

# Framework for Knowledge-Based Fault Detection and Diagnostics in Multi-Domain Systems: Application to HVAC Systems

Mark Austin

The  
Institute for  
**Systems**  
Research



**A. JAMES CLARK**  
SCHOOL OF ENGINEERING

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the A. James Clark School of Engineering. It is a graduated National Science Foundation Engineering Research Center.

[www.isr.umd.edu](http://www.isr.umd.edu)



## ISR Technical Report 2017-06

# Framework for Knowledge-Based Fault Detection and Diagnostics in Multi-Domain Systems: Application to HVAC Systems

By Parastoo Delgoshaei<sup>1</sup> and Mark Austin<sup>2</sup>

**Abstract.** State-of-the-art fault detection methods are equipment and domain specific and non-comprehensive. As a result, the applicability of these methods in different domains is very limited and they can achieve significant levels of performance by having knowledge of the domain and the ability to mimic human thinking in identifying the source of a fault with a comprehensive knowledge of the system and its surroundings. This technical report presents a comprehensive semantic framework for fault detection and diagnostics (FDD) in systems simulation and control. Our proposed methodology entails of implementation of the knowledge bases for FDD purposes through the utilization of ontologies and offers improved functionalities of such system through inference-based reasoning to derive knowledge about the irregularities in the operation. We exercise the proposed approach by working step by step through the setup and solution of a fault detection and diagnostics problem for a small-scale heating, ventilating and air-conditioning (HVAC) system.

**Keywords:** Fault Detection and Diagnostics; Heating Ventilating and Air-Conditioning (HVAC); Inference-Based, Knowledge Base, Ontologies; Reasoning.

---

<sup>1</sup>Graduate Student, Ph.D. Program in Civil Systems, Department of Civil and Environmental Engineering, University of Maryland, College Park, MD 20742, USA. E-mail: p.delgoshaei@gmail.com

<sup>2</sup>Associate Professor, Department of Civil and Environmental Engineering, and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA. E-mail: austin@isr.umd.edu

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Problem Statement . . . . .	4
1.2	Objectives and Scope . . . . .	5
<b>2</b>	<b>Background</b>	<b>7</b>
2.1	Related Work . . . . .	7
2.2	The Semantic Web . . . . .	8
2.2.1	Semantic Web Technologies . . . . .	8
2.3	Semantic Model . . . . .	10
2.3.1	Ontologies . . . . .	10
2.3.2	Individuals . . . . .	10
2.3.3	Inference Rules . . . . .	11
<b>3</b>	<b>Methodology</b>	<b>12</b>
3.1	State-of-the-Art Development of Semantic Models . . . . .	12
3.1.1	Integrated Model-Centric Engineering Ontologies (IMCE) . . . . .	12
3.2	Proposed Architecture Framework . . . . .	15
3.3	Working with Jena and Jena Rules . . . . .	16
3.4	Data-Driven Approach to Generation of Individuals in Semantic Graphs . . . . .	17

<b>4</b>	<b>Meta-Domain Ontologies and Rules</b>	<b>19</b>
4.1	Spatial Ontology and Rules . . . . .	19
4.1.1	Experimental Ontology and Rules for Spatial Reasoning . . . . .	20
<b>5</b>	<b>Domain Ontologies and Rules</b>	<b>22</b>
5.1	Engineering Ontologies and Rules . . . . .	22
5.1.1	Building Ontology and Rules . . . . .	22
5.1.2	Mechanical Equipment Ontology and Rules . . . . .	24
5.1.3	Sensor Ontology and Rules . . . . .	26
5.1.4	Fault Detection and Diagnostic Ontologies, Rules, and Procedures . . . . .	29
5.2	Surrounding Environment Ontologies and Rules . . . . .	30
5.2.1	Occupant Ontology and Rules . . . . .	30
5.2.2	Weather Ontology and Rules . . . . .	33
<b>6</b>	<b>Case Study Problem</b>	<b>34</b>
6.1	Problem Description . . . . .	34
6.2	Snapshot of Semantic Graph Model Assembly . . . . .	35
6.3	Test Problem Scenario and Hypothesis Evaluation Procedure . . . . .	39
6.4	Synthesis of Multi-domain Rules . . . . .	40
6.5	Multi-domain Rule Evaluation . . . . .	42
6.5.1	Fault Detection . . . . .	42
6.5.2	Fault Diagnostics . . . . .	43
<b>7</b>	<b>Conclusions and Future Work</b>	<b>44</b>
7.1	Conclusions . . . . .	44
7.2	Future Work . . . . .	45

7.3 Acknowledgment . . . . .	45
<b>Appendices</b>	<b>50</b>
<b>A Building System Data Model</b>	<b>50</b>
A.1 System Data Model (SystemDataModel.xml) . . . . .	50

# Chapter 1

## Introduction

This technical report is concerned with the development of ontology and rule-based modeling abstractions, procedures, and prototype software for automated fault detection and diagnostic (FDD) analysis of condition-based maintenance in multi-domain systems (e.g., buildings, health monitoring, power plants and aviation systems). The report builds upon our previous work on requirements engineering [6, 5, 18], system of systems [36], and behavior modeling and analysis of engineering systems [4, 14, 15, 29, 32] with semantic web technologies.

### 1.1 Problem Statement

Automated fault detection and diagnostic (FDD) techniques provide a means of detecting unwanted conditions (i.e., “faults”) in systems by recognizing deviations in real-time or recorded data values from expected values, and then diagnosing the causes leading to the faults. Automated fault detection and diagnostic (FDD) techniques provide mechanisms for condition-based maintenance of engineered systems (e.g., buildings, health monitoring, power plants and aviation systems). Proper implementation of FDD can enable pro-active identification and remediation of faults before they become significantly deleterious to the safety, security, or efficiency of the operating system.

Within the building sector, degraded or poorly-maintained equipment currently accounts for 15 to 30 % of energy consumption in commercial buildings [21]. Approximately 50 to 67 % of air conditioners (residential and commercial) are either improperly charged or have airflow issues [37] and [22]. Faulty heating, ventilating, air conditioning, and refrigeration (HVAC&R) systems

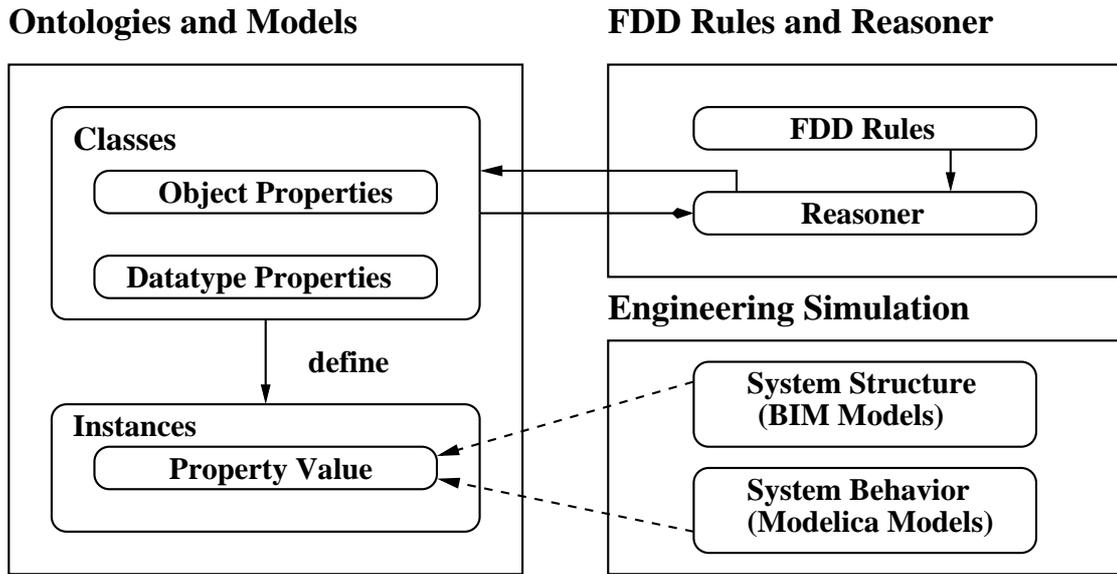


Figure 1.1: Architecture of engineering simulations connected to semantic models (ontologies and rules) reasoners for fault detection and diagnostic analysis (Adapted from Delgoshaei, Austin and Pertzborn [15]).

contribute to 1.5 to 2.5 % of total commercial building consumption [43]. Much of this energy usage could be prevented by utilizing automated condition-based maintenance. During the last decade, considerable research has focused on the development of FDD methods for HVAC&R systems. This work has been driven, in part, by the historically less-than-optimal operation of many state-of-the-art HVAC systems. Yet, in spite of recent advances in building simulation, automation and control (see the arrangement of ontologies, rules, reasoning and simulation software in Figure 1.1, automatic methods for FDD of building systems remain at a relatively immature stage of development. As a result, we require more advanced FDD techniques that leverage the untapped capabilities of building automation integrated with methods in artificial intelligence and semantic modeling. These interdisciplinary FDD systems can benefit from utilizing knowledge repositories for storing automation/simulation data and the inference-based reasoning techniques to obtain additional higher information, such as sensors location, equipment service area.

## 1.2 Objectives and Scope

This report describes a framework for knowledge-based fault detection and diagnostics in multi-domain systems, with a focus on applications to HVAC Systems. In a departure from state-

of-the-art developments in ontology engineering, which place a priority on the development and testing of ontologies alone, our objective is to create a modeling framework that supports:

1. Concurrent data-driven development of domain models, ontologies and rules, and
2. Inference-based reasoning for detection of faults and their causes.

The proposed method employs the Web Ontology Language (OWL) [31] and Jena API [2] for the development of semantic models (ontologies and rules) spanning the building, mechanical equipment, sensor, fault detection and diagnostics (FDD), occupant and weather domains. Support for spatial reasoning among entities is provided at the meta-domain level.

The remainder of this report proceeds as follows: Chapter 2 describes the work in FDD, and a brief introduction to the uses of the Semantic Web and its enabling technologies. The proposed methodology is described in Chapter 3. Chapters 4 and 5 cover: (1) the meta-domain and domain-specific ontologies and rules, respectively, and (2) a step-by-step procedure for detection and analysis of system faults. Chapter 6 presents a case study problem that involves detection of faults in a simple building – procedures for reasoning across multiple domains are presented. Finally, the conclusions of this study and a discussion of next steps is presented in Chapter 7.

## Chapter 2

# Background

### 2.1 Related Work

Recent advances in building automation technologies provide a means for sensing and collecting the data needed for software applications to automatically detect and diagnose faults in buildings. During the past few decades a variety of FDD techniques have been developed in different domains, including model-based, rule-based, knowledge-based, and simulation-based approaches. Katipamula and Brambley summarizes FDD research for HVAC systems [21]. Their work also describes different fundamental FDD methods under the two main categories of model-based and empirical (history-based) approaches. The major difference in these approaches lies in the nature of the knowledge used to formulate the diagnostics. Model-based diagnostics evaluate residuals between actual system measurements and *a priori* models (e.g., first principle models). Data-driven empirical strategies, on the other hand, do not require *a priori* models.

Model-based methods can be quantitative or qualitative. Quantitative models represent the requisite *a priori* knowledge of the system in terms of mathematical equations, typically as explicit descriptions of the physics underlying system components. Qualitative models, conversely, combine concepts such as descriptive “states” and “rules” into statements that are axiological instead of mathematical, expressing operational correctness or desirability through an axiology, a value system, appropriate to each physical application. As a result, the building system operation can be continuously classified as being either faulty or not faulty.

Rule-based strategies are one example of qualitative model-based FDD methods. Rules can

be based on first principles or they can be inferred from historical experiments, but in either case they represent expert qualitative knowledge that no purely quantitative representation could model. The first diagnostic expert systems for technical fault diagnosis were developed at MIT by Scherer and White [34]. Since then, diagnostic systems have evolved from rule-based to model-based and expert systems approaches. Semantic models offer a means for the representation of distributed and explicit knowledge and provide ways through inference-based rules to derive implicit knowledge.

Berners-Lee and co-workers [10] point to the benefits of ontology usage for knowledge representation, and utilizing high-level reasoning capabilities in the area of agent-based control solutions. Exploitation of semantics and ontologies in the area of agent-based engineering systems has become one of the hot topics recently. The main reason behind this trend is the success and promotion of Semantic Web technologies to enable languages that are both machine and human processable. Semantic Web-based applications have been developed in the areas of health care [16], biology [39, 24], and transportation [12]. In the area of fault detection and diagnostics, Batic [8] has developed an ontology-based fault detection and diagnosis systems and tested it on airport ontologies to detect the high level irregularities in the operation of airport heating/cooling plants. Also, Schumann [35] highlights the potential impacts of artificial intelligence techniques such as ontologies on tackling the challenges in obtaining a unified diagnosis framework. The benefit of this approach is that ontologies are an essential technology guaranteeing data and information interoperability in heterogeneous and content-rich environments [28] which is at heart of comprehensive fault detection and diagnostic methods.

## **2.2 The Semantic Web**

### **2.2.1 Semantic Web Technologies**

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 [10]. At that time, Berners-Lee identified two main goals for the World Wide Web:

- 1.** To make the Web a collaborative medium and,
- 2.** To make the Web understandable and automatically processable by machines.

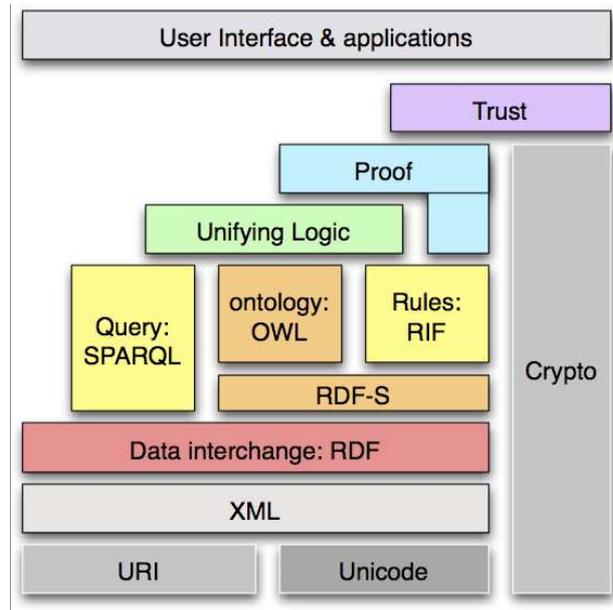


Figure 2.1: Technologies in Semantic Web Layer Cake [17].

Over the past twenty years the first part of this vision has come to pass. The development of Web browsers created a means for humans to retrieve and render information, and then manually interpret and understand the meaning of the content. A second more ambitious vision for the Web is support for semantic data structures and pathways for machine-to-machine communications that carry the semantic meaning for data in addition to its values. Thus, instead of broadly searching for someone based, perhaps based on a few keywords, semantic web provides a means to search precisely for someone based on their name, plus semantic relationships to places of employment, attendance at events, age, and so forth.

