POLITECNICO DI TORINO

SCUOLA DI DOTTORATO

Dottorato in Ingegneria Informatica e dell'Automazione – XXIV ciclo

Tesi di Dottorato

The Role of Semantic Web Technologies in Smart Environments

Faisal Razzak

Tutore
Prof. Fulvio Corno

Coordinatore del corso di dottorato
Prof. Pietro Laface

February 2013

to my Family

Acknowledgements

I will start *"In the name of GOD, the Most Gracious and the Most Merciful"*. I thank GOD for giving me objectives in life and more importantly, the strength and the knowledge to achieve those objectives.

I feel great privilege and pleasure to extend my heartfelt thanks and sense of gratitude to my family. From beginning my parents taught me the worth of one simple, yet a powerful concept. The concept of seeking knowledge and achieving wisdom. It separates us (humans) from other species. It has helped us evolve over uncountable number of years and it will help us, in the future, to evolve further in order to explore the vast universe around us, and to bring the ability to seek knowledge in other species. My wife has always been very understanding and supportive of all my endeavors. No amount of words can describe the sense of gratitude, I feel towards her.

Last but not the least, I would also like to acknowledge efforts of my supervisor (Prof. Fulvio Corno) whose profound interest, guidance and encouragement helped me in every aspect of my PhD. His active supervision and inspirational guidance proved to be essential for the completion of this thesis. Furthermore, I appreciate all the members of e-Lite Research group for their useful and constructive feedback on several topics of interest. A special thanks to all the teachers who taught courses during my 4 years of PhD degree.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last decade, both the industry and academia have focused on brining two important and necessary changes in the global IT scene. The first change was the ubiquity of computing technology for general masses, which mainly helped trigger the second change; the advent of intelligent/personalized services for individuals within general masses. The former effort is being driven by models of Ubiquitous computing, Pervasive computing, Internet of Things etc. While, the latter effort is driven by methods of artificial intelligence, i.e., enriching the available data, and using algorithms or heuristics to bring intelligence.

On one hand, the drive to make intelligent distributed applications on a global scale has provided impetus to the adoption of explicit semantic modeling of concepts represented in web documents, and in general information systems. The semantic modeling of concepts ensures that the data is machine readable, processable and widely accessible. *Semantic Web* envisions the availability of semantically enriched data on a large scale and it was put forward by Tim-Berners Lee [1]. In the beginning, different architectures ranging from a layered approach [2] to a tower based approach [3] were proposed for the development of semantic web specifications and applications. However, in recent years (post 2006) the vision of semantic web has shifted from a traditional "logic+reasoning" approach, mainly proposed by research groups coming from classical artificial intelligence community, to a more pragmatic and engineered approach of having shared data semantics, and a web of data derived from it. The proponents of engineered approach argue that intelligent agents can flourish once languages, formalism and standards for data semantics and integration are defined [4].

Focusing on the need of data semantics, standard organizations like the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) have put major effort at specifying, developing, and deploying languages for defining and sharing meaning of data. Hence, providing a technological foundation for semantic

interoperability. This technological foundation mainly consists of *Resource Description Framework (RDF)* [5], *Resource Description Framework - Schema (RDFS)* [6], *Ontology Web Language (OWL)* [7], and SPARQL query language [8].

While the aforementioned technologies provide formalism for data semantics, the integration of data on a global-scale is achieved by using Linked Data (LD) [9]. LD approach [10] is based on Linking (i.e., using the RDF for creating references to information stored in different databases) Open Data (i.e., information freely retrievable in RDF format through the SPARQL query language over the http protocol). The simplicity of the LD approach stimulated the quick and enormous growth of the number of data set providers joining the initiative[1]. Formally, Linked Data is used to describe recommended best practices for exposing, sharing, and connecting pieces of data, information and knowledge over the web using URIs and RDF.

On the other hand, the emergence of economically viable and efficient sensor technology, that can be integrated with appliances, has enabled system designers to build smart environments [11]. The term "smart" refers to increased connectivity among diverse elements of the environment and to give users intelligent services. In literature, the vision of *Smart Environments* has been around since 1991 and was first proposed by Mark Wiser in his paper [12]. He anticipated environments interwoven with sensors, actuators, displays and computational elements, embedded seamlessly in our every day lives and connected through a network.

Today our daily spaces are filled with sensors, device and computational gadgets that measure or generate unstructured data over time, consequently presenting a two-front opportunity for system designers and integrators. The first is by acting on these islands of unstructured data and transforming them into structured data with semantics. The outcome will be to enable automated and intelligent agents to extract and act on the structured data. The second opportunity is to develop automated and intelligent agents that can process the structured data and provide some intelligent services to the users. Semantic web technologies have the potential to provide support for the representation of structured data, explicit context representation, expressive context querying, and flexible context reasoning [13].

This thesis outlines the role of semantic web technologies in smart environments like smart homes and smart energy systems. The potential of semantic web technologies in addressing some of the problems in smart environments is studied by proposing some solutions.

[1] http://richard.cyganiak.de/2007/10/lod

1.1 Contribution

This thesis makes two major contributions in the context of smart environments. The first comes in the form of a *Domotic Effects* framework, which provides the ability to control and monitor a smart environment in terms of user intelligible goals. The second contribution of the thesis is to describe mechanisms for exchanging semantically enriched data in a smart environment.

Bader et al. [14] described smart environments as heterogeneous dynamic ensembles: group of co-located devices of different device types, which evolve over time. The presence of diverse devices and the associated complexity has given rise to a major problem in the past years, i.e., the problem of providing users with the ability to control and manage their respective environments. The first major contribution of this thesis is to provide users with the ability to control and monitor their respective environments in terms of user intelligible goals. As acknowledged in [15], this research trend has received little attention. A "Domotic Effects" (DE) framework is being proposed. It models user intentions or goals in an environment, and it provides a unified model for both control and monitoring. The framework has several novelties.

First, at the modeling level, it addresses both the concerns of end-users and system designers using a unified model. End-users have the ability to define their spaces according to their intentions. On the other hand, system designers have the flexibility to define governing rules for diverse smart environments, at a generic level. The modeling is provided using a new "DogEffects" ontology. It is scalable and extensible depending on the smart environment.

Second, at the monitoring level, users have the ability to monitor their respective environments in terms of their intentions or goals, in near real-time. The novelty comes from the ability to monitor each and every device in the environment and then presenting a bigger picture to the user, i.e., pre-defined user goal. It is among the few approaches present in the literature that provides a complete picture, i.e., conception, architecture, implementation and experimentation using use cases.

Third, at the control level, users have the ability to control their respective environments by automatically enforcing their intentions or goals in near real-time. It is among the few approaches present in the literature that provides a complete picture, i.e., conception, architecture, implementation and experimentation using use cases. The enforcement mechanism itself is also novel.

Extending the work at the control level, the *DE* framework also provides the provision of optimizing enforcement with some criteria, in near real-time. In this thesis, energy optimization is considered as the criteria. A novel heuristic is proposed and tested for energy optimization.

The second major contribution of this thesis comes within the context of Energy Management System (EMS). Two novel mechanisms are proposed for the exchange

of semantically enriched data. In recent years, the energy management has become a key requirement for smart environments. Managing energy needs is a rising concern these days for many countries around the world. The resources needed to generate energy, their scarcity and the rising impact of those resources on the global environment have made energy management a top agenda on the tables of high government officials around the world. An approach to this energy management issue is "Demand Side Management", proposed by the Smart Grid community [16] which allows customers to make informed decisions regarding their energy consumption, by adjusting both the timing and quantity of their electricity consumption [16, 17]. This flexibility is enabled by pricing policies for electricity consumption over time [18, 19] and/or by dynamic demand scheduling algorithms to optimize energy services in buildings [20]. Such scenarios underscore the need in which the appliances can share the information about the energy usage with their energy provider. EMS provides a complementary approach to energy management in an environment by providing graphical illustrations [21–23] of consumed energy to ease consumer understanding, hence making an energy conscious society.

