `tankStatus()` computation determines that a tank is overflowing (or has inadvertently becomes completely empty), the adjoining links to the tank are temporarily closed. This, in turn, can

completely affect the distribution of flows that are possible in the hydraulic network. A better computational strategy would be to use rules to prevent undesirable tank states in the first place.

4.2.4 Representation and Evaluation of Rules

Understanding the implementation of rule-based control mechanisms in EPANET is a prerequisite to extension formal decision making strategies to include factors that extend beyond hydraulic network simulation. As a first step in this process, this section summarizes the general format for representation of rules, and strategies for evaluation of actions.

Representation of Rules. The fragment of code:

```
IF SYSTEM CLOCKTIME GREATERTHAN 8AM
AND  SYSTEM CLOCKTIME LESSTHAN 6PM
AND  TANK 1 LEVEL BELOW 12
THEN PUMP 2 STATUS IS OPEN
```

illustrates the specification of a simple rule in EPANET. Rules begin with premises, a sequence of simple logical expressions. Simple logical expressions can be combined with the operators `AND` and `OR`. When the premises evaluate to true, one or more actions will be taken.

The language for EPANET rules is limited in the sense that it only knows about system- and component-level entities of concern to the hydraulic simulation. Clock time is an example of a system level entity. The status of tanks and links are component-level concerns.

Condition Clause Format. The key words of condition clause include IF, AND, OR. The condition clause in a Rule-Based Control takes the form of:

Object id attributes relation value

Action Clause Format. The key words of action clause include THEN, ELSE, AND. The action clause in a Rule-Based Control takes the form of:

Object id STATUS/SETTING IS value, where:

object= a category of network object
id= the object's ID label
attribute= an attribute or property of the object
relation= a relational operator
value= an attribute value

Note that only the IF, THEN key words are required, other words are optional.

Simple Example. Figure 4.8 shows a simple example of rule-based control statements with notations of its elements.

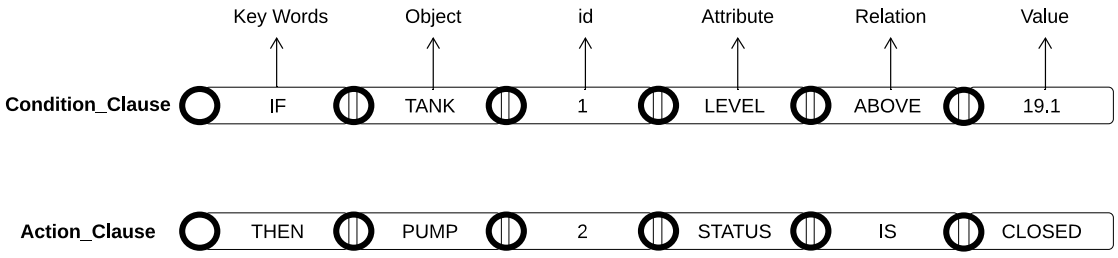

Figure 4.8: Simple Rule Statements Example.

A completed rule usually contains several lines that could be either a condition clause or action clause, e.g.,

IF condition 1
AND condition 2
OR condition 3
AND condition 4
etc.
THEN action 1
AND action 2
etc.
ELSE action 3
AND action 4
etc.

Assessment of Evaluation of Rules in EPANET. Each rule is parsed and processed into its premise - action components. In the procedure for evaluation of rules, each rules is classified as having premises that either evaluate to true or false. Actions are taken on rules having premises that have evaluated to true.

As a first-cut to implementation of rules for a simulation program, EPANET does what you would expect – it simply walks through the list of items on the action list and takes action. If the overall effect corresponds to forward chaining of actions, then this is purely accidental. EPANET does not have any formal support for forward and backward chaining of rules.

4.3 Jena Semantic Models and Rules

4.3.1 Water Network Ontologies

In our project, the research object and scopes are on the water network. In other quarters, the basis for conducting research is the development of the knowledge-based representation of water supply networks.

At first, it is obvious that we create the water network ontology class which is a generalized research object. After that, we combine the knowledge both from the software structure of EPANET (Figure 4.5) and system data model to create an abstract component ontology class with several concrete network ontology subclasses like: pump ontology class and tank ontology class which have the subclass relationship with the component ontology class. In the light of requirements of whistle software visualization and rule-based control mechanism, we also define the functional ontology classes like: shape ontology class and rule ontology class that describe the shape information for visualizing the water network components and rule sentences separately.

Secondly, with the similar working processes, the data properties are defined for each ontology class to describe the feature or practical data of them, which are used to water network simulation or visualization. As for the object properties for each ontology class, it pictures the logical relationship among all kinds of ontology classes later. Figure 4.9 shows a simplified example of water network ontology, the selected

ontology statements are as follows:

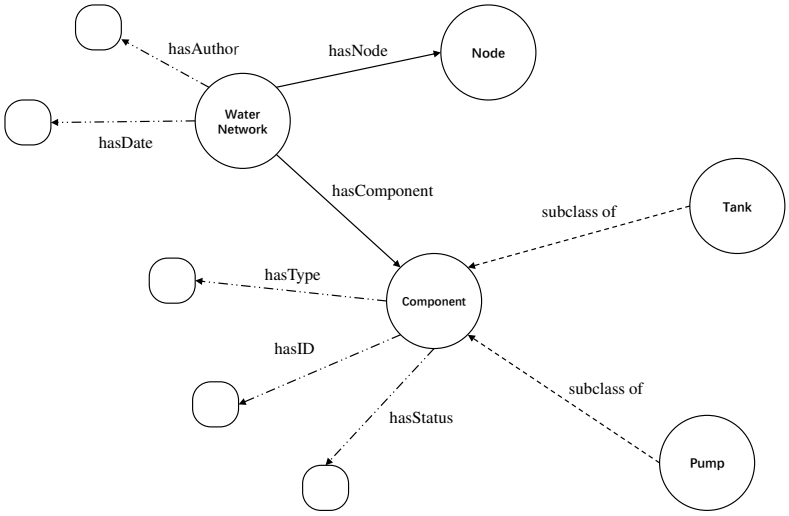

Figure 4.9: Simplified water network ontology.

- Water Network has component.
- Pump has status.
- Tank has initial level.
- Pump is the subclass of component.
- Tank is the subclass of component.

4.3.2 Populating Semantic Graphs with Individuals

The visitor software design pattern is used for data retrieval (and transfer) in both the development of semantic models, and by the Whistle backend software platform.

One such use is in the retrieval of data associated with the individuals of a semantic model. Figure 4.10 shows an example of employment of the visitor design pattern in our project. In this example, we have a total of three Java classes:

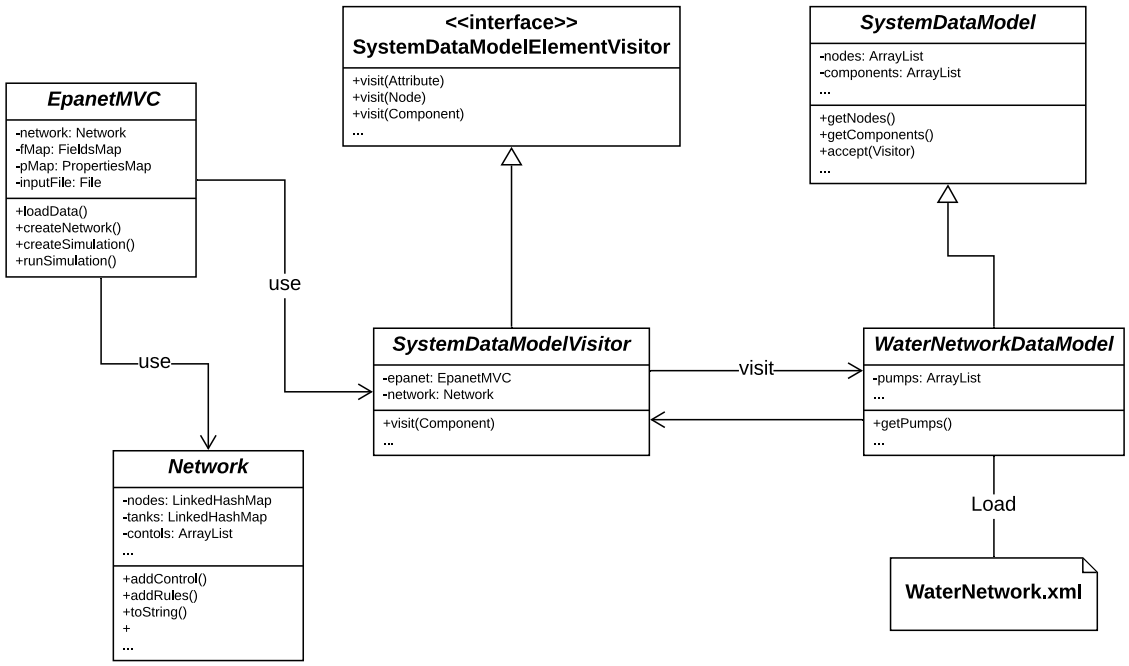

Figure 4.10: Software Architecture of Visitor Design Pattern.

- SystemDataModelElementVisitor.java – This is an interface used for declaring the visit operation for all types of visitable classes.

- SystemDataModelEpanetVisitor.java – This is a concrete visitor for visiting the data for EPANET simulation. It not only implements all the visit methods declared in the visitor interface, but also create its specific visit methods.

- WaterNetworkDataModel.java – It implement the visitable interface and define the accept operation. The visitor object is passed to this object using the accept operation.

Chapter 5: **Case Studies**

To illustrate the capabilities of our experimental software architecture, this chapter presents three case study problems. Case Study 1 describes basic semantic interactions between a pump and a water tank. Case Study 2 describes simulation modeling and rule-based control of a simple water network system in the Whistle implementation of EPANET. Case Study 3 covers semantic modeling for the simple water network system introduced in Case Study 2.

5.1 Case Study 1: Evaluation of Water Tank and Water Pump Operations

This case study explores the interaction among the elements in water network. One of the most important problem presented in water network is the status or operation change of one element based on the status of other elements. From this perspective, the interaction relationship among the water network elements can be divided into two categories: one-to-one and one-to-many relationships. The one-to-one relationship refers to changes in one object's feature that will only be affected by another object. As a case in point, during the operation of pump, the stop of

pump can be uniquely determined by the factor that if the tank capacity is full. A slightly more complicated relationship is the many-to-one relationship which stands for one object's feature can be influenced by multiple factors. For instance, the operational status of pump depends not only on the tank's capacity and also the external environmental conditions, like: system time and weather. Figure 5.1 shows these two relationships.

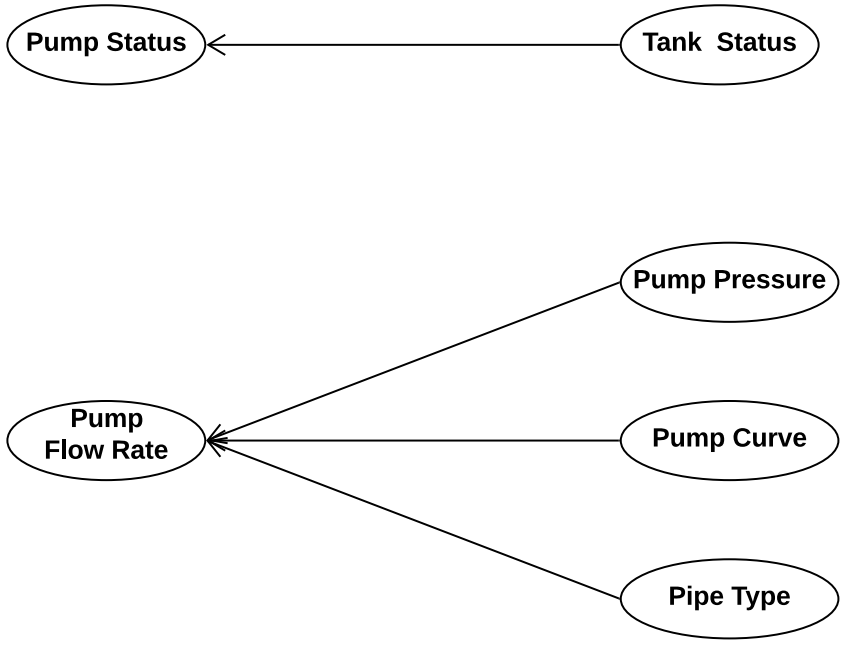

Figure 5.1: One-to-one and many-to-one rules logic relationships.

The above graph is the one-to-one relationship example for pump state we mentioned before. The bottom graph of one-to-many relationship demonstrates the pump's flow rate are affected by other three factors: the pump's water pressure, pump's efficiency curve and pump connected pipe's type. Within these three factors, pump's output water pressure directly decide the volume velocity of pump's flow rate. The pump's efficiency curve can decide the flow rate when pump's efficiency retain

the maximum. The pipe's type limit the pump's flow rate. Necessary conditions for starting the pump are decided by the tank's capacity and system is in working hours. With the logical relationship among the water networks and the development of semantic modeling, we can control the working status of specific elements automatically.

5.1.1 Water Tank and Pump Ontology Models

Ontology data models are comprised of classes and their data and object properties, and the relationships among them. The data properties define the characteristics of classes and the object properties help to describe the relationship or interaction among the classes. In our project, we use the Web Ontology Language (OWL) to define ontology data model. Complete water network ontology data model can be found in Appendix C.

Figures 5.2 and 5.3 are detailed visualizations of the tank and pump ontology models, respectively. Since the pump and tank are two components types in a water network, they are also part of the same semantic model framework. The tank ontology has water level related data properties and area. The max level and min level of decide the volume capacity of tank. In the middle of that two levels, current level is used for monitoring the water level of tank constantly so as to change the status of tank. The pump ontology's data properties mainly has an effect to adjust the operation of pump itself. The output water pressure is directly linked to the flow rate. The pump's curve can well define its operation efficiency.

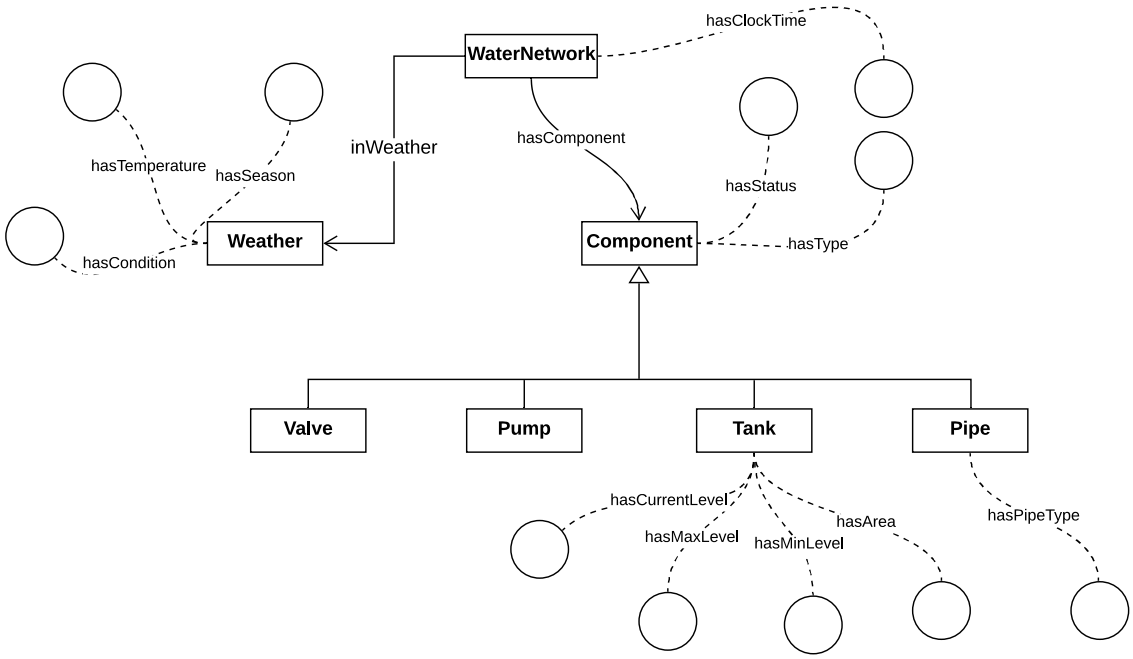

Figure 5.2: Tank ontology model.