Figure 2.1 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. The extended markup language (XML) enables the construction and management of documents composed of structured portable data. The resource description framework (RDF) allows for the modeling of graphs of resources on the Web. An RDF Schema (RDF-S) provides the basic vocabulary for RDF statements, and the machinery to create hierarchies of classes and properties. Our semantic models make extensive use of the Web Ontology Language (OWL) and expressive features of descriptive logic (DL) formalisms. Inference-based mechanisms allow a system to infer a new statement from existing statements. Together, these features and language capabilities provide the foundations for representing knowledge bases

(e.g., in the building, HVAC equipment and weather domains) and reasoning over that knowledge to detect faults and systematically verify hypotheses through evaluation of supporting evidence.

## 2.3 Semantic Model

Semantic models consist of ontologies, graphs of individuals (specific instances), and inference-based rules in the form of *if <conditions> then <consequences>*. Together, these entities and mechanisms allow for the construction and execution of domain-specific knowledge bases.

### 2.3.1 Ontologies

An ontology is a formal and explicit representation of the concepts, referred to as “classes” (e.g., cooling coil, valve), and their interrelationships in a domain. The classes may have attributes that are stored as a simple data type properties” (e.g., coil setpoint). Support for semantic relationships between classes is provided by object properties (e.g., a coil has as a valve). For the representation of domains where there are many variations to be represented, but common data properties among those variants, ontology languages provide support for the organization of similar concepts into hierarchies, and support for propagation of data and object properties through hierarchies via inheritance mechanisms. We may wish to state, for example, that a cooling coil is a type of coil. In this hierarchy, the class cooling coil is a subclass of the class coil. And the class coil is superclass of the class cooling coil. The details of the classes, data properties and object properties can be summarized as follows:

- **Classes:** Valve, Cooling Coil
- **Datatype properties:** coilTemperature (double), isClosed (Boolean), coilSetpoint(double)
- **Object Property:** hasValve

### 2.3.2 Individuals

Individuals are instances of ontology concepts, and their purpose is to represent the data in a domain, e.g.,

- **Individuals:** ValveI, ValveII, Ccoil, Hcoil
- **Storing individuals:** ¡Hcoil hasValve ValveII¡

One common syntax for representing facts about a domain is the triple structure <subject, predicate, object>.

### 2.3.3 Inference Rules

Inference rules and their associated reasoning mechanisms provide a way derive new information based on the existing data stored in the ontology in the form of: if <conditions> then <consequent>. For example, the script:

Logical Rule:

```
(?coil rdf:type coil) (?coil setPoint ?sp)
(?coil coilTemperature ?cp) equal(?cp,?sp)
(?coil hasValve ?valve) -> (?valve isClosed true)
```

```
Stored individuals : <Hcoil hasValve ValveII>
                   <Ccoil coilTemperature 35>
                   <Ccoil coilSetpoint 35>
```

```
Inferred Knowledge: <ValveII isClosed true>
```

takes existing facts and rule that covers the setpoint and temperature of a coil to infer that a valve is closed.

A key benefit of semantic modeling frameworks is that the ontologies and rules are human readable, yet they can also be compiled into code that is executable on machines.

# Chapter 3

## Methodology

### 3.1 State-of-the-Art Development of Semantic Models

In state-of-the-art development of semantic models, a common strategy is to provide classes and data properties for all possible configurations within a domain, as well as linkage to related domains. Considering the maturity of present-day software engineering techniques, it is somewhat surprising that motions of “simplicity in system design” through modularity of semantic models (e.g., bundling of ontologies and rules) do not seem to exist.

#### 3.1.1 Integrated Model-Centric Engineering Ontologies (IMCE)

The integrated model-centric engineering ontologies (IMCE) developed at the JPL (Jet Propulsion Laboratory) during the 2000-2010 era [9, 41] are a representative examples of state-of-the-art practice for development of semantic models and associated software tools. For the team-based development of semantic models we are concerned with two aspects of state-of-the-art development: (1) over-specification of dependency (import) relations among the ontologies, and (2) over use of multiple inheritance relationships in the specification of new ontologies, and specifically note:

- 1. Dependency (Import) Relationships Among Key Ontologies.** Figure 3.1 shows the graph of import (dependency) relationships among the key ontologies. Loosely speaking, the IMCE ontologies are organized into two groups: foundational ontologies and discipline

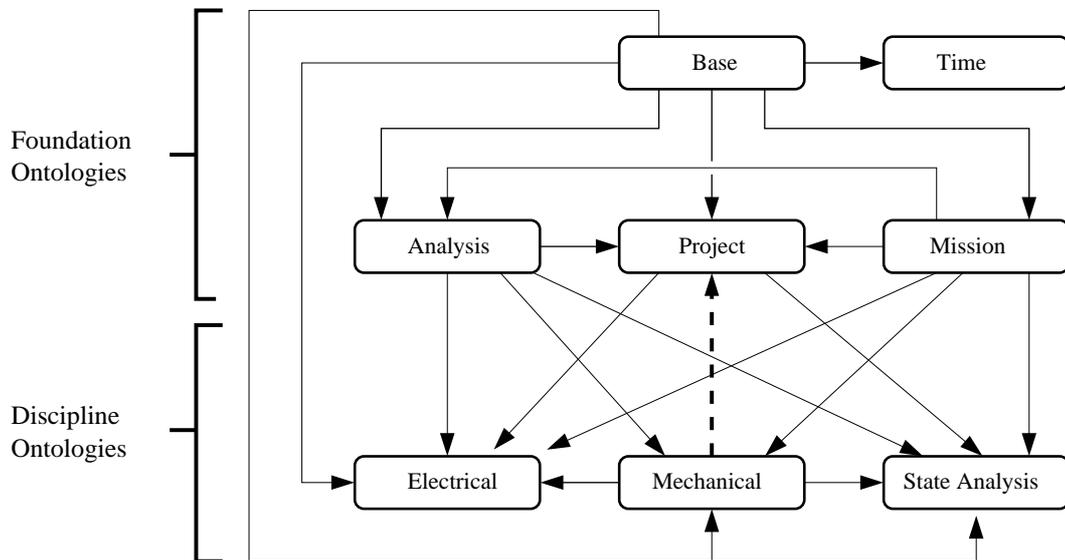


Figure 3.1: Graph of dependency relationships among ontologies in the IMCE ontologies developed at JPL.

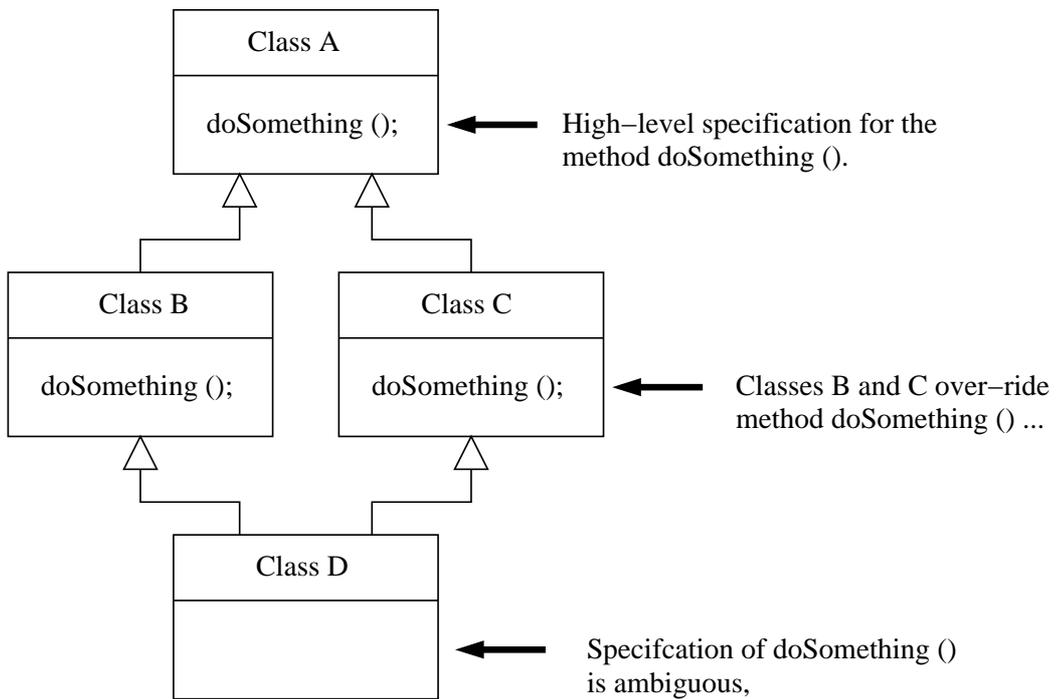


Figure 3.2: Schematic of multiple inheritance in action.

ontologies. Notice that 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 two hundred classes, with some classes containing three or four dozen data and object properties. The relationship among ontologies is more complicated than it needs to be.

- 2. Over Use of Multiple Inheritance in Ontology Specification.** Multiple inheritance in system modeling languages and some computer programming languages (e.g., C++) is a feature which allows an object or class to inherit characteristics and from more than one parent object or parent class. At the very least, the question of whether or not to provide support for multiple inheritance in system development is a controversial topic. The first problem is that multiple inheritance introduces ambiguity into system specification. The upper half of Figure 3.2 shows, for example, a scenario where class A is sub-classed by classes B and C, and the the method `doSomething()` in class A is over-ridden by customized versions of `doSomething()` in classes B and C. We have no objection to this part of the model setup. The problems begin in the lower half of Figure 3.2 where class D inherits data and methods from both classes B and C. At this point, the specification of `doSomething()` within D is ambiguous – should we use the version inherited from class B, or perhaps the version inherited from class C? One way of resolving this ambiguity is to add a custom version of `doSomething()` in class D, but then efficiency through reuse and organization of system concepts into hierarchies is lost and has zero benefit.

In a multiple inheritance ontology, like object-oriented software, classes can have more than one superclass. A second area of concern is related to ontology subsumption: that is, given an ontology O and two classes A, B, we wish to formally verify that class A is a subset of the interpretation of B in every model of O. The graph import relationships shown in Figure 3.1 shows how easy it is to assemble complex relationships among individual ontologies. This specific example does not contain loops, but given the number of unnecessary import relationships shown in Figure 3.1, it is not hard to see (intentional or not) how loops in dependencies among ontologies could be created. For sets of ontologies containing hundreds of classes, it is simply impractical to manually verify these relations.

We believe that the latter greatly complicates the challenge in creating rules that can effectively operate on individual (and across) semantic domains.

### 3.2 Proposed Architecture Framework

In a first step toward mitigating these complexities, we propose the semantic modeling framework shown in Figure 3.3.

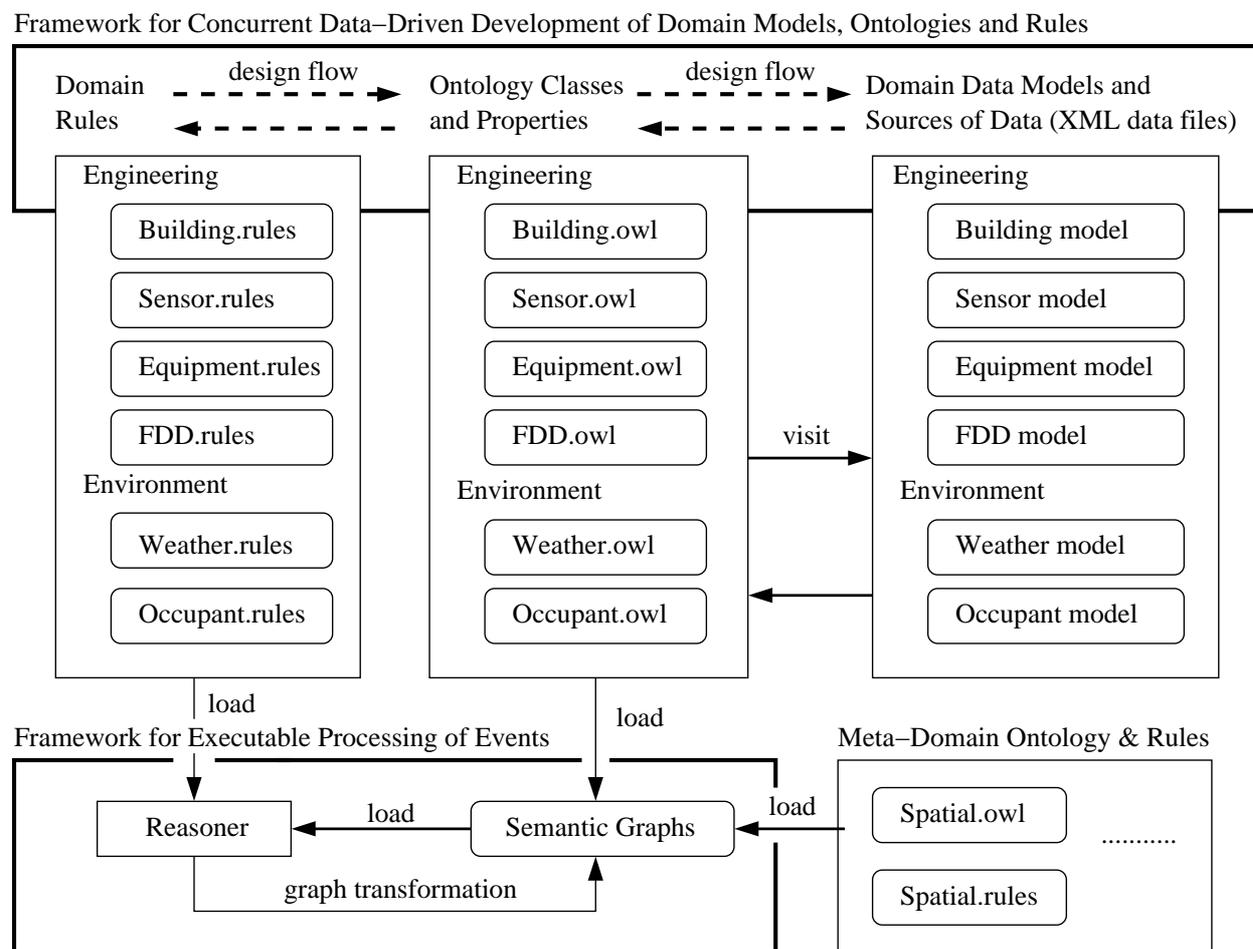


Figure 3.3: Proposed architecture for: (1) concurrent data-driven development of domain models, ontologies and rules, and (2) executable processing of events.

