UNIVERSITÀ DEGLI STUDI DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

Contents

Introduction

Industrialised countries are facing challanges that are strictly related to energy consumption. Although great efforts in studying novel energy sources are profused by both public and private entities, these solutions need to be sustained by efforts aimed at the reduction of energy wasting. Around 32% of total energy consumption in industrialized countries is used for electricity, heating, ventilation and air-conditiong (HVAC) in commercial buildings. Studies have shown that early detection of faults in those systems and their operation could lead to energy savings between 15% to 32%, improving the overall buildings efficiency. This scenario does not only offer great opportunities related to environmental protection but gives companies a chance for sensible expense reduction. Around 14% of commercial buildings in the US (as of 2012) deployed Building Management System (BMS), supported by the increasing number of available sensors and improvement in the Internet of Things technologies. Still those data are juste barely used for simple analytics due to the lack of common schema for data coming from a variety of building, vendor and location specific sources that prevent the developers to easily integrate and process those data in a more detailed way. Energy Management Systems (EMS) are usually monolithic and poorly integrated solutions tailored for specific buildings, usually these solutions are costly in terms of both money and time. They requires efforts to encode expert knowledge that can't be resued.

Chapter 1

Semantic Web

In this chapater will follow a brief description of the concept of ontology and of Semantic Web as defined by W3C. This chapter will then furter detail the ontologies and metadata schemata used in the thesis project, namely Semantic Sensor Network (SSN) and Brick. It is to note that the work presented in this thesis is not directly using all of these technologies, but it draws from their key concepts to reach some goals. It is therefore worth to mention such concepts here.

1.1 Introduction to the Semantic Web

Semantic Web is a term that identify an evolution of the web in which every resource available through the web is associated with a semantic meaning. Semantic Web technologies enable people to create data stores on the Web, build vocabularies and write rules for handling data. Semantic web layers are defined by W3C as:

- linked data
- vocabularies
- query
- inference

1.1.1 Linked data

Linked Data lies at the heart of what Semantic Web is all about: large scale integration of, and reasoning on, data on the Web. To make the Web of Data a reality, it is important to have the huge amount of data on the Web available in a standard format, reachable and manageable by Semantic Web tools. Furthermore, not only does the Semantic Web need access to data, but relationships among data should be made available, too, to create a Web of Data (as opposed to a sheer collection of datasets). This collection of interrelated datasets on the Web can also be referred to as Linked Data.

Resource Description Framework

Resource Description Framework (RDF)[11] is a description language, which aims to capture the semantic of linked data. The RDF standard is based on several fundamental concepts. The first such concept is "resource". Resources are the basic objects, or "things", that have to be described in the domain (e.g. rooms, floors, sensors etc.). All resources are identified through a unique, global identifier called the Universal Resource Identifier (URI). In most applications, the URI is the Uniform Resource Locator (URL) of a Web page, a part of a Web page or a link to a document available on a Web server. However, the URI is a more general concept, the only condition is that it uniquely identifies a resource. Examples of resources could be "unibs:florenzi.002" as well as "unibs:thesis1234" or "https://www.w3.org/RDF/". Another essential concept is "property". Properties describe relations between resources and, following the previous example, we could say that "has_author", "has_name", "has_subject" are properties. Having defined both resources and properties, statements can be derived. Statements, also known as triples, in RDF have the basic form: subject-predicate-object, where the subject is an RDF resource, the predicate is a RDF property, and the object is another RDF resource or a literal (a name, a number, a code, etc.). Still referring to the example, the statements could be "unibs:thesis1234 has_author unibs:florenzi.002", "unibs:thesis1234 has_subject `https://www.w3.org/RDF`" and "unibs:florenzi.002 has_name "Fabio Lorenzi"". RDF data models end up being represented as graphs where subjects and objects

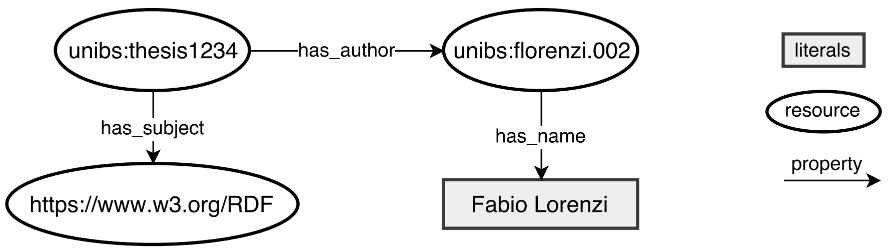


Figure 1.1: Graph representation of the example RDF model

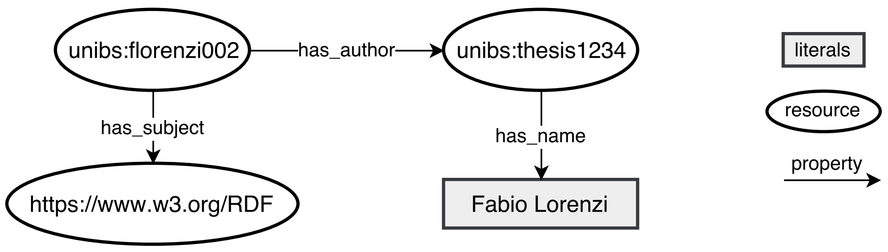


Figure 1.2: Correct RDF model with wrong semantic

are nodes while properties are edges. The resulting graph derived from the description above is shown in Figure 1.1. It is to note that RDF is a conceptual model for representing linked data, but as far as it goes, no semantic is implied in this representation. Looking at this model it is impossible to understand the meaning of the resources or properties, nor to understand if those elements reflect and adhere to rules of a particular domain. In the example above there is no explicit information telling us that the resource "unibs:florenzi.002" is a person and that "the unibs:thesis1234" is a book. Furthermore the RDF model represented with "unibs:florenzi.002 has_author unibs:thesis1234", "unibs:florenzi.002 has_subject `https://www.w3.org/RDF`" and "unibs:thesis1234 has_name "Fabio Lorenzi"", yelding the graph representation in Figure 1.2, is perfectly fine, altough not semantically correct[1]. Adding semantic to an RDF model, that is detailing the domain ontology, is left to other frameworks as explained in subsection 1.1.2.

[1]From a common sense point of view. No semantic is actually explicitly declared.

1.1.2 Vocabularies

On the Semantic Web, vocabularies define the concepts and relationships (also referred to as “terms”) used to describe and represent an area of concern. Vocabularies are used to classify the terms that can be used in a particular application, characterize possible relationships and define possible constraints on using those terms. There is no clear division between what is referred to as “vocabularies” and “ontologies”. The trend is to use the word “ontology” for more complex and possibly quite formal collection of terms, whereas “vocabulary” is used when such strict formalism is not necessarily used or only in a very loose sense. Vocabularies are the basic building blocks for inference techniques on the Semantic Web.

Resource Description Framework Schema

Resource Description Framework Schema (RDFS) defines a set of RDF resources useful for the description of other RDF resources and properties. Through RDFS it is possible to define and detail the semantic of a RDF model. Among the various concepts introduced with RDFS there are:

- **rdfs:Resource**: everything that is described in RDF is a resource
- **rdfs:Literal**: it is a text string
- **rdfs:Property**: it represents the properties
- **rdfs:Class**: it is similar to the concept of type or class in an object oriented programming language
- **rdfs:subClassOf**: it specifies inheritance, eventually multiple, between classes
- **rdfs:range**: it tells which resources are permitted as object in a statement
- **rdfs:domain**: it tells which resources are permitted as subject in a statement.

It is to note that the RDFS is a RDF model itself and it is used for explicitly declare the domain knowledge of a given RDF model. We refer to the RDFS as an ontology and to the RDF model triples as the instances of such ontology. Using

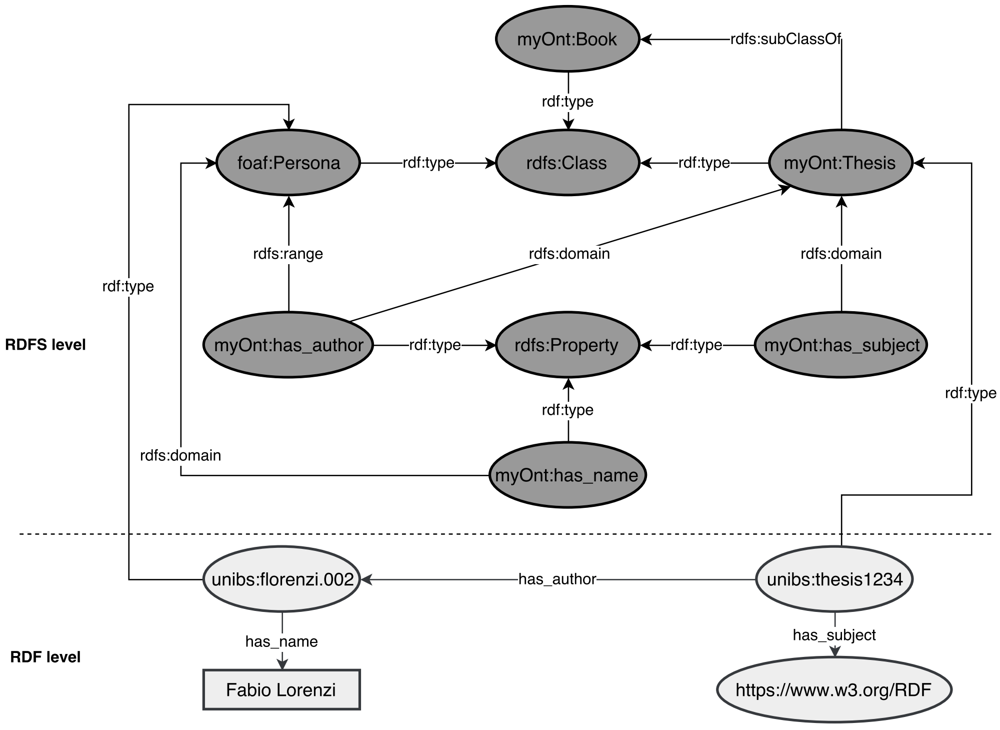


Figure 1.3: RDFS representation of an ontology

RDFS, it is possible to model domain knowledge for the example in **section 1.1.1** as in Figure 1.3. This "layering" approach is what allows for the design of reusable ontologies and gives opportunity to automatically infer new knowledge based on the RDFS model of the data. Still looking back at the example, lets suppose that the statement "unibs:florenzi.002 rdf:type foaf:Person" is not in the ground knowledge of the RDF model. Even if this explicit information wasn't available, a reasoner would have been able to infer that statement (dotted line) given the semantic description of the property "myOnt:has_author" (bold line). The process is represented in Figure 1.4.

1.1.3 Query

Since there exist technologies that represent knowledge bases and their semantics, it is useful, if not necessary, to have access to that knowledge through some kind of query language that allows the user to extract and consume that knowledge. If the knowledge base is stored as a RDF, the de-facto standard is represented by

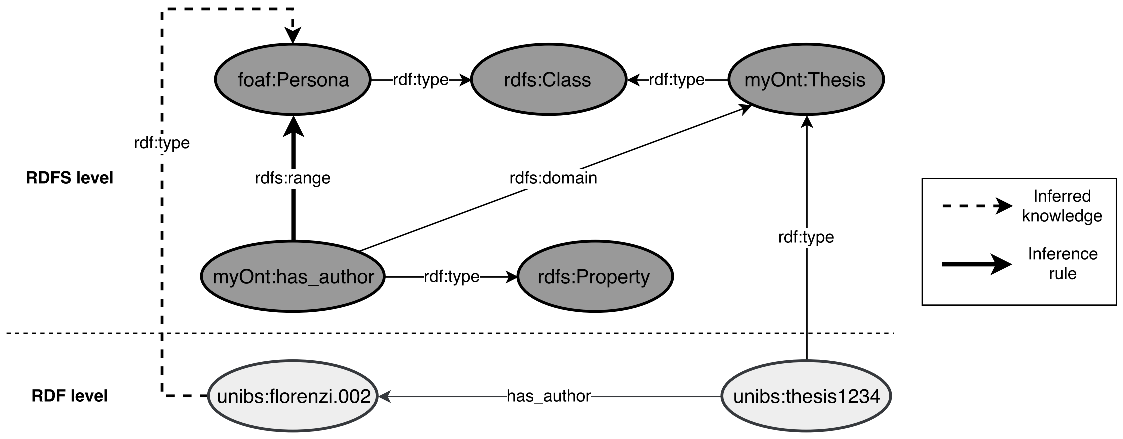


Figure 1.4: Inference process of new knowledge

SPARQL.

SPARQL Protocol and RDF Query Language

A mean for querying RDF models is defined by SPARQL [15] query language. SPARQL queries are based on patterns. As RDF can be seen as a set of relationships among resources, SPARQL queries provide one or more patterns against such relationships. A SPARQL engine would perform a pattern matching over the resources and it returns the matching triples. To represent a pattern (or query) to be run against the data model, SPARQL defines a series of keywords similar to those of SQL language:

- SELECT followed by the name of the variables that the user wishes to SELECT
- INSERT used to insert new triples pattern in the underlying RDF [defined in an extension called SPARQL/Update]
- WHERE followed by a set of conditions involving the variable that needs to be selected and that needs to be fulfilled in the underlying RDF.

As an example, from Figure 1.3, a standard query would be to retrieve every thesis and its author in the knwoledge base and it can be written as shown below.

```
SELECT ?thesis ?author
WHERE
  ?thesis a myOnt:Thesis
  ?thesis has_author ?author
```

1.2 Ontologies for smart buildings

After the brief (and not exhaustive) introduction to the core concept of the Semantic Web, here are introduced the two main ontologies that lie at the core of the project of this thesis. These ontologies are used to represent a smart building with all its relevant attributes (Brick) and the internal processes that take place involving the building sensors and physic variables (SSN and its extension). Both the ontologies are used since they are ortogonal one another and this help to cover every aspect of interest for a EMS application in a smart building.

1.2.1 Brick

Brick schema [1] defines domain specific concepts aimed to describe sensors in a building with its context. It is composed of three parts

- a class hierarchy of entities describing various building subsystems
- a minimal set of relationships for connecting entities
- a method for encapsulation in the form of Function Blocks.