First, a novel ontology driven framework based on the publisher-subscriber pattern [24], called LO(D)D[2] is presented. LO(D)D is a light-weight publishing framework that can be integrated with smart environments to expose semantically annotated sensor's or device's data being continuously updated. This work is done in the context of *SMILE-O* project[3]. LO(D)D is inspired from the lessons learned during the development of an earlier publishing framework called **S**emantic **E**nergy **I**nformation **P**ublishing **F**ramework (SEIPF). SEIPF has a client-server based architecture that is able to expose power consumed by different appliances installed in an environment, in a machine understandable format (using SPARQL endpoint), to support the development of 3^{rd} party applications. It presents a novel ontology based modeling mechanism for encoding power consumption of appliances in different states and then uses Linked Data principles to expose the data. This thesis includes complete details from conception to experimentation of both frameworks.

Both the aforementioned mechanisms underline the need to have energy consumption details in a semantically enriched format and accessible on a global-scale, in a structured format. Thus, enabling the energy provider or third party services to utilize this information in order to provide better graphical illustrations, to design better pricing policies and to perform dynamic demand scheduling.

[2]Linked Open (Dynamic) Data

[3]http://www.smile-o.org (a Regione-Piemonte project)

1.2 Structure of the Thesis

The remainder of this thesis is organized into three parts, consisting of twelve chapters.

For the easy comprehension of thesis, Part I describes the underlying concepts and technologies and comprises two chapters. **Chapter 2** introduces different Semantic Web standards and technologies which are used in the design of User Intelligible Goals and Semantic Data Exchange. **Chapter 3** briefly define technologies used to represent and emulate a smart environment.

Part II consists of six chapters describing user intelligible goals in smart environments. **Chapter 4** presents the state-of-the-art and makes the case for designing user intelligible goals. **Chapter 5** introduces the *DE* framework for modeling user intelligible goals. Formally, the modeling is explicated in **Chapter 6**. **Chapter 7** and **Chapter 8** move in parallel and contain details of conception, architecture and implementation of *Evaluation* and *Enforcement*, respectively. **Chapter 9** extends the work in the preceding chapter and discusses the privilege of advanced intelligent support in *DE* framework, in the context of energy management domain.

Part III consists of three chapters addressing the issue of semantic data exchange in smart environments. **Chapter 10** describes the need of semantic data exchange in smart environments. The case of energy management domain is specifically considered. In order to support the development of 3^{rd} party applications two frameworks are presented. **Chapter 11** presents an ontology driven framework, called LO(D)D. LO(D)D gives smart sensing and measuring environments the ability to expose semantically annotated sensor's or device's data being continuously updated; such updates might be issued at specific time intervals or be bound to some environment-specific event. **Chapter 12** presents a Semantic Energy Information Publishing Framework (SEIPF). SEIPF uses RDF publishing to enable residential gateways to expose power consumed by different appliances installed in a house to support the development of external applications.

In the end, **Chapter 13** concludes the thesis and provides possible future directions.

Part I

Background

Chapter 2

Semantic Web Technologies

This dissertation discusses the role of semantic web technologies' in smart environments. In order to develop a better and shared understanding on the topic, this chapter briefly highlights key Semantic Web technologies and concepts.

Semantic Web is often described as a web of data; it creates a universal medium for the exchange of data [25]. It envisions an automated negotiation and retrieval of machine understandable information among web applications, services and intelligence agents. This chapter discusses several key technologies that are developed by World Wide Web Consortium (W3C) to achieve the vision of Semantic Web.

2.1 Resource Description Framework

RDF [26] is a data model that is used to describe resources over the web. Its basic building block is an *object-attribute-value* triple, called a statement. XML syntax [27] is popularly used to represent and transmit RDF data model. However, other textual representations like Turtle[1] and Notation3[2] are also being used increasingly by the Semantic Web community. In RDF, no assumptions about a particular domain of use is made and therefore, RDF is domain independent in nature. It is up to the users to define their own terminology in a schema language called *RDF Schema (RDFS)*. RDFS defines the terms that can be used in a RDF data-model. RDFS can specify which objects exist and which properties can be applied to them, and what values they can take.

[1] http://www.w3.org/TR/turtle

[2] http://www.w3.org/TeamSubmission/n3

2.1.1 Concepts of RDF

Resources

A resource can be described as an object or a thing that need to be described. Resources may be authors, books, events, peoples, rooms, search queries, devices, and so on. Every resource has a URL, a Universal Resource Identifier. A URI can be a URL (Unified Resource Locater or Web address) or some other kind of unique identifier. URI schemes are defined not only for web locations but also for diverse objects like telephone numbers, ISBN numbers and geographic locations. The discussion on URI schemes is beyond the scope of thesis. In short, it is assumed that a URI is a unique identifier of a resource.

Properties

Properties are special kind of resources; they describe relationship between resources, for instance "written by","generated by", "author", "title", and so on. In RDF, properties are also described by URIs (and in practice URLs). The choice of using URLs (for both resources and properties) gives users the opportunity to adopt a global, worldwide and unique naming scheme.

Statements

Statements represent the properties of resources. A statement is an object-attribute-value triple, consisting of a resource, a property and a value. A Value can either be a resource or a literal. Literals are atomic values (string), that can have a specific XSD type[3].

2.1.2 Three Views of Statement

Consider a statement:

```
Faisal is the owner of the web page "http://polito.academia.edu/FaisalRazzak"
```

The simplest way to represent the preceding statement is to use the definition of a triple and encode the statement as ("http://polito.academia.edu/FaisalRazzak", "http://www.mydomain.com/site-owner","Faisal"). This triple (x, P, y) can be represented as a logical formula P(x,y), where the binary predicate P relates the object x to the value y. In fact, RDF only supports binary predicates.

[3]http://www.w3.org/TR/xmlschema-2

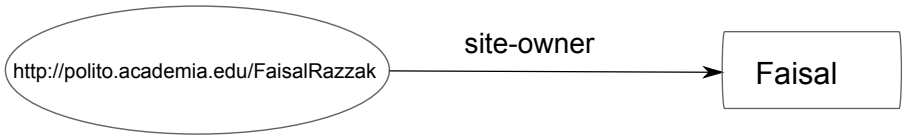


Figure 2.1. RDF Triple Structure

The second view of the statement is the graph model (Figure 2.1). It is a directed graph with labeled nodes and arcs; the arcs are directed from the resource (the subject of the statement) to the value (the object of the statement). This kind of graph is known in the Artificial Intelligence community as a semantic net.