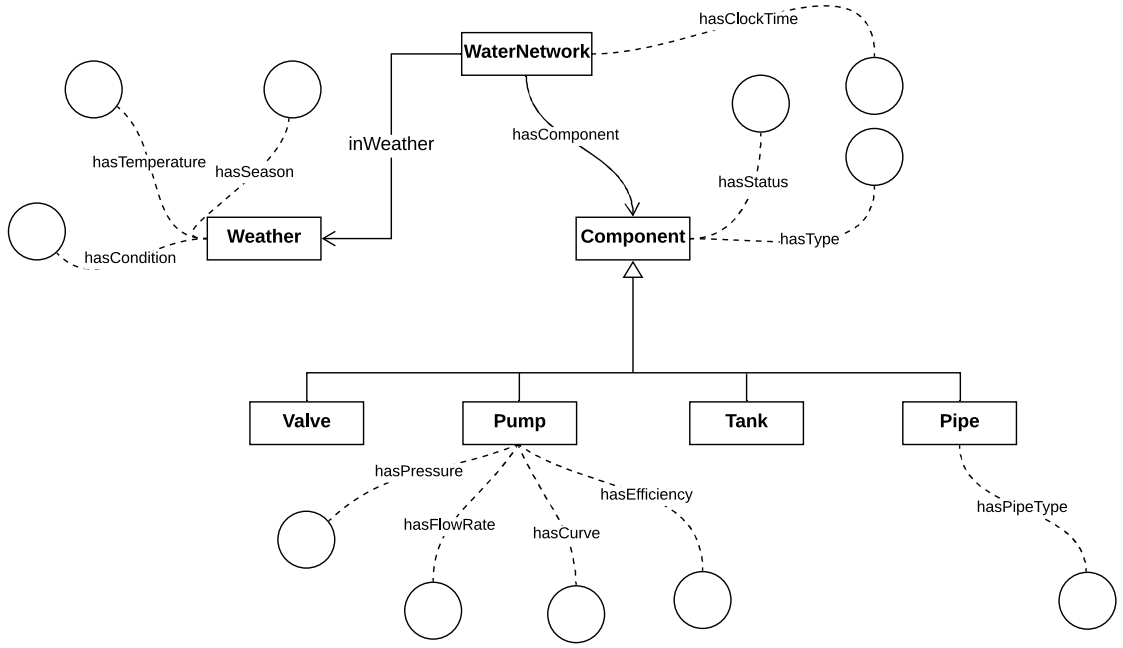

Figure 5.3: Pump ontology model.

In these semantic ontology graphs, it totally presents seven ontology classes: Weather, Component, Attribute, Valve, Pump, Tank and Pipe. Ontology component's two data properties describe its status and type, which inherited by subclass ontology valve, pump, tank and pipe. The pipe ontology has type data property based on its material. Within the water network system, the network components also have to be interacted with other factors, for example the clock time of Attribute ontology, the temperature, season and condition data properties of Weather ontology.

Instantiating the Tank and Pump Ontology Model

Figure 5.4 illustrates the process of generating pump and tank semantic models. Instantiation of ontology model refers to create the ontology individuals with the same properties but provided the real data from the data model which load the data from XML files. Visitor design pattern are used here to retrieve only the specific needed data from data model to create ontology individuals. At the same time, different kinds of rules are imported to the Jena ontology semantic models.

5.1.2 Pump and Tank Jena Rules

The semantic ontology model provides the framework of knowledge representation for specific system, but if can not directly achieve the function of evaluating the properties of classes and the relationships among the classes. In order to realize the rule-based control mechanism, importing the domain-specific rules into semantic model which defining the logical relationships among classes with their properties.

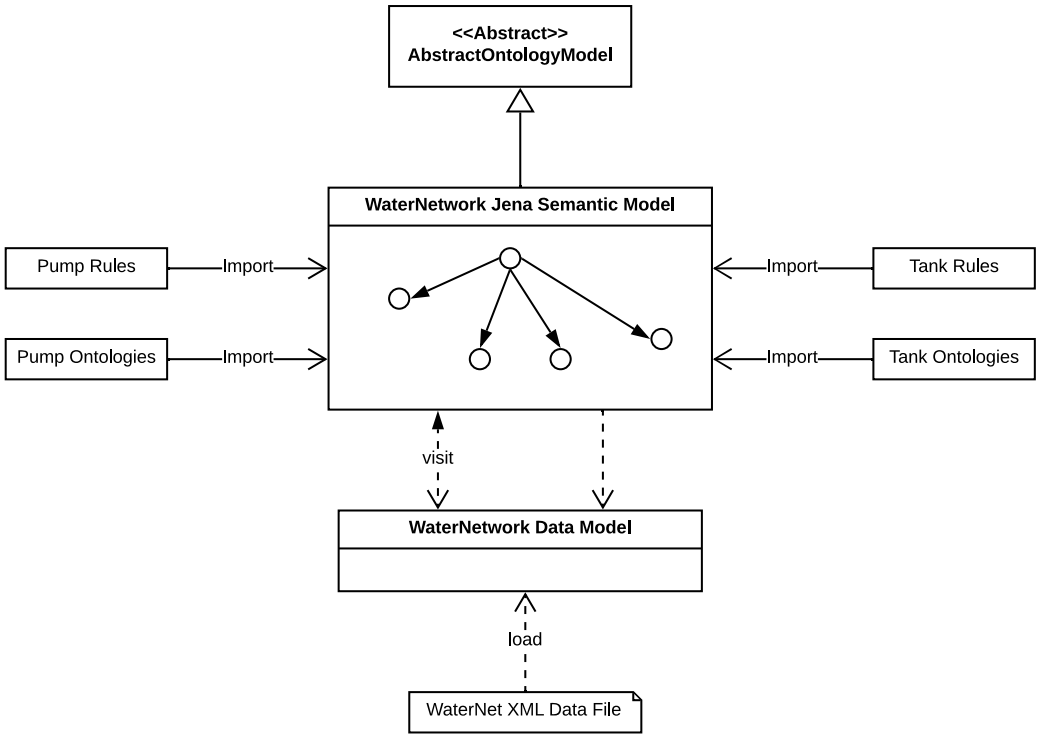

Figure 5.4: Generation of pump and tank semantic models.

With the semantic model reasoner and imported rules, graph transformation can be achieved. In this project, Jena rules are used to define the domain-specific rules. Figure XXXX (??) shows the simplified description of rules.

Pump Rules. The pump rules here mainly focus on the pump's status or characteristics control based on the pump's rest of features. In other words, it defines the logical relationship among the pump's data properties. Figure 5.5 demonstrates a list of pump rules. Within them, the pump's one property is controlled by others. For instance, pump's pressure and flow rate are mutual-decided by two built-in functions getFlowRate() and getPressure() which based on the mathematical relationships between them. In rule 03, a built-in function getMEFlowRate() is created for computing the maximum efficiency flow rate, which late used for updating the flow rate of pump.

```
@prefix wn:  <http://www.ontologies.org/waterNetwork#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

// Rule 01: Identify a component which is a pump ...

[ pump: (?co rdfs: type wn: Component) (?co wn: hasType ?t)
        Equal(?t, 'Pump') -> (?co rdfs: type wn: Pump)]

// Rule 02: Control Pump's flow rate and pressure based on their relationship ....

[ pressure: (?pu rdfs:type wn: Pump) (?pu wn: hasPressure ?pr)
             getFlowRate(?pr,?fl) -> (?pu wn: hasFlowRate ?fl)]

[ flow rate: (?pu rdfs: type wn: Pump) (?pu wn: hasFlowRate ?fl)
             getPressure(?fl,?pr) -> (?pu wn: hasPressure >pr) ]

// Rule 03: Control Pump's flow rate based on efficiency curve ...

[ curve: (?pu rdfs:type wn: Pump) (?pu wn: hasCurve ?cu)
          getMEFlowRate(?cu, ?fl) -> (?pu wn: hasFlowRate ?fl)]
```

Figure 5.5: Abbreviated list of Jena rules for transformation of the Pump Model.

Tank Rules. The tank's status commonly connected to its various kinds of water levels. Figure 5.6 list a set of tank rules. Basically, the tank's status consists of "Full", "Not Full" and "Lack", which are corresponded to three current water level conditions. In rule 02, the status "Full" and "Lack" are determined by comparison with tank's maximum and minimum water levels. Apart from that, in light of seasonal change, maximum and minimum water levels are adjusted.

WaterNetwork Components Interaction Rules

In this part, rules related to elements interaction are described. In other words, the basic elements that make up the water network are mutual-interacted with each other. Figure 5.10 shows a list of water network elements interaction examples. As a

```
@prefix wn:  <http://www.ontologies.org/waterNetwork#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

// Rule 01: Identify a component which is a tank ...

[ tank: (?co rdfs: type wn: Component) (?co wn: hasType ?t)
        Equal(?t, 'Tank') -> (?co rdfs: type wn: Tank)]

// Rule 02: Update tank's status based on current water level tank ....

[ statusFull: (?tn rdfs:type wn: Tank) (?tn wn: hasCurrentLevel ?cl)
              (?tn wn:hasMaxLevel ?ml) Equal(?cl,?ml) -> (?tn wn: hasStatus 'Full')]

[ statusLack: (?tn rdfs:type wn: Tank) (?tn wn: hasCurrentLevel ?cl)
              (?tn wn:hasMinLevel ?mi) lessThan(??cl,?mi) -> (?tn wn: hasStatus 'Lack')]

// Rule 03: Update tank's maximum water level based on season ...

[ maximumSpring: (?tn rdfs:type wn: Tank) (?we rdfs:type wn: Weather)
           (?tn wn: hasSpringLevel ?sl) (?we wn: hasSeason ?se)
           Equal(?se,'Spring') -> (?tn wn: hasMaxLevel ?sl)]
...
```

Figure 5.6: Abbreviated list of Jena rules for transformation of the Tank Model.

case in point, pump's status are decided by tank's volume. At the same time, pump's operational status is controlled by valve. Additionally, the different pipe material types correspond to various pump's upper pressure limit, which is showed in rule 04.

```
@prefix wn:  <http://www.ontologies.org/waterNetwork#>.
@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 ?x) -> (?a rdf:type ?y)]

// Rule 02: Control the pump status based on tank status ...

[ pump01: (?tn rdf:type wn:Tank) (?pu rdf:type wn:Pump)
          (?tn wn: hasStatus ?st) Equal(?st,'Full') -> (?pu wn: hasStatus 'Closed')]

[ pump02: (?tn rdf:type wn:Tank) (?pu rdf:type wn:Pump)
          (?tn wn: hasStatus ?st) notEqual(?st,'Full') -> (?pu wn: hasStatus 'Opened']

// Rule 03: Control valve status based on pump status ...

[ valve02: (?pu rdf:type wn:Pump) (?va rdf:type wn:Valve)
           (?pu wn: hasStatus ?st) Equal(?st,'Opened') -> (?va wn: hasStatus 'Closed')]

// Rule 04: Control the pump pressure based on pipe material type ...

[ pressurelimit(pvc): (?pu rdf:type wn:Pump) (?pi rdf:type wn:Pipe)
         (?pi wn: hasPipeType ?pt ) Equal(?pt,'pvc') -> (?pu wn: hasPressureLimit 100)]
```

Figure 5.7: Abbreviated list of Jena rules for transformation of the water network elements.

5.1.3 Simulation Steps of Water Network Semantic Model

In this section, concrete procedures of modeling and simulation of semantic graph are explained. Firstly, the empty water network semantic model framework are built by employment of JenaWaterNetworkSemanticModel with the process of loading on-

tology data model (Water-Network.owl) in its constructor. Secondly, the created w01Visitor by WaterNetworkDataModelJenaVisitor will visit and retrieve the data from water network data model and then populate them into semantic graph. This is the visitor design pattern in action. Thirdly, XML files are imported to created water network data model. Next, the water network semantic graph are filled with individuals which visited water network data model. Finally, various rules are imported to semantic model and then execute it for rule-based control graph transformations.

```
// Step 01. Create the empty semantic graph model, then load ontoloies.

   JenaWaterNetworkSemanticModel w01 = new JenaWaterNetworkSemanticModel();

// Step 02. Create visitor object model.

   WaterNetworkDataModelJenaVisitor w01Visitor = new WaterNetworkDataModelJenaVisitor();
   w01_visitor.addSemanticModel( w01 );

// Step 03. Get data from XML files.

   WaterNetworkDataModel wdm = new WaterNetworkDataModel();
   wdm.getData("data/WaterNetwork04.xml")

// Step 04. Populate semantic models with individuals.

   wdm.accept( w01_visitor); // Semeantic model visits the data model ...

// Step 05. Add rules. Then execute.

   wdm.addRules( "src/demo/rules/pump.rules",
                 src/demo/rules/tank.rules",
                 src/demo/rules/elementsInteraction.rules")
   wdm.executeRules();
```

Figure 5.8: Fragment of Whistle for instantiating water network data and semantic models, Jena visitors, and loading and executing domain-specific rules.

In order to clearly present the outcomes of execution for Jena reasoner with

rules. We create the selection object to query the semantic model and print the corresponded statements. The followings are the query process for updating the pump's status based on the tank's status and the statements printed results. We query and print the tank and pump's status before and after the rules execution.

```
// Step 01. Create the query selector s1, s2 and print the statement ...

   Selector s1 = new SimpleSelector( (Resource) wn.pump01,
   (Property) wn.hasStatus, (RDFNode) null);
   wn.printStatements("Pump 01 ...", s1);

   Selector s2 = new SimpleSelector ( (Resource) wn.tank01,
   (Property) wn.hasStatus, (RDFNode) null);
   wn.printStatements("Tank 01 ...", s2);

// Step 02. Add rules. Then execute ...

   wn.addRules( "src/demo/rules/pump.rules");
   wn.executeRules();

// Step 03. Create the query selector s2 and print the statement ...

   Selector s3 = new SimpleSelector ( (Resource) wn.pump01,
   (Property) wn.hasStatus, (RDFNode) null);
   wn.printStatements("Pump 01 ...", s3);

   Selector s4 = new SimpleSelector ( (Resource) wn.tank01,
   (Property) wn.hasStatus, (RDFNode) null);
   wn.printStatements("Tank 01 ...", s4);
```

Figure 5.9: Fragment of Whistle code querying and printing the initial semantic graph, loading and executing the pump rules, and then selecting and printing statements in modified semantic graph.