Our goal is to support two objectives:

1. Concurrent data-driven development of domain models, ontologies and rules, and

## 2. Executable processing of incoming faults.

Instead of creating ontologies and then developing a few rules for validation of model properties, our goal is to put the development of data, ontologies and rules on an equal footing. A key advantage of this approach is that it forces designers to provide semantic representations for data that are needed in decision making, and increases the likelihood that data not needed for decision making will be left out. Rules will be developed for verification of domain properties and processing of faults through reasoning with data sources, possibly from multiple domains. Implementation of the latter goal leads to semantic graphs that will dynamically adapt to the consequences of incoming data and events (e.g., changing occupant locations and weather events) acting on the system.

Our second strategy is to minimize the use of multiple inheritance in the specification of OWL ontologies and, instead, explore opportunities for replacing inheritance relationships by object property relations. In order for the architectural framework to be both scalable and adaptable to changing external conditions, the ontologies will need to be modular, and the rules will need to act both within a domain and across domains.

### 3.3 Working with Jena and Jena Rules

Our prototype software implementation makes extensive use of Apache Jena and Jena Rules. 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). The Jena rule-based inference subsystem is designed to allow a range of inference engines or reasoners to be plugged into Jena. Jena Rules is one such engine.

Jena Rules employs facts and assertions described in OWL to infer additional facts from instance data and class descriptions. As illustrated in Figure 3.4, it also provides support for the development of builtin functions that can link to external software programs and streams of data sensed in the real world. For the implementation of the vision implied by Figure 3.3, particularly support for spatial and temporal reasoning, the latter turns out to be crucially important because, by default, OWL only provides builtin datatype support for numbers (i.e., float and double),

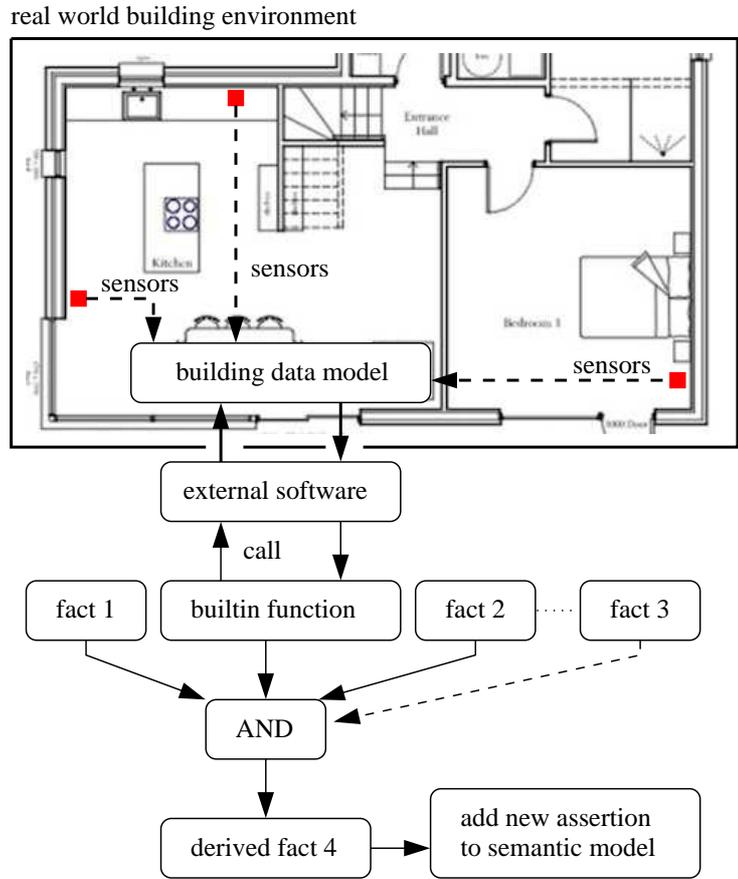


Figure 3.4: Framework for forward chaining of facts and results of builtin functions to new assertions (derived facts).

booleans (i.e., to represent true and false) and character strings (i.e., string). To combat the lack of support for complex data types, such as those needed to represent data for spatial and temporal reasoning, we adopt a strategy of embedding the relevant data in character strings, and then designing builtin functions and external software that can parse the data into spatial/temporal models, and then make the reasoning computations that are required.

### 3.4 Data-Driven Approach to Generation of Individuals in Semantic Graphs

In the proposed framework semantic models are the composition of ontologies, rules and data, which cooperatively work together to represent and model the response of multi-domain

environments to events.

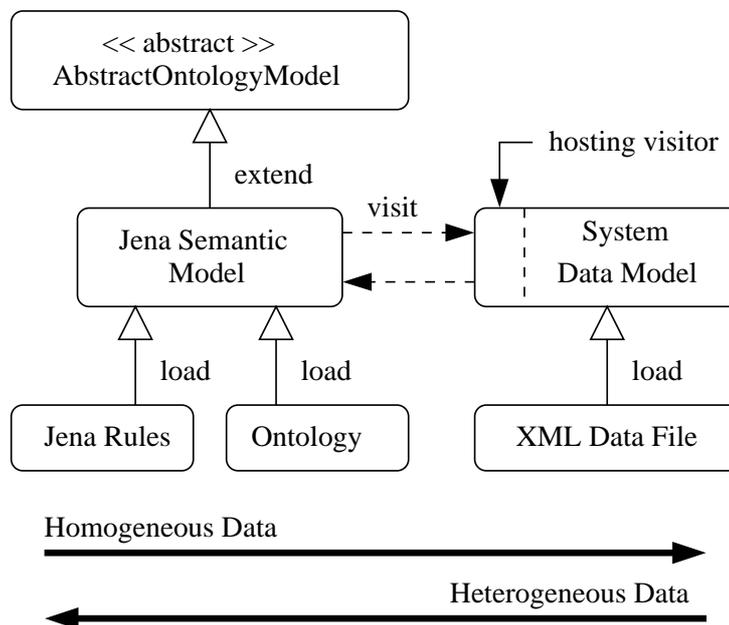


Figure 3.5: Data-driven approach to generation of individuals in semantic graphs.

Figure 3.5 illustrates a data-driven approach to the generation of individuals in semantic graphs. First, data is imported into Java Object data models using JAXB, the XML binding for Java [19]. We assume in this figure that the data is stored in an XML file but, of course, it could also come from other sources (such as streams of data shown in Figure 3.4). After the ontologies and rules have been loaded into the Jena Semantic Model, the semantic model creates instances of the relevant OWL ontologies by visiting the data model and gathering information on the individuals within a particular domain (e.g., building, sensor, occupant). Once the data has been transferred to the Jena Semantic Model and used to create an ontology instance, the rules are applied.

The lower section of Figure 3.5 shows the effect that this model has on the heterogeneity of data that needs to be managed. Generally speaking, it can be expected that the data models will be highly heterogeneous. When a semantic graph visits a data model to obtain the model individuals, only seven data types are available. Thus, as data moves from the data model into the semantic graph, the overall problem becomes homogeneous in the types of data stored. Thus, a key challenge in making this simplification of the problem data lies in the design and implementation of the appropriate visitor design pattern routines.

## Chapter 4

# Meta-Domain Ontologies and Rules

In systems analysis, a meta model defines the languages (semantics) and processes (structure and constraints) from which models can be formed. The meta model for SysML [26] defines, for example, more than 250 entities from which SysML diagrams can be constructed. The semantic modeling counterpart of software engineering meta-models is meta-domain ontologies and rules that have universal application to the implementation of targeted domain models. Sometimes the name fundamental is used instead of meta-domain. In either case, semantic descriptions of time, space, physical units and currency can all be thought of as essential elements for describing how our world actually works.

### 4.1 Spatial Ontology and Rules

This study employs spatial reasoning to determine the relationship of sensors and occupants to geometric entities such as rooms and building zones. Reasoning procedures are based on the logic of regions and their connectivity, allowing one to address issues of the form: what is true, and where? Formal theories for reasoning with space – points, lines, and regions – are covered by region connected calculus [33]. A robust implementation of two-dimensional spatial entities and associated reasoning procedures is provided by the Java Topology Suite (JTS) [20].

### 4.1.1 Experimental Ontology and Rules for Spatial Reasoning

Figure 4.1 shows an abbreviated representation of our experimental spatial (geometry) ontology and associated data and object properties. High-level classes – abstract concepts – are provided for entities that represent singular geometry (e.g., `AbstractGeometry`) and groups of entities (e.g., `AbstractGeometryCollection`).

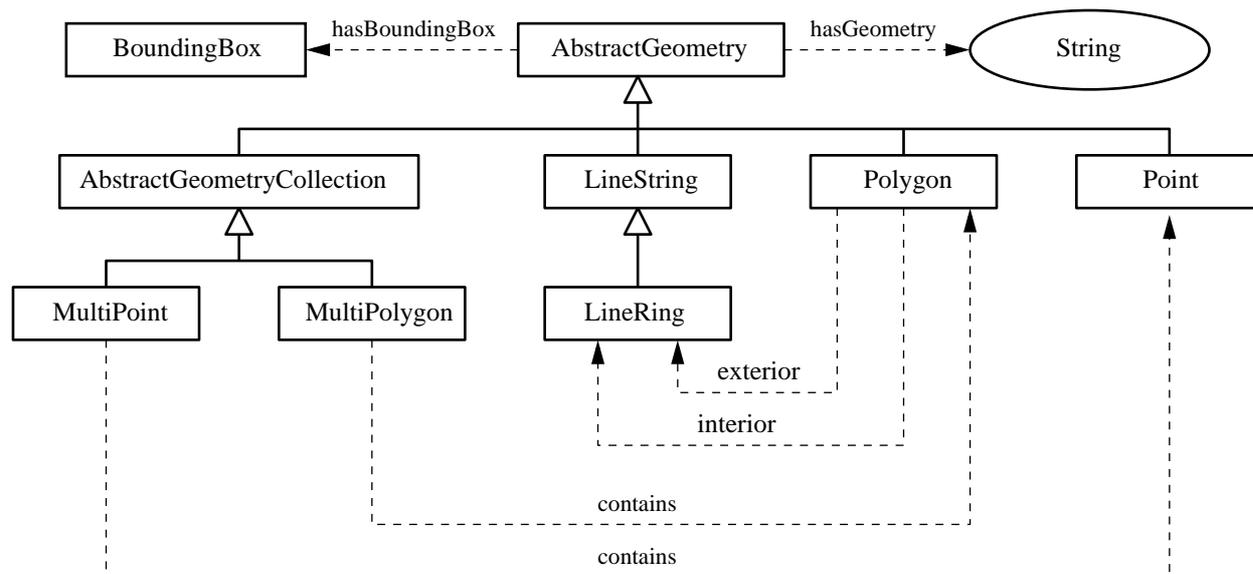


Figure 4.1: Abbreviated representation of spatial (geometry) ontology and associated data and object properties.

Specific types of geometry (e.g., `Polygon`, `MultiPoint`) are organized into a hierarchy similar to the Java implementation in JTS. The high-level class `AbstractGeometry` contains a `Datatype` property, `hasGeometry`, which stores a string representation of the JTS geometry. For example, the abbreviated string

```
POLYGON (( 0 0, 0 5, ... 0 0))
```

shows the format for pairs of (x,y) coordinates defining a two-dimensional polygon. This feature allows a semantic model to visit a domain data model, and gather a complete description of the two-dimensional geometry.

Within Jena Rules, families of builtin functions can be developed to evaluate the geometric

relationship between pairs of spatial entities (e.g., to determine whether or not a point is contained within a polygon).

---

```
Jena Rules
```

---

```
// Rule to check if a sensor is inside a room ...

[ BuildingRule01: (?r rdf:type bld:Room) (?r bld:hasGeometry ?rg) (?rg geom:hasGeometry ?rjts)
  (?s rdf:type sen:Sensor) (?s sen:hasGeometry ?sg) (?sg geom:hasGeometry ?sjts)
  getPointInPolygon(?sjts,?rjts,?t)
  equal(?t, "true"^^xs:boolean) -> (?s bld:isInRoom ?r)]
```

---

Figure 4.2: Rules to determine the rooms in which sensors have been placed.

Figure 4.2 shows, for example, the Jena Rule that identifies the room in which a sensor is placed. An English translation of the rule fragments is as follows: If (?r) is a room with geometry (?rg) and string representation (?rjts), and (?s) is a sensor with geometry (?sg) and string representation (?sjts), then the builtin function `getPointInPolygon(?sjts,?rjts,?t)` will determine if the sensor (point geometry) is inside the room (polygon geometry) and return the result as a boolean (?t). If (?t) is true, then the sensor is inside the room and a new relationship (`?s bld:isInRoom ?r`) is created. A similar rule would be written to establish the relationship between sensors and HVAC zones.

## Chapter 5

# Domain Ontologies and Rules

The domain-specific ontologies and rules are organized into two groups: (1) engineering ontologies and rules, and (2) surrounding environment ontologies and rules. In Figures 5.1 through 5.12 we use red rectangles with heavy dashed edges to highlight the classes that participate in the rule checking and/or the case study problem presented in Chapter 6.

### 5.1 Engineering Ontologies and Rules

The engineering ontologies and rules cover four domains: (1) buildings, (2) mechanical equipment, (3) sensors, and (4) procedures for fault detection and diagnosis.

#### 5.1.1 Building Ontology and Rules

The prototype building ontology and rules (see Figures 5.1 and 5.2) provide computational support for the representation of two-dimensional floorplan geometry, modeling relationships between elements of floorplan geometry and sensors, zones for HVAC control, and building elements such as doors, windows and walls. The latter are modeled as subclasses of a component that has geometry described by a JTS string.

Connections to the mechanical equipment and occupancy domains are achieved through data properties for the building environment state; see, for example, `hasRoomSetpoint` and `isOccupied`. Object properties record the relationship of a room to relevant HVAC zones and sensors.