The main concept in Brick is the concept of tag introduced with project Haystack [10], which is further enriched by an ontology that cristallizes the concepts defined by tags. A tag is a flexible framework for annotating metadata to building data points. In Brick tags are grouped together as sets, named tagsets, to represent entities; the tags room, temperature and sensor can be grouped together as room temperature sensor representing a single entity. The concept of tagset is well integrated with the class hierarchy concept of the ontologies: a room temperature sensor is a subclass of a generic temperature sensor. As the main goal of Brick is to represent points in a building's BMS, Points is one of the main classes of Brick,

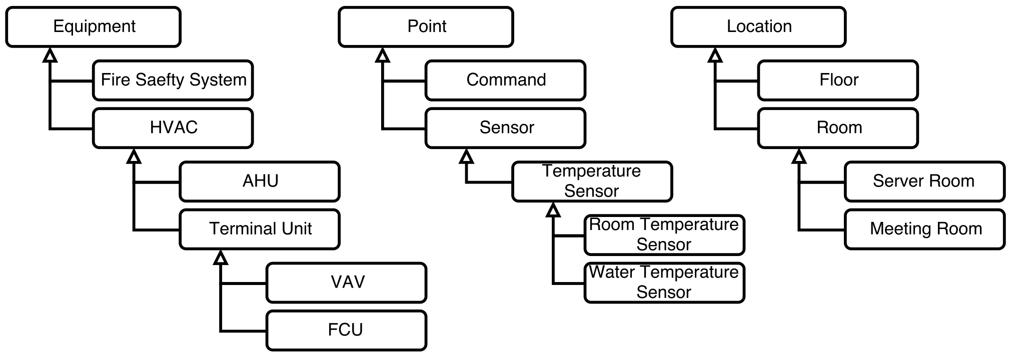


Figure 1.5: Subset of Brick hierarchy

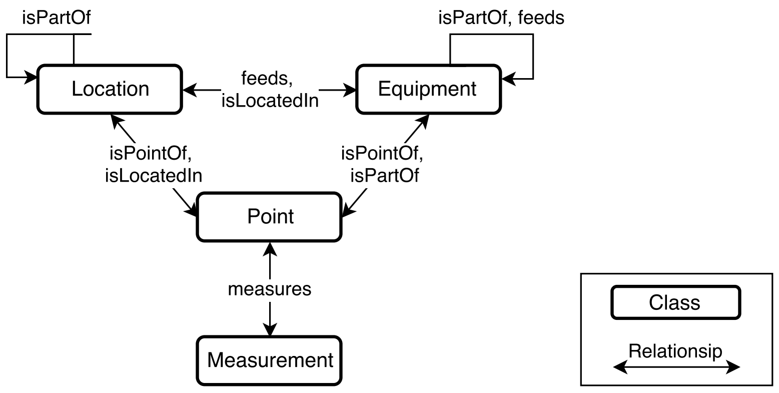


Figure 1.6: Brick schema core concepts

which subclasses define specific type of points like Sensor. The most common concepts a Point can be related to are Location, Equipement and Measurement so those classes, together with the Point class, form the core classes of Brick. Those classes are further subclassed to form a hierarchy. A subset of the hierarchy is shown, as an example, in Figure 1.5. this taxonomy is able to comprehend almost every entity that can be found in a BMS of a buidling[1]. Alongside classes, relationships play a crucial role in connecting entities and thus providing a context for many applications. Brick designes a set of minimal, intuitive and multipurpose relationship fundamental in capturing existing relationships in a real building. Figure 1.6 shows the Brick core concepts and the designed set of relationships.

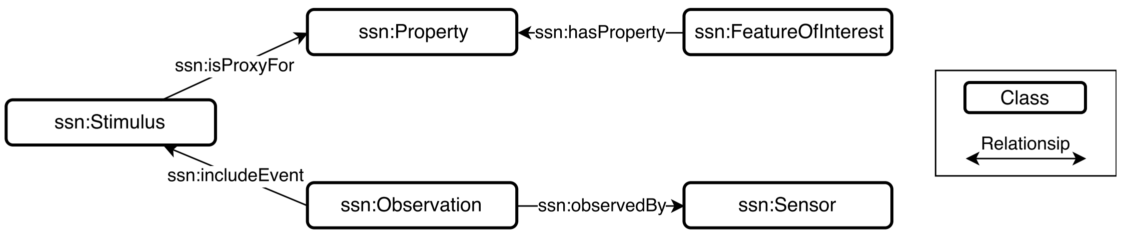


Figure 1.7: Semantic Sensor Network (simplified) ontology

1.2.2 Semantic Sensor Network ontology

While Brick schema is used to model the building domain, the Semantic Sensor Network (SSN) ontology [2] has been chosen because it provides a different view of the various aspect of the sensing process, beyond the specific building domain. Peculiar to the SNN Ontology (SSNO) is the Stimulus-Sensor-Observation design pattern. This pattern differentiates between the concept of ssn:Sensor for physical devices that take measurements in form of ssn:Observation and ssn:Stimulus that is the actual change in the enviroment that caused a particular observation, so that it is possible to differentiate that stimuli occur in the reality even if no sensor is monitoring them, thus modelling unobservable states of a system, and that observation made from a sensor can be different from the real event (e.g errors, failures, noise). These concepts aside, SSN provides other useful classes that are ssn:FeatureOfInterest that represents generic entities which status is to be monitored (e.g rooms, fans or floors) and ssn:Property that embodies specific physical quantities that a sensor ought to monitor. All the classes are connected by a set of relationships that are ssn:hasProperty that connects a ssn:FeatureOfInterest with meaningful physical properties, like a temperature in a room; ssn:isProxyFor represents that a given ssn:Stimulus influences a ssn:Property; ssn:includeEvent create a dependency between an observation and an event that causes it; ssn:observedBy links an observation to the sensor that could measure it. Figure 1.7 shows the resulting SSN ontology core.

Extending SSNO

As far as it goes, Brick and SSN are good ontologies for modelling a building and its sensor network, but in a Energy Management System (EMS) context, and for

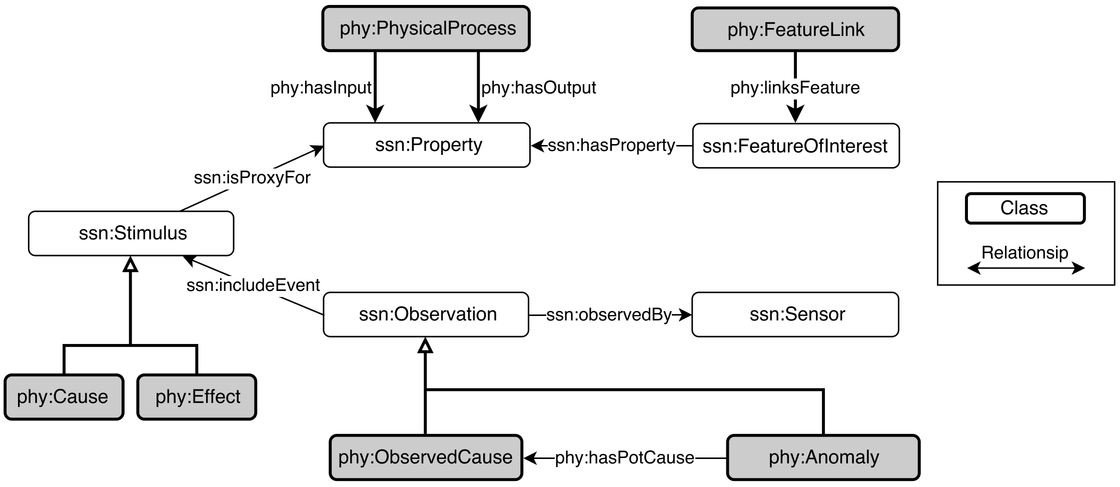


Figure 1.8: Extended SSN ontology. Extensions are highlighted with bold line and a light grey background.

FDD applications in particular, it is essential to understand the cause-effect relationships in the building. In order to do that, IBM Research extended the SSNO through the introduction of additional classes and references as in Figure 1.8. phy:PhysicalProcess represents laws of physics through a simplified model. Different subclasses of physical processes are derived from LTI system processes and will be described in chapter 2. phy:FeatureLink tells which kind of relationship exists between two ssn:FeatureOfInterest, such a link could be a wall between two rooms or a window between a room and the outside. phy:Cause and phy:Effect are subclasses of ssn:Stimulus useful to differentiate whether said stimulus is a cause or an effect of a ssn:Observation. phy:Anomaly is a subclass of ssn:Observation; it describes out-of-range observations that need to be diagnosed, like a high temperature in a room. phy:ObservedCause is still a subclass of ssn:Observation; it is used for describing a measurment that can be a cause of an anomaly.

Chapter 2

Model creation

This chapter will outline the approach suggested by IBM Reserch for tackling some of the challanges introduced in Introduction regarding fault detection and diagnosis (FDD) and for overcoming the limitations of the state of the art approaches. A step-by-step example illustrating the approach is provided at the end of the chapter.

2.1 State of the art

Nowadays, as extensively analyzed by Katipamula and Brambley [4], FDD approaches fall in either one of the following three categories: physycal model based approaches, data driven approaches and rule-based approaches. Still, each of these approaches shares common limitations that can be summarized as:

- they can only be applied to a single, specific building
- they require large manual effort and expertise
- their deployment can take several weeks
- they neglect strong interactions between systems.

Physycal model approach This approach requires a physycal model (e.g ordinary differential equations) of the building and its components. They are highly precise in diagnosing faults given a correct model. However, deriving the models is no trivial task and requires times and expertise. On top of this, derived models are building, system and location specific and they are difficult to adapt to other buildings than the one they are thought for, even if they share similar structure and similar components, thus limiting the scalability of this approach.

Data driven approach Data driven approaches completely rely on building's sensor data. They assume that access to a large dataset of hystorical data is granted. There is little to no need for any a priori knowledge of the processes involved. Black-box data driven approaches derive the model in the form of an input-output relationship whose parameters are not correlated with the actual physical parameter (e.g artificial neural networks, regression) while grey-box data driven approaches take advantage of simplified physical relationships between measured quantities (e.g principal component analysys) and rely on statistical methods for estimating their parameters. Even though these methods do not suffer from scalability issues, they are limited to fault detection and lack in diagnosis capabilities.

Rule-based approach The rule-based approach is the most common in traditional FDD applications. It uses domain knowledge and expertise in order to derive a series of simple *if-then-else* rules or some kind of decision trees along with a series of thresholds and confidence intervals; during system operations data are evaluated against this rules and countermeasures are eventually taken. Even though this is the most common approach, it still needs a lot of manual effort and requires access to domain knowledge so these requirements still limit portability and scalability of applications based on this technique.

2.2 IBM Research approach

The approach developed by IBM Research is based on a combination of the three aformentioned concepts to overcome their disadvantages. It models high level physical processes in the systems to derive diagnosis rules, it parametrizes the rules using data analytics techniques and applies them during system's operations.

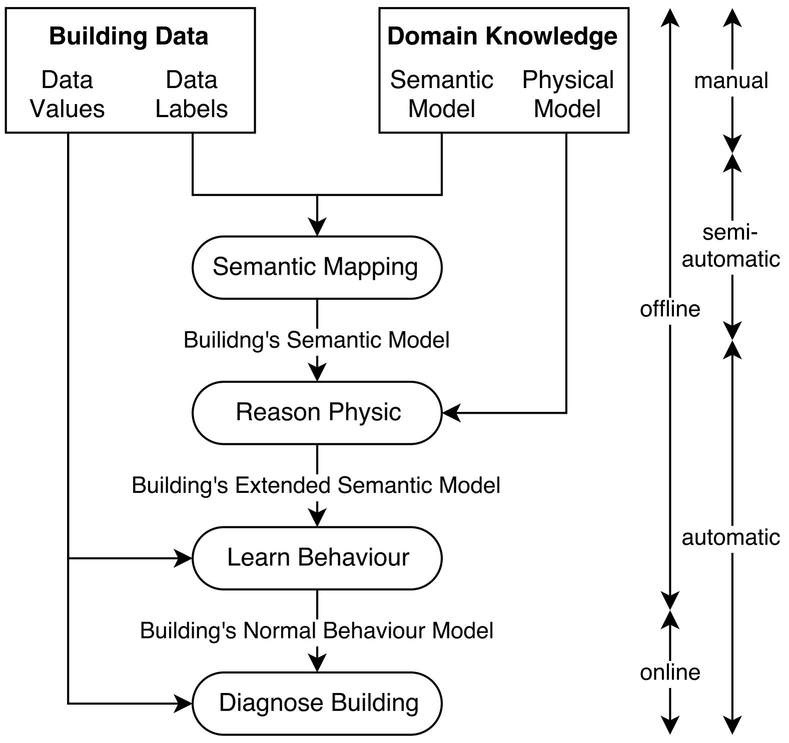


Figure 2.1: IBM Reserch approach, overview

The strength of the semantic approach lies in its capability to automatically infer knowledge given a building, its sensors and their data. This leads to a semi-automated process that limits the manual effort needed, as shown in Figure 2.1.

The proposed approach needs three inputs to produce the diagnosis of a given anomaly:

- building data: assumed available in the building's Building Management System (BMS)
- semantic model: specified through the use of concepts from domain ontologies Brick (subsection 1.2.1) and SSNO (subsection 1.2.2)
- physycal model: the model of the physical dependencies between subsystems of the building. The physical processes are modelled through concepts presented in the extended SSNO. (subsection 1.2.2).

These inputs are processed through the various stages to produce the complete model of the building and its internal processes. Details about the different phases are given in the following pages.

2.2.1 Semantic Mapping

Building a semantic model for a specific building needs knowledge on the type of sensors, meters and various equipment available in said building, their location and their semantic meaning in the chosen ontologies. These informations are usually stored in BMS through labels that, despite some kind of standardization efforts, are usually vendor specific; in some cases even the same vendor ends up changing its labelling schemes throughout the years. Even though there is no common ground to evaluate these labels, usually they are comprised of a series of abbreviations and acronyms, eventually separated by some special character (e.g "_") that gives informations about the device's ID, function and location (or the equipment it is part of). An air temperature sensor in a room on ground floor can be labeled AIR_TEMP_R3GF while another air temperature sensor in a room on the first floor may ends up being labeled as RATSF1R9; both the sensors represent a common semantic resource in an ontology and need to be mapped to that concept. This process is called semantic mapping. Existing approaches are differentiated in three types[13]:

- semi-automatic: they offer a tool assisting the user in labelling the point to corresponding semantic types and are based on advanced text mining techniques (e.g regular expressions, classifiers)
- data driven: these approaches try to recover the meta-data given the time-series. They are based on the concept that different data-points exhibiting similar timeseries behaviour should be similar themselves
- active feedback: in these kind of approaches a series of known events are injected into the system and datapoints semantic model is derived observing the effect of the event injection.