Followings are the results of status' updating outcomes. We can see the status of pump is changed from **Opened** to **Closed** by the semantic graph transformation.

```
[java]
[java] Before execution of rules ...
[java]
[java] Statements: Pump 01 ...
[java] ============================================================
[java] Statement[   1]
[java]   Subject  : http://www.ontologies.org/waterNetwork#pump01
[java]   Predicate: http://www.ontologies.org/waterNetwork#hasStatus
[java]   Object   : "Opened"
[java] ============================================================
[java]
[java] Statements: Tank 01 ...
[java] ============================================================
[java] Statement[   2]
[java]   Subject  : http://www.ontologies.org/waterNetwork#tank01
[java]   Predicate: http://www.ontologies.org/waterNetwork#hasStatus
[java]   Object   : "Full"
[java] ============================================================
[java]
[java] After execution of rules ...
[java]
[java] ============================================================
[java] Statements: Pump 01 ...
[java] ============================================================
[java] Statement[   1]
[java]   Subject  : http://www.ontologies.org/waterNetwork#pump01
[java]   Predicate: http://www.ontologies.org/waterNetwork#hasStatus
[java]   Object   : "Closed"
[java] ============================================================
[java]
[java] Statements: Tank 01 ...
[java] ============================================================
[java] Statement[   2]
[java]   Subject  : http://www.ontologies.org/waterNetwork#tank01
[java]   Predicate: http://www.ontologies.org/waterNetwork#hasStatus
[java]   Object   : "Full"
[java] ============================================================
```

Figure 5.10: Snapshot of semantic model for tank and pump status, before and after execution of Jena rules.

5.2 Case Study 2: Simple Water Network System (Simulation)

This case study demonstrates how a water network data model is imported into the Whistle implementation of EPANET, simulated for 24 hours, and evaluated with respect to rules for controlling the minimum and maximum permissible water level in a storage tank.

5.2.1 Problem Statement

Figure 5.11 is an elevation view of the simple water network system. The network is simple in the sense that there is one reservoir, one storage tank, one pump, four junctions (that place demands on water supply), and one control to regulate water level in the storage tank.

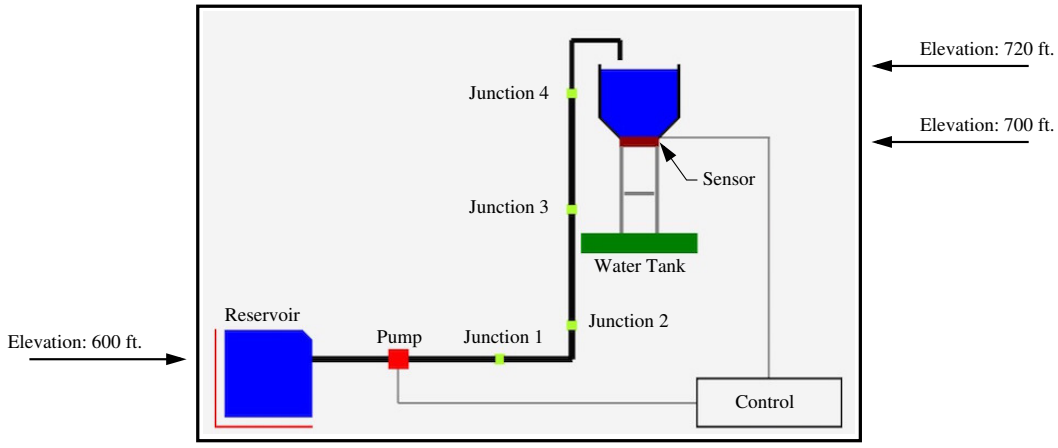

Figure 5.11: Elevation view of simple water network system.

The model assumes that water level in the reservoir will remain constant (at head 600 ft). The base of the water storage tank is at elevation 700 ft. And the maximum depth of water in the storage tank is 20 ft. A pipe with diameter four inches connects

the reservoir to the storage tank. The pump is modeled with a head-flow performance curve that interpolates the data points:

Flow (GPM)	Head (ft)
0.0	150.0
2000.0	100.0
4000.0	60.0

Notice that the 150 ft head-lift specification of the water pump is compatible with the topography of the water network system.

During a 24 hr time cycle (i.e., 12 am to 12 pm), Junctions 1 through 4 have behavior defined by profiles of water demand that vary between 6.0 GPM (off-peak demand) and 30.0 GPM (peak demand).

In the public-domain release of EPANET, pump operations are temporarily closed when either the water level in the tank falls below zero, or the water level exceeds the maximum capacity of the tank. These two rules are hard coded into the EPANET software. This case study employs to rules to regulate the water level of the tank, while also satisfying profiles of customer demand. The logic of the control is very simple:

Rule R01: If the water level in the storage tank falls below 3 ft, then the water pump will be turned on.

Rule R02: If the water level in the storage tank exceeds 17 ft, then the water pump will be turned off.

No attempt is made to optimize scheduling of the pump operations to minimize energy

consumption and/or the cost of pumping operations.

5.2.2 Water Network System Data Model

State-of-the-art approaches to water network simulation with EPANET employ an INP (input) data model format, dedicated to the requirements of EPANET simulation. Here, we need a data format that can support multiple purposes: (1) provide data to EPANET for simulation, (2) provide data to Whistle for simulation and visualization (e.g., see Figure 5.11), and (3) provide data to Apache Jena for semantic model and reasoning.

To this end, Appendix A contains a slightly abbreviated description of XML for the network components and their properties and connectivity, prescribed demands on water supply, and rules to limit minimum/maximum levels of water level in the storage tank. The system data model is capable of serving these multiple purposes primarily because of: (1) the attribute (<attribute>) tags embedded inside the node, way, relation, component and behavior tags, and (2) the use of visitor design patterns to retrieve problem-specific data. Figure 5.12 shows, for example, fragments of <node> and <way> data for the section of pipeline connecting Junction 4 (Node 008) to the Water Tank (Component C03). The <node> tags contain attribute data on elevation, demand and pattern – these detail are employed by the EPANET hydraulic network simulation. Similarly, the <way> tag contains references to sequences of nodes 008 through 011, and engineering parameters defining pipeline properties. The attribute keys `end1` and `end2` are endpoints of link segments in EPANET.

```
<node ID = "008" x = "370.0" y = "345.0">
   <description text="Network Junction 4" />

   <attribute key = "type"      value = "Junction" />
   <attribute key = "elevation" value = "660" units ="ft" />
   <attribute key = "demand"    value = "30.0" />
   <attribute key = "pattern"   value = "P001" />

   ... details of junction (rectangle) shape removed ...

</node>

... details of nodes 009 and 010 removed ...

<node ID = "011" x = "420.0" y = "380.0">
   <attribute key = "type"      value = "Point"/>
   <attribute key = "elevation" value = "660" units ="ft" />
</node>

<way ID="006">
   <description text="Connect Junction 4 to Water Tank." />
   <attribute key = "type" value = "Pipeline"/>

   <!-- Connectivity of pipe ends -->

   <attribute key = "end1" value = "008" />
   <attribute key = "end2" value = "C03" />

   <!-- Sequence of nodal coordinates -->

   <node ID="008" />
   <node ID="009" />
   <node ID="010" />
   <node ID="011" />

   <!-- Engineering parameters -->

   <attribute key = "diameter"  value =    "4" />
   <attribute key = "roughness" value =  "100" />
   <attribute key = "minorloss" value =    "0" />
   <attribute key = "status"    value = "Open" />

   ... details of pipeline (linestring) shape removed ...

</way>
```

Figure 5.12: Abbreviated definition of pipeline segment connecting Junction 4 (node 008) to the water storage tank (component C03).

5.2.3 Specification of Nodal Demands

The following fragment of Whistle code:

```
<!-- ================================================================ -->
<!-- Load pattern for time-intervals of consumer demand (24 hrs) ... -->
<!-- ================================================================ -->

<behavior ID = "P001" type = "Pattern">
    <description text = "Consumer demand load pattern (24 hrs)" />
    <attribute key = "multipliers"  value = "0.20  0.20  0.20  0.30  0.40  0.50
                                             0.70  0.80  1.00  1.00  1.00  1.00
                                             1.00  1.00  1.00  1.00  1.00  1.00
                                             1.00  1.00  0.80  0.70  0.50  0.20"/>
</behavior>

<!-- ================================================== -->
<!-- Load pattern for pump operations (that repeats) ...   -->
<!-- ================================================== -->

<behavior ID = "P002" type = "Pattern">
    <description text = "Pattern for pump operations" />
    <attribute key = "multipliers"  value = "1.0 1.0 1.0 1.0" />
</behavior>
```

defines patterns of behavior for: (1) consumer (nodal) demand over a 24 hr time period, and (2) pump operations. As illustrated in Figure 5.13, we assume that consumer nodal demands will peak during daylight/working hours. For demonstration purposes, Junctions 1 through 4 are assumed to have a base demand of 30 GPM. The actual demand at a particular time is the base demand (30 GPM) times the multiplier value for pattern P001.

The pattern for pumping operations (P002) is specified for time intervals lasting 4 hrs; during a 24 simulation time frame, the pattern for pumping operations will automatically repeat six times. A multiplier value of one indicates a desire for the pump to operate. In practice, this pattern should be over-ruled by controls and rules.

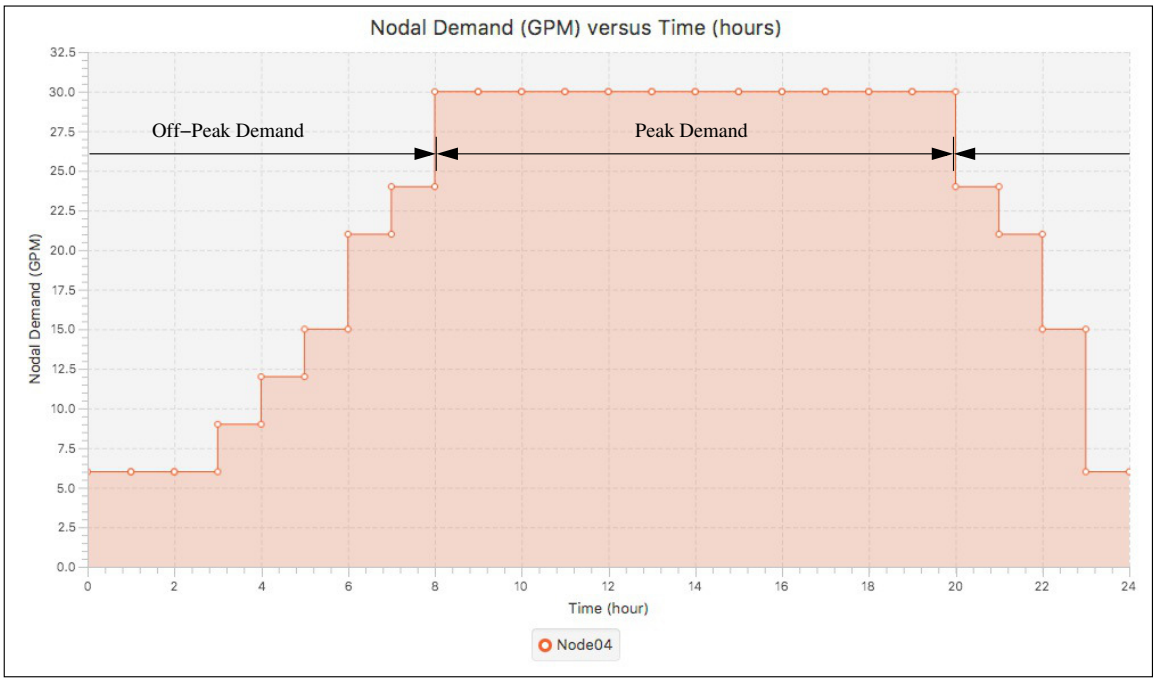

Figure 5.13: Plot of nodal demand (GPM) versus time (hours).

5.2.4 Specification of Water Network Rules

The System Data Model (see Section 2.4.3) provides computational support for the representation and evaluation of mathematical constraints. Three common types of constraint are inequality, equality and logical expressions. Here, we make a minor extension of the framework to include textual descriptions of rules that can be parsed into the EPANET data model. To illustrate this capability, the fragment of Whistle code:

```
<constraint ID = "R01" type="Rule">
   <description text="Open pump (turn on) when tank water level is low" />
   <attribute key = "expression" value = "IF TANK C03 LEVEL BELOW 3.0
                                          THEN PUMP C04 STATUS IS OPEN"/>
</constraint>

<constraint ID = "R02" type="Rule">
   <description text="Close pump (turn off) when tank water level is high" />
   <attribute key = "expression" value = "IF TANK C03 LEVEL ABOVE 17.0
                                          THEN PUMP C04 STATUS IS CLOSED"/>
```

```
  </constraint>
```

defines rules for: (1) turning the pump on when the tank water level falls below 3 ft, and (2) turning the pump off when the water level reaches below 17 ft.

5.2.5 Assembly and Execution of Simulation Model in Whistle

The step-by-step procedure for assembly and execution of the EPANET simulation model in Whistle is as follows:

```
// ============================================================================
// Exercise EPANET model ...
// ============================================================================

import whistle.application.epanet.EpanetMVC;

import whistle.gui.chart.AreaChart;

import whistle.util.system.model.SystemDataModel;
import whistle.util.system.visitor.SystemDataModelEpanetVisitor;
import whistle.util.data.Curve;
import whistle.util.data.DataModel;

program ("Case Study 2: Simulate Small Water Network with EPANET") {

   // Step 01: Create EpanetMVC and empty EPANET network model ...

   epanet01 = EpanetMVC();
   epanet01.createNetwork();

   // Step 02: Import and print data model for small water network ...

   sdm01 = SystemDataModel();
   sdm01.getData ("data/umd-water-network03.xml");

   print sdm01.toString();

   // Step 03: Create epanet model visitor ...

   epavisitor01 = SystemDataModelEpanetVisitor();
   epavisitor01.add( epanet01 );

   // Step 04: Populate epanet model with system data model info ....
```

```
sdm01.accept ( epavisitor01 );

// Step 05: Compile and print EPANET network model ...

epanet01.compile();
print epanet01.toString();

// Step 06: Create hydraulic simulation model ...

epanet01.createSimulationModel();

// Step 07: Run hydraulic simulation model ...

epanet01.runHydraulicSimulation();

// Step 08: Build models of network performance ...

epanet01.networkPerformance();
epanet01.buildPerformanceCurves();

// Step 09: Transfer network performance to data models ...

dm01 = epanet01.getNodeDemands();
dm01.setTitle( "Node Demand (GPM) versus Time (hours)");
dm01.setYLabel("Flow Rate Q (GPM)");
dm01.setXLabel("Time (hours)");

print dm01.toString();

... etc ...

// Step 10: Create area charts of network performance versus time ...

jfx01 = AreaChart();
jfx01.setTitle("Nodal Demand (GPM) versus Time (hours)");
jfx01.setSize( 1000, 600 );
jfx01.setXLabel("Time (hour)");
jfx01.setYLabel("Nodal Demand (GPM)");
jfx01.setXRange(    0.0,  24.0 );
jfx01.setYRange(    0.0,  40.0 );

// Transfer data models to plot component ...

jfx01.addCurve( "Node04" );
c01    = dm01.getCurve ("Junction(004)");
nsteps = c01.getNoPoints();
for (i = 0; i < nsteps-1; i = i + 1) {

   x1 = c01.getX(i); y1 = c01.getY(i);
   x2 = c01.getX(i + 1);

   jfx01.addPoint( x1, y1 );
   jfx01.addPoint( x2, y1 );
```

```
   }

   // Display area chart ...

   jfx01.display();

   ... etc ...

   print "--- ";
   print "--- ======================================================";
   print "--- Finished !! ... ";
}
```

The step-by-step procedure begins with import of the appropriate Java classes into Whistle. For example, the statement:

```
import whistle.application.epanet.EpanetMVC;
```

makes methods in the class `whistle.application.epanet.EpanetMVC` available to the Whistle script. Steps 1 and 2 show the flow of computations needed to create an empty EPANET network model, and load the contents of `umd-water-network03.xml` into the System Data Model. An EPANET system data model visitor is created in Step 3. In Step 4, the EPANET system data model visitor visits the system data model and extracts the data needed to populate the EPANET network model. The compilation procedure (Step 5) resolves references (i.e., connects) among the various data model elements.