Graphs are a powerful tool for human understanding, but the Semantic Web vision requires machine accessible and machine processable representations. Therefore, thee is a third representation possibility based on XML. According to this possibility, an RDF document is represented by an XML element with the rdf *rdf:RDF*. The content of this element is a number of descriptions, which use *rdf:Description* tags. Every description makes a statement about a resource.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:mydomain="http://www.mydomain.net/my-rdf-ns">
        <rdf:Description about="http://polito.academia.edu/FaisalRazzak">
                        <mydomain:site-owner>
                                        Faisal
                        </mydomain:site-owner>
        </rdf:Description>
</rdf:RDF>
```

2.2 Ontology (OWL)

To capture domain knowledge in a generic way, and provide a commonly agreed understanding of a domain, which may be reused and shared across applications and groups, the concept of *Ontology* is used. An ontology is a formal specification of a shared conceptualization [28]. W3C provides the Ontology web Language (OWL) to describe ontologies about a particular domain [29]. OWL has three increasingly-expressive sub languages: *OWL Lite*, *OWL DL*, and *OWL Full*.

2.2.1 Why OWL?

The expressiveness of RDF and RDFS is very limited (and this is a deliberate choice): RDF is roughly limited to model binary relationships and RDF-S is limited to sub-class hierarchies and property hierarchies, with restrictions on the domain and range of the lasts.

However a number of research groups have identified different characteristic use-cases for the Semantic Web that would require much more expressiveness than RDF and RDF-S offer. Initiatives from both Europe and United States came up with proposals for richer languages, respectively named OIL and DAML-ONT, whose merging DAML+OIL was taken by the W3C as the starting point for the Web Ontology Language OWL.

Ontology languages must allow users to write explicit, formal conceptualizations of domain knowledge, the main requirements are therefore:

- a well defined syntax,
- a formal semantics,
- an efficient reasoning support,
- a sufficient expressive power,
- a convenience of expression.

The importance of a well-defined syntax is clear, and known from the area of programming languages: it is a necessary condition for "machine understandability" and thus for machine processing of information. Both RDF/RDF-S and OWL have this kind of syntax. A formal semantics allows to describe the meaning of knowledge precisely. Precisely means that semantics does not refer to subjective intuitions and is not open to different interpretations by different people (or different machines). The importance of a formal semantics is well known, for example, in the domain of mathematical logic. Formal semantics is needed for allowing people to reason about knowledge. This, for ontologies, means that we may reason about:

- Class membership. If x is an instance of a class C, and C is a subclass of D, we can infer that x is also an instance of D.
- Equivalence of classes. If a class A is equivalent to a class B, and B is equivalent to C, then A is equivalent to C, too.
- Consistency. Let x be an instance of A, and suppose that A is a subclass of $B \cap C$ and of D. Now suppose that B and D are disjoint. There is a clear inconsistence in our model because A should be empty but has the instance x. Inconsistencies like this indicate errors in the ontology definition.

- Classification. If we have declared that certain property-value pairs are sufficient conditions for membership in a class A, then if an individual (instance) x satisfies such conditions, we can conclude that x must be an instance of A.

Semantics is a prerequisite for reasoning support. Derivation such as the preceding ones can be made by machines instead of being made by hand. Reasoning is important because allows to:

- check the consistency of the ontology and of the knowledge model,
- check for unintended relationships between classes,
- automatically classify instances.

Automatic reasoning allows to check much more cases than could be checked manually. Such checks become critical when developing large ontologies, where multiple authors are involved, as well as when integrating and sharing ontologies from various sources.

Formal semantics is obtained by defining an explicit mapping between an ontology language and a known logic formalism, and by using automated reasoners that already exist for that formalism. OWL, for instance, is (partially) mapped on description logic, and makes use of existing reasoners such as Fact, Pellet and RACER. Description logics are a subset of predicate logic for which efficient reasoning support is possible.

2.2.2 OWL in a nutshell

OWL documents are usually called OWL *ontologies*. Since they are built on RDF and RDFS, they are essentially RDF documents. The following sections explain key elements of an OWL document.

Header

The root element of an ontology is an `rdf:RDF` element, which specifies a number of namespaces:

```
<rdf:RDF
     xmlns:owl = "http://www.w3.org/2002/07/owl#"
     xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">
```

An OWL ontology can start with a set of assertions for house keeping purpose. These assertions are grouped under an `owl:Ontology` element, which contains comments, version control, and inclusion of other ontologies.

```
<owl:Ontology rdf:about="http://elite.polito.it/ontologies/simpleHomeEffects.owl">
     <rdfs:comment>A sample Ontology </rdfs:comment>
     <owl:priorVersion
          rdf:resource="http://elite.polito.it/ontologies/effectsOld.owl"/>
     <owl:imports
          rdf:resource="http://elite.polito.it/ontologies/effects.owl"/>
     <rdfs:label>Domotic Effects</rdfs:label>
</owl:Ontology>
```

The most important of the above assertions is the `owl:imports`, which lists other ontologies whose content is assumed to be part of the current ontology. It is important to be aware that the `owl:imports` is a transitive property: if the ontology A imports the ontology B, and the ontology B imports the ontology C, then A is also importing C.

Classes

Classes are defined using the `owl:Class` element and can be organized in hierarchies by means of the `rdfs:subClassOf` construct.

```
<owl:Class rdf:ID="associateProfessor">
     <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
</owl:Class>
```

It is also possible to indicate that two classes are completely disjoint such as the *associateProfessor*,*assistantprofessor* and the *Professor*, using the `owl:disjointWith` construct.

```
<owl:Class rdf:about="#associateProfessor">
     <owl:disjointWith rdf:resource="#assistantProfessor"/>
     <owl:disjointWith rdf:resource="#Professor"/>
</owl:Class>
```

Equivalence of classes may be defined using the `owl:equivalentClass` element.

```
<owl:Class rdf:ID="#faculty">
     <owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>
```

Eventually there are two predefined classes, `owl:Thing` and `owl:Nothing`, which, respectively, indicate the most general class containing everything in a OWL document, and the empty class. As a consequence, every `owl:Class` is a *subclass* of `owl:Thing` and a *superclass* of `owl:Nothing`.

Properties

In OWL are defined two kinds of properties:

- Object properties, which relate objects to other objects. Example are `isTaughtBy` and `supervises` relationships.
- Datatype properties, which relate objects with datatype values. Examples are `age`, `name`, and so on. OWL has not any predefined data types, nor does it provide special definition facilities. Instead, it allows the use of XML-Schema data types.

Here there are two examples, the first for a Datatype property while the second is for Object properties:

```
<owl:DatatypeProperty rdf:ID="age">
     <rdfs:range rdf:resource="&xsd;#nonNegativeInteger"/>
</owl:DatatypeProperty>


<owl:ObjectProperty rdf:ID="isTaughtBy">
     <rdfs:domain rdf:resource="#course"/>
     <rdfs:range rdf:resource="#academicStaffMember"/>
</owl:ObjectProperty>
```

Restrictions on properties

In RDFS it is possible to declare a class C as a subclass of a class C', then every instance of C will be also an instance of C'. OWL allows to specify classes C' that satisfy some precise conditions, i.e., all instances of C satisfy the conditions. This is done by defining C as a subclass of the class C'' which collects all the objects that satisfy the conditions. In general, C'' remains anonymous. In OWL there are three specific elements for defining classes basing on restrictions, they are `owl:allValuesFrom`, `owl:someValuesFrom` and `owl:hasValue`, and they are always nested into a `owl:Restriction` element. The `owl:allValuesFrom` specify a universal quantification ($\forall$). For example, the following element requires first-year courses to be taught by Professors only.

```
<owl:Class rdf:about="#firstYearCourse">
     <rdfs:subClassOf>
          <owl:Restriction>
               <owl:onProperty rdf:resource="#isTaughtBy"/>
               <owl:allValuesFrom
               rdf:resource="#Professor"/>
          </owl:Restriction>
     </rdfs:subClassOf>
</owl:Class>
```

The `owl:someValuesFrom` defines an existential quantification ($\exists$). For example, *there exist* an undergraduate course taught by an instance of the class of academic staff members (existential quantification).