Building Energy Asset Discovery tool

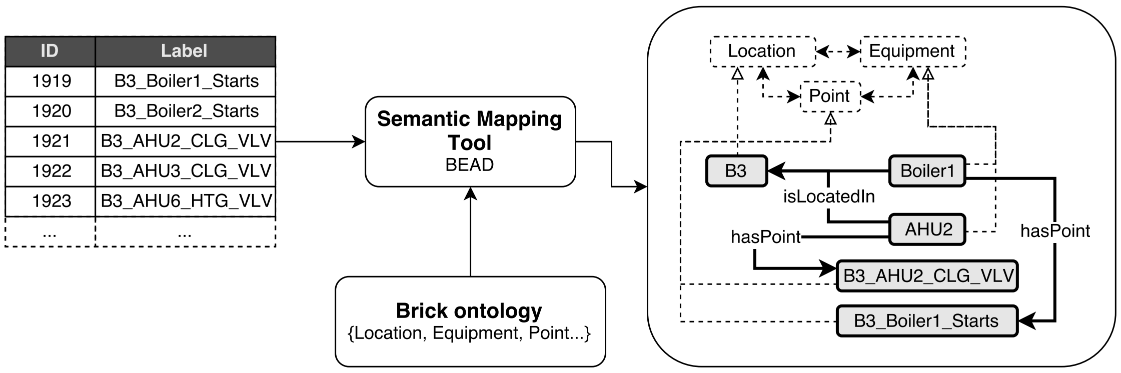
the Building Energy Asset Discovery (BEAD) tool[7] is the tool developed by IBM Research for assisting user in the process of discovery and tagging of sensors. The tool is based on dictionaries of the type $\mathcal{D} : \mathcal{A} \rightarrow \mathcal{MS}$ where $\mathcal{A}$ is the set of all relevant acronyms and $\mathcal{MS}$ is the set of markerset; each markerset is a set of markers or keywords. These concepts are closely related to those of tagset and tag found in the Brick ontology and this duality allows the association of the sensor with the correct semantic type as described by Brick (see subsection 1.2.1). The dictionary contains the most common acronyms found in BMS and it is further extended by the markers (tags) from the Brick ontology, such that a tag maps to itself, e.g Temperature→{Temperature}. The tool takes a label from the BMS, computes a similarity score against the dictionary entries and guides the user in the labelling process. For example, given the following dictionary

d1: RAT→{Return, Air, Temperature}
d2: RAT→{Room, Air, Temperature}
d3: SAT→{Supply, Air, Temperature}
d4: OAT→{Outdoor, Air, Temperature}
d5: Temp→{Temperature}
d6: Temperature→{Temperature}

the label RATSF1R9 (Room Air Temperature Sensor, Floor 1, Room 9) has a high similarity with both RAT→{Return, Air, Temperature} and RAT→{Room, Air, Temperature}, so it is up to the user to understand the meaning and choose the right markerset. In a similar fashion, it is possible to extract information on the location of a sensor or the asset it is related to, as shown in Table 2.1. This process clearly requires human supervision, hence the definition of semi-automatic, but through this approach the operator has to manually inspect a smaller subset of all the possible marketsets, thus taking minutes instead of several weeks to complete the semantic mapping. Practical experiments demonstrated that through this approach the decrease in number of labels to analyze is the 7.5% of the total number of BMS labels [13]. The output of this process is the semantic model of the specific building according to the chosen ontology. A flowchart illustrating this process is shown in Figure 2.2.

Table 2.1: Dictionary extended with assets' informations

Label	**Asset**	**Marketset**
U6_RAT	AHU6	{Return, Air, Temperature}
U6_DAT	AHU6	{Discharge, Air, Temperature}
AHU7_RAT	AHU7	{Return, Air, Temperature}
AHU7_SAT	AHU7	{Supply, Air, Temperature}
AU9_RET_TEMP	AHU9	{Return, Air, Temperature}
AU9_SUP_TEMP	AHU9	{Supply, Air, Temperature}



ID	Label
1919	B3_Boiler1_Starts
1920	B3_Boiler2_Starts
1921	B3_AHU2_CLG_VLV
1922	B3_AHU3_CLG_VLV
1923	B3_AHU6_HTG_VLV
...	...

Figure 2.2: Semantic mapping process

2.2.2 Physic model inference

The semantic model obtained from the semantic mapping phase is a model of the building architecture, assets, sensors and their loaction across the building. The model still lacks information about the physic processes taking place in the building and how the actual physic model is monitored and influenced by the points (sensors, setpoints etc.) in the building. Manually building the physical model of a building is unpractical and costly, thus the developed framework approaches this challange through the use of semantic inference techniques. This phase requires as inputs the semantic model of the building and an appropriate ontology that models the physics variables. Given those inputs, through the reasoning process is possible to automatically add the correct dependencies and physical processes to the model. This process is separated in two phases:

- physical properties inference: it recreates the physical variables that are involved in physical processes. The properties are either mandatory or optional
- physical processes inference: once the physical properties are set in place, it is possible to link them together according to the domain knowledge specified in the upper ontology.

Further informations about the concept of reasoning and reasoner and further details about implementation choices for this approach are given in chapter 4.

Physical properties inference Properties represent physical quantities in the real world and can be mandatory or optional. Mandatory properties are characteristic of a ssn:FeatureOfInterest and need to be created for each instance of that feature. This behaviour is specified in the ontology of the physic through the use of the phy:requiresProperty relationship, so that during the reasoning mandatory properties can be automatically created. For example, in Figure 2.3 it is shown that the concept of :Room phy:requiresProperty :Temperature, that means that every instance of a :Room needs to be associated with a property that is an instance of :Temperature, or in plain English that every room has a temperature. Optional properties, on the other hand, need to be created only if some requirements are met, for example a :Room have a :Cooling property only if a

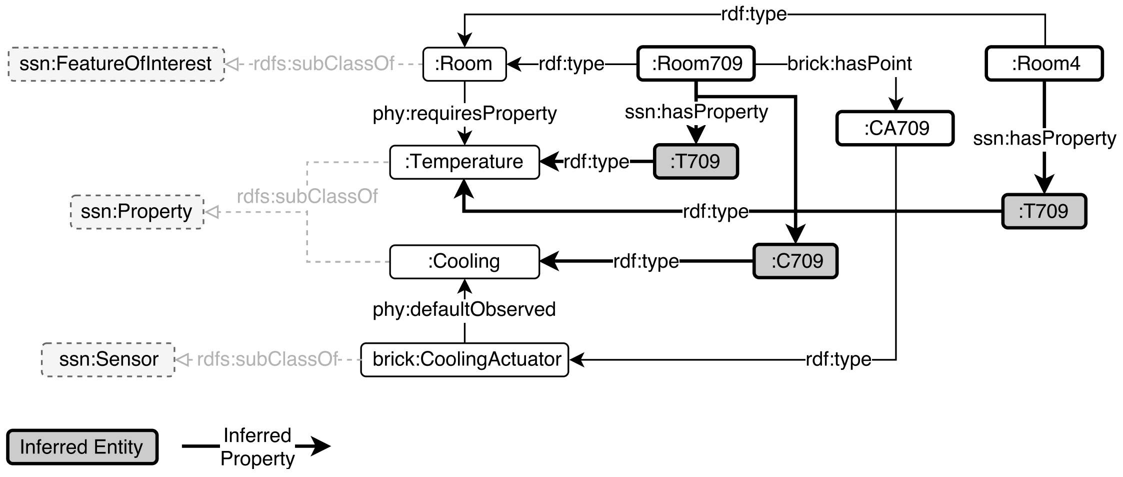


Figure 2.3: Process of inference of mandatory and optional properties

:CoolingActuator is present in said room, and it is modelled in the ontology as :CoolingActuator phy:defaultObserved :Cooling. In Figure 2.3 it can be seen that only :Room709 has a :Cooling property since it is the only room that has a :CoolingActuator. The :requiresProperty and :defaultObserved properties[1] are concept level references and model general knowledge. General knowledge describes the domain without bothering about the ground truth. This means, for example, that the concept of :Room requires the existance of the concept of :Energy even if there are no rooms instances in the building's model. The existence of said properties is the key for the automatic reasoning process.

Physical processes inference Once all the actual properties have been infered for a specific configuration, it is time to append to the model informations about the physical processes involving such properties. As already stated in subsection 1.2.2, physical processes are modelled as Linear Time Invariant (LTI) systems based on the fact that Multiple Input Multiple Output (MIMO) processes can be

[1]Intended as the semantic concept of relationship between entities, not to be confused with domain knwoledge properties like :Temperature.

Table 2.2: Taxonomy of the semantic representation of LTI processes

Semantic concept	**Parent**	**Formal definition**
MISO	Process	$y_i(t) = \boldsymbol{c_i} \cdot \boldsymbol{x_i} + \boldsymbol{d_i} \cdot \boldsymbol{u_i}(t) = f(\boldsymbol{u}(t))$
SISO	MISO	$y_i(t) = \boldsymbol{c_i} \cdot \boldsymbol{x_i} + d_i \cdot u_i(t) = f(u(t))$
Positive Correlted (PC)	SISO	$y(t) \propto u(t)$
Negative Correlated (NC)	SISO	$y(t) \propto -u(t)$
Proportional (P)	SISO	$y(t) = k_p \cdot u(t)$
Positive Proportional (PP)	PC, P	$y(t) = k_p \cdot u(t), k_p \geq 0$
Negative Proportional (NP)	NC, P	$y(t) = k_p \cdot u(t), k_p < 0$
Negation (N)	NP	$y(t) = k_p \cdot u(t), k_p = -1$
Integral (I)	P	$y(t) = k_I \int u(t)dt$
Derivative (D)	P	$y(t) = k_D \cdot \dot{u}(t)$
Lag (PTn)	P	$\sum_{i=0}^{n} C_i \cdot T^i \cdot y^{(i)}(t) = k_{PT} \cdot u(t)$
1st order lag (PT1)	PTn	$T \cdot \dot{y}(t) + y(t) = k_{PT} \cdot u(t)$
Delay (τ)	SISO	$y(t) = u(t - \tau), \tau > 0$
Multiplicative (M)	MISO	$y(t) = u_1(t) \cdot u_2(t)$

decomposed into multiple Multiple Input Single Output (MISO) processes

$$\dot{x} = A \cdot x + B \cdot u$$
$$y = C \cdot x + D \cdot u$$

that splitted on the output rows and in case of indipendent inputs becomes

$$y_i = \sum_{j \in J} \left[\sum_{l \in L_j} c_{i,l} \cdot x_l + \sum_{k \in K_j} d_{i,k} \cdot u_k \right] = \sum_{j \in J} f_{i,j}(u_j) \tag{2.1}$$

A MISO process can be further subclassed as a Single Input Single Output process. Each process follows the rules

Rule 1 a process has at least one input and exactly one output
Rule 2 when a process has multiple inputs, they superpose additively.

Given these rules, it is possible to define a class hierarchy of LTI processes that are used to model a physical system. A subset of said taxonomy is available at Table 2.2.

2.2.3 Learning the normal behaviour

This phase is not strictly related to the ones detailed in the previous sections in this chapter, yet it is of utmost importance for the diagnosis process. This phase allows for the definition of general rules that represent a confidence interval of the normal behaviour of the sensors. These rules are derived from different statistical and machine learning methods. Ploennigs and Schumann [6] suggest the use of Artificial Neural Networks (ANN) for modelling the temperature comfort in a room. Generalized Addictive Models (GAM) can be used for modelling energy processes so that a subsequent Autoregressive Moving Avarage (ARMA) model can characterize the normal behaviour[8]. The approach determines a series of upper and lower bounds that characterize the normality interval for a given sensor. Behaviours outside these ranges are classified as anomaly and they trigger the diagnosis phase. Studying this methods is outside the scope of the thesis and so these confidence intervals are assumed to be known without bothering about how they were derived.

2.3 Example of the whole approach

In order to better explain the concepts presented in the previous sections, it is provided an example of the full process applied to a room, facing the outside environment, with a cooling Air Handling Unit (AHU). At the end of the section it is shown that a full model of the room and its internal and external processes are derived. This model will then be used during the explanation of the diagnosis process. Looking back at Figure 2.1, we start by defining the inputs needed for the process (sharp top boxes).

Building data and semantic model It is assumed that some kind of BMS is available in the building so that through the BMS is it possible to retrieve both labels (for semantic mapping) and data of various sensors. The room analyzed in the example will be named ROOM1. In ROOM1 an air temperature sensor and an occupancy sensor are available. Room1 is also equipped with an AHU, named AHU709, whose internal components are observed by a fan mode sensor, a supply

Table 2.3: Example of BMS content

ID	**Label**
1234	CCPC1B3
1235	CWFU709
1236	CWTU709
1237	SAFAHU709C1
1238	FMAHU709
1239	OCCR1
1240	ATSROOM1
1241	ATS_Outside

air flow sensor a coil water temperature sensor and a coil water flow sensor. This AHU709 is fed by a chiller named C1 equipped with a power sensor. Let the BMS contain the data as in Table 2.3 and lets use Brick (1.2.1) as a domain ontology. Through the semantic mapping tool (subsection 2.2.1) it is possible to map the labels in the BMS to the Semantic type in the Brick ontology. The user can also give the system additional informations about the architecture of the building and the location of equipments and sensors. It is then possible to map :B3 as a brick:Location, :ROOM1 still as brick:Location and then link one to the other as :ROOM1 brick:isPartOf :B3 and so on. The tool will also discover the semantic type of the sensor and will be able to create the correct instances. For example it will add to the model an air temperature sensor named :ATSROOM1 and will link it as a point to :ROOM1. The output semantic graph can be seen in Figure 2.4 where the instances created from the BMS and their mutual relationships are bold, while the Brick ontology concepts and relationships are light grey. This is the building semantic model, that will be used together with the physic model described in the next paragraph to infer knowledge about the physics properties and relationships in the building.

Physic model The physic model is a model of the physic phenomena occuring in an enviroment. It defines physical variables and processes and their interaction in the context of a specific domain. Creating a model of physic's laws is a time consuming task that involves expertise from different fields, from engineering to physics, that are very domain specific. The good side of the semantic approach is

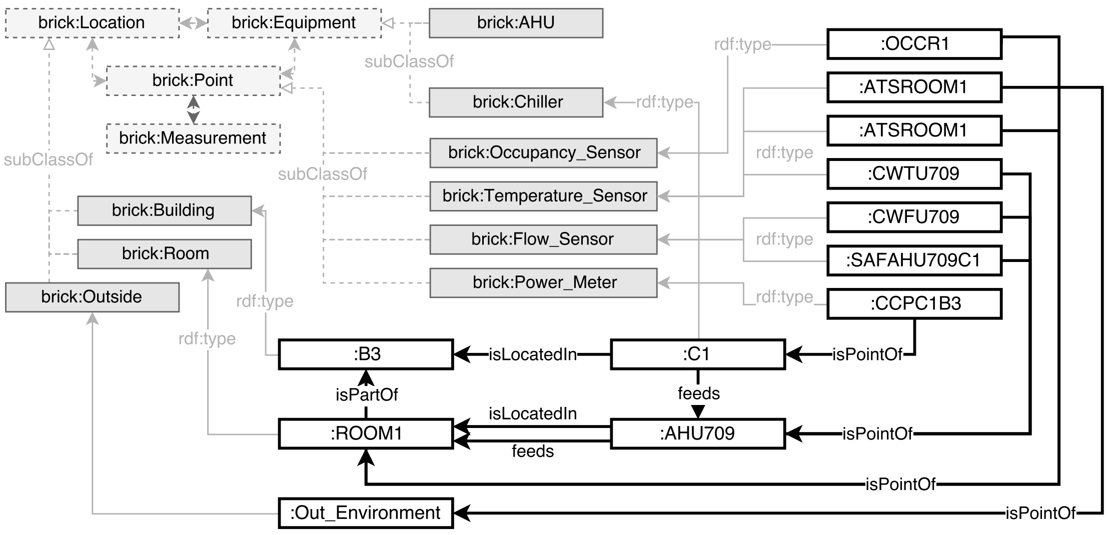


Figure 2.4: Complete semantic model of a room in a building equipped with an AHU according to Brick ontology

that, once the model is derived from a general point of view, it is automatically extended for every instance of a building, indipendently from its architecture. It is to note that the domain model for the physic does not necessarily model exact physical relationships but just the principal ones, since those are the most relevant to the approach. However this choice is purely practical rather then a limitation of the approach itself, and given sufficient time and knowledge, it is possible to enrich the model in order to close the gap with the reality. Here will be presented a model for the physic of a room, equipped with a AHU fed by a chiller. The model is based on a simplification that makes the assumption of fully mixed air, that allows for the modelling of a whole space as a single node with uniform temperature. Wetter [17] provides an insight of the physics involved in the example. It starts describing a heat balance equation of a room as

$$m_r c_p \dot{\vartheta}_r = Q_{Energy} \tag{2.2}$$

$$Q_{Energy} = Q_{Occ} + Q_{Cool} + \sum A_i h_i (\vartheta_i - \vartheta_r) \tag{2.3}$$

$$\tag{2.4}$$

and from Equation 2.2 and Equation 2.3 it derives

$$m_r c_p \dot{\vartheta}_r + \sum A_i h_i \vartheta_r = Q_{Occ} + Q_{Cool} + \sum A_i h_i \vartheta_i \tag{2.5}$$

that says that the temperature of a room ϑ_r is a first order derivative of the room's inner energy Q_{Energy} that is the combination of the heat gain of occupants Q_{Occ}, the heat removed by the cooling system Q_{Cool} and the heat transfer between adjacent environments. Given these equations, semantic informations can be extracted. From Equation 2.2 it can be seen that

1. rooms have an internal energy [Property, mandatory]
2. rooms have a temperature [Property, mandatory]

while from Equation 2.3 it follows that

3. a room's temperature is influenced by the energy in a first order lag process [Process, PT1]
4. rooms can be cooled, if they have a cooling actuator [Property, optional]
5. cooling is negative proportional (NP) with the room's energy
6. room energy depends from the adjacents enviroment's temperature proportional to the shared surface A_i and the heat transfer coefficient h_i, hence the dependency is a positive proportional process [Process, PP].