Figure 5.14 shows a side-by-side comparison of the system data model and EPANET network models. In the system data model, pipeline profiles are modeled as sequences of nodes within ways (<way>). EPANET simplifies the geometry

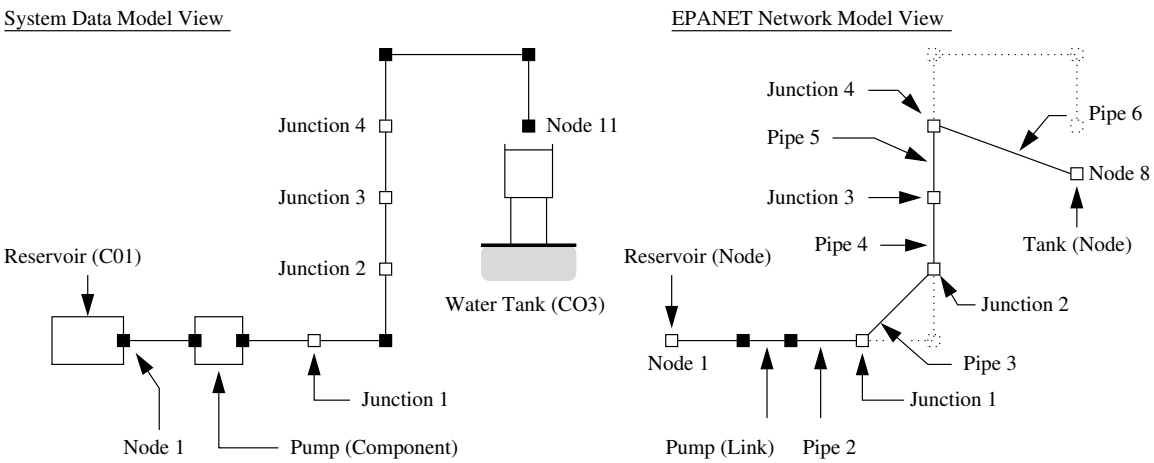

Figure 5.14: Side-by-side comparison of system data model and EPANET network model views.

by considering only the endpoints of link segments. Hence, while the former has 11 nodes, the EPANET network has only 8 nodes. The system data model models pumps, reservoirs and tanks as components. EPANET models tanks and reservoirs as tanks, which, in turn are extensions of the class node. EPANET models pumps as an extension of the class link.

Steps 6 and 7 are dedicated to creation and execution of the hydraulic simulation model. It is important to notice that at this point, the entire time-stepping procedure is embedded within `epanet01.runHydraulicSimulation()`. Future versions will provide support for elaboration of time-stepping procedures from the Whistle script. By default, EPANET exports results of the hydraulic simulation to a data file. The methods `networkPerformance()` and `buildPerformanceCurves()` print summaries of the hydraulic simulation and assemble data models of water network system performance. In Steps 9 and 10, curves of network performance are transfered data model curves, which, in turn, are transfered to JavaFX charts.

5.2.6 EPANET Simulation Results

Figure 5.15 shows the time-history of water depth in the tank (ft) versus time (hours), along with intervals of time where the pump status is on and off.

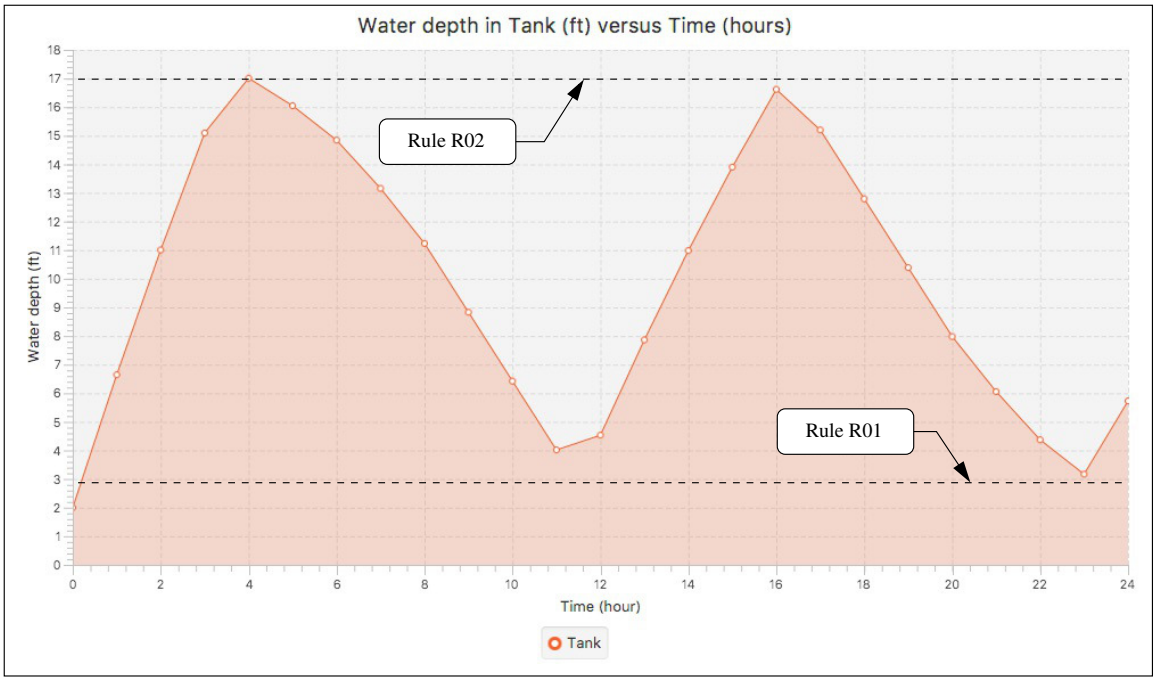

Figure 5.15: Plot of water depth in tank (ft) versus time (hours).

When the simulation begins (12 am) the demand for water is off-peak, and the tank water level is very low (2 ft). The pump takes approximately 4 hour to raise the water level depth up to 17 ft, whence Rule R02 is activated and the pump is closed. From 4 am to 11 am the water is removed from the tank (due demands at Junctions 1 through 4) and the pump is turned back on when Rule R01 is activated. the cycle repeats during the latter half of the day.

Figures 5.16 and 5.17 show flows in/out of the reservoir and water storage tank, and the corresponding flows along pipes. Notice that when the pump is on, water is

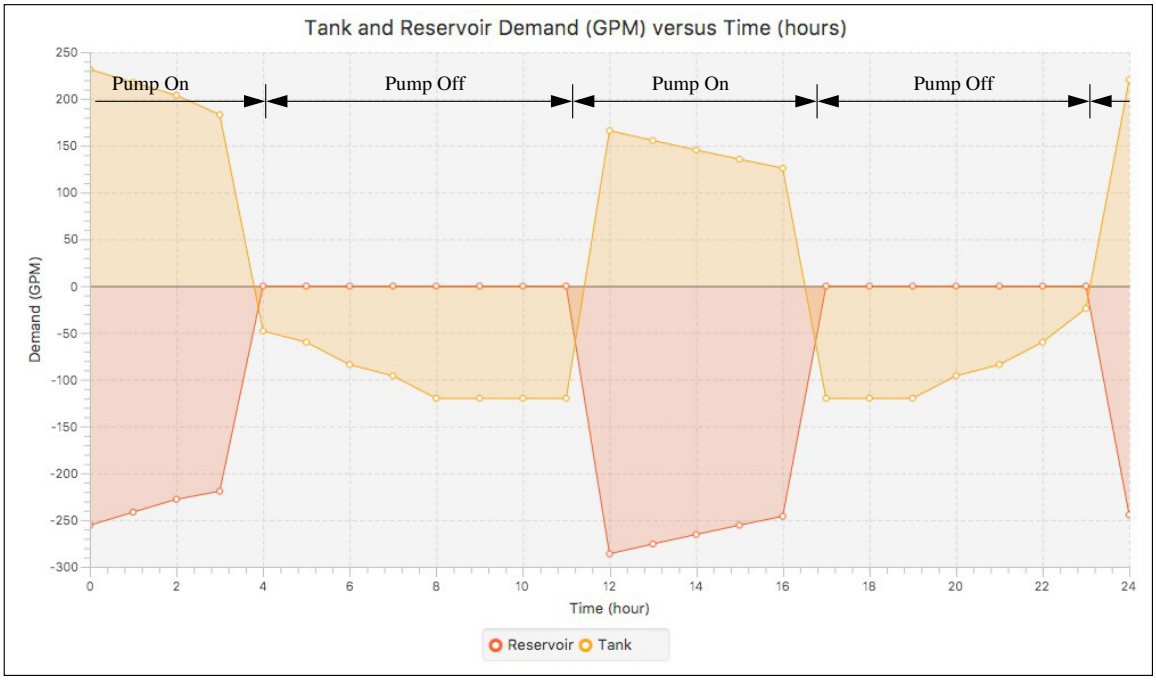

Figure 5.16: Plot of tank and reservoir demand (GPM) versus time (hours).

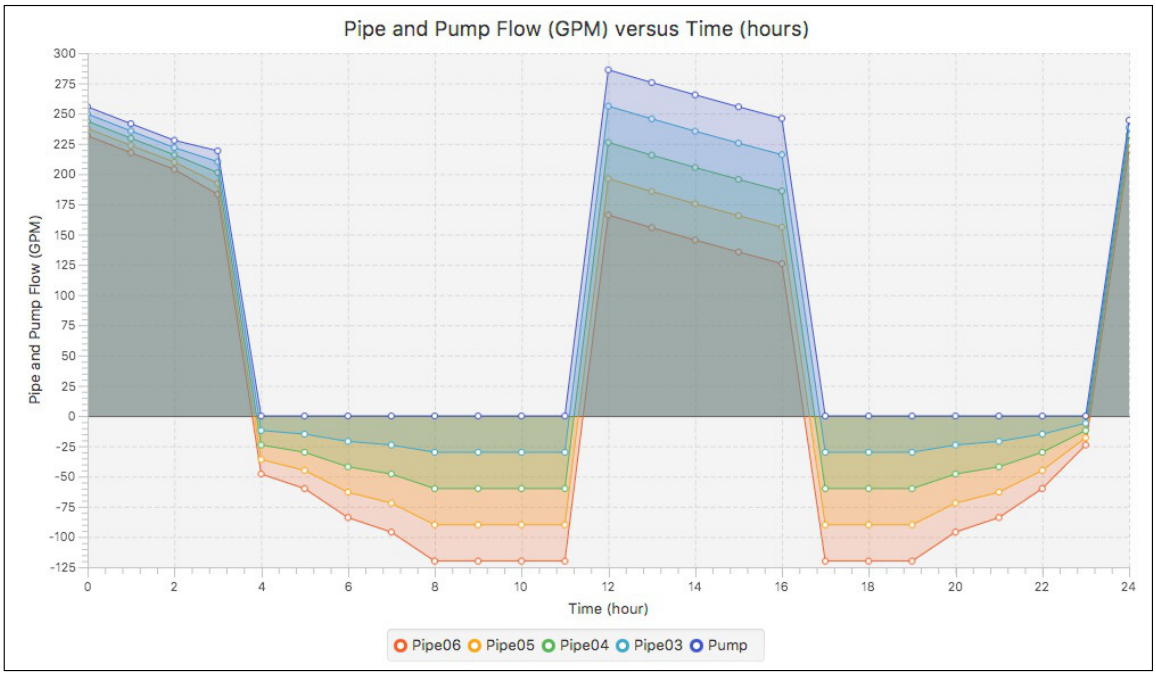

Figure 5.17: Plot of pipe and pump flow (GPM) versus time (hours).

drawn from the reservoir to satisfy both the demand for water at the junctions and fill the storage tank. Turning the pump off closes the link connecting nodes 2 and 3 in the EPANET model. This forces water to be extracted from the tank to satisfy demands by Junctions 1 through 4.

5.3 Case Study 3: Simple Water Network System (Semantic Model)

The purpose of Case Study 3 is to systematically assemble an Apache Jena Semantic Model for the simple water network system (see Figure 5.11). Rule-based control is applied for the two tank rules discussed in Case Study 2. The development process includes the following steps: (1) process of building semantic model, (2) simple query of existed statements, and (3) evaluation of Jena rule-based control mechanism.

5.3.1 Manual Synthesis of Jena Semantic Model + Rules

To simplify the development process, the semantic graph stores only ontology classes and associated data/object properties directly related to the evaluation of water network rules. Two steps are needed to manually create the classes and participating data properties, and then populate the semantic graph with individuals.

Step 1: Create Ontology and Data Properties. The manual specification of classes and data properties is encapsulated in the method **buildOntology()**:

```
public void buildOntology() {

    // Define Classes and data properties ...

    waterNetwork = model.createClass( ns + "WaterNetwork" );
    component    = model.createClass( ns + "Component" );
    pump = model.createClass( ns + "Pump" );
    tank = model.createClass( ns + "Tank" );
    elementsStatus = model.createClass( ns + "ElementsStatus" );

    // Define the relationship among classes ...

    component.addSubClass ( pump );
    component.addSubClass ( tank );
```

```
    // Create data and object properties for the class WaterNetwork ...

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

    hasStatus = model.createDatatypeProperty(ns + "hasStatus");
    hasStatus.setDomain(component);
    hasStatus.setRange(XSD.xstring);

    ... source code removed ...
}
```

Figure 5.18 shows the hierarchy of classes and data properties:

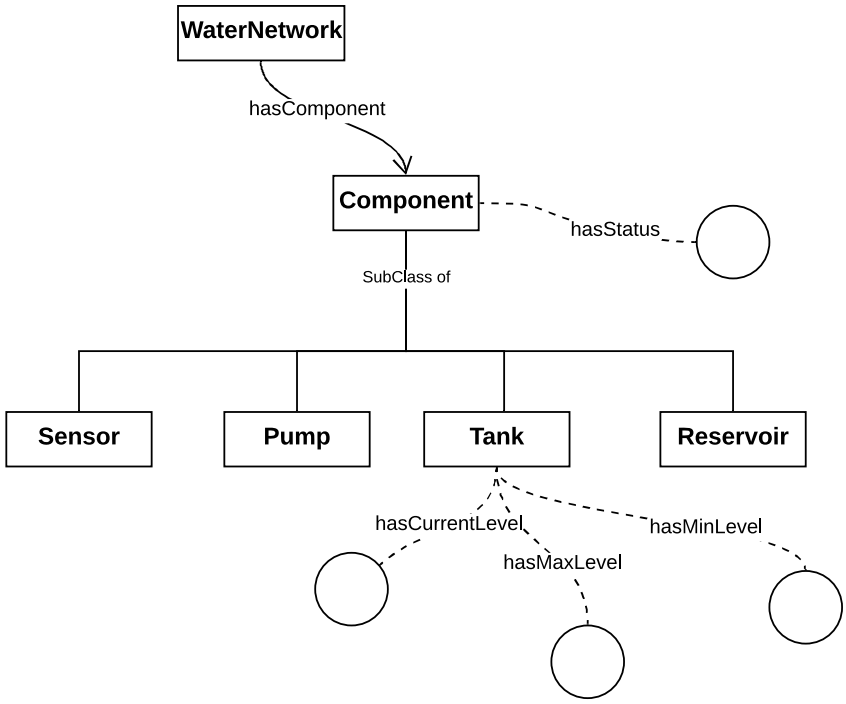

Figure 5.18: Simple Water Network Graph.