```
<owl:Class rdf:about="#academicStaffMember">
     <rdfs:subClassOf>
          <owl:Restriction>
               <owl:onProperty rdf:resource="#teaches"/>
               <owl:someValuesFrom
               rdf:resource="#undergraduateCourse"/>
          </owl:Restriction>
     </rdfs:subClassOf>
</owl:Class>
```

The `owl:hasValue` defines a specific value that the property must have. In general an `owl:Restriction` element contains an `owl:onProperty` element and one ore more restriction declarations. Restrictions for defining the cardinality of a given class are also supported through the elements:

- `owl:minCardinality`,
- `owl:maxCardinality`,
- `owl:Cardinality`.

The latter is a shortcut for a cardinality definition in which `owl:minCardinality` and `owl:maxCardinality` assume the same value.

Special properties

Some properties of the property element can be defined directly:

- `owl:TransitiveProperty` defines a transitive property, such as “has better grade than”, “is older than”, etc.

- `owl:SymmetricProperty` defines a symmetric property, such as "has same grade as" or "is sibling of".
- `owl:FunctionalProperty` defines a property that has at most one value for each object, such as "age", "height", "directSupervisor", etc.
- `owl:InverseFunctionalProperty` defines a property for which two different objects cannot have the same value, for example "is identity ID for".

Instances

Instances of classes, in OWL, are declared as in RDF:

```
<rdf:Description rdf:ID="160850">
     <rdf:type rdf:resource="#academicStaffMember"/>
</rdf:Description>
```

OWL, unlike typical database systems, does not adopt a *unique-names assumption* therefore two instances that have different names are not required to be actually two different individuals. Then, to ensure that different individuals are recognized by automated reasoners as such, inequality must be explicitly asserted.

```
<lecturer rdf:ID="160850">
     <owl:differentFrom rdf:resource="#187833"/>
</lecturer>
```

Because such inequality statements frequently occur, and the required number of statements would explode for stating the inequality of a large number of individual, OWL provides a shorthand notation to assert the pairwise inequality for all the individuals in a list: `owl:AllDifferent`.

```
<owl:AllDifferent>
     <owl:distinctMembers rdf:parseType="Collection">
          <lecturer rdf:about="#160850"/>
          <lecturer rdf:about="#187833"/>
          <lecturer rdf:about="#160596"/>
     </owl:distinctMembers>
</owl:AllDifferent>
```

Note that `owl:distinctMembers` can only be used in combination with the `owl:AllDifferent` element.

2.3 SPARQL

The SPARQL Protocol and RDF Query Language (SPARQL) is a query language and protocol for RDF [8]. The protocol allows a SPARQL endpoint which acts as a gateway for RDF knowledge base. On the other hand, the query language is used to retrieve and manipulate data stored in RDF format. SPARQL permits four kind of queries, i.e., SELECT, ASK, CONSTRUCT and DESCRIBE queries. SELECT queries is used to extract raw values from the RDF information and the results are returned in a table format. CONSTRUCT query is used to extract information from the SPARQL endpoint and transform the results into valid RDF. ASK query is used to provide a simple True/False result for a query on a SPARQL endpoint. DESCRIBE query is Used to extract an RDF graph from the SPARQL endpoint, the contents of which is left to the endpoint to decide based on what the maintainer deems as useful information.

SPARQL is based on matching graph patterns against RDF graphs. In order to define graph pattern, we must first define triple patterns. A *triple pattern* is like an RDF triple, but with the option of a variable in place of RDF terms, i.e., IRIs, literals or blank nodes, in the subject, predicate or object positions. For example, "?title" represents a variable in the triple pattern shown in Figure 2.2.

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title> ?title .
```

Figure 2.2. An example of Triple pattern

There are four elementary graph patterns over which group graph patterns can be defined. They are i) BASIC graph pattern ii) FILTER graph pattern iii) OPTIONAL graph pattern iv) ALTERNATIVE graph pattern .

A *BASIC* graph pattern (BGP) is a set of triple patterns written as a sequence of triple patterns (separated by a period if necessary). A BGP is understood as the conjunction of its triple patterns. See Figure 2.3 for a BGP example.

```
?x foaf:name ?name . ?x foaf:mbox ?mbox
```

Figure 2.3. An example of Basic Graph Pattern

A group graph pattern is a set of graph patterns delimited with braces **{}**. Figure 2.4 shows examples of group graph patterns. All of them are equivalent since they are only made of BGPs and therefore, these patterns are interpreted conjunctively.

```
{ ?x foaf:name ?name . ?x foaf:mbox ?mbox }

or

{ ?x foaf:name ?name . ?x foaf:mbox ?mbox . }

or


{ { ?x foaf:name ?name . }
{ ?x foaf:mbox ?mbox . } }
```

Figure 2.4. Examples of Group Graph Pattern

A SPARQL query matches variables against its values in the data. Consider the RDF data shown in Figure 2.5. The SPARQL SELECT query (Figure 2.6) matches the values of the variables request in the query construct (*?name and ?mbox*) from the RDF data. The result of the query is shown in Figure 2.7.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Johnny Lee Outlaw" .
_:a foaf:mbox <mailto:jlow@example.com> .
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

Figure 2.5. Sample RDF data

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name . ?x foaf:mbox ?mbox }
```

Figure 2.6. SPARQL Query

The *FILTER* construct restricts variable bindings to those for which the filter expression evaluated to **TRUE**. Figure 2.9 shows a SPARQL SELECT query with a filter construct, over the data (Figure 2.8). The result is shown in Figure 2.10. Another example is shown in Figure 2.11.

The *OPTIONAL* graph pattern matching allows information to be added to the answer where the information is available, but do not reject the answer because

```
?name                ?mbox
-----------------------------------------------
Peter Goodguy        <mailto:peter@example.org>
Johnny Lee Outlaw    <mailto:jlow@example.com>
```

Figure 2.7. Result of the SPARQL query

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/book/> .
@prefix ns: <http://example.org/ns#> .
:book1 dc:title "SPARQL Tutorial" .
:book1 ns:price 42 .
:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
```

Figure 2.8. Sample RDF data

some part of the query pattern does not match. if the optional part does not match, it creates no bindings but does not eliminate the solution. Figure 2.12 shows an example of SPARQL SELECT query with OPTIONAL graph pattern and its associated data and results.

SPARQL provides a means of forming the disjunction of graph patterns so that one of several *ALTERNATIVE* graph patterns may match. If more than one of the alternatives match, all the possible pattern solutions are found. Pattern alternatives are syntactically specified with the keyword **UNION**. Figure 2.13 and Figure 2.14 show examples of SPARQL SELECT query with UNION graph pattern and its associated data and results.

For more detailed insight on SPARQL, readers are referred to [8, 30–32].

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x ns:price ?price .
FILTER (?price < 30.5)
?x dc:title ?title . }
```

Figure 2.9. SPARQL Query with FILTER construct

```
?title                  ?price
-------------------------------
The Semantic Web        23
```

Figure 2.10. Result of the SPARQL query

```
Query:

PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
FILTER regex(?title, "^SPARQL")
}

 Result:

  ?title
-----------------------
SPARQL Tutorial
```

Figure 2.11. An example of FILTER Construct

2.4 Linked Data

According to Tim Berners-Lee's web architecture note [9], the Semantic web is not just about exposing machine understandable information over the Internet but making links between different exposed information sets, so that a machine, an application, a service or a person can find related information. The availability of data from different sources in a universal format and linked together is known as 'Linked Data'. Linked Data[4] (LD) assumes that information is available in RDF format.

The term Linked Data refers to a set of best practices for publishing and interlinking structured data on the Web. Tim Berners-Lee presented following Linked Data principles:

1. Use URIs as names for things.
2. Use HTTP URIs, so that people can look up those names.

[4]http://linkeddata.org

```
Data:

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntaxns#>
.
_:a rdf:type foaf:Person .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@example.com> .
_:a foaf:mbox <mailto:alice@work.example> .
_:b rdf:type foaf:Person .
_:b foaf:name "Bob" .