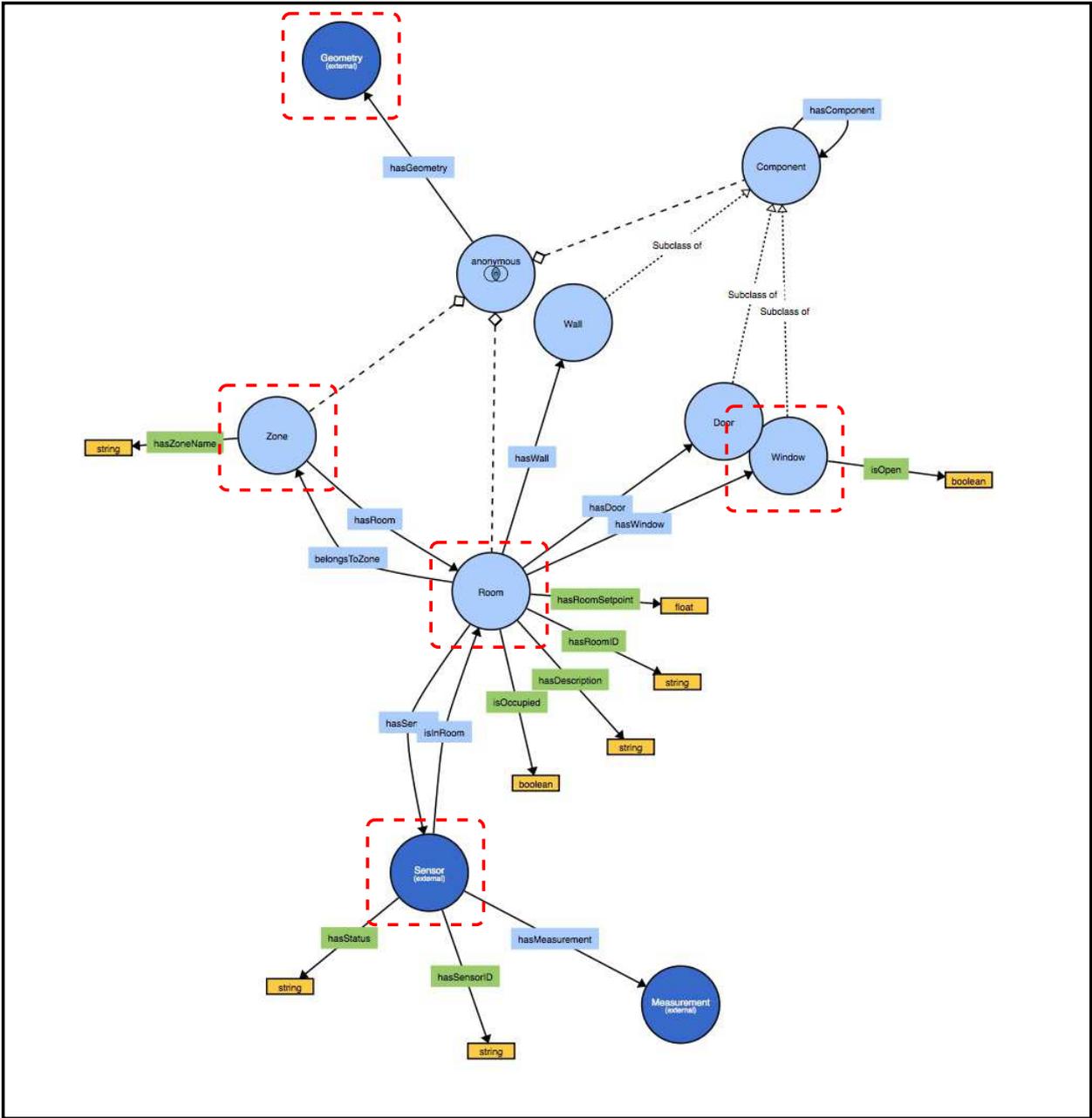


Figure 5.1: Schematic of building ontology classes and properties.

```
// Rule to check if two zones intersect ...  
  
[ BuildingRule02: (?r1 rdf:type bld:Zone) (?r1 bld:hasGeometry ?r1g) (?r1g geom:hasGeometry ?r1jts)  
  (?r2 rdf:type bld:Zone) (?r2 bld:hasGeometry ?r2g) (?r2g geom:hasGeometry ?r2jts)  
  notEqual( ?r1jts, ?r2jts ) getPointInPolygon( ?r1jts, ?r2jts, ?t)  
  equal(?t, "true"^^xs:boolean) -> (?r1 bld:intersects ?r2)]
```

Figure 5.2: Rule for Zone Intersect.

Windows have the boolean data property `isOpen` to record whether or not a particular window is open. As we will soon in the case study problem, this parameter plays a pivotal role in diagnostic analysis of the causes leading to a fault in mechanical equipment.

The prototype software implementation has one rule for determining the spatial relationship among zones of the building. The rule systematically retrieves the JTS geometry of each zone, verifies they are not equal, and then uses the builtin function `getPointInPolygon()` to verify their geometric relationship. As previously noted, these backend computations are handled by the Java Topology Suite software [20].

### 5.1.2 Mechanical Equipment Ontology and Rules

Figures 5.3 and 5.4 illustrate the concepts (i.e., ontology classes), properties (i.e., data and object properties) and rules governing the operation and identification of faults in mechanical systems equipment. In practice, datatype property values associated with the various ontologies will be set from streams of data either performed by a simulation tool (e.g. EnergyPlus, Dymola, TRN-SYS) [13, 23, 40], or perhaps from measurements taken in a real building, working in conjunction with BACnet protocols [7] and a co-simulation middleware.

The semantic graph shown in Figure 5.3 is quite broad, covering concepts from chillers and fans to zones. The scope of our investigation focuses on faults associated with valves, coils and air handling units. Basic rules (see Figure 5.4) are provided for: (1) controlling the flow in a valve, (2) determining if a valve is leaky, and (3) identifying situations where the normal operational status of a valve is false. Thus, we are able to determine that when a cooling coil valve is faulty, the associated air handling unit is also faulty.



```

// Close the valve when the coil temperature is the same as coil setpoint.

[ EquipmentRule01: (?coil rdf:type eq:Coil) (?coil eq:hasCoilSetpoint ?sp)
  (?coil eq:hasCoilTemperature ?cp) equal(?sp,?cp) (?coil eq:hasValve ?valve) ->
  (?valve eq:isShutOff "true"^^xs:boolean) print('valve is shut')]

// If the valve is shut, the temperature of the air that passes through the coil
// has to be the same. Otherwise, the valve is leaky

[ EquipmentRule02: (?hvw rdf:type eq:Valve) (?hvw eq:isShutOff "true"^^xs:boolean)
  (?c rdf:type eq:Coil)(?c eq:hasValve ?hvw) (?c eq:Tad ?t1)
  (?c eq:Tas ?t2) notEqual(?t2 ?t1) -> (?hvw eq:isLeaky "true"^^xs:boolean)
  (?hvw eq:hasNormalOperationalStatus "false"^^xs:boolean) print('valve is Leaky') ]

// If the a valve fails, the AHU fails too ...

[ EquipmentRule03: (?hvw rdf:type eq:Valve) (?AHU eq:hasCoil ?c) (?c eq:hasValve ?v)
  (?v eq:hasNormalOperationalStatus "false"^^xs:boolean) ->
  print('AHUMalfunction') (?AHU eq:hasNormalOperationalStatus "false"^^xs:boolean)]

```

---

Figure 5.4: Rules for establishing the operational status and simple operations of mechanical equipment.

### 5.1.3 Sensor Ontology and Rules

Figure 5.5 shows the classes and properties in our experimental sensor ontology. Our goal is to provide computational support for modeling: (1) sensor operation, including when a sensor reading might be outside an acceptable working range, and (2) determining the location of a sensor relative to the environment in which it is embedded. These objectives are achieved with three classes: Sensor, Measurement, and the external class Geometry. Support for modeling various types of sensor (e.g., temperature sensor, flow sensor, and CO<sub>2</sub> sensor) is provided through the definition of specialized sensor classes that subclass Sensor. The class Measurement has data properties to keep track of the current sensor value, the time, and the units associated with the measurement.

Two sensor rules (see Figure 5.6) are supported: (1) To determine if a sensor reading is beyond the acceptable range, (2) To determine the room in which the sensor is located. The first rule uses the classes Sensor and Measurement and associate properties. The second rule uses the classes Sensor and Geometry.

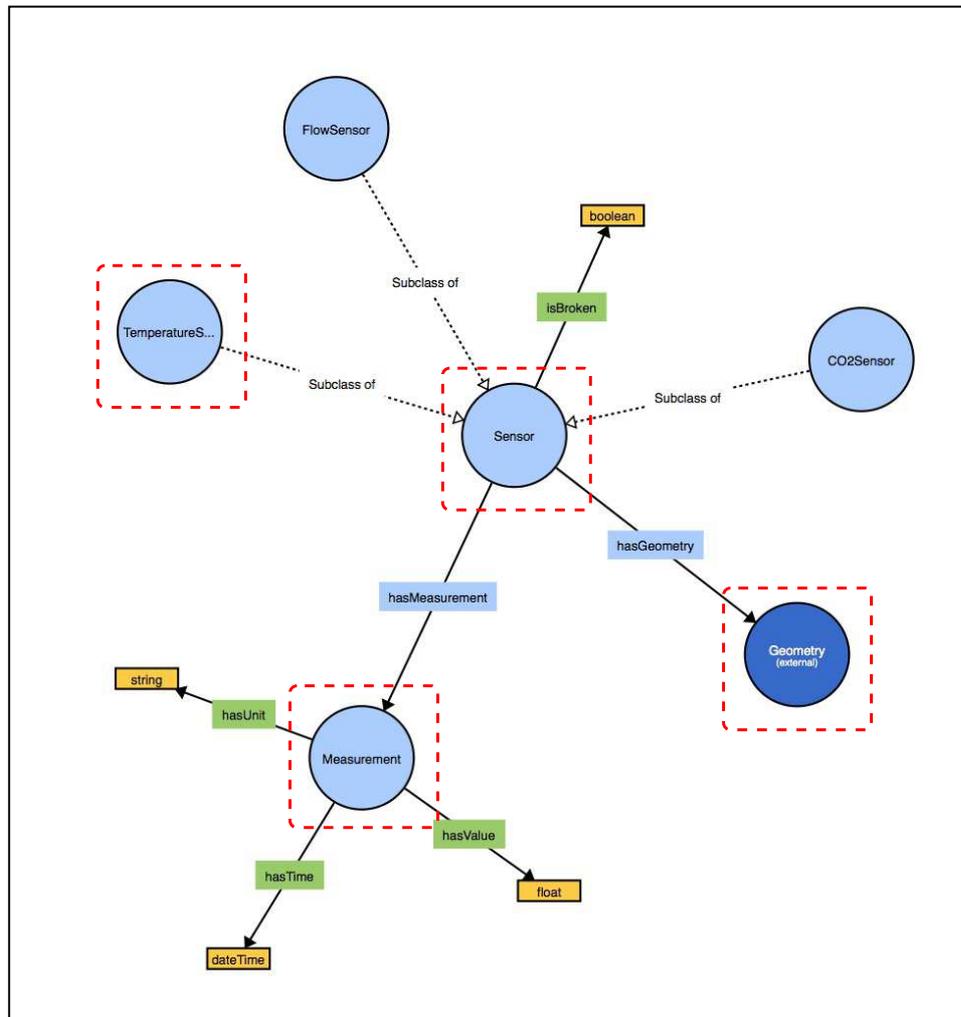


Figure 5.5: Sensor ontology classes and properties.

---

Jena Rules

---

```
// Simple rule to check if a sensor is broken ...
[ SensorRule01: (?s rdf:type sen:Sensor) (?s sen:hasMeasurement ?m) (?m sen:hasValue ?r)
  isOutOfRange(?m ?t) -> (?s sen:isBroken ?t) ]
```

---

Figure 5.6: Rule for intersection of HVAC zones.

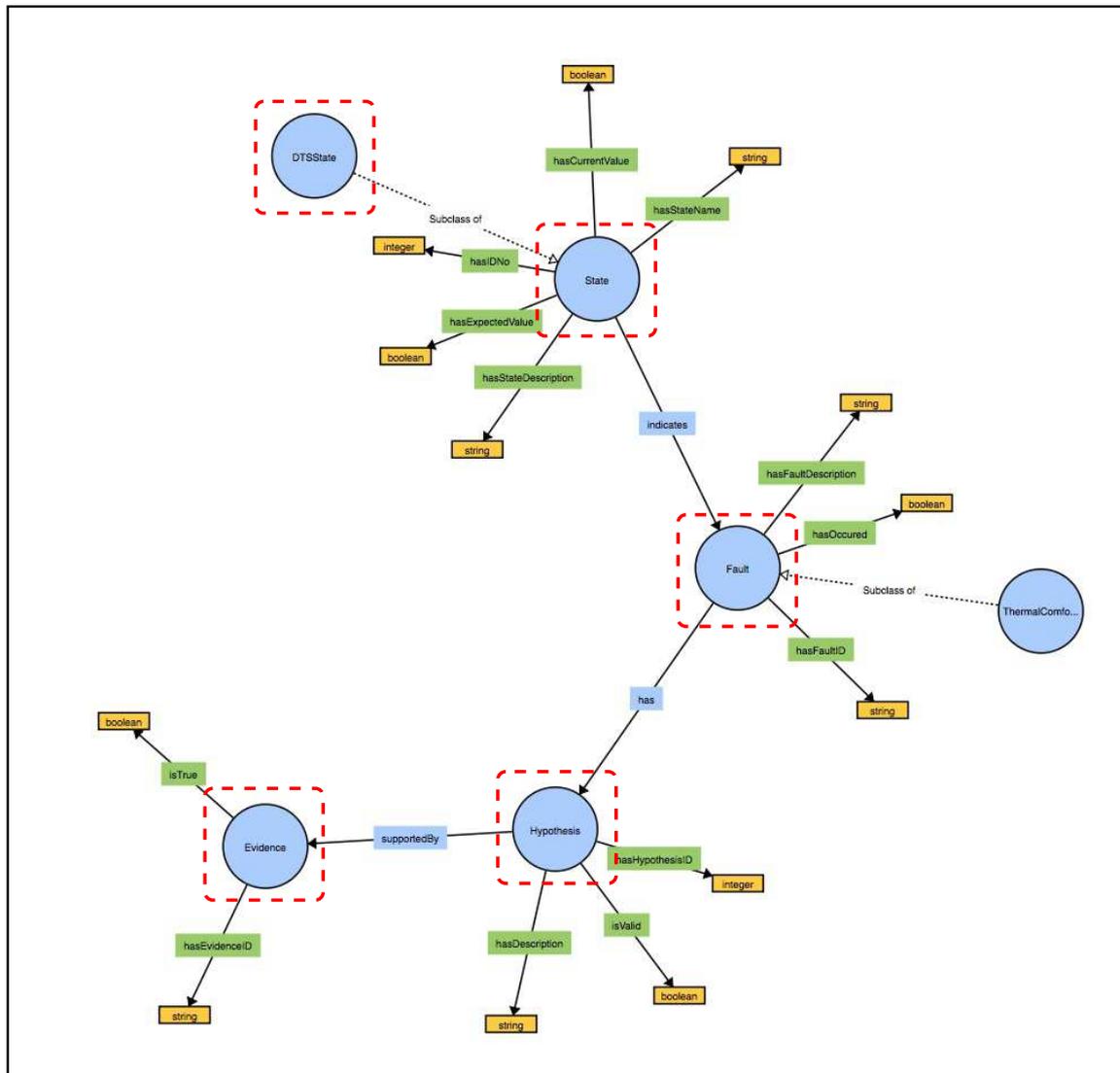


Figure 5.7: Fault detection and diagnostic ontology classes and properties.