Further analyzing the physics involved bring to the conclusion that

$$Q_{Occ} = q_{lat} n_{occ} \tag{2.6}$$

where q_{lat} is the latent heat gain and n_{occ} the number of occupants in the room. From Equation 2.3 and Equation 2.6 it follows that

7. rooms have a number of occupants [Property, Mandatory]
8. a room's internal energy is positively correlated with the number of occupants [Process, PP].

Additional knowledge can be added if the physics of the system in known; for example, it is possible to further enrich the model by specifying the internal processes that regulate a AHU. Wang et al. [16] provide a simplified model for modelling the cooling load of an AHU through Equation 2.7

$$Q_{Cool} = \frac{c_a \dot{m}_a^e c_c \dot{m}_c^e}{c_a \dot{m}_a^e + c_c \dot{m}_c^e} (\vartheta_a - \vartheta_c) \tag{2.7}$$

This, given that ϑ_a is the mixed air temperature, $\dot{m}_a$ the air flow rate, ϑ_c the chilled water temperature and $\dot{m}_c$ the chilled water flow rate, leads to the following conclusions

9. cooling is a complex function of the air temperature, the chilled water temperature, the supply air flow and the water flow in the coil [Process, MISO].

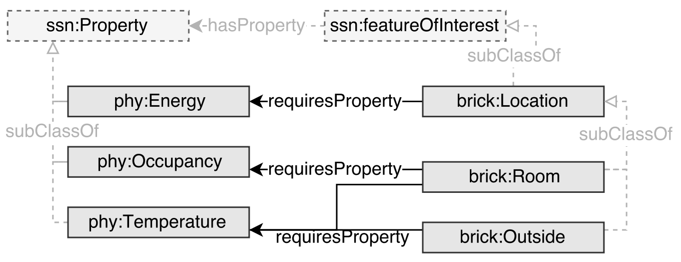
Finally, Cui and Wang [3] provide a simple model of a chiller's physics, whose equation can be expressed as

$$\dot{m}_c c_w (\vartheta_{cret} - \vartheta_c) = \eta_{COP} W \tag{2.8}$$

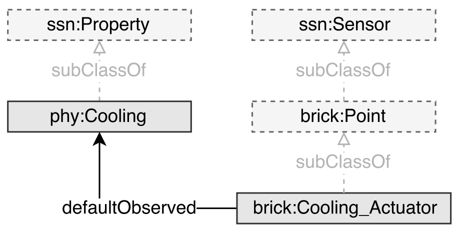
from which it can be derived that

1. the chilled water tempearture decreases with the increase of the power consumption [Process, NP]
2. the water flow rate increases as the power consumption of the pump increases [Process, PP].

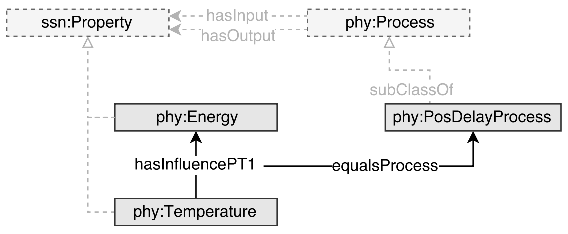
This dependencies can be semantically mapped to a domain ontology that still use concept from Brick, SSN and its extended version and Figure 2.5 shows how the discovered domain knowledge can be represented using those ontologies. It is clear that this part is the most costly of the whole process, especially timewise, since it requires very specific expertise. In the example, for instance, it is not captured that a room's internal energy is influenced by the radiant heat of the sun but it would have been possible to do so. The good side of this approach is that the modelling efforts need to be done just once per domain as the ontology is reusable and will adjust itself to the bulding instance trough the automatic reasoning process.



(a) Mandatory properties



(b) Optional properties



(c) Process

Figure 2.5: Patterns for definition of a model for a system's physics

Reasoning process The reasoning process is divided in two main phases: the discovery of the physical properties and the creation of the physical processes involving those properties. This requires access to the building semantic model created during the semantic mapping phase and the physic model derived by expert for the particular domain (e.g physic model of HVAC system). Both the phases work on the ontology (metadata) to draw conclusions about the instances. The first phase has to be the properties discovery one, since those properties are inputs and outputs for the processes. Based on the ontology derived in the previous paragraph and the semantic model of the building (Figure 2.4), through the application of simple reasoning rules, it is possible to derive the whole semantic graph of both the building and its actual physic beahviour, as seen in Figure 2.6, in a simplified form[2]. The rules to be applied are:

- if there is a room, it has an internal energy, $Room(r) \Rightarrow Energy(e) \wedge hasProperty(r, e)$
- if there is a room, it has a temperature, $Room(r) \Rightarrow Temperature(t) \wedge hasProperty(r, t)$

[2]Since even the degree of complexity of the semantic graph of small example is too high, the fidelity to the real graph is dropped for clairty's sake.

- if there is a temperature sensor in a room, the room has a temperature and the temperature is observed by the sensor, $Room(r) \wedge Temperature\_Sensor(ts) \wedge hasPoint(r, ts) \Rightarrow Temperature(t) \wedge hasProperty(r, t) \wedge observes(s, t)$

- energy influences the temperature through a PT1 process, $Room(r) \wedge Energy(e) \wedge hasProperty(r, e) \wedge Temperature(t) \wedge hasProperty(r, t) \Rightarrow PT1(p) \wedge hasInput(p, e) \wedge hasOutput(p, t)$

- energy of a room depends on the outside temperature if they are adjacent, $Room(r) \wedge Outside(o) \wedge isNextTo(r, o) \wedge Temperature(t) \wedge hasProperty(o, t) \wedge Energy(e) \wedge hasProperty(r, e) \Rightarrow PP(p) \wedge hasInput(p, t) \wedge hasOutput(p, e)$

- if a AHU has a coil water flow sensor then it has a coil water flow property, $AHU(a) \wedge CoilWaterFlowSensor(s) \wedge hasPoint(a, s) \Rightarrow CoilWaterFlow(f) \wedge hasProperty(a, f) \wedge observes(s, f)$.

The reasoning rules can be completed for the remaining AHU's components reusing similar patterns as those shown in the list above. These inference rules are explicitly written as implications among instances (no ontological reasoning is involved) because it is easier to convey the concept this way and, since this example is really small, it was easy and fast enough to be done by hand. Even a slightly more complex example would have required a more refined reasoning scheme involving the concepts, the instances and their mutual relationships. However it is already possible to get a grasp of the portability of this approach just noticing that, just with these simple rules, it does not matter how many instances of rooms, AHUs or chillers there can be. The reasoner would still enforce the correct properties in an automatic fashion. Looking back at the examples in this chapter, it can be noticed that, even in a small example, there are many informations encoded in the graph that starts to grow in size and complexity. Therefore it starts to surface one of the challanges of this approach, addressed in chapter 4, that is the scalability issue for big commercial buildings.

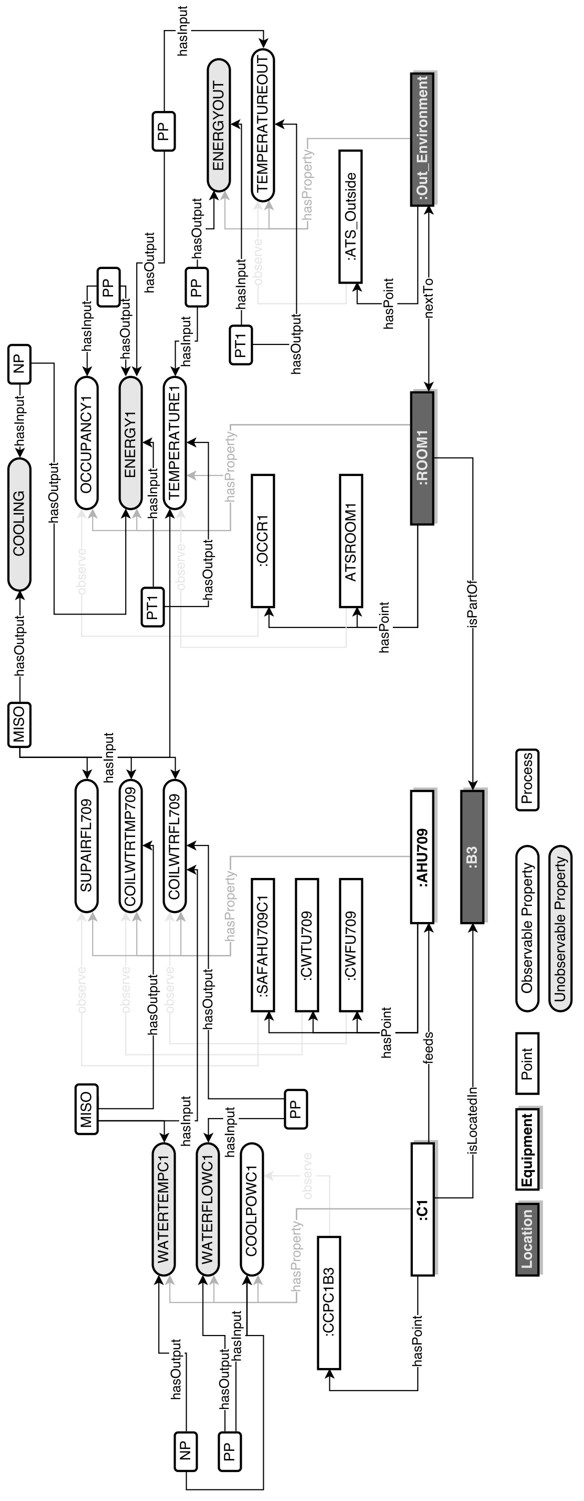


Figure 2.6: Semantic model of the building and its physics

Chapter 3

Diagnosis Algorithm

In this chapter will be outlined the diagnosis algorithm[9] developed by IBM Research with the collaboration of Michael Maghella, a fellow student at Università degli Studi di Brescia. This algorithm is based on a novel reputation-based approach that makes use of the informations available in the semantic graph to identify cause-effect relationships and use these relationships to isolate relevant causes basing the diagnosis on the time series data.

3.1 General description

The diagnosis activity is comprised of three main steps:

- fault detection, that is the discovery of the fault. A diagnoser needs to be able to tell apart the normal behaviours from faulty ones. This is done, in this approach, through the rules discovered while learning the normal behaviour of a building (see subsection 2.2.3). Whenever a fault is detected the diagnosis process ought to start.

- fault isolation, that is the discovery of the causes of the fault. It is a key step for enabling a precise diagnosis of the problem. In this approach informations derived from the semantic graph are used to narrow down the set of possible faults to the most relevant ones.

- fault identification, that is to determine with some confidence which are

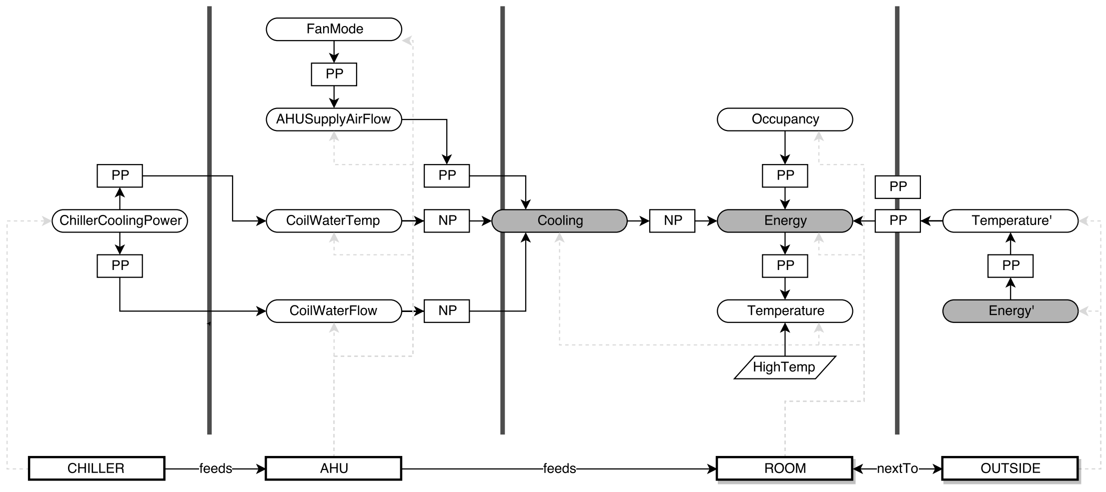


Figure 3.1: Internalized model of the example building

the actual causes of a fault among all the possible causes. The proposed diagnoser implements a voting scheme to identify these causes.

While explaining the diagnosis approach, it will be used the same model presented in chapter 2 (Figure 2.6), that will be represented in a compact notation shown in Figure 3.1, where every component is an instance of a concept with the same name (internalized representation). Moreover, for the sake of simplicity, the physic model of the system is simplified to a more lax one, dropping the MISO processes and the PT1 ones in favor of the PP and NP ones. The diagnoser is run every time an anomaly is detected in the time series. An anomaly is intended as a deviation from the normal behaviour clearly detectable by some preset upper and lower bounds, therefore an anomaly can be classified as High or Low. It is assumed that an anomaly is symptom of some kind of fault in the system. The algorithm starts, as soon as the anomaly is detected, by identifying the semantic type of the anomaly and the potential causes in the graph. It proceeds computing the voting vector and, eventually, it is recursively executed for all those properties that get a voting score equals or greater then one. When no other causes with a high score are left, the algorithm terminates and produces a list of the detected causes along with the name of the faulty properties, the semantic type of the faults and the estimatation, expressed in percentage, of the responsability of those causes towards the diagnosed anomaly. The relative pseudocode is in Algorithm 1.