Step 2: Add Individuals to Semantic Graph. In semantic modeling, individuals are the counterpart of objects in object-oriented programming. The method

addIndividuals():

```
public void addIndividuals() {

    // Add pump01, tank01, w01, e01 to the model ...

    pump01 = pump.createIndividual(ns + "pump01");
    tank01 = tank.createIndividual(ns + "tank01");

    // Create statement "pump01 has status Opened" ...

    Literal statusOpen = model.createTypedLiteral("Opened", XSDDatatype.XSDstring);
    Statement pumpStatus = model.createStatement( pump01, hasStatus, statusOpen);

    // Create statement "tank01 has max level 17.0" ...

    Literal maxLevel = model.createTypedLiteral("17.0", XSDDatatype.XSDdouble);
    Statement tankMaxLevel = model.createStatement( tank01, hasMaxLevel, maxLevel);

    // Create statement "tank01 has min level 3.0" ...

    Literal minLevel = model.createTypedLiteral("3.0", XSDDatatype.XSDdouble);
    Statement tankMinLevel = model.createStatement( tank01, hasMinLevel, minLevel);

    // Add statements to semantic model ...

    model.add(pumpStatus);
    model.add(tankMaxLevel);
    model.add(tankMinLevel);
}
```

creates one pump (pump01) and one tank (tank01), sets the pump status to open (which means the pump is operating), and two statements for permissible minimum and maximum values of water level.

A simplified test program implementation will call these two methods as follows:

```
TestWaterNetwork wn = new TestWaterNetwork();
wn.buildOntology();
wn.addIndividuals();
```

Step 3: Add Rules. With the ontologies, data properties and individuals in place, rules can be written to keep the water level in a tank within permissible bounds.

```
// Rule 01: Update the pump status based on the tank's current water level ...

[ tank01: (?p rdf:type wnw:Pump) (?t rdf:type wnw:Tank) (?t wnw:hasCurrentLevel ?c)
          (?t wnw:hasMinLevel ?mi) (?p wnw:hasStatus ?ps) lessThan(?c, ?mi) ->
          drop(4) (?p wnw:hasStatus "Open")]

[ tank02: (?p rdf:type wnw:Pump) (?t rdf:type wnw:Tank) (?t wnw:hasCurrentLevel ?c)
          (?t wnw:hasMaxLevel ?ma) (?p wnw:hasStatus ?ps) greaterThan(?c, ?ma) ->
          drop(4) (?p wnw:hasStatus "Closed")]
```

Jena rules are evaluated left to right. Statements to the left of the arrow are premises; those to the right of the arrow are actions. In English, rule `tank01` says: If $?p$ and $?t$ are of type Pump and Tank, respectively, and the current water level in the tank ($?c$) is less than the permissible minimum value ($?mi$), then drop (remove) the statement ($?t$ wnw:hasMinLevel $?mi$) from the model, and replace it with ($?p$ wnw:hasStatus "Open"). In other words, if the current water level in the tank is too low, then turn the pump on. Rules tank01 and tank02 are the Jena Rules counterpart of rules R01 and R02 in Case Study 2.

5.3.2 Exercising the Jena Semantic Model + Rules

The section exercises the Jena Semantic Model and Rules in a four-step procedure: (1) query the condition of the pump and tank, (2) dynamically update the current water level of tank, (3) load rules into the model and execute, and (4) query the

modified semantic graph for condition of pump and tank and to check the execution result. A user-defined function **changeWaterLevel()** simulates a time sequence of changes to the water level.

Step 1: Query Condition of Tank and Pump. The fragment of code:

```
Selector s1 = new SimpleSelector( (Resource) wn.tank01,
                                  (Property) null, (RDFNode) null);
Selector s2 = new SimpleSelector( (Resource) wn.pump01,
                                  (Property) wn.hasStatus, (RDFNode) null);

wn.printStatements("Tank 01...", s1);
wn.printStatements("Pump 01...", s2);
```

queries the semantic graph for statements relating to tank (tank01) and pump (pump01) resources. The printed output is as follows:

```
Statements: Tank 01...
============================================================
Statement[   1]
  Subject  : http://www.ontologies.org/waterNetwork#tank01
  Predicate: http://www.ontologies.org/waterNetwork#hasCurrentLevel
  Object   : "2.0^^http://www.w3.org/2001/XMLSchema#double"
Statement[   2]
  Subject  : http://www.ontologies.org/waterNetwork#tank01
  Predicate: http://www.ontologies.org/waterNetwork#hasMinLevel
  Object   : "3.0^^http://www.w3.org/2001/XMLSchema#double"
Statement[   3]
  Subject  : http://www.ontologies.org/waterNetwork#tank01
  Predicate: http://www.ontologies.org/waterNetwork#hasMaxLevel
  Object   : "19.0^^http://www.w3.org/2001/XMLSchema#double"
Statement[   4]
  Subject  : http://www.ontologies.org/waterNetwork#tank01
  Predicate: http://www.ontologies.org/waterNetwork#hasInitialLevel
  Object   : "2.0^^http://www.w3.org/2001/XMLSchema#double"
Statement[   5]
  Subject  : http://www.ontologies.org/waterNetwork#tank01
  Predicate: http://www.ontologies.org/waterNetwork#hasStatus
  Object   : "NotFull"

Statements: Pump 01...
```

```
============================================================
Statement[   1]
  Subject  : http://www.ontologies.org/waterNetwork#pump01
  Predicate: http://www.ontologies.org/waterNetwork#hasStatus
  Object   : "Opened"
============================================================
```

Here we see that the initial water level in the tank is 2 ft and the pump status is Opened (i.e., the pump is on).

Step 2: Dynamically Update Water Level in Tank. The fragment of code:

```
wn.changeCurrentLevel();
```

dynamically updates the water level in the tank.

Step 3: Load Rules into Semantic Model and Execute. The fragment of code:

```
wn.addRules();
wn.executeRules();
```

adds rules to the semantic model and then executes them.

Step 4: Query Modified Semantic Graph. The fragment of code:

```
Selector s3 = new SimpleSelector( (Resource) wn.tank01,
                                  (Property) wn.hasCurrentLevel, (RDFNode) null);
Selector s4 = new SimpleSelector( (Resource) wn.pump01,
                                  (Property) wn.hasStatus, (RDFNode) null);

wn.printStatements("Tank 01...", s1);
wn.printStatements("Pump 01...", s2);
```

queries the modified semantic graph for the tank water level and pump status. It generates the output:

```
================================================================
Statements: Tank 01...
================================================================
Statement[   1]
  Subject  : http://www.ontologies.org/waterNetwork#tank01
  Predicate: http://www.ontologies.org/waterNetwork#hasCurrentLevel
  Object   : "21.0^^http://www.w3.org/2001/XMLSchema#double"
================================================================

Statements: Pump 01...
================================================================
Statement[   1]
  Subject  : http://www.ontologies.org/waterNetwork#pump01
  Predicate: http://www.ontologies.org/waterNetwork#hasStatus
  Object   : "Closed"
================================================================
```

5.3.3 EPANET Ontology and Rules

Figure 5.19 builds on Figure 5.14 and drives the need for water network system ontologies and rules, designed from system data model and EPANET perspectives.

Appendix B contains a draft of an ontology and Jena Rules specfication for EPANET. The classes have names (e.g., Junction) familiar to users of EPANET, and would be instantiated (see right-hand side of Figure 5.19) by visiting the EPANET network model. The rules capture interactions between the water tank level and status of the pump.

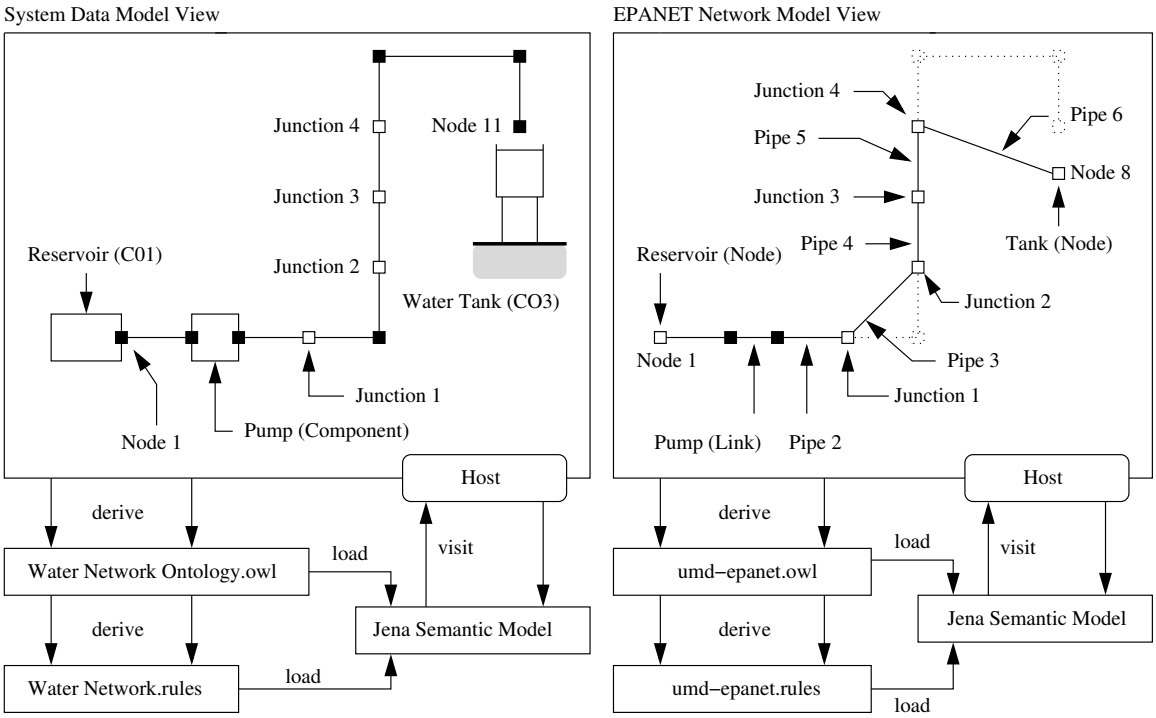

Figure 5.19: Two perspectives of development for water network system ontologies and rules. Left: systems data model view, Right: EPANET network model and hydraulic simulation.

Chapter 6: **Conclusions and Future Work**

6.1 Summary and Conclusions

Water resources play a pivotal role in the operation of urban systems because they are necessary for sustainability of life and economic development. From a day-to-day operations standpoint, we need to understand what strategies of operation lead to high levels of efficiency? And from a long-term planning perspective, accurate estimation of the future demand and availability of water resources is essential for achieving healthy and sustainable urban behavior. To this end, the long-term goals of this project are to develop a platform infrastructure where decision making for short- and long-term planning is supported by state-of-the-art simulation working alongside semantic representations of water network system knowledge and rules-based reasoning. Such approaches to decision making (see Figure 1.2) are expected to be deep and broad in their consideration of knowledge and rules, and ideally, also transparent.

This thesis has focused on simulation of hydraulic networks with the Java-based implementation of EPANET. While the software is now almost twenty years old, it remains a work in progress. During the course of this study, many enhancements to

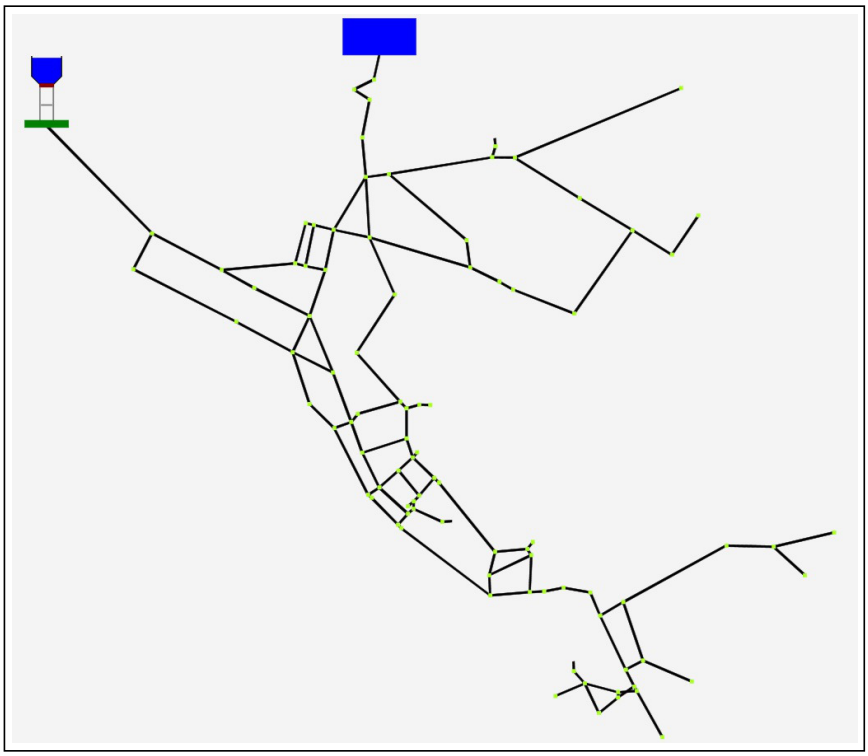

Figure 6.1: Plan view of urban water network system.

the source code have been made. Some of them are very simple (e.g., a toString() method for the network model) and some are more involved (e.g., development of system data model visitors for EPANET and Jena). State-of-the-art practice with EPANET is to define problems in a `.INP` format. For our purposes, however, we needed a problem description that would provide data for multiple purposes: (1) to seed the hydraulic simulation, (2) to provide visual representations of the network layout and components, and (3) to provide a framework for creating semantic models and evaluation of rules. In our approach, data is stored in a general purpose system data model. We instantiate data for hydraulic simulations in EPANET by visiting the system data model. Similarly, we instantiate individuals in semantic models of EPANET by having Jena visit the system data model. Finally, visual representations

of the hydraulic network are created in Whistle by extracting composite hierarchies from the system data model, and then displayed as domain-specific layers (e.g, pipe layout, junctions, pumps, tanks, reservoirs). The framework is extensible to layers for GIS (e.g., Open Street Map) These models can be loaded into Whistle as composite hierarchies and spatial distributions of domain (e.g., residential homes).