---

Jena Rules

---

```
// General purpose rule for recording when a fault has occurred.
[FDDRule01: (?st rdf:type fdd:State) (?st fdd:hasCurrentValue ?csv)
 (?st fdd:belongsToFault ?F) (?st fdd:hasExpectedValue ?esv)
 notEqual(?csc,?esv) -> (?F fdd:hasOccured ''true'') print('faultoccured')]
```

---

Figure 5.8: Rule for detecting a faulty state.

### 5.1.4 Fault Detection and Diagnostic Ontologies, Rules, and Procedures

The fault detection and diagnostic (FDD) ontology (see Figure 5.7) captures the knowledge needed for: (1) identifying that a fault exists, and (2) systematically diagnosing the fault to find the root causes. The main classes in this process are State, Fault, Hypothesis and Evidence. State is a high-level state representation that has data values – see, for example, the boolean properties `hasExpectedValue` and `hasCurrentValue` – common to many types of state representation. Our experimental FDD ontology also supports `DTSSState`, a subclass of `State`, designed to represent states associated with dynamic thermal sensation (DTS).

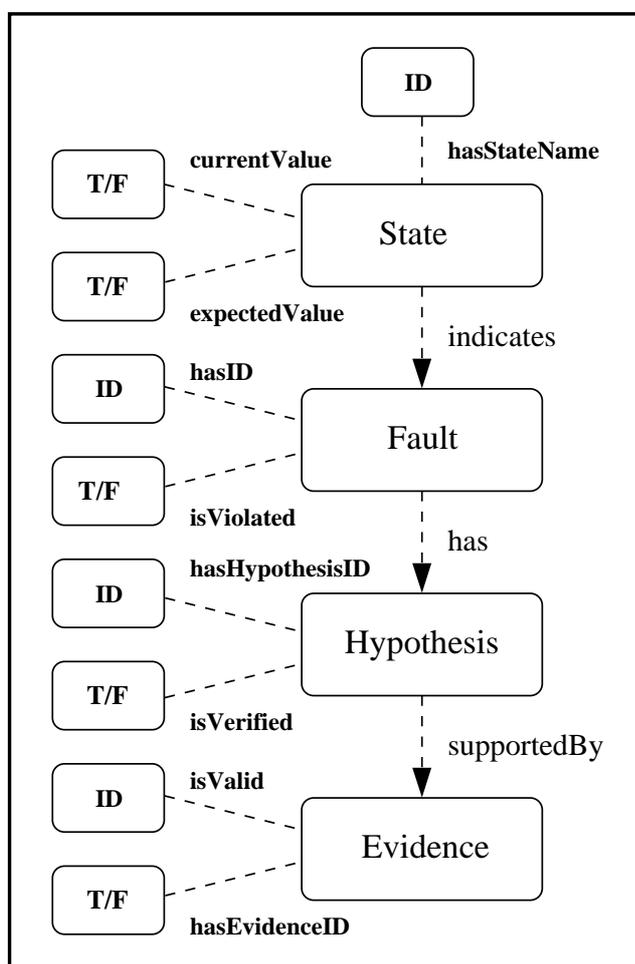


Figure 5.9: Flow chart for identification of faults and identification and verification of hypotheses and supporting evidence.

Figure 5.9 is a flowchart for fault detection and the identification and verification of rel-

evant hypotheses and supporting evidence. The step-by-step procedure for detecting a fault and diagnosing its causes corresponds to a traversal through the classes State, Fault, Hypothesis and Evidence. A fault is indicated when the current and expected values of a state are in conflict. Each fault has a hypothesis that needs to be supported by evidence. The evaluation procedure works backwards. Verification of the evidence is a prerequisite to validating a hypothesis. In an implementation of the procedure, data properties indicate whether or not a fault has been verified, whether or not an hypothesis has been verified, and whether or not supporting evidence is valid. This procedure is mirrored by set of rules shown in Figure 5.8.

## 5.2 Surrounding Environment Ontologies and Rules

The surrounding environment ontologies and rules include model support for the building occupants and weather phenomena.

### 5.2.1 Occupant Ontology and Rules

While several studies [1, 25] have recently identified the importance of including inhabitants as an integral part of simulation and control of energy systems and indoor environments, present-day procedures rely on predetermined occupancy schedules and/or empirical estimates based on sensors. For fault detection and diagnostic analysis of mechanical equipment in buildings, solutions are complicated by the strong coupling of human presence, comfort and behavior, to details of the building state (e.g., whether or not a window is open) and surrounding environment (e.g., what side of the building is in the sun).

Figures 5.10 and 5.11 take a first step toward the development of an ontology and rules for modeling occupant presence. The ontology expands upon the work of Mahdavi and Taheri [27], and considers four sub-category problems: (1) location, (2) actions (e.g., open/close window), (3) attitudes (e.g., thermal sensation) and (4) preferences in terms of temperature and moisture of the air. We model occupant location with a point geometry in the building, Figure 5.11 shows two rules that infer occupant's location and thermal comfort respectively.

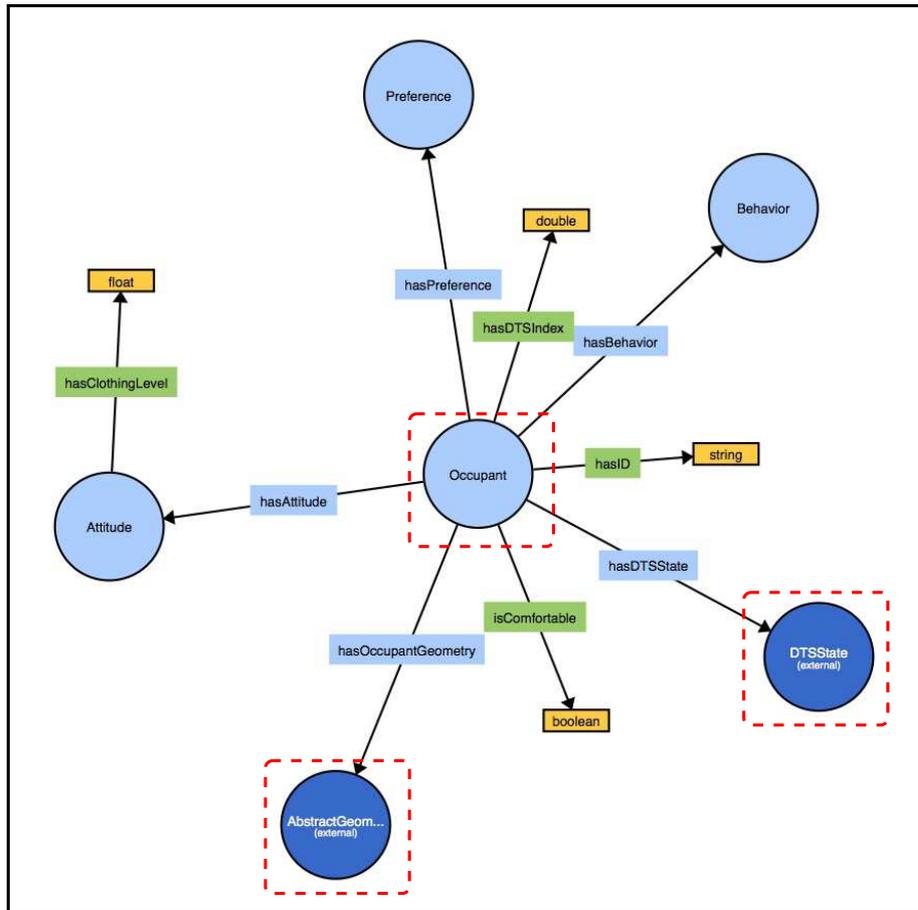


Figure 5.10: Schematic of occupant ontology classes and properties.

---

Jena Rules

```
// Determine room in which an occupant is located.

[ OccupantRule01: (?r rdf:type bld:Room) (?o rdf:type occ:Occupant)
  (?o occ:hasOccupantGeometry ?og) (?og geom:hasGeometry ?ojts)
  (?r bld:hasGeometry ?rg) (?rg geom:hasGeometry ?rjts)
  getPointInPolygon(?ojts,?rjts,?t) equal(?t, "true"^^xs:boolean) ->
  (?r bld:hasOccupant ?o) print(?o, 'Occupant is in Room', ?r, ?t)]

// When positive values of DTSIndex are greater than 0.3, an occupant is not comfortable.

[ OccupantRule02: (?oc rdf:type occ:Occupant) (?oc occ:hasDTSIndex ?v) greaterThan(?v, 0.3)
  (?oc occ:hasDTSSState ?dts) -> print(?oc, 'isComfortable' "false"^^xs:boolean)
  (?oc occ:isComfortable "false"^^xs:boolean)
  (?dts fdd:hasCurrentValue "false"^^xs:boolean)]
```

---

Figure 5.11: Rule for occupants location and thermal comfort.

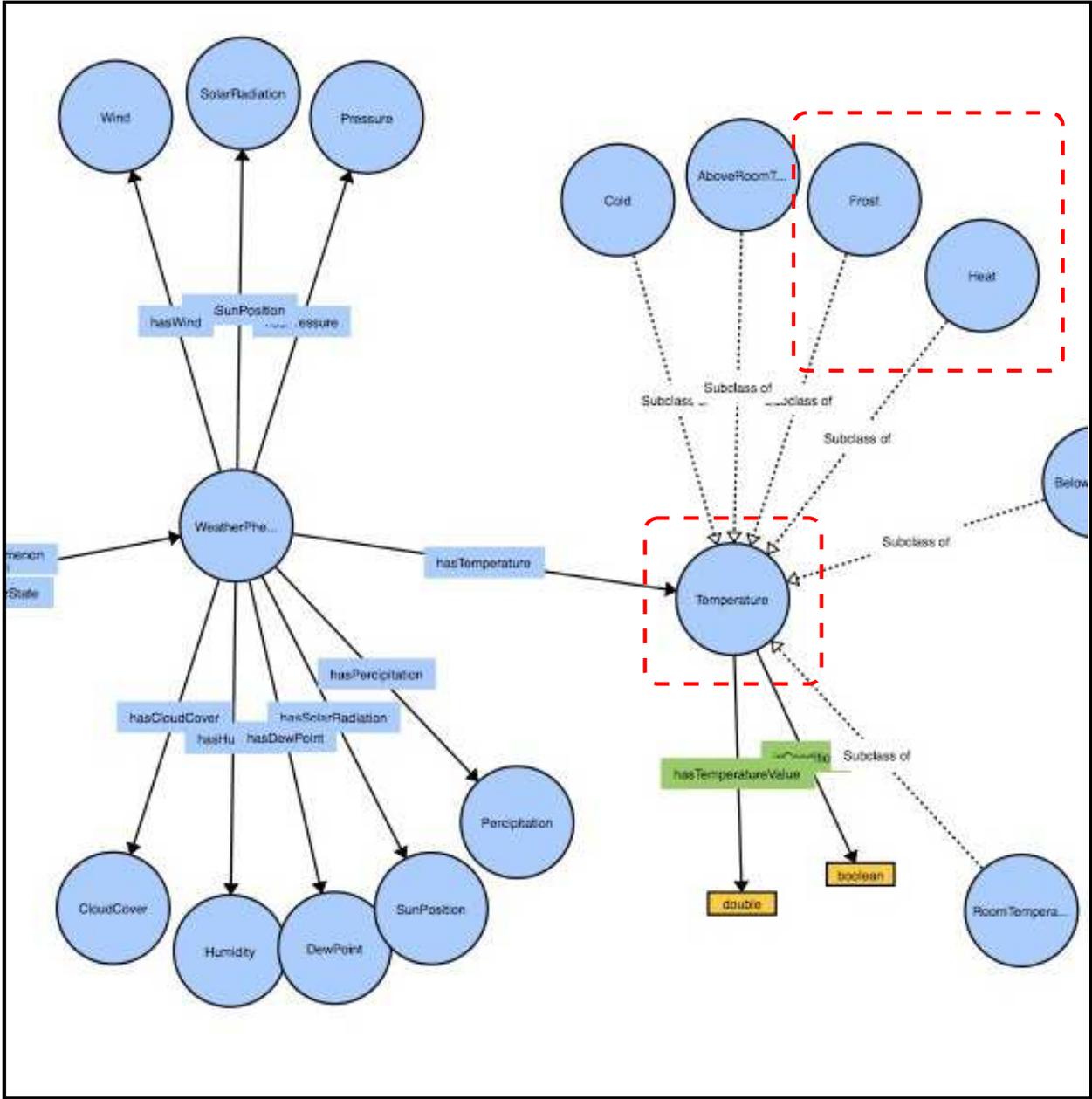


Figure 5.12: Partial view of weather ontology classes and properties (Source: Adapted from Staroch [38]).

## 5.2.2 Weather Ontology and Rules

Based upon the work of Staroch [38], Figure 5.12 presents the concepts that are used in Weather Ontology. The main concepts are Weather Phenomenon, Weather Report, and Weather State. The weather state is composed of different Weather phenomenon class holds the physical attributes regarding the weather such as the temperature, pressure, solar radiation, wind and cloud. Weather data is obtained from current Weather [42], a free and open source API (application programming interface) that provides access to historical as well as current and future forecast weather data from an online server. A Weather report can include data about the current weather or a forecast, specified in terms of start time and duration. For example, a medium range weather report has duration of more than 3 hours, with a start time of less than 12 hours into the future.

---

```
Jena Rules
// Use current temperature value to identify a frosty temperature condition ...

WeatherRule01: (?t rdf:type we:Temperature) (?t we:hasTemperatureValue ?tv)
lessThan(?tv,0) -> (?t rdf:type we:Frost)
(?t, we:isCondition, "true"^^xs:boolean) print(?tv,'FrostCondition')]

// Use current temperature value to identify a heat temperature condition ...

WeatherRule02: (?t rdf:type we:Temperature) (?t we:hasTemperatureValue ?tv)
greaterThan (?tv,30) -> (?t rdf:type we:Heat)
(?t, we:isCondition, "true"^^xs:boolean) print(?tv,'Heat')
```

---

Figure 5.13: Rules to detect weather condition.

Figure 5.13 presents two rules that use the current temperature value to identify a frosty and heat temperature conditions. A Frost temperature condition occurs when observed temperature is below 0 C. A Heat temperature condition occurs when observed temperature is above 30 C. Similar intervals of temperature range can be defined for cold, below room temperature (at least 10 C and less than 20 C), and so forth.

## Chapter 6

# Case Study Problem

To examine capabilities of the framework for knowledge-based fault detection and diagnostic analysis, this chapter presents a case study test problem where faults in HVAC equipment are triggered by occupant discomfort in a conditioned space. The case study shows how heterogeneous data and knowledge from a variety of sources and domains can be integrated into a single semantic graph, how ontologies and rules can work together to detect the existence of a fault, and then diagnose the causes by systematically considering hypotheses and the supporting evidence.