Algorithm 1 General diagnosis algorithm

```
Require:
    the anomaly $A$,
    the semantic graph $\mathcal{G}$
Ensure: a set $C_A$ of possible causes of the anomaly
 1: procedure diagnoseAnomaly($A, \mathcal{G}$)
 2:     $T \leftarrow$ getSemanticTypeOfAnomaly($A$)
 3:     $PC_A \leftarrow$ getPossibleCausesInGraph($A, T, \mathcal{G}$)
 4:     $C_A \leftarrow \emptyset$
 5:     $\boldsymbol{v} \leftarrow$ computeReputationVoteVector($A, T, PC_A$)
 6:     for all $p \in PC_A$ do
 7:         if $\boldsymbol{v}[p] \geq 1$ then
 8:             $C_p \leftarrow$ diagnoseAnomaly($p, \mathcal{G}$)          ▷ diagnose subtree
 9:             if $C_p \neq \emptyset$ then
10:                 $C_A \leftarrow C_A \cup C_p$
11:             else
12:                 $C_A \leftarrow C_A \cup \{p\}$                    ▷ $p$ is a root cause
13:             end if
14:         end if
15:     end for
16: end procedure
```

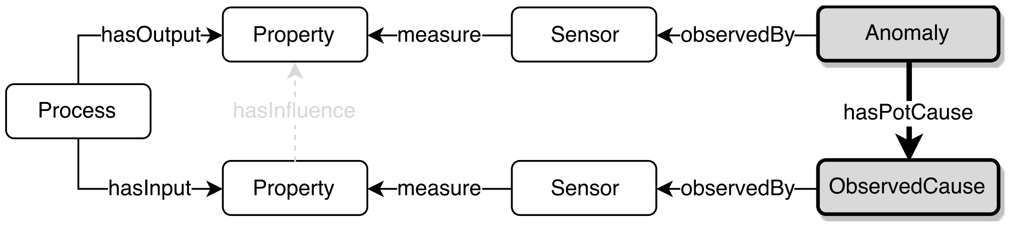
3.2 Potential causes

The first non-trivial operation of the algorithm is the identification of the possible causes of a given anomaly. A potential cause is an observation made by a sensor that observes a property and such that the following hold:

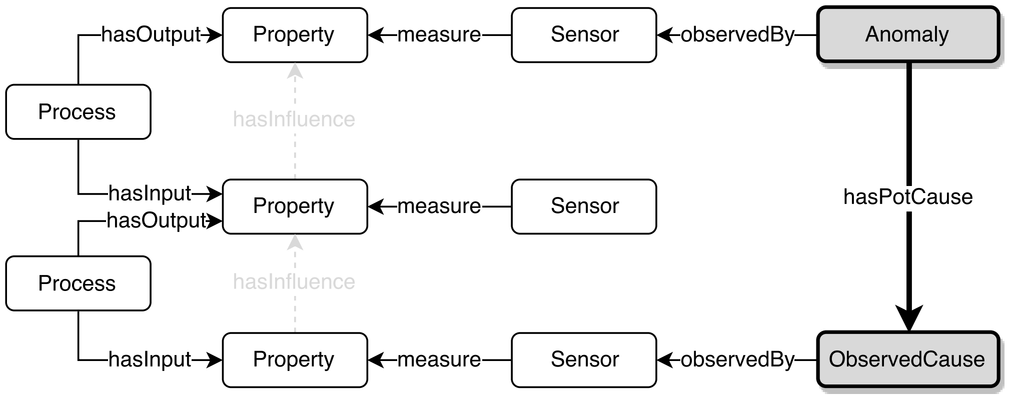
1. the property observed by the sensor is an input of a process that directly outputs to the anomaly, or it is connected to the anomalous property by a chain of unobservable properties

2. it is observable.

The properties and their relationships with the anomaly are all encoded in the semantic graph and are therefore accessible via reasoning. The approach is also interested in the semantic type of the anomaly and of the possible causes. Retrieving the possible causes of an anomaly is possible because in the graph anomalies are observations made by a sensor that measures a property. Figure 3.2 shows the possible ways to retrieve a potential cause of a property. In both graphs the situation respects the conditions given earlier regarding potential causes. The causes retrieved at this point are the ones that are correlated at various degrees to the anomaly and are the only ones that can help to produce a diagnosis. At this point it is important to preserve the semantic informations about the processes involved since these informations are at the base of the voting algorithm. When talking about "semantic informations" of the process it is meant the replacement of a chain of properties with a single, direct process whose type is infered from the ones of the chain. For example it can be seen in Figure 3.3 that the temperature in a room is influenced by the occupancy property of the room through a chain of a PT1 process and a PP process, both involving the unobservable energy property. This means that the temperature is influenced by the occupancy through a direct PT1 process. Table 3.1 shows the possible combinations for the most common type of processes used that are the positive correlation, the negative correlation and the lag processes. Given these new processes that directly correlate an anomaly to its potential causes, it is possible to infer the entity of the potential cause. This information is computed thanks to the intuition that the type of process correlating two observation carries qualitative information about the output. It is possible to



(a) Direct influence



(b) Chain of unobservables properties

Figure 3.2: Detection of potential causes

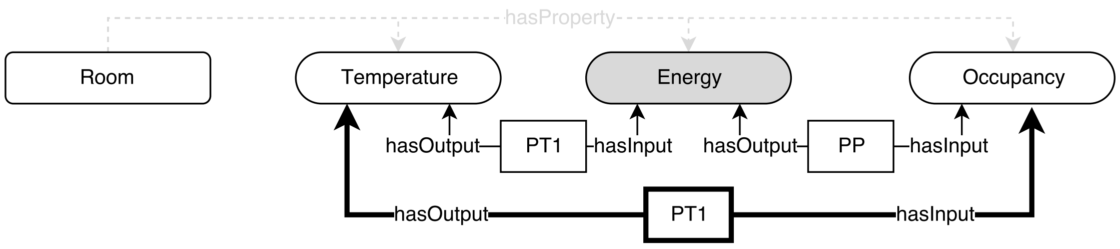


Figure 3.3: Process chain composition

Table 3.1: Derived processes from most common process chains

First process	**Second process**	**Derived process**
Positive correlated	Positive correlated	Positive correlated
Positive correlated	Negative correlated	Negative correlated
Negative correlated	Positive correlated	Negative correlated
Negative correlated	Negative correlated	Positive correlated
Any	Lag	Lag

Table 3.2: Predicted value of an observation given the influencing process and its input

Anomaly type	Process type	Cause type
High	Positive correlated	High
High	Negative correlated	Low
Low	Positive correlated	High
Low	Negative correlated	Low
Any	Lag	Lag

predict that an observation has to be high if it is influenced by a positive correlation process whose input is an observation that is high itself. The ontology defines the combinations listed in Table 3.2. Thanks to those informations it is possible to implement the fault identification capablities of the algorithm.

3.3 Fault identification

The whole approach is based on the intuitive assumption that an anomaly is caused by faults and that the measurement of the behaviour of such faults needs to be anomalous too. Moreover this anomalous behaviour needs to be semantically coherent with the behaviour predicted from the chain of process that links the anomaly to the cause. These two principles are at the base of the voting algorithm. The algorithm works by computing, from the set of anomaly-free time series, a world view (that is a subset of the historical data for each time series) where the property that presents an anomaly would be anomaly-free but close in range to the anomalous value. This can be done by selecting all the anomaly-free data of the sensor that observes the fault; these values are splitted in deciles and the decile closest to the anomalous value is chosen, that is the first decile if the anomaly is of type Low or the tenth if it is of type High. For every potential cause are then extracted the samples measured at the same timestamp as the samples of the anomalous property that fall in the chosen decile. Given this subset $S^{'} \subset S$, each potential cause will derive a second subset $S^{''}_p \subset S^{'}$ which represents the view of the system from the point of view of one of the potential causes, that provide its point of view assuming it is not anomalous itself. The method is the same

Algorithm 2 Voting algorithm

```
Require:
    the anomaly $A$,
    a set of possible causes of $A, PC_A$,
    the semantic type of the anomaly, $T$,
Ensure: the vector of votes, $\boldsymbol{v}$
 1: procedure COMPUTEREPUTATIONVOTEVECTOR($A, T, PC_A$)
 2:     $S \leftarrow$ GETHISTORICALTIME SERIES
 3:     $S' \leftarrow$ GETSIMILARHISTORICALDATA($S, A$)
 4:     $\boldsymbol{V} \leftarrow \emptyset$                                    ▷ the voting matrix
 5:     for all $p \in PC_A$ do
 6:         $S'' \leftarrow$ GETSIMILARHISTORICALDATA($S', p$)
 7:         for all $c \in PC_A$ do
 8:             $d_{p,c} \leftarrow$ CALCULATEDISTANCE($S'', c$)
 9:             $v_{p,c} \leftarrow$ CALCULATEDISTANCE($S'', c$)
10:         end for
11:         $\boldsymbol{r} \leftarrow$ CALCULATEREPUTATION($\boldsymbol{V}$)
12:         $\boldsymbol{v} \leftarrow$ WEIGHTVOTES($\boldsymbol{V}, \boldsymbol{r}$)
13:     end for
14: end procedure
```

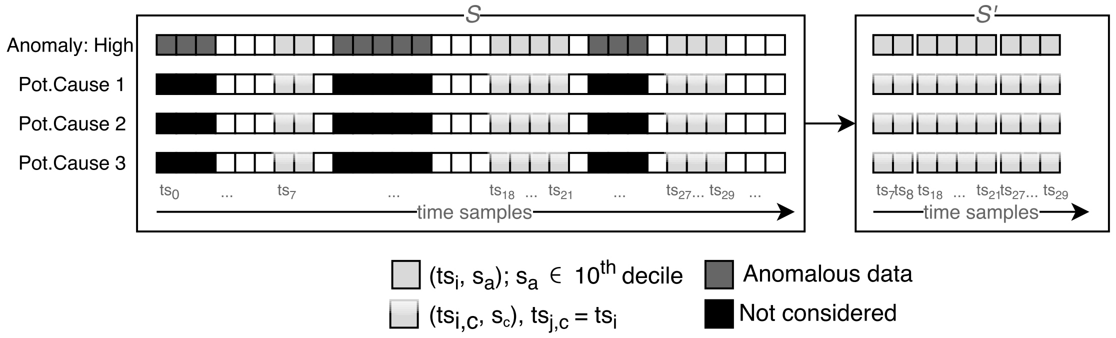

S
S′
Anomaly: High
Pot.Cause 1
Pot.Cause 2
Pot.Cause 3
ts_0 ... ts_7 ... ts_{18} ... ts_{21} ... ts_{27}... ts_{29} ...
time samples
$ts_7 ts_8$ ts_{18} ... $ts_{21} ts_{27}$... ts_{29}
time samples
(ts_i, s_a); $s_a \in 10^{th}$ decile
$(ts_{i,c}, s_c)$, $ts_{j,c} = ts_i$
Anomalous data
Not considered

Figure 3.4: Creation of the set S'

described for the computation of $S^{'}$, using the value observed for that property at the time of the anomaly instead of the one of the anomaly itself. At this point every potential cause p has the informations needed to give the opinion, that is to vote, about which is the actual cause c of the fault, according to its perceived state of the world $S^{''}_{p}$. The voting is expressed in terms of normalized mean distance $d_{p,c}$ between the actual value, observed by the suspect property c at the time t_0 of occurence of the anomaly, and the mean of the samples values that represents the behaviour of the potential cause expected by the voting variable.

$$d_{p,c} = 10\frac{s_c(t_0) - \operatorname{mean} S^{''}_{p,c}}{\max S_c - \min S_c} \tag{3.1}$$

Ploennigs et al. [9] suggests a modification to Equation 3.1 in order to tackle scenarios involving delays. This modification leads to

$$\begin{gathered} d_{p,c} = 10\frac{\operatorname{mean}(\sum_{\tau=0}^{\tau_U} s_c(t-\tau)) - \operatorname{mean} S^{'''}_{p,c}}{\max S_c - \min S_c} \\ \text{with } S^{'''} = \{s_c(t^{'''} \in S^{''}_{c} | t^{'''} \leq t - \tau_U\} \end{gathered} \tag{3.2}$$

so that all relevant previous values up to a maximum delay are taken into account and their mean value is used instead of the value at the time of the anomaly. The expected comparison value is calculated over a subset $S^{'''}$ including only samples before $t - \tau_U$ so that values that are in the scope of the delay are not used as comparison values. Normalization occurs in a range of [-10,10] with the maximum and minimum value of the whole time series in S (anomalous values included). These steps ensure that the different measurements can be comparable. The resulting distance is less then zero if the actual value observed for a property is lower than the value the voting property is expecting to see, it is greater than zero otherwise. This information about the signedess of the computed distance plays a key role during the fault identification because it helps discriminating between anomalous values that are the cause of a fault and those anomalous values that are effect of a fault. That means that if the behaviour of a property doesn't match the behaviour that a cause should have, said property can't be a cause and its vote is set to zero.

$$v_{p,c} = \begin{cases} d_{p,c} & \text{if } d_{p,c} \geq 0 \wedge \text{type}\, c \equiv \mathit{High} \\ -d_{p,c} & \text{if } d_{p,c} \leq 0 \wedge \text{type}\, c \equiv \mathit{Low} \\ \|d_{p,c}\| & \text{if } \text{type}\, c \equiv \mathit{Unknown} \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

For example, if a room's temperature is too high and the real faulty property is the outside temperature, there's a chance that the chiller power consumption will be anomalous too since the chiller is trying to mitigate the effect of the fault. Still the chiller is not the cause of the anomaly. Once the computation of the distances ends, the matrix of the votes is filled with the "opinions" that the potential causes have about each other. Since the votes have been calculated under the assumption that the potential cause that is voting is under normal circumstances, if said cause is the origin of the fault then it will flag as faulty all the others because its point of view will be so distorted that normal behaviours appear as anomalous. This tendency of faulty properties is misleading for the diagnoser and thus needs to be addressed. The way this algorithm deals with the problem is through the means of the concept of reputation. A potential cause has a high reputation r_p, hence it is trustworthy, if it doesn't receive any vote from other causes. The reputation decreases as the property receives more votes, as seen in Equation 3.4.

$$r_p = \frac{1}{1 + \sum\limits_{c \in PC_A} v_{p,c}} \tag{3.4}$$

Given the reputation vector $\boldsymbol{r} = \{r_i | i \in PC_A\}$, the total vote v_p received by a potential cause p is given by the sum of the votes of all the properties weighted by the reputation of the property that gave that vote (Equation 3.5). It is considered abnormal every property that has a value higher then 1.

$$v_p = \sum_{c \in PC_A} r_c \cdot v_{p,c} \tag{3.5}$$

An alternative kind of vote is the so called vote count. This method drops the magnitude of the votes and simply counts how many votes greater than one have been assigned to a given property p. (Equation 3.6).

$$\begin{aligned}\bar{v}_p &= \sum_{c \in PC_A} \operatorname{discretize}(r_c \cdot v_{p,c}) \\ \operatorname{discretize} x &= \begin{cases} 1 & \text{if } x \geq 1 \\ 0 & \text{otherwise} \end{cases}\end{aligned} \tag{3.6}$$

The formulae result in the vote vectors $\boldsymbol{v} = \{v_p | p \in PC_A\}$ and $\bar{\boldsymbol{v}} = \{\bar{v}_p | p \in PC_A\}$. Using one method or the other yelds different results and which one to use is decided upon execution on a per-case basis. Experiments [9] demonstrated that it is usually better to use the vote count whereas there could be multiple faults.