The scope of this study has been limited to the small water network considered in Case Studies 2 and 3. However, analysis of larger hydraulic network systems will also work (see Figure 6.1). The strategies of rule based control have been simple. For example, when the pump is turned off in Case Study 2, the pump link is also closed. As illustrated in Figures 5.16 and 5.17, this forces water to be drawn from the supply tank rather than the tank. A more realistic analysis [30] would provide a bypass around the pump, and use multiple pumps operating in parallel to provide redundancy of operations during periods of pump failure. Larger hydraulic network systems also have multiple points of sensing – and since sensor operations are not always 100% reliable, this raises the need for decision making under uncertainty and detection of faulty equipment. A limitation of the present study is that these factors were not taken into account.

6.2 Future Work

By modern-day standards, EPANET's builtin mechanisms for control and rule-based reasoning are very simple. Throughout a simulation, individual rules are classified as having premises that either evaluate to true or false. Actions are then taken on

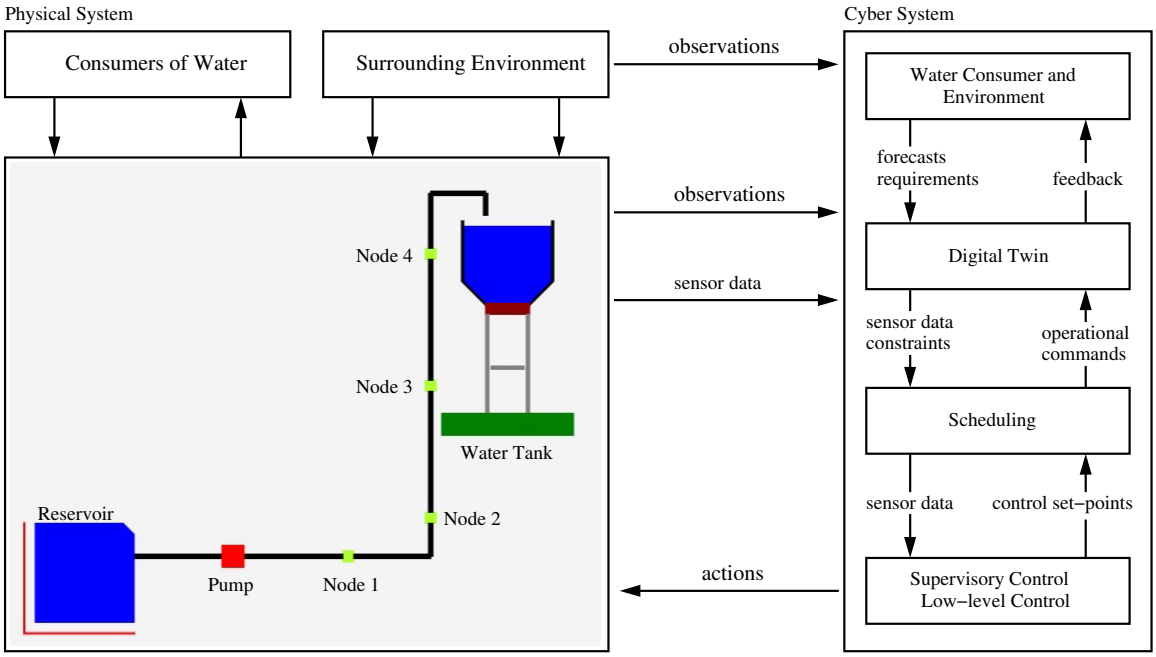

Figure 6.2: Water network management from a cyber-physical systems and digital twin perspective.

rules having premises that have evaluated to true. As a first-cut to implementation of rules for a simulation program, EPANET does what you would expect – it simply walks through the list of items on the action list and takes action. If the overall effect corresponds to forward chaining of actions, then this is purely accidental. EPANET does not have any formal support for forward and backward chaining of rules. By replacing EPANET's builtin mechanisms for rule evaluation with Jena Rules, the hope is that this shortcoming will be erased. We note that in Chapters 4 and 5, changes to state of the hydraulic network were temporally driven (e.g., the water level reaches a required level; a pump is turned off). In the behavior modeling of complex urban environments, however, notions of time and space – we need to know when and where events occur – are both critical to decision making.

Future work needs to carefully consider how analysis and decision making for wa-

ter network simulations and operations management will scale, and take advantage of modern computing and communications technology. This is particularly important for replacement of aging infrastructure with new types of systems physical networks connected to cyber components (data, information and software) for decision making. These advances have led to a multitude of new design challenges that arise from network structures that are spatial and interwoven, and dynamic behaviors and control that are distributed and concurrent. One way of helping to keep these complexities manageable is to structure system models in such a way that physical and cyber representation are treated as first class citizens, and decision making procedures are organized hierarchically. The right-hand side of Figure 6.2 shows, for example, cyber representations organized into a digital twin – scheduling – low-level/supervisory control hierarchy Low-level and supervisory control strategies are concerned with the control of component-level devices, and coordination of device settings to achieve a system purpose. Scheduling activities are concerned with the timing of activities, often to minimize operational cost. A digital twin is a cyber (or digital) representation of a system that mirrors its implementation in the physical world through real-time monitoring and synchronization of data associated with events. The associated algorithms work to provide superior levels of performance and devise strategies for avoiding unnecessary down time. The use of digital twin ideas and technologies for water network management is underway, but still in its infancy [11].

Chapter A: Small Water Network System

This appendix contains an abbreviated descriptions of the small water network system as modeled in: (1) The System Data Model, and (2) EPANET.

A.1 System Data Model Representation (WaterNetwork.xml)

```
<?xml version="1.0" encoding = "UTF-8"?>
<SystemDataModel author = "Zebo Peng" date = "2019-09"  source = "UMD">

  <!-- ======================================== -->
  <!-- EPANET modeling parameters ...           -->
  <!-- ======================================== -->

  <attribute key = "Units"       value = "GPM" />
  <attribute key = "Headloss"    value = "H-W" />

  <!-- ======================================================= -->
  <!-- Time-based attributes ...                               -->
  <!-- ======================================================= -->

  <attribute key = "Duration" value = "24:00" />
  <attribute key = "Hstep"    value = "1:00" />
  <attribute key = "Tstart"   value = "12 am" />
  <attribute key = "Pstep"    value = "1:00" />
  <attribute key = "Pstart"   value = "0:00" />
  <attribute key = "Rstep"    value = "1:00" />

  <!-- ======================================== -->
  <!-- Water network coordinates ...            -->
  <!-- ======================================== -->

  <node ID = "001" x = "100.0" y = "70.0">
     <attribute key = "type"      value = "Point"/>
     <attribute key = "elevation" value = "600" units ="ft" />
     <attribute key = "source"    value = "0.0"   />
```

```
   <attribute key = "demand"    value = "0.0"   />
</node>

<node ID = "002" x = "180.0" y = "70.0">
   <attribute key = "type"      value = "Junction"/>
   <attribute key = "elevation" value = "600" units ="ft" />
   <attribute key = "source"    value = "0.0"   />
   <attribute key = "demand"    value = "0.0"   />
   <attribute key = "pattern"   value = "P001"   />
</node>

<node ID = "003" x = "200.0" y = "70.0">
   <attribute key = "type"      value = "Junction"/>
   <attribute key = "elevation" value = "600" units ="ft" />
   <attribute key = "source"    value = "0.0"   />
   <attribute key = "demand"    value = "0.0"   />
   <attribute key = "pattern"   value = "P001"   />
</node>

<!-- ================================ -->
<!-- Junction 1 ...                   -->
<!-- ================================ -->

<node ID = "004" x = "295.0" y = "70.0">
   <description text="Network Junction 1" />

   <attribute key = "type"      value = "Junction"/>
   <attribute key = "elevation" value = "600" units ="ft" />
   <attribute key = "source"    value = "0.0"   />
   <attribute key = "demand"    value = "30.0"   />
   <attribute key = "pattern"   value = "P001"   />

   <shape type = "Rectangle">
      <attribute key =   "level" value = "48.0"/>
      <attribute key =   "width" value = "10.0"/>
      <attribute key =  "height" value = "10.0"/>
      <attribute key = "opacity" value = "1.0"/>
      <attribute key =   "color" value = "maroon"/>
   </shape>
</node>

<!-- ================================ -->
<!-- Junction 2 ...                   -->
<!-- ================================ -->

<node ID = "005" x = "370.0" y =  "70.0">
   <attribute key = "type"      value = "Point"/>
   <attribute key = "elevation" value = "600" units ="ft" />
</node>

<node ID = "006" x = "370.0" y = "105.0">
   <description text="Network Junction 2" />

   <attribute key = "type"      value = "Junction" />
```

```
   <attribute key = "elevation" value = "600" units ="ft" />
   <attribute key = "source"    value = "0.0"   />
   <attribute key = "demand"    value = "30.0"   />
   <attribute key = "pattern"   value = "P001"   />

   <shape type = "Rectangle">
      <attribute key =   "level" value = "48.0"/>
      <attribute key =   "width" value = "10.0"/>
      <attribute key =  "height" value = "10.0"/>
      <attribute key = "opacity" value = "1.0"/>
      <attribute key =   "color" value = "maroon"/>
   </shape>
</node>

<!-- ================================ -->
<!-- Junction 3 ...                   -->
<!-- ================================ -->

<node ID = "007" x = "370.0" y = "225.0">
   <description text="Network Junction 3" />

   <attribute key = "type"      value = "Junction" />
   <attribute key = "elevation" value = "650" units ="ft" />
   <attribute key = "source"    value = "0.0"   />
   <attribute key = "demand"    value = "30.0"   />
   <attribute key = "pattern"   value = "P001"   />

   <shape type = "Rectangle">
      <attribute key =   "level" value = "48.0"/>
      <attribute key =   "width" value = "10.0"/>
      <attribute key =  "height" value = "10.0"/>
      <attribute key = "opacity" value = "1.0"/>
      <attribute key =   "color" value = "maroon"/>
   </shape>
</node>

<!-- ================================ -->
<!-- Junction 4 ...                   -->
<!-- ================================ -->

<node ID = "008" x = "370.0" y = "345.0">
   <description text="Network Junction 4" />

   <attribute key = "type"      value = "Junction" />
   <attribute key = "elevation" value = "660" units ="ft" />
   <attribute key = "source"    value = "0.0" />
   <attribute key = "demand"    value = "30.0" />
   <attribute key = "pattern"   value = "P001" />

   <shape type = "Rectangle">
      <attribute key =   "level" value = "48.0"/>
      <attribute key =   "width" value = "10.0"/>
      <attribute key =  "height" value = "10.0"/>
      <attribute key = "opacity" value =  "1.0"/>
```

```
      <attribute key =   "color" value = "maroon"/>
   </shape>
</node>

<!-- ================================ -->
<!-- Nodes: Junction 4 to Tank ...    -->
<!-- ================================ -->

<node ID = "009" x = "370.0" y = "400.0">
   <attribute key = "type"      value = "Point"/>
   <attribute key = "elevation" value = "660" units ="ft" />
</node>

<node ID = "010" x = "420.0" y = "400.0">
   <attribute key = "type"      value = "Point"/>
   <attribute key = "elevation" value = "660" units ="ft" />
</node>

<node ID = "011" x = "420.0" y = "380.0">
   <attribute key = "type"      value = "Point"/>
   <attribute key = "elevation" value = "660" units ="ft" />
</node>

<!-- ======================================== -->
<!-- Control network coordinates              -->
<!-- ======================================== -->

<node ID = "012" x = "575.0" y =  "50.0" type="Point" />
<node ID = "013" x = "575.0" y = "300.0" type="Point" />
<node ID = "014" x = "460.0" y = "300.0" type="Point" />

<node ID = "015" x = "190.0" y =  "65.0" type="Point" />
<node ID = "016" x = "190.0" y =  "25.0" type="Point" />
<node ID = "017" x = "500.0" y =  "25.0" type="Point" />

<!-- ======================================== -->
<!-- Water network ways ...                   -->
<!-- ======================================== -->

<way ID="001">
   <description text="Reservoir to Pump." />
   <attribute key = "type" value = "Pipeline"/>

   <!-- Connectivity of pipe ends -->

   <attribute key = "end1" value = "C01" />
   <attribute key = "end2" value = "002" />

   <!-- Sequence of nodal coordinates -->

   <node ID="001" />
   <node ID="002" />

   <!-- Engineering parameters -->
```

```
   <attribute key = "diameter"  value =    "4" />
   <attribute key = "length"    value =   "80" />
   <attribute key = "roughness" value =  "100" />
   <attribute key = "minorloss" value =    "0" />
   <attribute key = "status"    value = "Open" />

   <shape type = "LineString">
        <attribute key = "level" value = "50.0"/>
        <attribute key = "width" value = "6.0"/>
        <attribute key = "color" value = "black"/>
   </shape>
</way>

<way ID="002">
   <description text="Connect Water Pump to Junction 1." />
   <attribute key = "type" value = "Pipeline"/>

   <!-- Connectivity of pipe ends -->

   <attribute key = "end1" value = "003" />
   <attribute key = "end2" value = "004" />

   <!-- Sequence of nodal coordinates -->

   <node ID="003" />
   <node ID="004" />

   <!-- Engineering parameters -->

   <attribute key = "diameter"  value =    "4" />
   <attribute key = "length"    value =   "95" />
   <attribute key = "roughness" value =  "100" />
   <attribute key = "minorloss" value =    "0" />
   <attribute key = "status"    value = "Open" />

   ... details of shape removed ...

</way>

<way ID="003">
   <description text="Connect Junction 1 to Junction 2." />
   <attribute key = "type" value = "Pipeline"/>

   <!-- Connectivity of pipe ends -->

   <attribute key = "end1" value = "004" />
   <attribute key = "end2" value = "006" />

   <!-- Sequence of nodal coordinates -->

   <node ID="004" />
   <node ID="005" />
   <node ID="006" />
```

```
   <!-- Engineering parameters -->

   <attribute key = "diameter"  value =    "4" />
   <attribute key = "roughness" value =  "100" />
   <attribute key = "minorloss" value =    "0" />
   <attribute key = "status"    value = "Open" />

   ... details of shape removed ...

</way>

<way ID="004">
   <description text="Connect Junction 2 to Junction 3." />
   <attribute key = "type" value = "Pipeline"/>

   <!-- Connectivity of pipe ends -->

   <attribute key = "end1" value = "006" />
   <attribute key = "end2" value = "007" />

   <!-- Sequence of nodal coordinates -->

   <node ID="006" />
   <node ID="007" />

   ... details of parameters and shape removed ...

</way>

<way ID="005">
   <description text="Connect Junction 3 to Junction 4." />
   <attribute key = "type" value = "Pipeline"/>

   <!-- Connectivity of pipe ends -->

   <attribute key = "end1" value = "007" />
   <attribute key = "end2" value = "008" />

   <!-- Sequence of nodal coordinates -->

   <node ID="007" />
   <node ID="008" />

   ... details of parameters and shape removed ...

</way>

<way ID="006">
   <description text="Connect Junction 4 to Water Tank." />
   <attribute key = "type" value = "Pipeline"/>

   <!-- Connectivity of pipe ends -->
```

```
   <attribute key = "end1" value = "008" />
   <attribute key = "end2" value = "C03" />

   <!-- Sequence of nodal coordinates -->

   <node ID="008" />
   <node ID="009" />
   <node ID="010" />
   <node ID="011" />

   ... details of parameters and shape removed ...

</way>

<!-- ======================================== -->
<!-- Control network ways ...                 -->
<!-- ======================================== -->

<way ID="007">
   <attribute key = "type" value = "Control"/>

   <!-- Sequence of nodal coordinates -->

   <node ID="012" />
   <node ID="013" />
   <node ID="014" />

   ... details of shape removed ...

</way>

<way ID="008">
   <attribute key = "type" value = "Control"/>

   <!-- Sequence of nodal coordinates -->

   <node ID="015" />
   <node ID="016" />
   <node ID="017" />

   ... details of shape removed ...

</way>

<!-- ========================================== -->
<!-- Water Reservoir Component ...              -->
<!-- ========================================== -->

<component ID = "C01" x = "0.0" y = "0.0">
   <description text="Water Reservoir" />

   <!-- Engineering parameters -->

   <attribute key = "type"  value = "Reservoir" />
```

```
   <attribute key = "elevation" value = "600.0" units ="ft" />
   <attribute key = "head"  value = "600.0" />
   <attribute key = "area"  value = "0.0" />

   <!-- Look-and-feel of component shape -->

   <compoundshape ID = "Reservoir-Shape01">
      <shape ID = "Reservoir" type = "Polygon">
          <attribute key =   "level" value = "48.0"/>
          <attribute key = "opacity" value = "1.0"/>
          <attribute key =   "color" value = "blue"/>
          <attribute key =    "fill" value = "true"/>
          <node ID="r01" x =  "10.0" y =  "10.0" type="Point" />
          <node ID="r02" x = "100.0" y =  "10.0" type="Point" />
          <node ID="r03" x = "100.0" y = " 90.0" type="Point" />
          <node ID="r04" x = " 90.0" y = "100.0" type="Point" />
          <node ID="r05" x =  "10.0" y = "100.0" type="Point" />
      </shape>
   </compoundshape>
</component>

<!-- ============================================ -->
<!-- Network Controller Component ...             -->
<!-- ============================================ -->

<component ID = "C02" x = "500.0" y = "0.0">
   <description text="Network Controller" />

   <attribute key = "type" value = "Control" />

   ... details of shape removed ...

</component>

<!-- ============================================ -->
<!-- Water Tank Component ...                     -->
<!-- ============================================ -->

<component ID = "C03" x = "400.0" y = "200.0">
   <description text="Elevated Water Tank" />

   <!-- Engineering parameters -->

   <attribute key = "type"      value = "Tank" />
   <attribute key = "elevation" value = "700.0" units ="ft" />
   <attribute key = "area"      value = "400.0" />
   <attribute key = "minlevel"  value =   "0.0" />
   <attribute key = "initlevel" value =   "2.0" />
   <attribute key = "maxlevel"  value =  "20.0" />
   <attribute key = "flowin"    value =   "0.0" />
   <attribute key = "flowout"   value =   "0.0" />

   <!-- Look-and-feel of component shape -->
```

```
   <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" />
          <node ID="n06" x = "80.0" y = "170.0" type="Point" />
      </shape>

      ... details of shape removed ...

   </compoundshape>
</component>

<!-- ============================================================ -->
<!-- Load pattern for time-intervals of consumer demand (24 hrs) ... -->
<!-- ============================================================ -->

<behavior ID = "P001" type = "Pattern">
    <description text = "Consumer demand load pattern (24 hrs)" />
    <attribute key = "multipliers"  value = "0.20  0.20  0.20  0.30  0.40  0.50
                                             0.70  0.80  1.00  1.00  1.00  1.00
                                             1.00  1.00  1.00  1.00  1.00  1.00
                                             1.00  1.00  0.80  0.70  0.50  0.20"/>
</behavior>

<!-- ===================================================== -->
<!-- Load pattern for pump operations (that repeats) ...   -->
<!-- ===================================================== -->

<behavior ID = "P002" type = "Pattern">
    <description text = "Pattern for pump operations" />
    <attribute key = "multipliers"  value = "1.0 1.0 1.0 1.0" />
</behavior>

<!-- ===================================================== -->
<!-- Typical water pump performance curves ...             -->
<!-- ===================================================== -->

<behavior ID = "B001">
    <description text = "Head-flow performance curve for Pump 1" />
    <attribute key = "type" value = "Curve" />
    <attribute key =   "x" value = "  0.0 2000.0 4000.0" />
    <attribute key =   "y" value = "150.0  100.0   60.0" />
</behavior>

<!-- ===================================================== -->
<!-- Water pump: simplified statechart model ...           -->
<!-- ===================================================== -->
```

```
<behavior ID="B002" type="FSM">
   <description text="Simple model of pump behavior ..." />

   ... details of finite state machine behavior removed ...

</behavior>

<!-- ================================================== -->
<!-- Water pump component model ...                     -->
<!-- ================================================== -->

<component ID = "C04" x = "190.0" y = "70.0">
   <description text="Water Pump" />

   <!-- Connectivity of pump to nodes -->

   <attribute key = "end1" value = "002" />
   <attribute key = "end2" value = "003" />

   <!-- Component-level attributes -->

   <attribute key = "type"   value = "Pump" />
   <attribute key = "height" value = "20.0" />
   <attribute key = "width"  value = "20.0" />

   <!-- Pattern for pump operations -->

   <attribute key = "pattern" value = "P002"   />

   <!-- Pump attributes            -->

   <attribute key = "minhead" value =    "0.0" />
   <attribute key = "maxhead" value =  "100.0" />
   <attribute key =   "power" value = "5000.0" />

   <!-- Pump head-flow performance curve .... -->

   <attribute key = "hcurve" value = "B001" />

   <!-- Statechart behavior model .... -->

   <behavior id="#B002"/>

   ... details of input and output ports removed ...

   <compoundshape ID = "Shape01">
      <shape ID = "sh01" x = "0.0" y = "0.0" type = "Rectangle">
          <attribute key =   "level" value = "50.0"/>
          <attribute key =   "width" value = "20.0"/>
          <attribute key =  "height" value = "20.0"/>
          <attribute key = "opacity" value = "1.0"/>
          <attribute key =    "fill" value = "true"/>
          <attribute key =   "color" value = "blue"/>
      </shape>
```

```
    </compoundshape>
  </component>

  <!-- ========================================================== -->
  <!-- Water network rules: Pump water when tank water level is low -->
  <!-- ========================================================== -->

  <constraint ID = "R01" type="Rule">
     <description text="Open pump (turn on) when tank water level is low" />
     <attribute key = "expression" value = "IF TANK C03 LEVEL BELOW 3.0
                                            THEN PUMP C04 STATUS IS OPEN"/>
  </constraint>

  <constraint ID = "R02" type="Rule">
     <description text="Close pump (turn off) when tank water level is high" />
     <attribute key = "expression" value = "IF TANK C03 LEVEL ABOVE 17.0
                                            THEN PUMP C04 STATUS IS CLOSED"/>
  </constraint>

</SystemDataModel>
```