### 6.1 Problem Description

Figure 6.1 is a plan view of the case study problem setup, consisting a small two-room building architecture, three sensors and three building occupants. See Appendix A for a complete description of the floorplan geometry and sensor characteristics. Not shown is the mechanical equipment responsible for conditioning the room temperature and achieving acceptable levels of occupant comfort. The mechanical equipment consists of an air handling unit (AHU). The AHU has a coil (i.e., for heating and cooling). The water temperature that flows to the coil is managed by a valve.

Three rules are responsible for the operation and classification of faults in the mechanical equipment:

- Close the valve when the coil temperature is the same as coil setpoint.

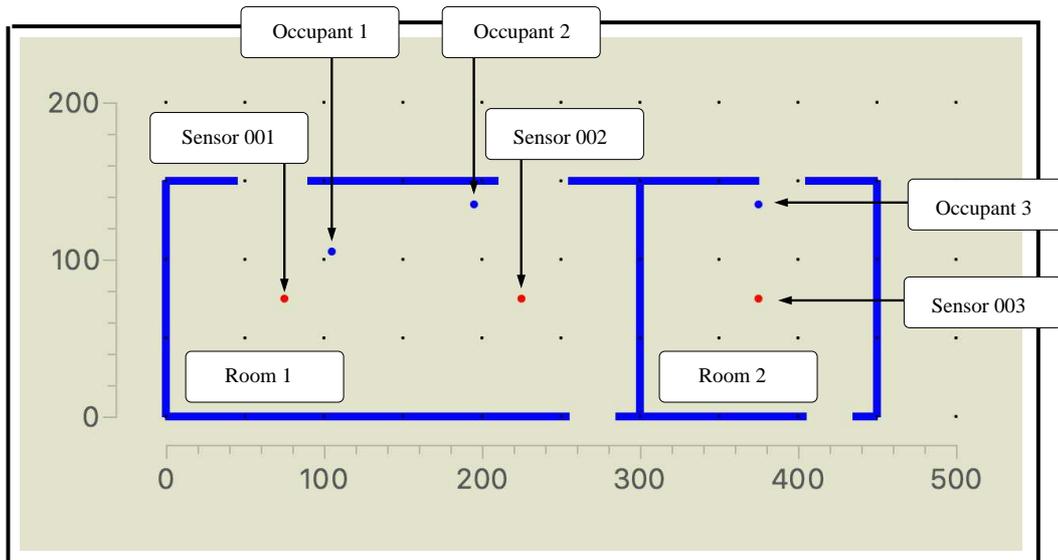


Figure 6.1: Plan view of two-room building architecture, sensors, and building occupants.

- If the valve is shut, the temperature of the air that passes through the coil has to be the same. Otherwise, the valve is leaky
- If the a valve fails, the AHU fails too.

One measure to evaluate thermal comfort for the occupants is through computing the thermal sensation as a function of environmental factors such as outdoor and indoor temperature and some personal factors such as clothing levels. A dynamic model to compute thermal sensation (DTS) index to was introduced by Chen and co-workers [11]. According to thermal sensation scale suggested by ASHRAE [3], an acceptable range for occupancy comfort is the interval  $[-0.3, 0.3]$ . By comparing the current and expected values in a DTS state, the rules in Figure 5.8 will infer the existence of a faulty state, and then systematically examine the evidence associated with each hypothesis to find a root cause.

## 6.2 Snapshot of Semantic Graph Model Assembly

Figure 6.2 shows a snapshot of the building, equipment, sensor, weather, and FDD ontologies integrated together, and populated with system data. The semantic graph model contains instances of ontologies (individuals), relationships among individuals (often spanning domains),

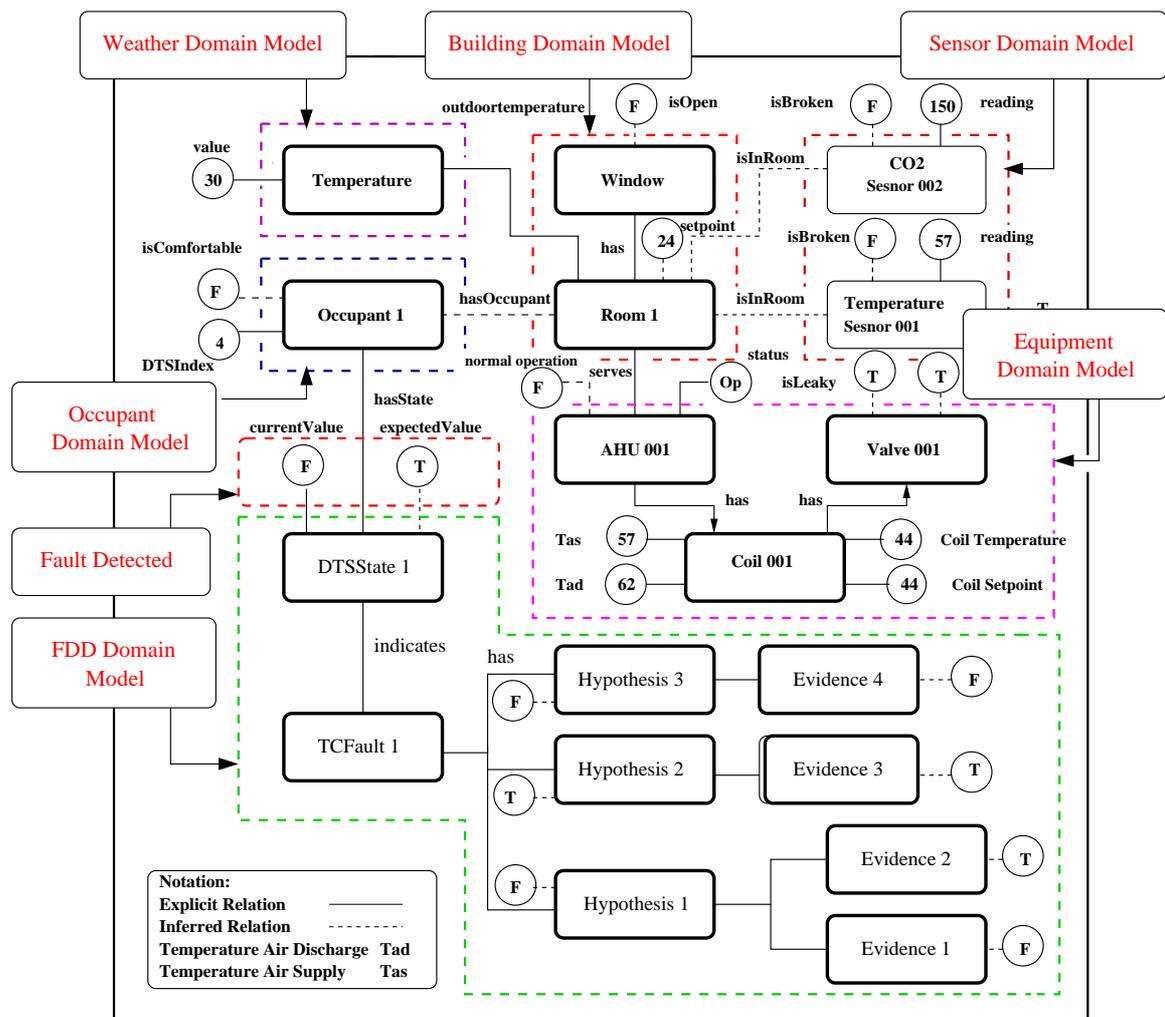


Figure 6.2: Snapshot of fully assembled semantic graph model. The data values will be computed and filled by the rules.

<b>Class</b>	<b>Individual</b>	<b>Description</b>
State	DTSSState 1	The DTS index in between $[-0.3, 0.3]$ .
Fault	TCFault 1	The DTS index lies outside the interval $[-0.3, 0.3]$ when the air-handling unit is operating.
Evidence	Evidence 1	The CO2 sensor reading is above the normal range the and that shows the window is open.
	Evidence 2	The outdoor temperature is greater than room setpoint.
	Evidence 3	A sensor's reading is outside the range that indicates the sensor is broken.
	Evidence 4	A component is AHU is malfunctioning that results in an abnormal operation of AHU.
Hypothesis	Hypothesis 1	Warm outside air is leaking into the room through an open window $\rightarrow$ Supported by Evidence 1 and Evidence 2.
	Hypothesis 2	The serving air-handling unit has abnormal operation. $\rightarrow$ Supported by Evidence 3.
	Hypothesis 3	The room sensor that provides feed-back to AHU reaching its target setpoint is broken $\rightarrow$ Supported by Evidence 4.

Table 6.1: Instances of states, hypotheses, and evidence for identifying the cause for abnormal occupant thermal comfort value.

```

// Evidence Rule 01: A window is open base on CO2 concentration in the room.
// -----

[ EvidenceRule01: (?cs rdf:type sen:CO2Sensor) (?cs bld:isInRoom ?room)
  (?r bld:hasWindow ?w)(?cs bld:hasReading ?m) lessThan(?m,600)
  greaterThan(?m,400) (?e fdd:hasEvidenceID ?n) equal("1"^^xs:integer,?n) ->
  (?w building:isOpen "true"^^xs:boolean) (?e fdd:isTrue "true"^^xs:boolean) ]

// Evidence Rule 02: Outside temperature is warmer than the setpoint.
// -----

[ EvidenceRule02: (?r rdf:type bld:Room) (?r bld:hasSetpoint ?sp)
  (?t rdf:type we:Temperature) (?t we:hasTemperatureValue ?tv)
  greaterThan(?tv,?sp) equal("2"^^xs:integer,?n) (?e rdf:type fdd:Evidence)
  (?e fdd:hasEvidenceID ?n) -> (?e fdd:isTrue "true"^^xs:boolean) ]

// Evidence Rule 03: Temperature sensor in a room is broken.
// -----

[EvidenceRule03: (?ts rdf:type sen:TemperatureSensor) (?ts bld:isInRoom ?room)
  (?ts bld:isBroken ?t) equal(?t, "true"^^xs:boolean) equal("3"^^xs:integer,?n)
  (?e rdf:type fdd:Evidence)
  (?e fdd:hasEvidenceID ?n ->(?e fdd:isTrue "true"^^xs:boolean) ]

// Evidence Rule 04: Malfunction is in the Air Handling Unit.
// -----

[EvidenceRule04: (?AHU rdf:type eq:AHU) (?v eq:hasNormalOperationalStatus "false"^^xs:boolean)
  equal(?t, "true"^^xs:boolean) equal("4"^^xs:integer,?n)
  (?e rdf:type fdd:Evidence)-> (?e fdd:isTrue "true"^^xs:boolean) ]

// FDD Rule 02: Indicate when thermal comfort in a conditioned room has expected value.
// -----

[FDDRule02: (?AHU rdf:type eq:AHU)(?AHU eq:servesRoom ?r)(?r bld:hasOccupant ?oc)
  (?oc occ:hasDTSSState ?dts) (?AHU eq:status ?s)
  equal(?s "Operating") -> print('Expected DTS',?oc)
  (?dts fdd:hasExpectedValue "true"^^xs:boolean) ]

```

---

Figure 6.3: Fault detection diagnostic rules for operation of a heating coil and for checking evidence 3 and evidence 4.

and data values associated with various individuals.

From a fault detection and diagnostics standpoint, the main points to note are as follows:

- Occupant 1 is located in Room 1.
- Room 1 has window, a temperature sensor (Sensor 001), and a carbon dioxide sensor (Sensor 002). HVAC services are provided to Room 1 by air handling unit AHU 001. AHU 001 has a coil (Coil 001); Coil 001 has a valve (Valve 001).
- The datatype property for AHU001 “normal Operation” is set to false. This setting is based on the system data and the result of equipment rules 01 through 03 being triggered.
- The setpoint temperature for Room 1 is 24 C, but the current temperature reading for Sensor 001 is 57 C.
- OccupantRule02 sets the ”isComfortable” datatype property for Occupant1 to “false” as the result of a DTSindex value of 4.
- Occupant 1 has dynamic thermal sensation (DTS) state DTSSState 1. DTSSState 1 indicates a thermal comfort fault (TCFault1), which will be diagnosed by looking at three hypotheses and their supporting evidence.
- The relationship between Hypotheses 1 through 3 and supporting evidence is shown along the bottom of Figure 6.2. Users may query the semantic graph to find the correct hypotheses and valid supporting evidence.

### 6.3 Test Problem Scenario and Hypothesis Evaluation Procedure

The test problem scenario assumes that the numerical value of occupant thermal comfort in a conditioned room has fallen outside the acceptable range. This is detected by FDD Rule 01. With this scenario in place, any one of three hypotheses could potentially be true. To correctly identify the correct hypothesis, the system requires to reason among the facts and identify the evidence existing in different domains,

- The outdoor temperature is higher than the setpoint (weather) and the window in the room is open (building, sensor, weather).

- The air-handling unit is malfunctioning (mechanical equipment),
- The room sensor providing feed-back to the air-handling unit to reach its target setpoint is broken (sensor).

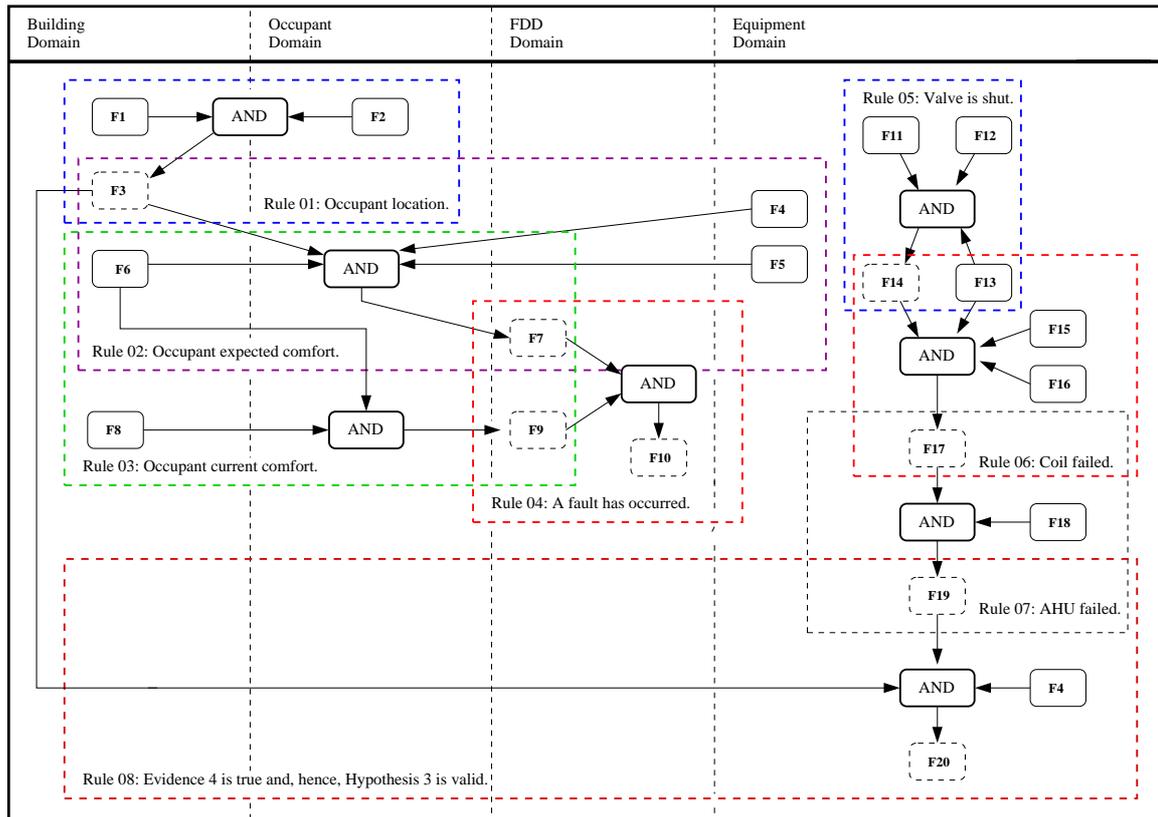
As a result, this task will require comprehensive reasoning over multiple domains and identifying the supporting evidence to the most probable hypothesis. To achieve this, we used the proposed framework and implemented ontologies for weather, building, occupant, sensor and equipment domains. The ontologies are populated with data. However, in general this data will be obtained from simulations or real buildings.