3.4 Diagnosis example

In this section it is shown how the diagnosis takes place in the context of a building as in Figure 3.1. It is assumed that data picked up by the sensors just before the anomaly occurs are distributed as in Figure 3.5. The rule that monitors the anomaly is "if the room's temperature is greater than 25° there's an anomaly" and the fault is due to the chiller that stopped to work at 1 pm. The room temperature starts rising and a first anomaly is detected, thus starting the diagnosis process. Since the anomaly is a temperature greater than 25°, the semantic type of the anomaly is High. Then the possible causes can be traced back, following the rules explained above. The anomaly is detected because of the room temperature and so the properties connected to that property are traced back. The room temperature is influenced by the internal energy of the room, that is unobservable. This violates the second rule and so energy can't be a potential cause and needs to be resolved. This leads to discover as potential causes the occupancy, that is a potential cause via a positive proportional process (see Table 3.1), the outside temperature, still via a positive proportional process, and the cooling property. Once again, cooling property is unobservable and needs to be traced back. All the backtracked causes and the derived processes are shown in bold in Figure 3.6. As soon as the potential causes are known, it is possible to predict the semantic value of said causes and according to Table 3.2 it is derived the graph shown in Figure 3.7. The diagnoser

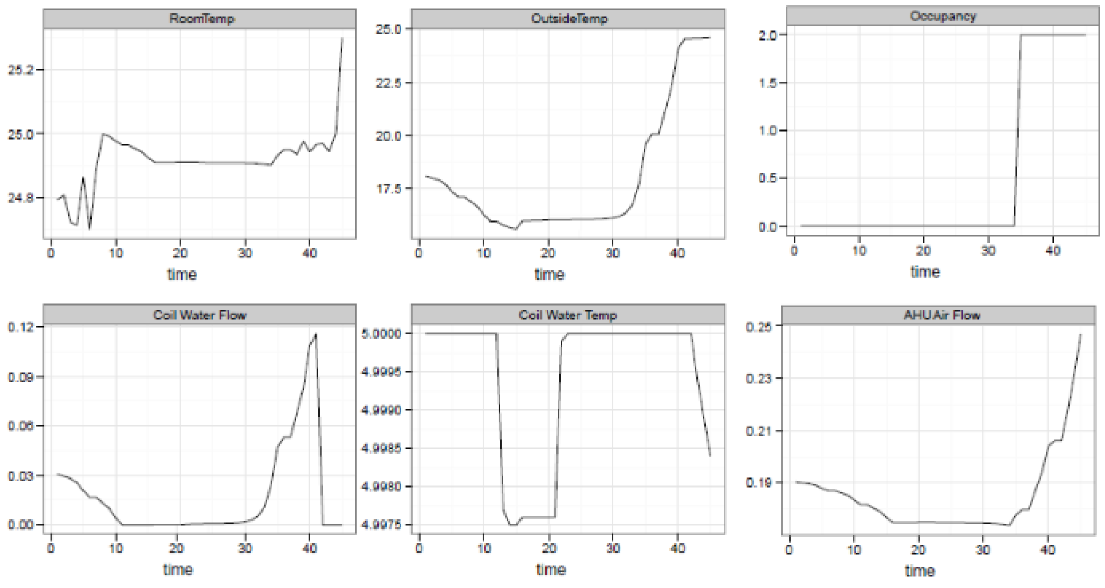


Figure 3.5: Example of time series of different sensors

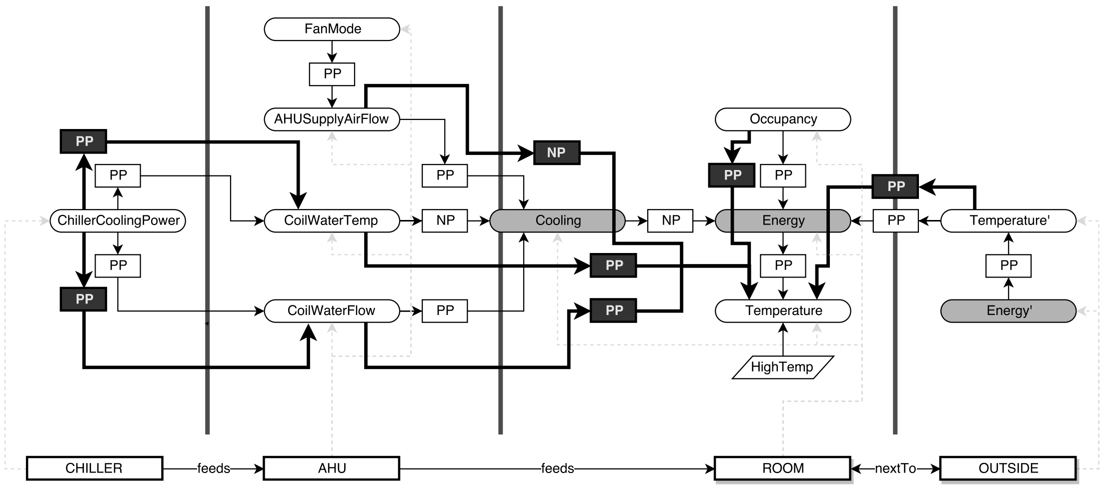


Figure 3.6: Backtracking the possible causes of a HighTemp anomaly

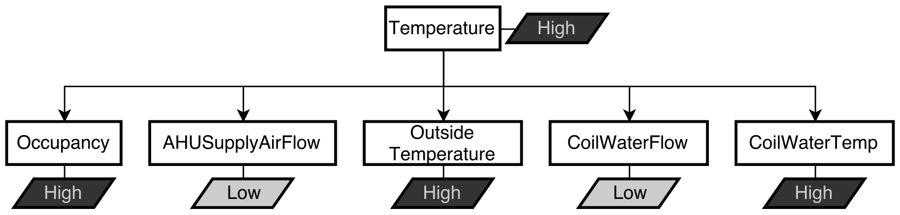
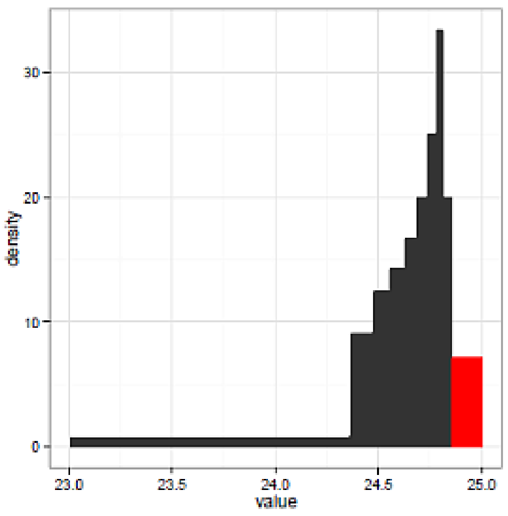
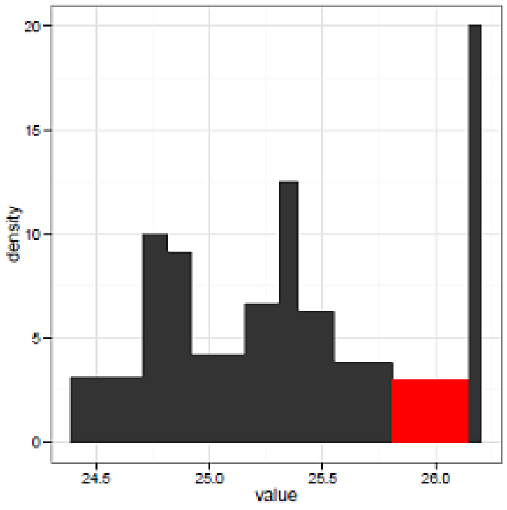


Figure 3.7: Semantic type of potential causes



(a) Deciles computed for anomalous property



(b) Deciles computed for outside temperature

Figure 3.8: Deciles ranges used for computation of S' and S''

has enough informations for computing the vote vector. It starts from collecting the time series measured by the sensors observing the involved properties (Figure 3.5) and thus populating the S set. From this set the diagnoser tries to extract the subset $S' \subset S$. This is done by extracting the anomaly-free historical time series and by computing the deciles over these data. Since the anomaly was of type High, it is selected the tenth decile ranging from 24.86°C to 25°C. The time series of other properties involved are derived by sampling the property time series at each timestamp such that at that time the room temperature is between 24.86°C to 25°C. This set S' represents the normal behaviour of the building that is closer to the anomalous one and is the common ground used by every potential cause to build its own point of view. Each potential cause p can use this common ground to derive a subset $S''_p \subset S'$ based on the similarity respect to the value of the property at the time of the anomaly. Once the subsets are derived, the properties can express their vote. The vote matrix is shown in Table 3.3. It is interesting to note how the supply air flow property doesn't get any vote even if its value is abnormally increasing after the anomaly, as see in Figure 3.5. This is due to the fact that the system expects the supply airflow to be lower than the usual for it to be a cause of the fault. The AHU tries to make up for the fault at the

Table 3.3: Vote matrix

	Occ	**CWT**	**SAF**	**CWF**	**OT**
Occupancy	0	0.0010	0	3.741	1.437
CoilWaterTemp	0.909	0	0	3.331	1.394
SupplyAirFlow	0	0	0	7.678	0.470
ChillerWaterFlow	0.909	0.001	0	0	2.555
OutsideTemperature	1.8182	0	0	8.747	0.003
Vote sum	3.636	0.002	0	23.497	5.861
Reputation	0.215	0.997	1	0.04	0.145

Table 3.4: Weighted votes and vote vectors $\boldsymbol{v}$ and $\bar{\boldsymbol{v}}$

	Occ	**CWT**	**SAF**	**CWF**	**OT**
Occupancy	0	0	0	0.807	0.310
CoilWaterTemp	0.906	0	0	3.322	1.390
SupplyAirFlow	0	0	0	7.678	0.470
ChillerWaterFlow	0.037	0	0	0	2.555
OutsideTemperature	0.264	0	0	1.275	0
Total Vote $\boldsymbol{v}$	1.208	0	0	13.081	2.276
Vote Count $\bar{\boldsymbol{v}}$	0	0	0	3	1

chiller by increasing the air flow to cool the room, hence it is a "good" behaviour. Since the supply air flow property contradicts the behaviour that a cause should have, it can't be a cause and its vote is set to zero according to Equation 3.3. In Table 3.3 is also shown the property reputation calculated as in Equation 3.4. According to Equation 3.5 and Equation 3.6 the vote vectors are calculated as shown in Table 3.4. The diagnoser will then diagnose those properties that are deemed to be causes and correctly traces back the root cause that is the chiller (see Figure 3.6). Ploennigs et al. [9] demonstrate that this approach has been able to adapt to delayed effects scenarios, multi faults scenarios with both unobservable and observable faults and to more complex buildings.

Chapter 4

IoT Brain

This chapter is going to give some informations about the design choice taken while implementing the approaches in **chapter 2** and **chapter 3**. Implementation efforts were aimed at providing a further degree of scalability to the approach and leveraging cloud based technologies as well as reducing the technical difficulties that usually arise while fiddling with complex tools like Semantic Web ontologies that are, in turn, too expressive for the scope of this project.

4.1 Fully integrated BMS

Previous chapters defined a possible approach, developed by IBM Research, for diagnosing smart buildings. It has been shown that such a method works as expected. However such instruments needs to be integrated in a broader panorama of services that should be present in a Smart Building, instead of being perceived as standalone, monolithic applications. Defining the term Smart Building is a challenging task and various organizations adopt their own definition. The US Government Services Administration (GSA):

> Technology alone won't do it. The GSA realises that the smartest part of smart buildings is people and wants to engage them. Providing feedback and information through a dashboard is a good start. With smart technology, we can learn anything we want about a building and optimise its performance. But real performance means happier, more

> productive tenants. And that requires insights into the hears and minds of the people inside. What a dashboard can really do is enable better decisions, inspire participation, spread knowledge and best practices, communicate at a human scale and propagate new norms in how we use our buildings.

while IBM states:

> Smart buildings are well managed, integrated physical and digital infrastructures that provide optimal occupancy services in a reliable, cost effective, and sustainable manner. Smarter buildings help their owners, operators and facility managers improve asset reliability and performance that in turn, reduces energy use, optimizes how space is used and minimizes the environmental impact of their buildings.

many others provide their own definitions that, however, do not change that much, if not in the form. What emerges is that a smart building has to be an integrated set of applications that works and communicate in order to enable the building to perceive the context it is in, being aware of its status and its occupants. Another key concept it is its ability to react, and adapt, autonomously to changes in the environment, possibly predicting meaningful events and optimizing its behaviour in order to optimize occupants comfort and energy consumption. IoT devices provide the context for a Smart Building. Commercial buildings are constantly monitored by IoT smart sensors and meters (see Figure 4.1) that provide a huge amount of data that needs to be stored and processed, aggregated and enriched with external data sources to extract useful information on the state of the building. Analytics provide the means that the building use to interpret the context and react in a correct way (diagnosis, user comfort models, energy consumption predictions). An example can help to better understand the concept of fully integrated and smart BMS. Let a fault occour at a Fan Coil Unit (FCU) in a room. A Smart Building should be able to detect it, diagnose it, alert the manager, eventually issue a repair request to the maintenance service with a description of the fault. Then, upon arrival of the maintenance worker, the building should be able to guide said worker through the building itself towards the location of the fault with a human-like conversational interface. Through the example it is higlighted

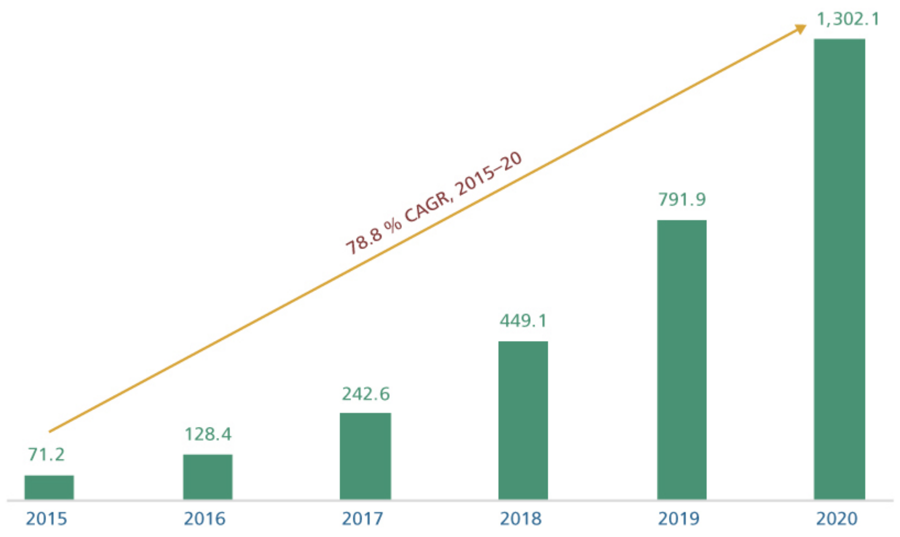


Figure 4.1: Potential growth in worldwide IoT sensor deployements for Commercial Real Estate, [millions]

Source: Chart created and analysis performed by the Deloitte Center for Financial Services based on Gartner research: "Forecast: Internet of Things, endpoints and associated services, worldwide, 2015", Gartner Inc., October 29, 2015. Graphic: Deloitte University Press — DUPress.com

the strong integrated nature of a Smart Building Management and Automation System.