Query:

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
OPTIONAL { ?x foaf:mbox ?mbox }
}

Result:


?name                ?mbox
_____________________________________________
Alice          <mailto:alice@example.com>
Alice          <mailto:alice@work.example>
Bob
```

Figure 2.12. An example of OPTIONAL graph pattern

3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).

4. Include links to other URIs, so that they can discover more things.

Linked Data builds directly on Web architecture [33] and applies this architecture to the task of sharing data on global scale. it intends to transform the current Web in to a Web of Data. For more details, readers are referred to [10, 34].

```
Data:
@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .
_:a dc10:title "SPARQL Query Language Tutorial" .
_:a dc10:creator "Alice" .
_:b dc11:title "SPARQL Protocol Tutorial" .
_:b dc11:creator "Bob" .
_:c dc10:title "SPARQL" .
_:c dc11:title "SPARQL (updated)" .


Query:

PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { { ?book dc10:title ?title }
UNION
{ ?book dc11:title ?title }
}

Result:

?title
-----------------------------------


SPARQL Protocol Tutorial
SPARQL
SPARQL (updated)
SPARQL Query Language Tutorial
```

Figure 2.13. An example of UNION graph pattern

```
Data:

@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .
_:a dc10:title "SPARQL Query Language Tutorial" .
_:a dc10:creator "Alice" .
_:b dc11:title "SPARQL Protocol Tutorial" .
_:b dc11:creator "Bob" .
_:c dc10:title "SPARQL" .
_:c dc11:title "SPARQL (updated)" .


Query:

PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?author ?title
WHERE { { ?book dc10:title ?title .
?book dc10:creator ?author . }
UNION
{ ?book dc11:title ?title .
                        ?book dc11:creator ?author . }
}