## 6.4 Synthesis of Multi-domain Rules

Table 6.1 describes the instances for key concepts of FDD ontology as they apply to the test case problem, and explains details of the individuals for FDD ontology. For the case study problem, the chain of dependency relationships between hypotheses and supporting evidence is as follows:

- Hypothesis 1 is that warm outside air is leaking into the room through an open window. Evaluation of this hypothesis is supported by execution of two evidence rules, EvidenceRule01 and EvidenceRule02.
- Hypothesis 2 is that the serving air-handling unit has abnormal operation. Evaluation of this hypothesis is supported execution of EvidenceRule03.
- Hypothesis 3 states that the room sensor that provides feedback to AHU reaching its target setpoint is broken. Supporting evidence is provided by the execution of EvidenceRule04.

Figure 6.3 presents the fault detection diagnostic rules for: (1) Operation of a heating coil, (2) Checking evidence 3 and evidence 4, and (3) Detecting when the thermal comfort in a conditioned room matches its expected value.



Legend:

- |                                   |                                    |   |
|-----------------------------------|------------------------------------|---|
| F1 = Room hasGeometry             | F8 = Occupant1 hasDTSIndex 4       | F15 = Coil001 Tas 62                        |
| F2 = Occupant hasGeometry         | F9 = DTSSState currentValue false  | F16 = Coil001 Tad 57                        |
| F3 = Room1 has Occupant1          | F10 = DTSSState indicates DTSFault | F17 = Valve001 isShut normalOperation false |
| F4 = AHU001 serves Room1          | F11 = Coil001 CoilSetpoint 44      | F18 = AHU001 hasCoil Coil001                |
| F5 = AHU001 status operating      | F12 = Coil001 CoilTemperature 44   | F19 = AHU001 normalOperation false          |
| F6 = Occupant1 hasState DTSSState | F13 = Coil001 hasValve Valve001    | F20 = Evidence1 isValid true                |
| F7 = DTSSState expectedValue true | F14 = Valve001 isShut true         |   |

Figure 6.4: Snapshot of multi-domain evaluation and forward chaining of rules.

## 6.5 Multi-domain Rule Evaluation

Figure 6.4 shows a snapshot of multi-domain evaluation and forward chaining of rules. From an evaluation standpoint, the eight rules can be clustered into two pathways, the first focusing on fault detection and the second focusing on diagnostic investigation of probable causes, represented as hypotheses and supporting evidence.

### 6.5.1 Fault Detection

The first pathway identifies the existence of a fault and is covered by rules 1 through 4:

- Rule 01: Use OccupantRule01 (see Figure 5.11) to determine when an occupant is located in a room.
- Rule 02: Use FDDRule02 (see Figure 6.3) to determine the expected comfort of an occupant.
- Rule 03: Use OccupantRule02 (see Figure 5.11) to determine the current comfort of an occupant.
- Rule 04: Use OccupantRule02 (see Figure 5.11) to compute when a fault has occurred.

determine in which room an occupant is located and whether or not the current value of occupant comfort matches the expected value of comfort. In the snapshot, activation of Rule 01 determines that: Occupant1 is located in Room1. A separate execution would also determine that Occupant2 is also located in Room1. Activation of Rule 02 is based upon the output of Rule 01, state data from the building domain, the relationship of the air handling unit to Room1. In the snapshot trace, the output of Rule 02 states that DTSSState for Occupant1 is true and that Occupant1 has a DTSIndex of 4. A fault occurs when there is a discrepancy between the current and expected values of comfort (see F7 and F9), as indicated by the values of current and expected values of DTSSState.

### 6.5.2 Fault Diagnostics

By systematically examining hypotheses and supporting evidence, the second pathway diagnoses the causes of a fault. For the scenario outlined in Figure 6.4, this procedure is covered by rules 5 through 8:

- Rule 05: Use EquipmentRule01 (see Figure 5.4) to determine if a valve is shut.
- Rule 06: Use EquipmentRule02 (see Figure 5.4) to determine if the coil has failed.
- Rule 07: Use EquipmentRule03 (see Figure 5.4) to determine whether or not the air handling unit has failed.
- Rule 08: If EvidenceRule04 (see Figure 6.3) evaluates to true then Hypothesis 3 is true.

The rule for determining whether or not the valve is shut takes input values from the Coil001 CoilSetpoint (44) and CoilTemperature (44) (see F12 and F13), and checks to verify that the coil has a valve. In our scenario, the rule output (F14) is true, indicating that Valve001 is shut, and hence in Rule 06 normal operation evaluates to false. A simple check to verify that the coil belongs to air handling unit AHU001 generates the conclusion that normal operation of the AHU is false (see F19). Finally, input from the room occupancy test and a test to verify that AHU001 is connected to Room1, leads to the conclusion Evidence 4 is supported and Hypothesis 3 is valid. Finally, we note that except for the room occupancy information feeding into Rule 08, the fault detection and diagnosis pathways operate independently.

## Chapter 7

# Conclusions and Future Work

### 7.1 Conclusions

We have proposed in this report a knowledge-based framework for fault detection and diagnostics. The underlying process closely mimics the “thinking process” that humans follow in identifying and diagnosing the causes of a fault. Thus, the steps of gathering data for the participating domains, populating ontologies with individuals, and using rules to detect and diagnose faults and their causes is easy for humans to understand and generally applicable to other domains (e.g., building energy, automotive, health care) for FDD purposes. Capabilities of the prototype implementation has been demonstrated by working step by step through the procedure of detecting and diagnosing the source of faults in an HVAC system.

Key advantages of this approach include: (1) it is decoupled from the system simulation, (2) it is comprehensive, and (3) it is scalable. In fact, the process for expanding an application to include new domains as they come along is very straight forward. The inference-based rules are guaranteed to check at anytime a changed occurred in a an ontology resulting in event-driven fault detection and diagnostic. Finally, inference-based rules provide mechanisms in capturing chain effects that exists in the nature of system failure – for example, if a valve is not operational, the evidence that AHU is not operating properly also holds true.

## 7.2 Future Work

In our prototype implementation, the small two-room building model extracted data from a custom “system data model” currently under development. We expect that a more mature version of this ontology would extract semantic information from instances of building information models (BIM) such as the Industry Foundation Class (IFC). Future work will also include deployment in real building systems. We anticipate that the proposed methodology will be integrated into building automation systems (BAS) and support investigations where analytic built-in functions are implemented in the condition part of inference-based rules. These functions will perform time-history analyses to identify a faulty state for the system. We anticipate a trend where formal approaches to analysis are used to irregularities in building performance, which are indicators of possible system faults. Moreover, we will investigate strategies for taking control actions based on recognized faults of the system.

## 7.3 Acknowledgment

The first author was supported by a fellowship award from the NIST Graduate Student Measurement Science and Engineering (GMSE) Program.

# Bibliography

- [1] Agarwal Y., Balaji B., Gupta R., Lyles J., Wei M., and Weng T. Occupancy-Driven Energy Management for Smart Building Automation. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building (BuildSys 2010)*, pages 1–6, Zurich, Switzerland, November 3-5 2010.
- [2] Apache Jena. An Open Source Java Framework for building Semantic Web and Linked Data Applications, Accessible at <https://jena.apache.org> (Accessed on 12/12/16). 2016.
- [3] ASHRAE Standard 552010 Thermal Environmental Conditions for Human Occupancy, American Society of Heating, Refrigerating and Air-Conditioning Engineers, 2010.
- [4] Austin M.A., Delgoshaei P., and Nguyen A. Distributed System Behavior Modeling with Ontologies, Rules, and Message Passing Mechanisms. *Procedia Computer Science*, 44:373 – 382, 2015. 2015 Conference on Systems Engineering Research.
- [5] Austin, M.A., Mayank V., and Shmunis, N. Ontology-Based Validation of Connectivity Relationships in a Home Theater System. *21st International Journal of Intelligent Systems*, 21(10):1111–1125, October 2006.
- [6] Austin, M.A., Mayank, V., and Shmunis, N. PaladinRM: Graph-Based Visualization of Requirements Organized for Team-Based Design. *Systems Engineering*, 9(2):129–145, May 2006.
- [7] BACnet: A Data Communication Protocol for Building Automation and Control Networks, ANSI/ASHRAE 135, 2004.
- [8] Batic M., Tomasevic N., and Vranes S. Ontology-based Fault Detection and Diagnosis System Querying and Reasoning Examples. In *ICKDDM 2015 : 17th International Conference on Knowledge Discovery and Data Mining*, volume 2. International Science Index, Industrial and Manufacturing Engineering, 2015.

- [9] Bayer T., Dvorak D., Friedenthal S., Jenkins S., Lin C. and Mandutianu S. Foundational Concepts for Building System Models. In *SEWG MBSE Training Module 3*, see <http://nen.nasa.gov/web/se/mbse/documents>, California Institute of Technology, CA, USA, 2012.
- [10] Berners-Lee T., Hendler J., and Lassa O. The Semantic Web. *Scientific American*, pages 35–43, May 2001.
- [11] Chen X., Wang Q., and Srebric J. Occupant Feedback-based Model Predictive Control for Thermal Comfort and Energy Optimization: A Chamber Experimental Evaluation. *Applied Energy*, 164:341 – 351, 2016.
- [12] Corsar D., Milan D., Edwards P., and Nelson J.D. *The Transport Disruption Ontology*, pages 329–336. Lecture Notes in Computer Science, vol 9367, Springer, 2015.
- [13] Crawley D.B., Lawrie L.K., Winkelmann F.C., Buhl W.F., Huang Y.J., Pedersen C.O., Strand R.K., Liesen R.J., Fisher D.E., Witte M.J., and Glazer J. EnergyPlus: Creating a New-Generation Building Energy Simulation Program. *Energy and Buildings*, 33(4):319 – 331, 2001. Special Issue: {BUILDING} SIMULATION’99.
- [14] Delgoshaei P., Austin M.A., and D. Veronica D. Semantic Models and Rule-based Reasoning for Fault Detection and Diagnostics: Applications in Heating, Ventilating and Air Conditioning Systems. *The Twelfth International Conference on Systems (ICONS 2017)*, pages 48–53, April 23-27 2017.
- [15] 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.
- [16] Dung T.Q. and Kameyama W. *Ontology-based Information Extraction and Information Retrieval in Health Care Domain*, volume 4654 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 323–333. 2007.
- [17] Feigenbaum L., 2006. Semantic Web Technologies in the Enterprise.
- [18] Fogarty K. and Austin M.A. System Modeling and Traceability Applications of the Higraph Formalism. *Systems Engineering*, 12(2):117–140, 2009.

- [19] Java Architecture for XML Binding (JAXB), See: <http://www.oracle.com/technetwork/articles/javase/index-140168.html> (Accessed November 5, 2017).
- [20] Java Topology Suite (JTS). See <http://www.vividsolutions.com/jts/> (Accessed August 4, 2017).
- [21] Katipamula S. and Brambley M.R. Review Article: Methods for Fault Detection, Diagnostics, and Prognostics for Building Systems A Review, Part I. *HVAC&R Research*, 11(1):3–25, 2005.
- [22] Kim W. and Braun J.E. Impacts of refrigerant charge on air conditioner and heat pump performance. In *Impacts of Refrigerant Charge on Air Conditioner and Heat Pump Performance*, pages 2433–2441, July 10–15 2010.
- [23] Klein S.A., Beckman W.A., et al., 1994. TRNSYS: A Transient Simulation Program, Engineering Experiment Station Report 38-12, University of Wisconsin, Madison.
- [24] Lord P., Bechhofer S., Wilkinson M.D., Schiltz G., Gessler D., Hull D., Goble C., Stein L., McIlraith, S.A., Plexousakis D., and Van Harmelen F. *Applying Semantic Web Services to Bioinformatics: Experiences Gained, Lessons Learnt*, pages 350–364. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [25] Lu J., Sookoor T., Srinivasan V., Gao G., Holben B., Stankovic J., Field E., and Whitehouse K. The Smart Thermostat: Using Occupancy Sensors to Save Energy in Homes. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys 2010)*, pages 211–224, Zurich, Switzerland, November 3-5 2010.
- [26] MagicDraw Architecture Made Simple: SysML Metamodel, Version 18.1, No Magic Inc, Allen, Texas, 2015.
- [27] Mahdavi A. and Taheri M. An Ontology for Building Monitoring. *Journal of Building Performance Simulation*, pages 1–10, October 2016.
- [28] Merdan M. Knowledge-based Multi-Agent Architecture Applied in the Assembly Domain. Ph.D. Dissertation, Vienna University of Technology, 2009.
- [29] 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.
- [30] Open Street Map (OSM). <https://www.openstreetmap.org> (Accessed April 3, 2017). 2017.