4.2 Requirements

In a Smart Building context it is important to understand what are the main goals that are to be achieved by the deployment of such a system. What emerged from this thesis work is that a Smart Building IT infrastructure needs to be build upon a scalable framework that provides a platform for easy and fast analytics deployement. There's a need of a middleware that provide every top level application with a common ground that abstract the physical sensors so that the whole system is integrated and top layer components are able to exchange informations. This middleware is born from the Semantic Web concept of ontology (see chapter 1) and of semantic reasoning. What is different here is that the scope is note the same. Semantic Web technologies have their focus on massive amount of diverse data that comes from different sources and their goal is somewhat to unify them and give them a meaning that is universally accepted. That is a hard goal to achieve for a single simple ontology. Therefore highly complex solutions like the Ontology Web Language (OWL)[5] have been developed. In that context standardization and rigid formality in their representation. In the context of Smart Buildings the focus is shifted and requirements become more lax. Building's data are sensible and are not supposed to be exchanged outside of the scope of a single organization (a company, a university, etc.), additionally, since the goal of this framework is to manage a building, the concepts involved are little in numbers and simpler in structure than those involved in the Semantic Web. Balaji et al. [1] showed how a little number of concepts and relationships (see Figure 1.6) is required for capturing the complexity of a building. Orthogonal to this, another simple ontology SSNO and its extension (see subsection 1.2.2), developed by IBM Research, suffice for capturing the complexity of physical interaction between buildings assets, as seen in Figure 1.8. Moreover, opposed to the Semantic Web context, it can't be forget that a part of the integrated system is comprised of human actors who work and live in the buildings and form a big part of the context of the building itself. It is important to identify and focus on the potential users of such a system. Those users are both those who benefit from the deployed analytics and those who are going to develop those analytics, still it is not required from them to have a strong computer science background of any sort; even developers in this case can be just domain expert (eg. physicists, civil engineers etc.) that for example needs to update, or change, the physical model rules or the model of the architecture. The system should aim at providing those users the most straightforward default way to interact with the system, trough high-level languages and possibily human language interfaces (see section 4.4) yet it should still allows for low level interaction where needed. To sum it up, the system that is the end goal of the project needs:

- easy development tools
- high level scripting languages
- semantic capabilities to enable integration
- scalable back-end

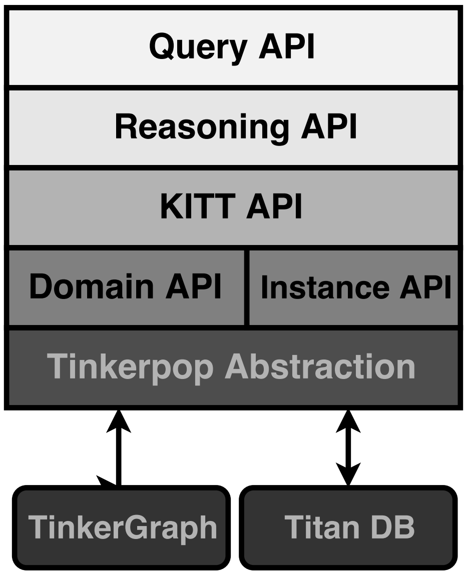


Figure 4.2: KITT internal structure

4.3 Knowledge Inference Technology for IoT

Knowledge Inference Technology for IoT (KITT) is a layered framework that provide an API to model, manage and reason on the semantic knowledge of IoT Systems. It has been developed by IBM Research and expanded with further capabilities as the core of this thesis. It is designed to meet the requirements outlined in the previous section and to provide a service for integrated deployment of analytics in IoT contexts. KITT adopt a layered approach as seen in Figure 4.2. Each of these layers implements different functionalities that will be detailed just below

4.3.1 TinkerPop abstraction layer

This layer is strictly related to the low level implementation of the graph on the physical store. What is important to note is the choice of representation of the underlying RDF graph that is the base of the semantic approach. Altough many RDF stores and semantic databases are available, the choice made fell towards a graph Data Base (DB) implementation of said RDF graph. The main reason is to be searched in the scalability needs of the project and the scope of it. What has been developed is an integrated semantic framework for Smart Buildings and not a multipurpose layered semantic engine.

GraphDB vs. RDFStore It is important to note that the word semantic doesn't necessary mean RDF. When talking about semantic it is intended a way to give a meaning to otherwise unmeaningful data so that more complex operations can be performed, thanks to the abstraction that such additional knowledge provide. It has been said in section 1.1.1 that RDF is a way to store triples, and that those triples represent a graph. Ontologies, then, are still triples that makes use of some well known concepts and relationships (e.g Class and subClassOf) to give a meaning to RDF triples. It follows that RDF data, both ground truth and ontologies, can be safely stored in a graphDB. There are also some advantages in using a graph representation. For instance, graph DB implements the so called Labeled Property Graph (LPG) whose main advantage is that of storing vertexes and labels that can have an internal structure. In practice, this allows for the chance of storing additional data, whose semantic description is not essential, without complicating the ontology. It is possible, for example, to store information about geographical coordinates of an equipment directly in the node, without having to model such data in the ontology (see subsection 4.3.3). This way applications can also append (e.g through JSON string) other application-dependant data at runtime, keeping the model small in size.

Apache TinkerPop The TinkerPop abstraction layer has been developed in order to abstract the underline storing engine and provide vendor indipendance. This layer provide a series of methods that can be used by upper layers to directly operate on the graph without worrying about the underline technology. Changing the graphDB engine just requires the right implementation of the abstraction layer without further modification of the upper layers. Apache TinkerPop itself is an open source, vendor-agnostic, graph computing framework distributed under the commercial friendly Apache2 license.

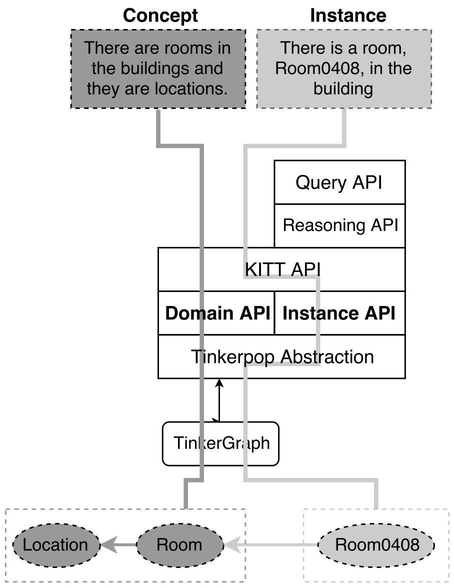


Figure 4.3: Usage of the Domain and Instance API

4.3.2 Domain API and Instance API

These layers both work upon the TinkerPop Abstraction layer as they are the main component involved while building the model of a building. The domain API acts when defining the taxonomy of a domain. For example, whenever it is needed to express that a new concept exists, like a Room is a subClassOf a Location, that is Domain API duty. The Instance API is, instead, in charge of the creation of new facts, that are the specific knowledge about a system. This API is the one responsible, for example, of the creation of new instances, such that Room0408 is a instance of the concept Room. In Figure 4.3 it is detailed which API an instance and a concept are going through when they are created in the graph.

4.3.3 KITT API

KITT API is the middle ground between lower layers and upper layers. At this point of the stack everything is abstract enough so that the main focus shift towards the user of the system, be it human or an application. KITT API provide high level endpoints that allow for interactions with the lower level domain and instance endopoints through high level text base format (JSON, KITT scripting).

KITT scripting KITT scripting language is a way for the end user to inject in the system a whole model in a straightforward way. A end user should be able to load a whole model without worrying about implementation details or technical aspects of reasoning and ontology modelling. This lead to the development of a simplified scripting language for ontology creation, that is simple yet expressive enough for implementing the needed relationships of Brick (subsection 1.2.1) and SSN (subsection 1.2.2) ontologies. It is based on the following assumptions:

- Domain concepts and relationships are structured in a taxonomy and so are disjointed
- Relationships can be established between concepts or between instances
- Relationships between concepts and instances are only the basic ones connecting an instance to its parent concept
- Instances have to be instantiated from the defined model.

Thanks to these assumptions the scripting language can define a full model in a human readable text format. Under these assumptions the model is powerful enough to create a digital twin of the building. A KITT script is divided into three main parts

- Concepts definition
- Model instantiation
- Ruleset definition.

In the concept level definition a series of concepts, that is the vocabulary of the model, are defined as shown in Listing 4.1

Listing 4.1: KITT script for domain definition

```
[Init]
[Dimensions]
Point
Location
Equipment
```

```
Property
[InstanceReferenceTypes]
hasLocation = in
hasPoint
isFedBy
hasProperty
[Concepts]
Location
  Room
Equipment
  HVAC
    AHU
Point
  Sensor
    Flow_Sensor
      Supply_Air_Flow_Sensor
    Temperature_Sensor
Property
  Temperature
```

the script start with the definition of the main dimensions of the ontology. In Listing 4.1 the definition of Point, Location, Equipment and Measurement suggests that the ontology used is going to be Brick. The `[InstancereferenceTypes]` section introduce which possible relationships are going to be instantiated later, shorthands can be defined so that defining instances, in the appropriate section of the script, becomes easier. The `[Concept]` tag start the part of the script where the taxonomy of the ontology is defined. The tabulation at the start of each line has a semantic meaning and defines the hierarchy of the concepts; An indent means that the following concepts are sub-concepts of the upper one, while concepts on the same line represent disjointed classes. In it is represented the scripting module that correspond to the declaration of the ground truth of the knowledge base, that in the case of a smart building that means defining the architecture, the spatial relationships between locations and the various equipment and points distributed in the buildings.

Listing 4.2: Definition of instances

```
[Instances]
Room1708 a Room ''the room''
  TS1708 a Temperature_Sensor hasPoint Room1708 X{"type": "
     Point", "coordinates": [32.76473, 35.01616] }X {oBMS:"
     ts\1708\."}
AHU1 a AHU isFedBy Room
  SAF_U1 a Supply_Air_Flow_Sensor hasPoint AHU1
...
```

the instantiation is done through the keyword `a`, that specify the concept an isntance belongs to. While creating the instance the KITT interpreter looks for `X{...}X` GeoJSON coordinates and `{"key":"value",...}` dictionaries and store them in the underline graph node. This allow for a powerful annotation feature enabled by underline property graph structure. For example coordinates can be used by a navigational interface that guides a user in the building. The next section in a KITT script is the `[Reasoning]` one, whose goal is to define and execute the ruleset that infer the physics model of the building. Listing 4.3 defines the syntax in the script, while the syntax and the meaning of the rules is described in subsection 4.3.4.

Listing 4.3: Definition of reasoning ruleset

```
[Reasoning]
Room(x)->Temperature(y)&hasProperty(x,y)
...
```

When fed to the KITT API the KITT script goes through a parser that build the model and execute the reasoning. At the time of model creation, the parser itself automatically compute subsumption relationships for the instance such that every instance is directly bound to its parent class and to other ancestor classes.

4.3.4 Reasoning API

Whenever reasoning is involved, it is intended as the capability of inferring new knowledge about the system given previous asserted facts and an ontology. The main duty of the reasoning engine in this project is that of applying knowledge about general physics to the given model. The second one is it's capacity of answering complex query that the KITT API level can't natively handle. Therefore KITT reasoning language describe an union of production rules languages like SWRL[14] and query languages like SPARQL[15]. On top of that ad-hoc functions have been added so that such language could be the most straightforward for the operations involved in this project.

KITT reasoning language The reasoning language developed borrowed syntax from classic rule languages like SWRL, yet it needed to expose methods useful to explicit complex query and path traversal. A KITT rule is expressed as a form of entailment

$$body \implies head$$

where *body* and *head* are both expressed in conjunctive form, so that a rule assume the form

$$A_1 \wedge \ldots \wedge A_n \implies B_1 \wedge \ldots \wedge B_m$$

where A_i and B_j are either statements about, or relationships between, facts (see Equation 4.1).

$$Room0408 \wedge Temp0408 \implies hasProperty(Room0408, Temp0408) \tag{4.1}$$

That means that wether the condition in the body holds, the condition on the left must hold too. This characteristic, typical of production rules, is used to entail new knowledge according to domain expertise. When coupled with the concept of variable, rules becomes a powerful tool to enforce knowledge on a knowledge base. Thanks to variable, rules can be wrote refering to concepts, that in turn are set of instances; this allows for the encoding of domain expertise into procedural rules.