Result:
?author    ?title
-------------------------------------------

Alice     SPARQL Query Language Tutorial
Bob       SPARQL Query Language Tutorial
```

Figure 2.14. An example of UNION graph pattern

Chapter 3

DogOnt & Dog

The work presented in this thesis is not an isolated research effort, but it is a work well integrated into a more general theme of research that is taking place in the e-Lite research group of the Turin's Polytechnic. The work, both user intelligible goals and semantic data exchange, builds upon two important contribution made by former members of the e-Lite research group. The first is the DogOnt [35] ontology which models the structure of an environment (In particular, a home). The second is an ontology driven home automation system, called Domotic OSGi Gateway (Dog) [36]. Therefore, before discoursing on the role of semantic web technologies in smart environments, this chapter briefly defines the technologies used to emulate a smart environment.

3.1 DogOnt

DogOnt ontology provides the semantic core to model the domotic system of an environment (house). It focuses on representing devices, appliances, states and functionalities. Modularity is exploited for integrating the core set of modeling primitives with additional features for energy profiling, user modeling, privacy management, etc. DogOnt is organized along eight main hierarchies of concepts (see Figure 3.1) respectively rooted at:

1. "Building Environment": models structural elements of the environment. For instance building, flat, garage, garden, rooms (bathroom, bedroom, dining-room, kitchen, living-room, etc);

2. "Building Things": models physical objects (devices) present in the environment. The physical objects may be electrically controllable or uncontrollable. The electrically controlled devices (like coffee maker, boiler, cooker, fan, lamp, actuator, sensors and various others) are categorized under the Controllable

category. Physical objects (like table, sofa, wall, ceiling and others), which can not be electrically controlled are categorized under the Uncontrollable category.

3. "Functionality": models different functionalities provided by the controllable devices. DogOnt further classifies them as Control Functionality, Notification Functionality or Query Functionality. Control functionalities are the actions that devices can perform (like a Lamp has "on" and "off" functionalities). Most controllable devices have the ability to send a notification back, as an acknowledgment of the completion of the assigned task. These notification capabilities are modeled under the Notification functionalities classification. For performing the task intelligently, usually, it is required to have a mechanism through which the status of devices can be queried at any time, which are modeled under the Query functionalities classification;

4. "Commands": Controllable devices perform their Control functionalities by receiving some particular commands, whose modeling and classification is performed under this category;

5. "Notification": Controllable devices send different types of notifications, such as notifying when their state changes, which are modeled and classified under this category;

6. "State" and "State Values": At any instant, a controllable device possesses an internal state, which is modeled as a set of orthogonal state spaces (called "States"), with different "State Values" each. A complete description of the state of a device is therefore a valid State Value for each of the States defined for that device. State Values can be discrete (e.g., in Lamp "onState" and "offState") or continuous within a specific range of values (e.g, a DimmerLamp device has a "Light-Intensity-State" whose value ranges from 0% to 100%).

7. "Domotic Network Component": Controllable devices adopt widely different network protocols. This modeling dimension describes the protocol characteristic and network addressing scheme.

3.1.1 Device Modeling in DogOnt

According to the DogOnt classification dimensions, each device is modeled by creating instances for all relevant DogOnt classes, and according to the ontology constraints. The device modeling process is briefly explained by illustrating the model of a "Dimmer Lamp" device (Figure 3.2), which is a subclass of "Lamp" and "Controllable", and has all inherited features of these super classes.

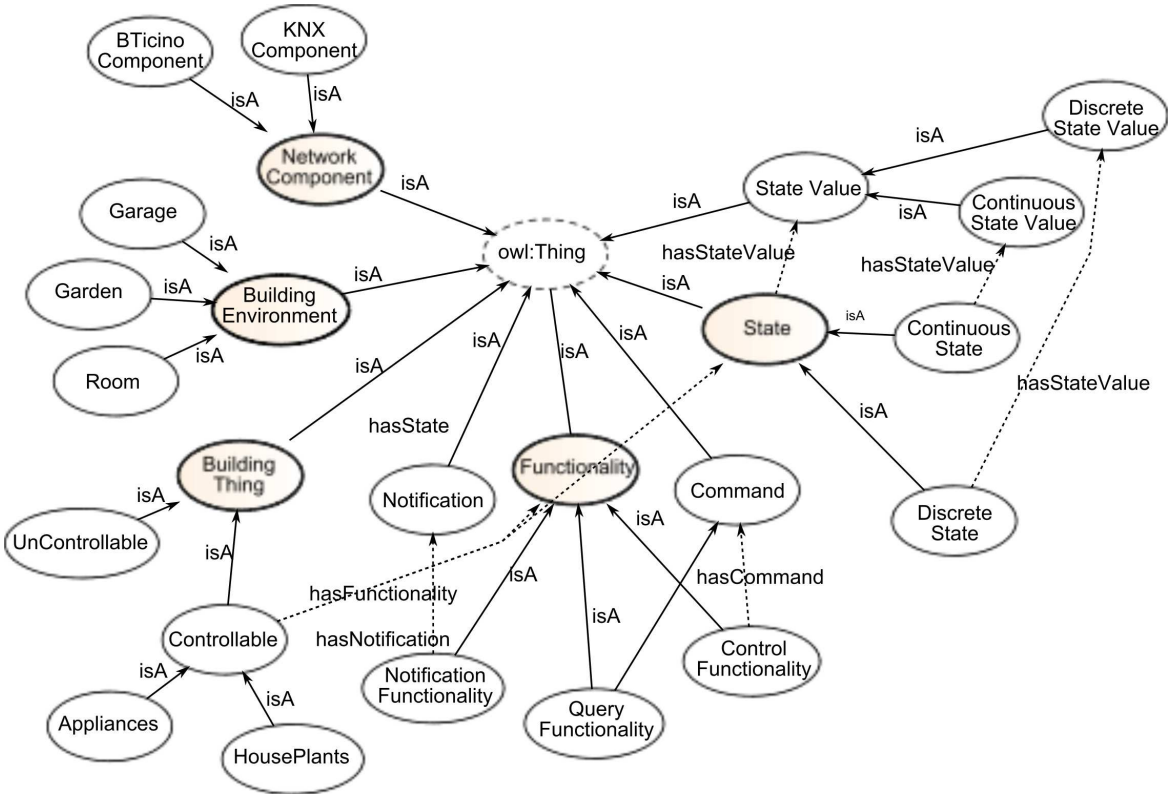


Figure 3.1. DogOnt top-level concepts.

A dimmer lamp has all the functionalities which a “Lamp” can hold, such as, it can be (switched) “on” and “off”, and can be placed at a certain location in the IE. Furthermore, “Dimmer Lamp” has an extra “Light-Regulation-Functionality” by which the “Light-Intensity” of the lamp can be managed. The value of “Light-Intensity” ranges from 0 to 100. With the “Light-Regulation-Functionality” it may be increased or decreased with a step 10 through “stepUp” or “stepDown” commands respectively, or it may also be directly set to a specific value with “set(value)” command. As “Dimmer Lamp” is a type of Controllable device, two more functionalities are inherited from the class of “Controllable”, these are “Query-Functionality” and “State-Change-Notification-Functionality”.

3.2 Domotic OSGi Gateway

Domotic OSGi Gateway (Dog) is an ontology-powered home automation gateway. It is empowered by the DogOnt ontology and therefore, it is able to expose different

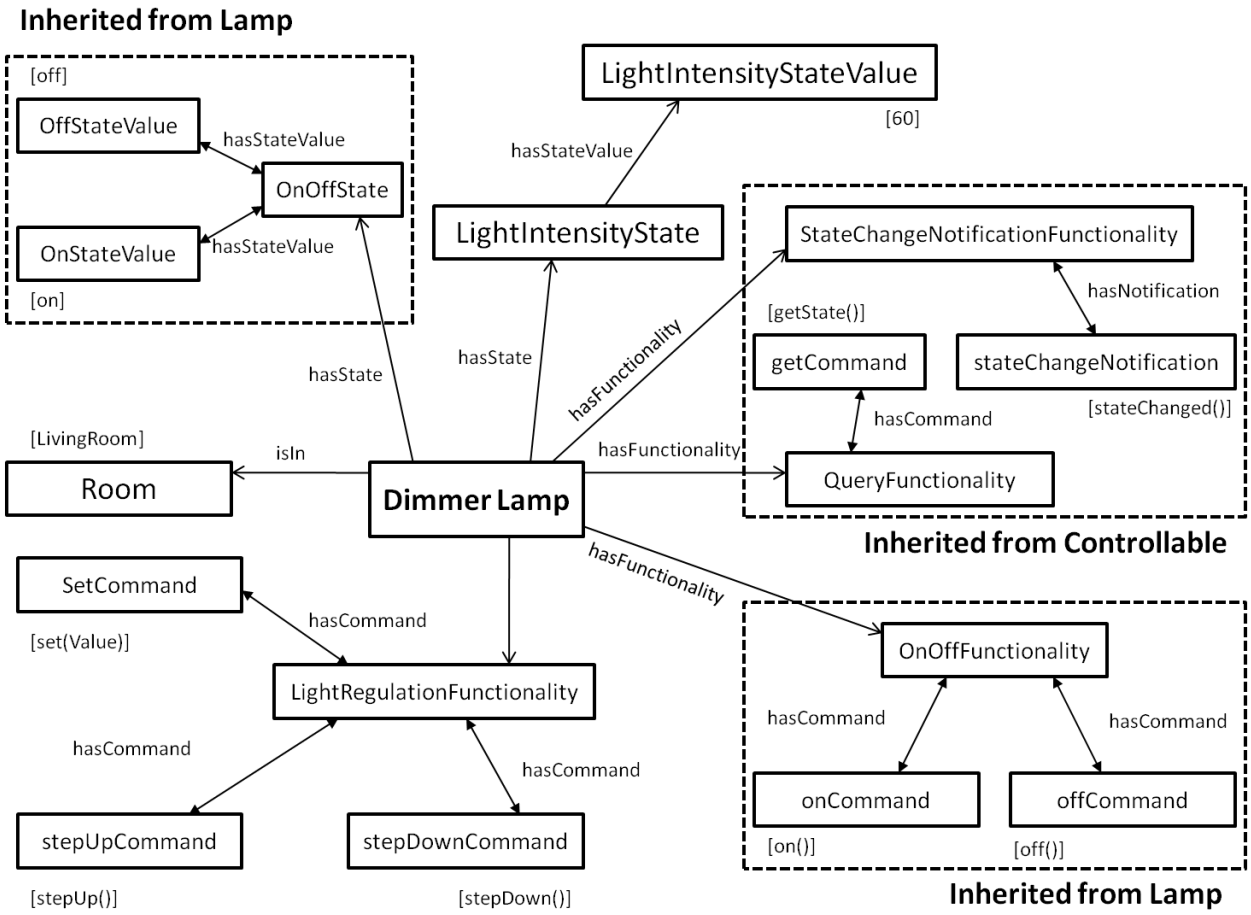


Figure 3.2. Dimmer Lamp in DogOnt

domotic networks as a single and technology neutral home automation system. Dog is versatile as it is built on top of the OSGi framework[1] and the adoption of semantic modeling techniques allows Dog to support intelligent operations inside the home environment. It has a modular architecture (Figure 3.3) and it is divided in five main categories: API, Device Management, Device Control, StartUp and Library.

3.2.1 Api

This category includes bundles offering technology independent programming interfaces for both external applications and OSGi-based plugins.

[1] http://www.osgi.org

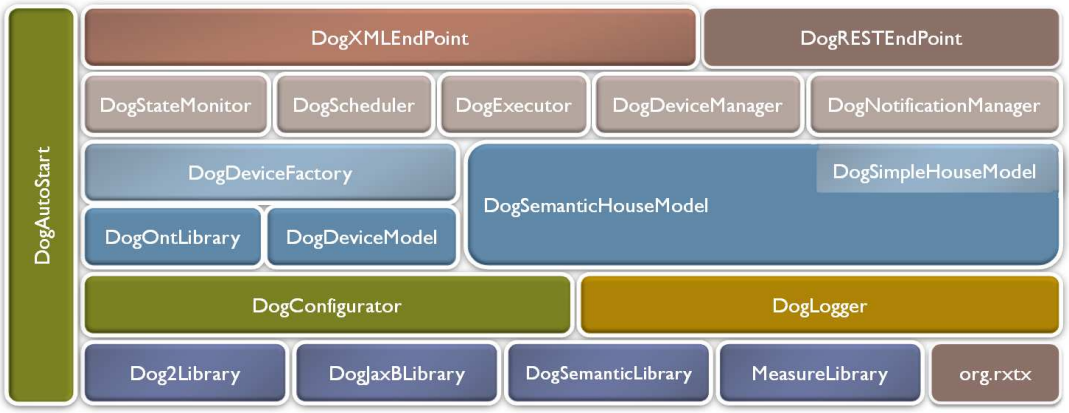


Figure 3.3. Dog core bundles

DogRESTEndPoint: provides a REST endpoint for services offered by Dog. It is based on JSON and XML bases messaging system, thus enabling DOG access for non-OSGi or non-Java applications.

DogXmlEndPoint: provides an XML-RPC endpoint for services offered by the DogApi bundle, thus enabling DOG access for non-OSGi or non-Java applications. it helps retrieve the house configuration, send commands to devices managed by DOG and receive house events.

3.2.2 Device Control

This category includes bundles dedicated to device control and monitoring. They encompass:

DogStateMonitor: provides information about the current states of the devices connected to the Dog platform. It keeps the snapshot of the states of all the devices. The device state notification is compliant with the OSGi Monitor Admin Service Specification [37].

DogExecutor: allows the other Dog bundles to execute commands on devices. Executing commands means calling methods of the DogDeviceModel classes. Thanks to DogSemanticHouseModel, DogExecutor can verify both the syntax and semantics of received commands. In the first case command compliance with the corresponding ontology definition is analyzed (e.g., a simple lamp can only accept an *on* command with no parameters), whereas in the latter the received commands are checked against the set of allowed commands deduced from the DogOnt device functionalities (e.g., a door cannot be switched on).

DogScheduler: offers a centralized service for scheduling both command execution and device monitoring jobs. DogScheduler exploits the DogExecutor and DogStateMonitor services.