A.2 Water Network System Model in EPANET

```
Water Network ...
================================================ ...
--- No junctions = 6 ...
--- No nodes     = 8 ...
--- No links     = 7 ...
--- No patterns  = 3 ...
--- No curves    = 1 ...
--- No controls  = 0 ...
--- No rules  = 2 ...
--- No tanks  = 2 ...
--- No pumps  = 1 ...

Network Nodes ...
======================================== ...

Node(ID = 002, type = JUNCTION) ...
================================================= ...
--- (x,y)     = ( 180.000 ft,   70.000 ft) ...
--- elevation = ( 600.000 ft) ...
---
--- Ke (emitter coefficient) =  0.000 ...
--- Initial demand      = 0.00 ...
```

```
--- demand pattern = P001 ...
--- base demand    = 0.000000 GPM ...
--- source = null ...
================================================== ...

Node(ID = 003, type = JUNCTION) ...
================================================== ...
--- (x,y)     = ( 200.000 ft,   70.000 ft) ...
--- elevation = ( 600.000 ft) ...
---
--- Ke (emitter coefficient) =  0.000 ...
--- Initial demand      = 0.00 ...
--- demand pattern = P001 ...
--- base demand    = 0.000000 GPM ...
--- source = null ...
================================================== ...

Node(ID = 004, type = JUNCTION) ...
================================================== ...
Description: Network Junction 1 ...
-------------------------------------------------- ...
--- (x,y)     = ( 295.000 ft,   70.000 ft) ...
--- elevation = ( 600.000 ft) ...
---
--- Ke (emitter coefficient) =  0.000 ...
--- Initial demand      = 0.00 ...
--- demand pattern = P001 ...
--- base demand    = 30.000000 GPM ...
--- source = null ...
================================================== ...

Node(ID = 006, type = JUNCTION) ...
================================================== ...
Description: Network Junction 2 ...
-------------------------------------------------- ...

     ... details removed ...

Node(ID = 007, type = JUNCTION) ...
================================================== ...
Description: Network Junction 3 ...
-------------------------------------------------- ...

     ... details removed ...

================================================== ...

Node(ID = 008, type = JUNCTION) ...
================================================== ...
Description: Network Junction 4 ...
-------------------------------------------------- ...
--- (x,y)     = ( 370.000 ft,  345.000 ft) ...
--- elevation = ( 660.000 ft) ...
---
```

```
--- Ke (emitter coefficient) =  0.000 ...
--- Initial demand      = 0.00 ...
--- demand pattern = P001 ...
--- base demand    = 30.000000 GPM ...
--- source = null ...
================================================== ...

Node(ID = C01, type = TANK) ...
================================================== ...
Description: Water Reservoir ...
-------------------------------------------------- ...
--- (x,y)     = (   0.000 ft,    0.000 ft) ...
--- elevation   = ( 600.000 ft) ...
--- initial water elevation =  600.000 ft ...
--- minimum water elevation =  600.000 ft ...
--- maximum water elevation =  600.000 ft ...
--- tank area   = (     0.0 ft^2) ...
--- tank volume = (     0.0 ft^3) ...
---
--- water elevation =  600.000 ft ...
================================================== ...

Node(ID = C03, type = TANK) ...
================================================== ...
Description: Elevated Water Tank ...
-------------------------------------------------- ...
--- (x,y)     = ( 400.000 ft,  200.000 ft) ...
--- elevation   = ( 700.000 ft) ...
--- initial water elevation =  702.000 ft ...
--- minimum water elevation =  700.000 ft ...
--- maximum water elevation =  720.000 ft ...
--- tank area   = (   400.0 ft^2) ...
--- tank volume = (  8000.0 ft^3) ...
---
--- initial water volume    =  800.000 ft^3 ...
================================================== ...

Network Links ...
========================================== ...

Link(ID = 001, type = PIPE) ...
================================================== ...
Description: Reservoir to Pump. ...
-------------------------------------------- ...
--- link type   = PIPE ...
--- status      = OPEN ...
--- diameter    = 4.000000 in ...
--- length      = 80.000000 ft ...
--- roughness   = 100.000000 ...
--- resistance  = 15.766425 ...
--- no vertices (for rendering) = 2 ...
--- first  = Node(C01): elevation =  600.0 ft ...
--- second = Node(002): elevation =  600.0 ft ...
================================================== ...
```

```
Link(ID = 002, type = PIPE) ...
================================================= ...
Description: Connect Water Pump to Junction 1. ...
------------------------------------------------- ...
--- link type   = PIPE ...
--- status      = OPEN ...
--- diameter    = 4.000000 in ...
--- length      = 95.000000 ft ...
--- roughness   = 100.000000 ...
--- resistance  = 18.722629 ...
--- no vertices (for rendering) = 2 ...
--- first  = Node(003): elevation =  600.0 ft ...
--- second = Node(004): elevation =  600.0 ft ...
================================================= ...

Link(ID = 003, type = PIPE) ...
================================================= ...
Description: Connect Junction 1 to Junction 2. ...
------------------------------------------------- ...

     ... details removed ...

================================================= ...

Link(ID = 004, type = PIPE) ...
================================================= ...
Description: Connect Junction 2 to Junction 3. ...
------------------------------------------------- ...

     ... details removed ...

================================================= ...

Link(ID = 005, type = PIPE) ...
================================================= ...
Description: Connect Junction 3 to Junction 4. ...
------------------------------------------------- ...

     ... details removed ...

================================================= ...

Link(ID = 006, type = PIPE) ...
================================================= ...
Description: Connect Junction 4 to Water Tank. ...
------------------------------------------------- ...
--- link type   = PIPE ...
--- status      = OPEN ...
--- diameter    = 4.000000 in ...
--- length      = 125.000000 ft ...
--- roughness   = 100.000000 ...
--- resistance  = 24.635039 ...
--- no vertices (for rendering) = 4 ...
```

```
--- first  = Node(008): elevation =  660.0 ft ...
--- second = Node(C03): elevation =  700.0 ft ...
================================================== ...

Link(ID = C04, type = PUMP) ...
================================================== ...
--- status = OPEN ...
--- first  = Node(002): elevation =  600.0 ft ...
--- second = Node(003): elevation =  600.0 ft ...
---
--- pump curve    = CUSTOM ...
--- flow exponent =  0.00000 ...
--- shutoff head  =     0.00 ft ...
--- maximum head  =   150.00 ft ...
--- initial flow  =     4.46 GPM ...
--- maximum flow  =     8.91 GPM ...
---
--- head-flow curve: B001 ...
--- efficiency-flow curve = null ...
--- energy-cost pattern = null ...
================================================== ...

Tanks ...
========================================== ...

Node(ID = C01, type = TANK) ...
================================================== ...
Description: Water Reservoir ...
-------------------------------------------------- ...
--- (x,y)     = (   0.000 ft,    0.000 ft) ...
--- elevation   = ( 600.000 ft) ...
--- initial water elevation =  600.000 ft ...
--- minimum water elevation =  600.000 ft ...
--- maximum water elevation =  600.000 ft ...
--- tank area   = (     0.0 ft^2) ...
--- tank volume = (     0.0 ft^3) ...
---
--- water elevation =  600.000 ft ...
================================================== ...

Node(ID = C03, type = TANK) ...
================================================== ...
Description: Elevated Water Tank ...
-------------------------------------------------- ...
--- (x,y)     = ( 400.000 ft,  200.000 ft) ...
--- elevation   = ( 700.000 ft) ...
--- initial water elevation =  702.000 ft ...
--- minimum water elevation =  700.000 ft ...
--- maximum water elevation =  720.000 ft ...
--- tank area   = (   400.0 ft^2) ...
--- tank volume = (  8000.0 ft^3) ...
---
--- initial water volume    =  800.000 ft^3 ...
================================================== ...
```

```
Pumps ...
======================================== ...

Link(ID = C04, type = PUMP) ...
================================================== ...
--- status = OPEN ...
--- first  = Node(002): elevation =  600.0 ft ...
--- second = Node(003): elevation =  600.0 ft ...
---
--- pump curve    = CUSTOM ...
--- flow exponent =  0.00000 ...
--- shutoff head  =     0.00 ft ...
--- maximum head  =   150.00 ft ...
--- initial flow  =     4.46 GPM ...
--- maximum flow  =     8.91 GPM ...
---
--- head-flow curve: B001 ...
--- efficiency-flow curve = null ...
--- energy-cost pattern = null ...
================================================== ...

Curves ...
======================================== ...

Curve(ID = B001, type = H_CURVE) ...
================================================ ...
--- (x,y) = (    0.00,   150.00) ...
--- (x,y) = ( 2000.00,   100.00) ...
--- (x,y) = ( 4000.00,    60.00) ...
================================================ ...

Patterns ...
======================================== ...

Temporal Pattern(Default) ...
================================================ ...
Time  1 hr, value =     1.00 ...
================================================ ...

Temporal Pattern(P001) ...
================================================ ...
Time  1 hr, value =     0.20 ...
Time  2 hr, value =     0.20 ...

     ... details removed ...

Time 22 hr, value =     0.70 ...
Time 23 hr, value =     0.50 ...
Time 24 hr, value =     0.20 ...
================================================ ...

Temporal Pattern(P002) ...
================================================ ...
```

```
Time  1 hr, value =     1.00 ...
Time  2 hr, value =     1.00 ...
Time  3 hr, value =     1.00 ...
Time  4 hr, value =     1.00 ...
================================================ ...

Controls ...
========================================== ...

Rules ...
========================================== ...

Rule(R01) ...
================================================== ...
Description: Open pump (turn on) when tank water level is low ...
-------------------------------------------------- ...
Expression: IF TANK C03 LEVEL BELOW 3.0
            THEN PUMP C04 STATUS IS OPEN
================================================== ...

Rule(R02) ...
================================================== ...
Description: Close pump (turn off) when tank water level is high ...
-------------------------------------------------- ...
Expression: IF TANK C03 LEVEL ABOVE 17.0
            THEN PUMP C04 STATUS IS CLOSED
================================================== ...
```