For example, a human expert would not say something as in 4.2

$$\begin{aligned} Room0408 &\implies hasProperty(Room0408, Temp0408) \wedge Temp0408 \\ Room709 &\implies hasProperty(Room709, Temp709) \wedge Temp709 \\ &\dots \end{aligned} \tag{4.2}$$

but would instead say “Every room has an internal temperature”; from this sentence it emerge that there exist a concept of room, a concept of internal temperature and that this concepts are related. Introducing variables it becomes a rule as in Equation 4.3

$$Room(x) \implies Temperature(y) \wedge hasProperty(x, y) \tag{4.3}$$

where x and y are variables that can be bound to any given fact in the instance model; if the variables appears on the right-hand side of the rule and can’t be bound to any given fact in the knowledge base, they are created so that the entailment is satisfied. The second fundamental function of the reasoner is its capability of answring complex queries. Complex queries are questions whose answers are not imeediately retrievable from the knowledge base through simple relationships. Complex queries are, in contrast with production rules, usually focused on retrieving information about well known facts instead of general knowledge thus requiring for a way of mixing in the same rule both named facts and variables. A good example of this sounds like “Which are the temperature sensors of the rooms fed by AHU1”; translating it like explained just above

$$AHU(a) \wedge Room(r) \wedge isFedBy(r, a) \wedge TempSensor(t) \wedge hasPoint(r, t) \tag{4.4}$$

lead to an erroneous interpretation of the query as this would fetch “every temperature sensor of the rooms that are fed by either one of all the AHUs”

$$AHU(AHU1) \wedge Room(r) \wedge isFedBy(r, AHU1) \wedge TempSensor(t) \wedge hasPoint(r, t) \tag{4.5}$$

```
Property(start,#Temperature)&_CHAIN({hasInput(v),hasOutput(^)},start,end)[2]&Property(end)
```

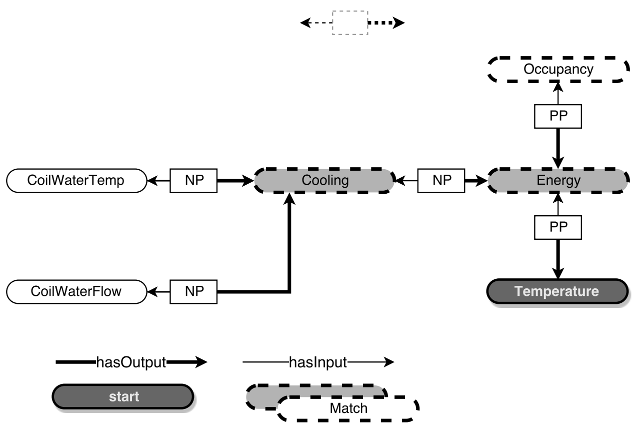


Figure 4.4: Property chain in a rule

is the right way to go and it has been implemented in the reasoner as

$$AHU(a, \#AHU1) \wedge Room(r) \wedge isFedBy(r, a) \wedge TempSensor(t) \wedge hasPoint(r, t) \tag{4.6}$$

where $AHU(a, \#AHU1)$ binds the variable a to the named individual $AHU1$. A less traditional ways to use this kind of rules is for the evaluation of property chains. Property chains, in the context of physical modelling and diagnosis represents important knowledge (see section 3.2) and the ability to traverse them is essential for reasoning in such environments. An ad-hoc operator has been introduced so that such traversal are possible

```
_CHAIN({R(dir),R(dir),...}, start, end)[condition]
```

where `R` is a relationship in the ontology, `dir` $\in$ `[^,v]` tells wether the relationship to chase is an incoming or out-coming one and when are combined as in `{R(dir)...}` they specify a pattern to match; `start` identify a known variable in the head of a rule and so is `end`, respectively identifying the first and last instances of the chain, `condition` is either a number specifying the maximum number of pattern matching to perform before stopping, a condition the `end` variable needs to uphold in order to stop the recursion or both. Such function is visible in Figure 4.4. In the case that a cycle is identified during the traversal it is natively dealt with using standard cycles detection algorithms.

Reasoner known limitations It is important to note that enabling the reasoner to perform update/modification of a knowledge base is bound to end up in the domain of undecidable languages. This is easily demonstrated by a counterexample like

$$A(x) \implies A(y) \tag{4.7}$$

or again

$$\begin{aligned} A(q) &\implies B(r) \\ B(s) &\implies A(t) \end{aligned} \tag{4.8}$$

and whatever ruleset such that there's some kind of cicles involved in the resolutions of such rules that would start to be fired endlessly and never comes to an halt. Such problem is well known in literature and needs to be evaluated on a case-by-case basis. It boils down to balancing whether is better to achieve high expressivness or remain in the domain of decidability. In this project, given the requirements in section 4.2 has been deemed more appropriate the first than the latter and this follows from the main purpose of the reasoner, that is to fill incomplete knowledge. Another limitation of the current reasoner is represented by the method used for executing the ruleset, that is the most naive possible and is a simple execution based on declaration order.

4.4 Towards conversations

The whole project has been built upon the concept of semantic layer. This semantic layer is the common ground that allows applications to communicate. The actors involved in a Smart Building, as stated in section 4.2, are both applications and human users, that are consumers of informations that the building produce on par with other applications. It then makes sense to exploit this semantic common ground to allows end user to interact with the buildings in the most natural way, that is through speech, thus tearing down the barrier between humans and cyber physical systems (CPS). This allows for improved integration of the human resource in the Smart Building system, improving productivity and comfort of the occupants. Figure 4.5 shows how the possibility of communicating through a conversational interface allows for increased mobility and it cuts on learning curves,

Figure 4.5: Advantages of conversational interface

diminishing user frustration when faced with the problem of learning new things. The main idea behind the approach is that natural language carries a meaning and if the domain context is well defined and agreed upon, it is possible to extract from a natural language sentence enough informations so that it is possible to start a dialogue. In the case of smart buildings the domain context is defined by the vocabularies of Brick and SSN. As soon as a user wants to interact with the system, it can state its question through natural language; the statement is parsed looking for:

- concepts, that are the subjects, predicates and objects (thus can be searched for in the ontology)
- intents, that specify some kind of processing executed on the subjects and objects.

This high-level idea has been implemented thanks to the IBM Watson Conversation services and led to the development of a succesful demo application. More details will be presented in Chapter 5.

Chapter 5

Experiments and results

In this chapter will be outlined the experimentation phase of the project. The experimentation has been divided in two types. The first one assessed the performance of the model platform and the reasoner whilst the second one has been the development of a functioning proof of concept of a conversational natural language interface. Each section will describe the setup and the result of one of the experimantations trying to interpret the results.

5.1 Reasoner performance

For the sake of performance evaluation KITT has been deployed on a local liberty webserver, simulating the cloud based running environment of real deployments. The performances have been evaluated in respect to the two classical measures, time and space requirements. Since real case scenarios for commercial buildings have considerably larger dimensions than the examples presented in previous chapter and that deployment data of real case wasn't available due to privacy reasons, in order to run a simulation that likely reflect real world data, the following route has been taken. A given architecture, representing a floor of a building, has been repeated over and over until reaching the complexity of a two hundred storeys building, each of which is comprised of up to a hundred rooms. For comparison, the tallest building in the world, as of 2018, is the Burj Khalifa in Dubai with 163 storeys. From Figure 5.1 it can be seen the structure of the first synthetic example

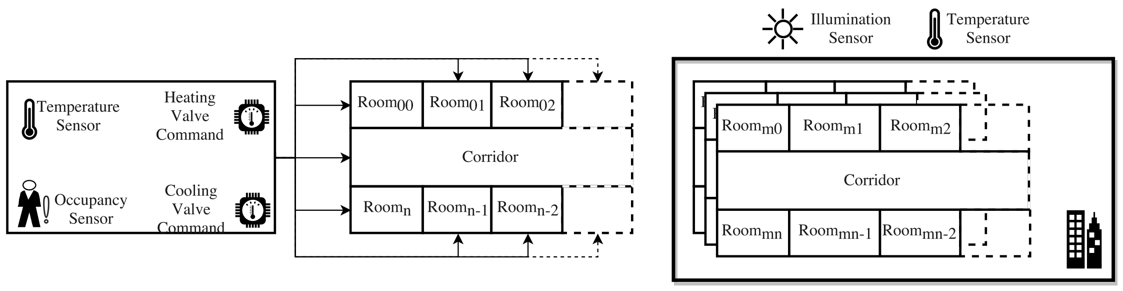


Figure 5.1: Construction of experiment data

Table 5.1: Performance test result for the simple model

Floors	**Rooms**	**No. Sensors**	**Vertexes**	**Edges**	**Vertexes reasoning**	**Edges reasoning**	**Memory create [MB]**	**Memory reasoning [MB]**	**Time create [ms]**	**Time reasoning [ms]**
1	1	10	123	381	143	477	8	54	16	95
1	10	46	168	588	251	1008	10	98	18	166
1	100	406	618	2658	1331	6318	47	228	156	2047
10	10	82	672	2973	1448	7011	53	151	186	2587
10	100	442	5172	24483	12248	61731	169	300	15850	256934
20	100	482	10232	48733	24496	48733	62	284	73692	1352861
50	100	602	25412	121483	60768	308011	143	267	492255	12631754
80	100	722	40592	194233	97158	492721	261	416	1312923	34668894
100	100	802	50712	242733	121418	615861	194	510	2172877	54904640
200	100	1202	101312	485233	242718	1231561	922	1014	8742782	225609628

of the test building. Every room is equipped with four Points: a temperature sensor, an occupancy sensor, a cooling valve command and a heating valve command. The rooms are located to the side of a corridor and are adjacent up to by two. This configurations represent a floor. Floors are then stacked one upon the other to form the final building. The outside environment is supposed to monitored by a temperature sensor and an illumination sensor. Given this configurations a full stack of reasoning rules are executed so that a physical model can be derived. The rules can be seen in

both the experiments shown that even though the graph dimension augments linearly with the number of sensor in a building, as well as the memory needs, that increase with the dimension of the graph,

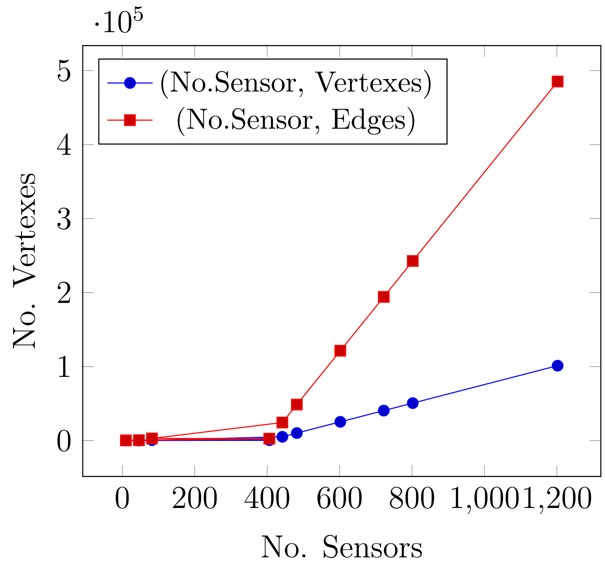


Figure 5.2: Correlation between the number of sensors and the graph vertices and edges count

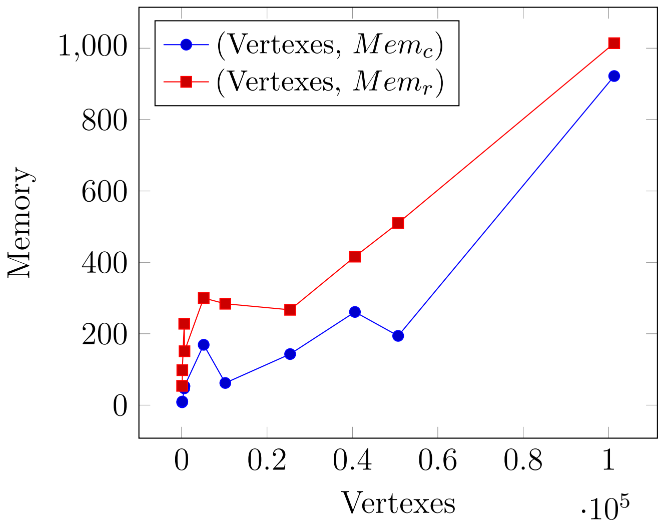


Figure 5.3: Correlation between the number ground truth vertices and the reasoning memory usage

Chapter 6

Conclusions and further developments

In this thesis has been presented a proof of concept of a novel framework, for the deployment of analytics in the context of Smart Buildings, that is based on semantic technologies. It has been pointed out the need of integration for enabling the transition from standard BMS to SmartBMS. The proposed solution, called KITT, is a layered, cloud based service that allows for the creation, storing and management of semantic models of a given building. Even though this approach has been validated and the proof of concept was happily acknowledged by professionals in the field, the project is still in its infancy and far from being mature. Being as new as it is there are lot of different directions to take in order to expand and improve the approach. Some of these expansion directions have already been highlighted in the previous chapters. Let sum up the ones that, me being the author, I think are the most interesting ones:

- Reasoner optimization; the proposed approach needed an ad-hoc reasoner that could go over the limitations that other reasoners imposed. The current KITT reasoner is far from being optimal, as seen in . One of the main issue is that it has been implemented, in the most naive way, as a simple full search in the space of the instances. No caching or heuristic is involved during the resolution of the rules. Studying whether classical optimization algorithm for production rules engine, like RETE, or novel ones yet to discover can improve the performances is one of the key challanges to tackle in the future.

- Ontology expansion; the ontology used in the project has been trimmed so that only essential relationships and concepts are left. This reduce both the complexity of the reasoning tasks and the development phase. It can happen that these restrictions can lead to difficulties in the deplyment of certain services. Studying the expressiveness of the ontology in the current status with respect to other well known ontology, like RDF-S, could help to find the optimal balance between expressivness and ease of use.
- Dynamic behaviour of the model; as the semantic model that sit at the center of the approach is the "digital twin" of the specific building it models, its status changes in time as the real building evolves or even faster. Sensors can be changed, sometimes added, sometimes removed and the same goes for locations and equipments too. On the other hand it can occur that the physical model is too coarse and it need to be replaced by a more detailed one. Every change means that the knowledge base can lose consistency and thus arise the need to think of ways to solve conflicts and update the model. It would be interesting to know if it is possible to repair the model instead of recomputing it from scratch with the new information as a ground truth.

as stated before these are just few of possible development directions that this project can take.

Bibliography

[1] Bharathan Balaji et al. “Brick: Towards a Unified Metadata Schema For Buildings”. In: (Nov. 2016), pp. 41–50.

[2] Michael Compton et al. “The SSN ontology of the W3C semantic sensor network incubator group”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 17 (2012), pp. 25–32. ISSN: 1570-8268.

[3] Jingtan Cui and Shengwei Wang. “A model-based online fault detection and diagnosis strategy for centrifugal chiller systems”. In: *International Journal of Thermal Sciences* 44.10 (2005), pp. 986–999. ISSN: 1290-0729.

[4] Srinivas Katipamula and Michael R. Brambley. “Methods for fault detection, diagnostics, and prognostics for building systems — a review”. In: *HVAC&R Research* 11.1 (2005), pp. 3–25.

[5] *Ontology Web Language.* URL: `https://www.w3.org/Submission/OWL`.

[6] Joern Ploennigs and Anika Schumann. “From Semantic Models to Cognitive Buildings”. In: *AAAI*. 2017.

[7] Joern Ploennigs et al. “Demo Abstract: BEAD - Building Energy Asset Discovery Tool for Automating Smart Building Analytics”. In: (Nov. 2014).

[8] Joern Ploennigs et al. “e2-Diagnoser: A System for Monitoring, Forecasting and Diagnosing Energy Usage”. In: 2015 (Jan. 2015), pp. 1231–1234.

[9] Joern Ploennigs et al. “Semantic Diagnosis Approach for Buildings”. In: PP (July 2017), pp. 1–1.

[10] *Project Haystack.* URL: `https://project-haystack.org/`.

[11] *Resource Description Framework.* URL: `http://www.w3.org/RDF`.

[12] Ioana Robu, Valentin Robu, and Benoit Thirion. "An introduction to the Semantic Web for health sciences librarians". In: 94 (May 2006), pp. 198–205.

[13] Anika Schumann, Joern Ploennigs, and Bernard Gorman. "Towards Automating the Deployment of Energy Saving Approaches in Buildings". In: (Nov. 2014).

[14] *Semantic Web Rule Language*. URL: `https://www.w3.org/Submission/SWRL`.

[15] *SPARQL Protocol and RDF Query Language*. URL: `https://www.w3.org/TR/rdf-sparql-query`.

[16] Yao-Wen Wang et al. "A simplified modeling of cooling coils for control and optimization of HVAC systems". In: 45 (Nov. 2004), pp. 2915–2930.

[17] Michael Wetter. "Multizone Building Model for Thermal Building Simulation in Modelica". In: 2 (Jan. 2006).