3.2.3 Device Management

The Device Management category groups all the bundles needed to comply with the *Device Access Specification Version 1.1* defined in the OSGi Service Compendium [37]. This specification defines the logic and the entities (i.e., bundles) that must be designed and implemented by Dog to support automatic detection and attachment of existing devices and to assist the dynamic plugging of new devices. It comprises:

DogDeviceManager: detects registration of Device services and associates these devices with an appropriate Driver service.

DogNotificationManager: dispatches notification and state change notifications. It is based on the publisher subscriber model and it filters inner state change notifications from outer ones (visible to applications).

DogDeviceFactory: creates device instances according to the runtime home configuration (defined using DogOnt). It is usually provided by *DogSemanticHouseModel*.

DogOntLibrary: is programmatically generated from DogOnt. It consists of all device interfaces, functionalities classes, state classes etc.

DogDeviceModel: is the Dog object representing DogOnt-defined device classes. It is a software proxy of a physical device that can be attached by a Driver service.

DogSemanticHouseModel: can be seen as the Dog nervous system. It exploits standard DogOnt classes and DogOnt instances referred to a specific IDE environment to provide knowledge-rich access to the environment properties and capabilities. Inference and reasoning tasks are carried at this level, both at the gateway start-up, for computing consistency checking and transitive closure, and at runtime. In particular, runtime reasoning is adopted for generating inter-operation rules and pre-defined scenarios, and for dealing with unknown devices through classification reasoning thus detecting compatible device types on the basis of formalized capabilities (`dogont:Functionality`). A SPARQL endpoint allows Dog to be extended by additional knowledge-based policies offering un-filtered query access to the whole DogOnt IDE model. This allows, for example, to support advanced policies based on additional context information, e.g., energy saving policies.

Dog exploits semantics mainly through the *DogSemanticHouseModel* bundle, which manages all DogOnt related tasks, for supporting the device/driver attachment paradigm. In case of micro-Dog installations where the computational power of the hardware running the gateway is too low for supporting on-line ontology management and inference, the *DogSemanticHouseModel* bundle is replaced by a *SimpleHouseModel* bundle that encodes static ontology information (queried from the ontology at configuration time); in this case all inference tasks are carried off-line, thus trading-off computational power with support for on-line inference.

3.2.4 StartUp

This category comprises of:

DogConfigurator: manages bundle-specific configurations. For instance, specific property files, XML files and/or Additional files (ontology, images, etc.). It provides startup configuration to all bundles of Dog.

DogLogger: provides logging facilities to all Core bundles.

3.2.5 Library

The category consists of bundles that act as simple repositories of classes and interfaces needed by the other Dog bundles. *Dog2Library* defines all the message types for inter-bundle communication. It also provides utility classes to other bundles. *DogJaxBLibrary* provides XML serialization / de-serialization for all message types. *DogSemanticLibrary* encapsulates and makes available all semantics-related libraries like Jena, Pellet, SPARQL query facilitator etc. *MeasureLibrary* exports the JScience library[2] to all Dog bundles.

[2]http://jscience.org

Part II

User Intelligible Goals

Chapter 4

State of the art

In this part of the dissertation, a unified approach based on user goal modeling is presented (Figure 4.1). The approach addressing the concerns of both AmI designers and end-users. User goal modeling is a higher level design for user interaction and control, which has received little attention till now, as acknowledged in [15].

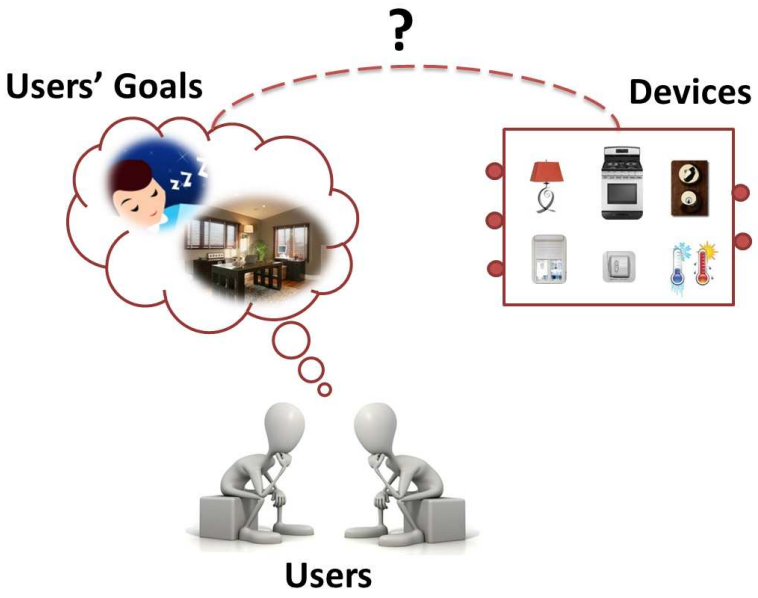


Figure 4.1. User Goal Modeling

In 2006, DavidOff et al. [38] narrated that personal spaces, such as home play an important role in group and individual self-definition: rather than just using personal spaces for a specific function, users pour their personalities and lives in the way they use and transform their personal environments. The outcome was the focus

on developing user programmable spaces. Such spaces either focused on allowing users to control and manage their environments or learning users' preferences, as the environments become populated with computational elements.

Garcia-Herranz et al. [39, 40] proposes an application-independent indirect control programming system to program complex behaviors with the simplicity required to allow novice users to program their smart environments. The motive is to allow users to create powerful and personal behavior without expert assistance.

In [41] an Artefact framework is proposed which allows end users to deploy ubicomp systems easily in a Do-it-yourself fashion. Secondly, it allows developers to write applications and to build augmented artefacts in a generic manner. The Artefact framework provides a layered architecture where basic artefact functionalities are combined in a core component. Additional augmented features can be added as plugins into the core. Each augmented feature is called a profile. Each profile defines a specific functionality and implements the underlying logic of the functions, e.g., room temperature, lamp brightness.

Katasonov [42] motivates to build *Digital fluency* in smart environments by enabling the non programmers to design, create and modify their smart environments. The paper proposes a higher level of abstraction in application design, on-the-fly development, flexibility with respect to adding new devices and software components.

Rashidi et al. [43] proposes a software architecture which incorporates learning techniques to discover patterns in user's daily activities. While the patterns of user's activities are observed and stored, the user can also define their own activity patterns as well. The activity pattern are observed as changes in states of devices occur in the house. Similarly, Salomons et al. [44] introduces a generic model for intelligent homes that describes the current state, the target state and the transition. The model is based on storing preferences on individual devices.

Cheng et al. [45] proposes a smart home reasoning system called ASBR system. The system learns user's preferences by adaptive history scenarios and put forwards a way to rebuild reasoned knowledge in other smart homes. They propose that contextual information can be extracted and reasoned as a set of scenarios. In addition, the system can derive personalize habits and store them in OWL files. Similarly, Dey et al. [46] proposes a software infrastructure solution to detect the current states of the environment (called Context) and take action based on it. The infrastructure is focused on developing context aware applications.

All the aforementioned approaches focused on user programmable spaces, but suffered from a "device-centric" vision. The "device-centric" vision limited the ability of the users to program their environments in terms of specific devices and their functionalities. In 2003, the Ambient Intelligence (AmI) community had identified key research directions for building intelligence in different environments, i.e., homes, offices, schools and control centers [47]. These directions include the need