Chapter B: EPANET Ontologies and Rules

B.1 EPANET Water Network Ontology (umd-epanet.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#" >
]>

<rdf:RDF xmlns="http://www.ontologies.org/epanet#"
     xml:base="http://www.ontologies.org/epanet"
     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:waterNetwork_ontology="http://www.ontologies.org/epanet#">
    <owl:Ontology rdf:about="http://www.ontologies.org/eepanet"/>

    <!--
    ////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://www.ontologies.org/epanet#WaterNetwork -->

    <owl:Class rdf:about="http://www.ontologies.org/epanet#WaterNetwork">
    </owl:Class>
```

```
<!-- http://www.ontologies.org/epanet#Component -->

<owl:Class rdf:about="http://www.ontologies.org/epanet#Component">
</owl:Class>

<!-- http://www.ontologies.org/epanet#Pump -->

<owl:Class rdf:about="http://www.ontologies.org/epanet#Pump">
   <rdfs:subClassOf rdf:resource="http://www.ontologies.org/epanet#Component"/>
</owl:Class>

<!-- http://www.ontologies.org/epanet#Tank -->

<owl:Class rdf:about="http://www.ontologies.org/epanet#Tank">
   <rdfs:subClassOf rdf:resource="http://www.ontologies.org/epanet#Component"/>
</owl:Class>

<!-- http://www.ontologies.org/epanet#Reservoir -->

<owl:Class rdf:about="http://www.ontologies.org/epanet#Reservoir">
   <rdfs:subClassOf rdf:resource="http://www.ontologies.org/epanet#Component"/>
</owl:Class>

<!-- http://www.ontologies.org/epanet#Pipe -->

<owl:Class rdf:about="http://www.ontologies.org/epanet#Pipe">
   <rdfs:subClassOf rdf:resource="http://www.ontologies.org/epanet#Component"/>
</owl:Class>

<!-- http://www.ontologies.org/epanet#Node -->

<owl:Class rdf:about="http://www.ontologies.org/epanet#Node">
   <rdfs:subClassOf rdf:resource="http://www.ontologies.org/epanet#Component"/>
</owl:Class>

<!--
///////////////////////////////////////////////////////////////////////////////
//
// Object Properties for class WaterNetwork
//
///////////////////////////////////////////////////////////////////////////////
 -->

<!-- http://www.ontologies.org/epanet#hasComponent -->

<owl:ObjectProperty rdf:about="http://www.ontologies.org/epanet#hasComponent">
   <rdfs:domain rdf:resource="http://www.ontologies.org/epanet#WaterNetwork"/>
   <rdfs:range rdf:resource="http://www.ontologies.org/epanet#Component"/>
</owl:ObjectProperty>

<!--
///////////////////////////////////////////////////////////////////////////////
//
```

```
    // Data Properties for class Component
    //
    ///////////////////////////////////////////////////////////////////////////
     -->

   <!-- http://www.ontologies.org/epanet#hasStatus -->

    <owl:DatatypeProperty rdf:about="http://www.ontologies.org/epanet#hasStatus">
       <rdfs:domain rdf:resource="http://www.ontologies.org/epanet#Component"/>
       <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>

    <!--
    ///////////////////////////////////////////////////////////////////////////
    //
    // Data Properties for class Tank
    //
    ///////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://www.ontologies.org/epanet#hasCurrentLevel -->

    <owl:DatatypeProperty rdf:about="http://www.ontologies.org/epanet#hasCurrentLevel">
       <rdfs:domain rdf:resource="http://www.ontologies.org/epanet#Tank"/>
       <rdfs:range rdf:resource="&xsd;double"/>
    </owl:DatatypeProperty>

    <!-- http://www.ontologies.org/epanet#hasMaxLevel -->

    <owl:DatatypeProperty rdf:about="http://www.ontologies.org/epanet#hasMaxLevel">
       <rdfs:domain rdf:resource="http://www.ontologies.org/epanet#Tank"/>
       <rdfs:range rdf:resource="&xsd;double"/>
    </owl:DatatypeProperty>

    <!-- http://www.ontologies.org/epanet#hasMinLevel -->

    <owl:DatatypeProperty rdf:about="http://www.ontologies.org/epanet#hasMinLevel">
       <rdfs:domain rdf:resource="http://www.ontologies.org/epanet#Tank"/>
       <rdfs:range rdf:resource="&xsd;double"/>
    </owl:DatatypeProperty>

</rdf:RDF>
```

B.2 EPANET Water Network Jena Rules (umd-epanet.rules)

```
@prefix epa:  <http://www.ontologies.org/epanet#>.
```

```
@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: Update the pump's status based on the tank's status ...

[ UpdateStatus: (?p rdf:type epa:Pump) (?t rdf:type epa:Tank)
                (?e rdf:type epa:ElementsStatus) (?e epa:hasTankFStatus ?tf)
                (?t epa:hasStatus ?ts) (?p epa:hasStatus ?ps) equal(?ts, ?tf) ->
                (?p epa:hasStatus "Closed")]

// Rule 03: Update the Pump's status based on the tank's currentLevel ...

[ tank01: (?p rdf:type epa:Pump) (?t rdf:type epa:Tank) (?t epa:hasCurrentLevel ?c)
          (?t epa:hasMaxLevel ?ma) (?p epa:hasStatus ?ps) greaterThan(?c, ?ma) ->
          (?p epa:hasStatus "Closed")]

[ tank02: (?p rdf:type epa:Pump) (?t rdf:type epa:Tank) (?t epa:hasCurrentLevel ?c)
          (?t epa:hasMinLevel ?mi) (?p epa:hasStatus ?ps) lessThan(?c, ?mi) ->
          (?p epa:hasStatus "Open") ]
```

Bibliography

[1] Alexander C. *A Pattern Language: Towns, Buildings and Construction*. Oxford Press, 1977.

[2] Apache Jena:. An Open Source Java Framework for building Semantic Web and Linked Data Applications. For details, see https://jena.apache.org/ (Accessed, May 10, 2020).

[3] Austin M.A., Baras J.S., and Kositsyna N.I. Combined Research and Curriculum Development in Information-Centric Systems Engineering. In *Twelth Annual International Symposium of The International Council on Systems Engineering (INCOSE 2002)*, Las Vegas 2002.

[4] Austin M.A., Delgoshaei P. and Nguyen, A. Distributed System Behavior Modeling with Ontologies, Rules, and Message Passing Mechanisms, Conference on Systems Engineering Research (CSER 2015), Hoboken, NJ, USA. *Procedia Computer Science*, 44:373 – 382, 2015.

[5] Austin, M.A., Mayank, V. and Shmunis, N. Ontology-enabled Validation of Connectivity Relationships in a Home Theater System. *International Journal of Intelligent Systems*, 21(10):1111–1125, October 2006.

[6] Avila-Melgar E.Y., Cruz-Chavez M.A. and Martinez-Bahena B. General Methodology for using EPANET as an Optimization Element in Evolutionary Algorithms in a Grid Computing Environment for Water Distribution Network Design. *Water Science and Technology: Water Supply*, 17(1), 2017.

[7] Berners-Lee T., Hendler J., and Lassila O. The Semantic Web. *Scientific American*, pages 35–43, May 2001.

[8] 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)*, pages 10–15, April 23-27 2017.

[9] 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):365–382, December 2017.

[10] 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*, pages 255–266. Proceedings of the 16th Annual Conference on Systems Engineering Research, Systems Engineering in Context, Chapter 21, Springer, 2019.

[11] Conejos Fuertes P., Alzamora F.M., Carot M.H., and Campos J.C.A. Building and Exploiting a Digital Twin for the Management of Drinking Water Distribution Networks. *Urban Water Journal*, pages 1–10, June 2020.

[12] Delgoshaei P., and Austin M.A. Software Patterns for Traceability of Requirements to Finite-State Machine Behavior: Application to Rail Transit Systems Design and Management. In *22nd Annual International Symposium of The International Council on Systems Engineering (INCOSE 2012)*, Rome, Italy, 2012.

[13] Delgoshaei P., Austin M.A., 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):223–238, December 2014.

[14] Delgoshaei P., Austin, M.A, and Veronica, D.A. A Semantic Platform Infrastructure for Requirements Traceability and System Assessment. *The Ninth International Conference on Systems (ICONS 2014)*, February 2014.

[15] Falquet G., Metral C., Teller J., and Tweed C. *Ontologies in Urban Development Projects*. Springer, 2005.

[16] Feigenbaum L., 2006. Semantic Web Technologies in the Enterprise.

[17] Fridenthal S., Moore A., and Steiner R. *A Practical Guide to SysML*. MK-OMG, 2008.

[18] Gajbhiye A., Hari Prasad Reddy P. and Sargaonkar A.P. Modeling Leakage in Water Distribution System Using EPANET. *Journal of Civil Engineering and Environmental Technology*, 4(3):211–214, April-June 2017.

[19] Gamma E., Helm R., Johnson R., and Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1995.

[20] Georgescu S.C., and Georgescu A.M. Pumping Station Scheduling for Water Distribution Networks in EPANET. *U.P.B.Sci. Bull., Series D*, 77(2):235–246, 2015.

[21] Geroimenko V., and Chen C. (Eds). *Visualizing the Semantic Web: XML-based Internet and Information Visualization*. Springer, 2003.

[22] Grolinger K., Capretz M.A.M., Shypanski A. and Gill G.S. Federated Critical Infrastructure Simulators: Towards Ontologies for Support of Collaboration. In *24th Canadian Conference on Electrical and Computer Engineering(CCECE)*, pages 1503–1506, Niagara Falls, ON, Canada, 2011.

[23] Hendler J. Agents and the Semantic Web. *IEEE Intelligent Systems*, pages 30–37, March/April 2001. Available on April 4, 2002 from http://www.computer.org/intelligent.

[24] Horrocks I. *Ontologies and the Semantic Web.* Communications of the ACM, 51(12):58-67, December, 2008.

[25] Horrocks I., Patel-Schneider P.F., and Van Harmelen F. *From SHIQ and RDF to OWL: The Making of a Web Ontology Language.*

[26] INCOSE Systems Engineering Vision 2025 (A World in Motion). *International Council on Systems Engineering*, 2014.

[27] Jackson D. *Dependable Software by Design. Scientific American*, 294(6), June 2006.

[28] Kernighan B.W. and Ritchie D.M. *The C Programming Language - Second Edition.* Prentice-Hall Software Series, Englewood Cliffs, NJ 07632, 1988. Based on Draft-Proposed ANSI C.

[29] Klise, K., Hart D., Moriarty D., Bynum M., and Murray R. Water Network Tool for Resilience (WNTR) User Manual. *U.S. EPA Office of Research and Development, Washington, DC, EPA/600/R-17/264*, 2017.

[30] Larock B.E., Jeppson R.W., and Watters G.Z. *Hydraulics of Pipeline Systems.* CRC Press, Boca Raton, Florida 33421, USA, 2000.

[31] Lee M.F. Pipeline Network Analysis. Technical Report Publication 77, Water Resources Research Center, Department of Environmental Engineering Sciences, University of Florida, 1983.

[32] Mahmoud Q.H. Getting Started With the Java Rule Engine API (JSR 94): Toward Rule-Based Applications. *Sun Microsystems*, 2005. For more information, see http://java.sun.com/developer/technicalArticles/J2SE/JavaRule.html (Accessed, March 10, 2008).

[33] Mount J., and Hanak E. Water Use in California, Just the Facts. *Public Policy Institute of California*, May 2019.

[34] 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.

[35] Open Street Map (OSM). https://www.openstreetmap.org (Accessed May 10, 2020). 2020.

[36] OWL:. The Web Ontology Language, See http://www.w3.org/TR/owl-features/ (Accessed, February 2017).

[37] Petnga L. and Austin M.A. *Ontologies of Time and Time-based Reasoning for MBSE of Cyber-Physical Systems.* 11th Annual Conference on Systems Engineering Research (CSER 2013), Georgia Institute of Technology, Atlanta, GA, 2013.

[38] Petnga L., and Austin M.A. Semantic Platforms for Cyber-Physical Systems. 24th Annual International Symposium of The International Council on Systems Engineering (INCOSE), Las Vegas, NV, USA, June 30 - July 03, 2014.

[39] Petnga L., and Austin M.A. An Ontological Framework for Knowledge Modeling and Decision Support in Cyber-Physical Systems. *Advanced Engineering Informatics*, 30(1):77–94, January 2016.

[40] Ramaswami, A., et al. Sustainable Urban Systems: Articulating a Long-Term Convergence Research Agenda. *Advisory Committee for Environmental Research and Education,* National Science Foundation, Washington, D.C., USA, January 2018.

[41] Rossman L.A. EPANET 2: Users Manual. Technical Report EPA/600/R-00/057, Water Supply and Water Resources Division National Risk Management Research Laboratory Cincinnati, OH 45268, September 2000.

[42] Rudolf G. Some Guidelines For Deciding Whether To Use A Rules Engine. 2003. Sandia National Labs.

[43] Salomons E., Hatchett S., and Eliades D.G. The EPANET Open Source Initiative. In *1st International WDSA/CCWI 2018 Joint Conference*, Kingston, Ontario, Canada, July 23-25 2018.

[44] Segaran T., Taylor J. and Evans C. *Programming the Semantic Web.* O'Reilly, Beijing, 2009.

[45] Tidwell D. *XSLT.* O'Reilly and Associates, Sebastopol, California, 2001.

[46] Wagner D.A, Bennett M. B., Karban R., Rouquette N., Jenkins S. and Ingham M. *An Ontology for State Analysis: Formalizing the Mapping to SysML* . IEEE Aerospace Conference, Big Sky, MT, USA, 2012.

[47] Whistle: A Java-enabled Scripting Language for Assembly and Execution of Multi-Domain Systems. See http://www.isr.umd.edu/∼austin/whistle.html (Accessed April 17, 2020).

[48] XML Stylesheet Transformation Language (XSLT). See http://www.w3.org/Style/XSL. 2002.