- [31] OWL: Web Ontology Language Overview, W3C Recommendation from February, 2004. For details, see <http://www.w3.org/TR/owl-features/> (Accessed, April 2017).
- [32] Petnga L., and Austin M.A. An Ontological Framework for Knowledge Modeling and Decision Support in Cyber-Physical Systems. *Advances in Engineering Informatics*, 30(1):77–94, January 2016.
- [33] Randell D.A., Cui Z., and Cohn A.G. A Spatial Logic based on Regions and Connectivity, 1994. Division of Artificial Intelligence, School of Computer Studies, Leeds University.
- [34] Scherer W.T. and White C.C. *A Survey of Expert Systems for Equipment Maintenance and Diagnostics*, pages 285–300. Springer US, Boston, MA.
- [35] Schumann S., Hayes J., Pompey P., and Verscheure O. Adaptable Fault Identification for Smart Buildings. In *2011 AAAI Workshop (WS-11-07)*, 2011.
- [36] Selberg S., and Austin M.A. Toward an Evolutionary System of Systems Architecture. In *Proceedings of Eighteenth Annual International Symposium of The International Council on Systems Engineering (INCOSE)*, Utrecht, The Netherlands, June 15-19 2008.
- [37] Siegel J.A. and Wray C.P. An Evaluation of Superheat-based Refrigerant Charge Diagnostics for Residential Cooling Systems/Discussions. *ASHRAE Transactions* 108(1), page 965, 2002.
- [38] Staroch P. A Weather Ontology for Predictive Control in Smart Homes, 2013. M.S. Thesis in Software Engineering and Internet Computing, Vienna University of Technology.
- [39] Taswell C. DOORS to the Semantic Web and Grid with a PORTAL for Biomedical Computing. *IEEE Trans Inf Technol Biomed*, 12(2):191–204, 2008.
- [40] TRNSYS: The Transient Energy System Simulation Tool. See: <http://www.trnsys.com/> (Accessed September 8, 2017)., 2017.
- [41] Wagner D.A., Bennett M.B., Karban R., Rouquette N., Jenkins S., and Ingham M.O. An Ontology for State Analysis: Formalizing the Mapping to SysML. In *Proceedings of 2012 IEEE Aerospace Conference*, Big Sky, Montana, March 2012.
- [42] Weather API. See <https://openweathermap.org/api> (Accessed September 14, 2017).
- [43] Wiggins M. and Brodrick J. Emerging Technologies: HVAC Fault Detection. *ASHRAE Journal*, pages 78–80, April 2012.

# Appendix A

## Building System Data Model

This appendix contains a complete description of the building system data model represented in XML. The data model formulation is inspired by OpenStreetMap [30] (OSM) and its use of only three types of tag – `<node>`, `<way>` and `<relation>` – and associated attributes to represent an extremely wide range of system structures found in urban areas. Thus, we begin with `<node>`, `<way>` and `<relation>` and add new tags for component (i.e., `<component>`) and shape (i.e., `<shape>`). In a departure from OSM, our long-term intent is that components will have various forms of continuous and discrete behavior. Attributes inside the shape (i.e., `<shape>`) specify how the component, way, or relation should be visualized.

We employ JAXB technology [19] to import the XML data files into the system data model. The data model formulation supports hashmaps of attribute data within each of the tagsets and, as such, is very general and naturally extensible.

### A.1 System Data Model (SystemDataModel.xml)

The following XML contains details (i.e., room geometry, sensors, doors, windows and portals) for the two-room model shown in Figure 6.1.

---

```
<?xml version="1.0" encoding="UTF-8"?>
<SystemDataModel author="Mark Austin" date="2017-06" source="UMD">
```

```

<!-- ===== -->
<!-- Part 01: Building Data Model Attributes -->
<!-- ===== -->

<attribute key = "latitude" value = " 45.00"/>
<attribute key = "longitude" value = "-46.00"/>
<attribute key = "orientation" value = "0.0"/>
<attribute key = "units" value = "m"/>

<!-- ===== -->
<!-- Part 02: Boundary Nodes and Sensors -->
<!-- ===== -->

<!-- Corner point nodes -->

<node ID="001" x = " 0.0" y = "0.0" type="Point" />
<node ID="002" x = " 8.5" y = "0.0" type="Point" />
<node ID="003" x = " 9.5" y = "0.0" type="Point" />
<node ID="004" x = "10.0" y = "0.0" type="Point" />
<node ID="005" x = "13.5" y = "0.0" type="Point" />
<node ID="006" x = "14.5" y = "0.0" type="Point" />
<node ID="007" x = "15.0" y = "0.0" type="Point" />
<node ID="008" x = " 0.0" y = "5.0" type="Point" />
<node ID="009" x = " 1.5" y = "5.0" type="Point" />
<node ID="010" x = " 3.0" y = "5.0" type="Point" />
<node ID="011" x = " 7.0" y = "5.0" type="Point" />
<node ID="012" x = " 8.5" y = "5.0" type="Point" />
<node ID="013" x = "10.0" y = "5.0" type="Point" />
<node ID="014" x = "12.5" y = "5.0" type="Point" />
<node ID="015" x = "13.5" y = "5.0" type="Point" />
<node ID="016" x = "15.0" y = "5.0" type="Point" />

<!-- ===== -->
<!-- Part 03: Ways for Wall Segments -->
<!-- ===== -->

<!-- Sequence of ways defining boundary of AVW 2206 ... -->

<way ID="001" type="Boundary">
  <description text="Interior Office Wall Segment." />
  <attribute key = "type" value = "InteriorWall"/>
  <attribute key = "material" value = "drywall"/>
  <attribute key = "thickness" value = "20"/>
  <attribute key = "units" value = "cm"/>
  <node ID="002" />
  <node ID="001" />
  <node ID="008" />
  <shape type = "LineString">
    <attribute key = "width" value = "6.0"/>
  </shape>
</way>

<way ID="002" type="Boundary">
  <description text="Exterior Wall Segment." />

```

```

    <attribute key =      "type" value = "ExteriorWall"/>
    <attribute key =  "material" value = "Masonry"/>
    <attribute key = "thickness" value = "30"/>
    <attribute key =      "units" value = "cm"/>
    <node ID="008" />
    <node ID="009" />
    <shape type = "LineString">
      <attribute key = "width" value = "9.0"/>
    </shape>
  </way>

<way ID="003" type="Boundary">
  <description text="Exterior Wall Segment." />
  <attribute key =      "type" value = "ExteriorWall"/>
  <attribute key =  "material" value = "Masonry"/>
  <attribute key = "thickness" value = "30"/>
  <attribute key =      "units" value = "cm"/>
  <node ID="010" />
  <node ID="011" />
  <shape type = "LineString">
    <attribute key = "width" value = "9.0"/>
  </shape>
</way>

<way ID="004" type="Boundary">
  <description text="Exterior Wall Segment." />
  <attribute key =      "type" value = "ExteriorWall"/>
  <attribute key =  "material" value = "Masonry"/>
  <attribute key = "thickness" value = "30"/>
  <attribute key =      "units" value = "cm"/>
  <node ID="012" />
  <node ID="013" />
  <shape type = "LineString">
    <attribute key = "width" value = "9.0"/>
  </shape>
</way>

<way ID="006" type="Boundary">
  <description text="Interior Office Wall Segment." />
  <attribute key =      "type" value = "InteriorWall"/>
  <attribute key =  "material" value = "drywall"/>
  <attribute key = "thickness" value = "20"/>
  <attribute key =      "units" value = "cm"/>
  <node ID="013" />
  <node ID="004" />
  <node ID="003" />
  <shape type = "LineString">
    <attribute key = "width" value = "6.0"/>
  </shape>
</way>

<!-- Sequence of ways defining boundary of AVW 2210 ... -->

<way ID="007" type="Boundary">
  <description text="Interior Office Wall Segment." />

```

```

    <attribute key =      "type" value = "InteriorWall"/>
    <attribute key =  "material" value = "drywall"/>
    <attribute key = "thickness" value = "20"/>
    <attribute key =      "units" value = "cm"/>
    <node ID="005" />
    <node ID="004" />
    <node ID="013" />
    <shape type = "LineString">
      <attribute key = "width" value = "6.0"/>
    </shape>
  </way>

<way ID="008" type="Boundary">
  <description text="Exterior Wall Segment." />
  <attribute key =      "type" value = "ExteriorWall"/>
  <attribute key =  "material" value = "Masonry"/>
  <attribute key = "thickness" value = "30"/>
  <attribute key =      "units" value = "cm"/>
  <node ID="013" />
  <node ID="014" />
  <shape type = "LineString">
    <attribute key = "width" value = "9.0"/>
  </shape>
</way>

<way ID="009" type="Boundary">
  <description text="Exterior Wall Segment." />
  <attribute key =      "type" value = "ExteriorWall"/>
  <attribute key =  "material" value = "Masonry"/>
  <attribute key = "thickness" value = "30"/>
  <attribute key =      "units" value = "cm"/>
  <node ID="015" />
  <node ID="016" />
  <shape type = "LineString">
    <attribute key = "width" value = "9.0"/>
  </shape>
</way>

<way ID="010" type="Boundary">
  <description text="Interior Office Wall Segment." />
  <attribute key =      "type" value = "InteriorWall"/>
  <attribute key =  "material" value = "drywall"/>
  <attribute key = "thickness" value = "20"/>
  <attribute key =      "units" value = "cm"/>
  <node ID="016" />
  <node ID="007" />
  <node ID="006" />
  <shape type = "LineString">
    <attribute key = "width" value = "6.0"/>
  </shape>
</way>

<!-- Window portals for AVW 2206 and AVW 2210 ... -->

<way ID="011" type="Portal">

```

```

        <description text="Window portal for AVW 2206." />
        <attribute key = "type" value = "WindowOpening"/>
        <node ID="009" />
        <node ID="010" />
    </way>

    <way ID="012" type="Portal">
        <description text="Window portal for AVW 2206." />
        <attribute key = "type" value = "WindowOpening"/>
        <node ID="011" />
        <node ID="012" />
    </way>

    <way ID="013" type="Portal">
        <description text="Window portal for AVW 2210." />
        <attribute key = "type" value = "WindowOpening"/>
        <node ID="014" />
        <node ID="015" />
    </way>

    <!-- Doorway portals for AVW 2206 and AVW 2210 ... -->

    <way ID="014" type="Portal">
        <description text="Doorway for AVW 2206." />
        <attribute key = "type" value = "Doorway"/>
        <node ID="002" />
        <node ID="003" />
    </way>

    <way ID="015" type="Portal">
        <description text="Doorway for AVW 2210." />
        <attribute key = "type" value = "Doorway"/>
        <node ID="005" />
        <node ID="006" />
    </way>

    <!-- ===== -->
    <!-- Part 04: AVW office relations -->
    <!-- ===== -->

    <!-- Relations for room boundaries ... -->

    <relation ID="001" type="CompositeBoundary">
        <description text="AVW Rm 2206 (Room Boundary)" />
        <attribute key = "function" value = "Office"/>
        <way ID="001" />
        <way ID="002" />
        <way ID="011" />
        <way ID="003" />
        <way ID="012" />
        <way ID="004" />
        <way ID="006" />
        <way ID="014" />
        <shape type = "MultiPolygon">
            <attribute key = "opacity" value = "0.3"/>

```

```

        <attribute key = "color" value = "LightBlue"/>
    </shape>
</relation>

<relation ID="002" type="CompositeBoundary">
    <description text="AVW Rm 2210 (Room Boundary)" />
    <attribute key = "function" value = "Office"/>
    <way ID="007" />
    <way ID="008" />
    <way ID="013" />
    <way ID="009" />
    <way ID="010" />
    <way ID="015" />
    <shape type = "MultiPolygon">
        <attribute key = "opacity" value = "0.3"/>
        <attribute key = "color" value = "LightBlue"/>
    </shape>
</relation>

<!-- Relations for individual rooms ... -->

<relation ID="003" type="Room">
    <description text="AVW Rm 2206 (Faculty Office)" />
    <attribute key = "function" value = "Office"/>
    <attribute key = "area" value = "50.0"/>
    <attribute key = "units" value = "m^2"/>
    <relation ID="001" />
</relation>

<relation ID="004" type="Room">
    <description text="AVW Rm 2210 (Faculty Office)" />
    <attribute key = "function" value = "Office"/>
    <attribute key = "area" value = "25.0"/>
    <attribute key = "units" value = "m^2"/>
    <relation ID="002" />
</relation>

<!-- ===== -->
<!-- Part 05: Architectural and Sensor Components -->
<!-- ===== -->

<!-- Sensor Components -->

<component ID="001" x = " 2.5" y = "2.5" type="Sensor">
    <description text="Smoke Detector." />
    <attribute key = "measurement" value = "Smoke"/>
    <attribute key = "name" value = "Smell-Smoke-Call-911"/>
    <attribute key = "status" value = "Broken"/>
    <shape type = "Circle">
        <attribute key = "radius" value = "3.0"/>
        <attribute key = "color" value = "Red"/>
    </shape>
</component>

<component ID="002" x = " 7.5" y = "2.5" type="Sensor">

```

```

    <description text="Room Thermometer." />
    <attribute key = "measurement" value = "Temperature"/>
    <attribute key =      "name" value = "Excellanto Temp-001"/>
    <attribute key =      "status" value = "Operating"/>
    <shape type = "Square">
      <attribute key =      "side" value = "5.0"/>
      <attribute key = "opacity" value = "1.0"/>
      <attribute key =      "color" value = "Green"/>
    </shape>
  </component>

<component ID="003" x = " 12.5" y = "2.5" type="Sensor">
  <description text="Room Thermometer." />
  <attribute key = "measurement" value = "Temperature"/>
  <attribute key =      "name" value = "Excellanto Temp-001"/>
  <attribute key =      "status" value = "Operating"/>
  <shape type = "Rectangle">
    <attribute key =      "width" value = "5.0"/>
    <attribute key =      "height" value = "5.0"/>
    <attribute key = "opacity" value = "1.0"/>
    <attribute key =      "color" value = "Green"/>
  </shape>
</component>

<!-- Building Architecture Components -->

<component ID="004" type="Window">
  <description text="Double-Paned Office Window" />
  <attribute key = "width" value = "1.0"/>
  <attribute key = "height" value = "2.0"/>
  <attribute key = "units" value = "m"/>
  <attribute key = "status" value = "Open" />
  <shape type = "Rectangle">
    <attribute key = "opacity" value = "1.0"/>
    <attribute key =      "color" value = "Green"/>
  </shape>
</component>

</SystemDataModel>

```

---

Key points to note are as follows:

1. Rooms are specified in two parts. Relations 001 and 002, i.e.,

```

<relation ID="001" type="CompositeBoundary"> ...
<relation ID="002" type="CompositeBoundary"> ...

```

define the geometry of Rooms 1 and 2, respectively. The relations are of type “composite boundary” because the room perimeters are mixtures of physical wall elements and portals, windows and doors. The abbreviated details of relations for Rooms 1 and 2 are specified as:

```
<relation ID="003" type="Room">
  <description text="AVW Rm 2206 (Faculty Office)" />

  <relation ID="001" />
</relation>

<relation ID="004" type="Room">
  <description text="AVW Rm 2210 (Faculty Office)" />

  <relation ID="002" />
</relation>
```

On the software backend, connections between the various types of relations and components, ways and sequences of nodes, is managed by a series of hashmaps.

2. Notice that the room specifications do not contain references to the sensors (i.e., components 001 through 003). Instead, we use spatial reasoning to infer in which room each sensor is located.