to develop and innovate new concepts and abstract models to address heterogeneity, intelligence, innovative interaction management techniques and human centric expressions of personal style. Though the role of users in personal environments is important, the role of AmI designers in order to build intelligence can not be ignored. In 2008, Doorn et al. [48] carried a series of workshops and interviews, and concluded that designers work top-down and like to start from abstract vague descriptions; therefore, approaches having device-centric vision limit the ability of AmI designers to design, implement and test general algorithms and solutions that might apply to a range of different environments, especially in large and complex buildings.

In [49,50] a goal based interaction has been proposed, and extended in [51], that takes a user's goal and finds a path achieving the goal.

Hemrik Dibowski et al. [52] proposes an automatic design approach for large building automation systems (BAS). The top-down approach initiates by defining the structure of the building, then the system integrators define requirements using ontologies.

A neuro-cognitive model for Environment Recognition, Decision-making, and Action execution inside automated buildings is proposed in [53–55]. They introduce separate models for perception known as Artificial Recognition System-PerCeption (ARS-PC) and decision making, identified as Artificial Recognition System-PsychoAnalysis (ARS-PA).

D-HTN [56] is a planning system for AmI applications, based on the hierarchical task network (HTN) approach, that is able to find courses of actions to address given goals.

Though [49, 50, 52–56] provide abstract design approaches for smart environments, the ability to allow users to program their environments is lacking.

In the literature the concerns of AmI designers and end-users are mostly addressed separately. Therefore, their exists a need for developing a unified approach that can address the issues of both AmI designers and end-users. On one hand, the approach should allow a user to program his/her own personal environment and on the other hand, the approach should allow AmI designers to work on an abstract level without focusing on a specific environment.

Chapter 5

Domotic Effects Framework

The last decade saw the emergence of economically viable and efficient sensor technology which can be integrated with appliances, enabling them to sense different parameters of their respective environments, i.e., temperature, luminosity, pressure etc. It helped realizing the vision of of smart environments [12] by developing heterogeneous dynamic ensembles: groups of co-located devices of different types which evolve over time [14]. Such environments promise to offer additional intelligent capabilities that go beyond the integrated and remote control of appliances present in the environment. But the presence of diverse devices and the associated complexity has given rise to a major problem in the past years, i.e., the problem of providing users with the ability to control and manage their respective environments.

State of the art revolves around the issues related to communication protocols and technologies [41, 43, 46]. Many approaches are furthermore based on abstract modeling of smart devices, resorting to some knowledge representation tool (e.g., ontologies [35,49,50]), but the research trend is moving from a traditional device-centric vision (bottom-up) to a vision of providing higher level design for user interaction and control [45, 47, 56, 57], i.e., user goal modeling. However, this research trend has received little attention as acknowledged in [15].

"Domotic Effects" (DE) framework provides a 3-tiered ontology driven modeling technique that models user intentions or goals (Figure 5.1). It addresses the concerns from perspectives of the AmI designer and the residents. It provides AmI designers with an abstraction layer that enables the definition of generic goals inside the environment, in a declarative way, and that can be used to design and develop intelligent applications. It provides a general framework for expressing functional properties, in a domain-dependent way: for each application domain, the AmI designer may choose the most suitable representation, and define suitable functional operators. Using these operators, various user goals are then defined in a specific environment. The high-level nature of the Domotic Effects, on the other hand, also allows the residents to program their personal, office or work spaces as they see fit:

they can define different achievement criteria for a particular generic goal, by using the domain-specific operators defined in the previous phase.

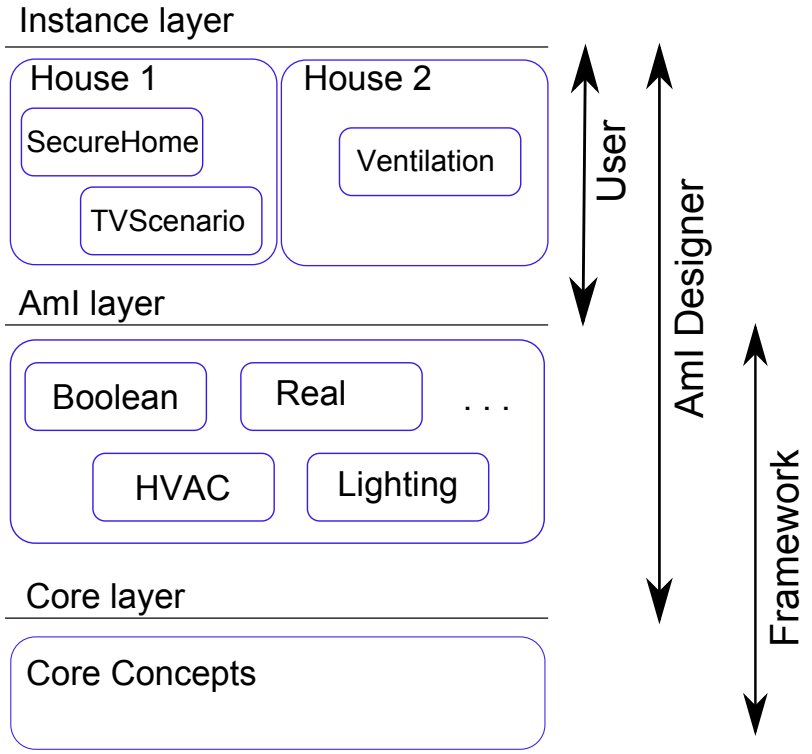


Figure 5.1. The DomoticEffects framework - Logic Architecture.

Every device in the environment is capable of providing certain visible (perceivable) effects for a user. These effects are fulfilled by possible states of the device. For example, an effect of *illumination* can be provided by a lamp in "ON" state. However, modern devices are complicated in nature and a single device can have a composite state, which may be modeled as concurrent sub-states. These sub-states are orthogonal regions combining multiple descriptions of a device. For example, a TV set may have an on-off state (with possible values On or Off), a volume state (with possible values 0 through 100), a channel state (with possible values depending on the set of programmed channels). A device state is therefore composite in nature and therefore it is modeled as the parallel composition of different sub-states.

There might be cases in which an effect can only be fulfilled by a combination of devices having particular states. For example, the effect of *securing* a building may require all the exit doors and windows to be closed. In the context of DE framework, an effect that depends upon a single device (having a state or sub-states) is called a simple effect (SE) and an effect dependent on a combination of devices (having particular states and sub-states) is called a complex effect (CE). A CE is described

by combining SEs and other CEs.

For discussion in this thesis, the applicability of the DE framework to Boolean application domains is considered, i.e., domains in which user goals (effects) can either be true (active) or false (inactive) depending on the value(s) of the involved states and sub-states, that may be Boolean, discretely enumerate or real-valued. This covers most control applications and many monitoring use cases in smart homes, offices and industrial plants.

The chapter is divided into two sections. Section 5.1 defines requirements for modeling human-intelligible effects (goals) and Section 5.2 introduces Domotic Effects and their formalization.

5.1 Requirements

Domotic Effects (DE) provide abstraction for modeling current (state) and future (goals) configurations of a smart environment. These configurations shall be expressed in a human-intelligible way and must support machine-based evaluation, solution and activation, with almost no human intervention. Formally, effects can either be *simple* or *complex*. Simple effects (SEs) are the terminal nodes of this functional representation and they correspond to functions applied to the state (or value) of a single device (sensor). Complex effects (CEs), on the other hand, derive from the application of domain-specific operators to SEs and other CEs. This permits the definition of modular and stratified effects built on top of simpler ones (Figure 5.2 and Figure 5.3).

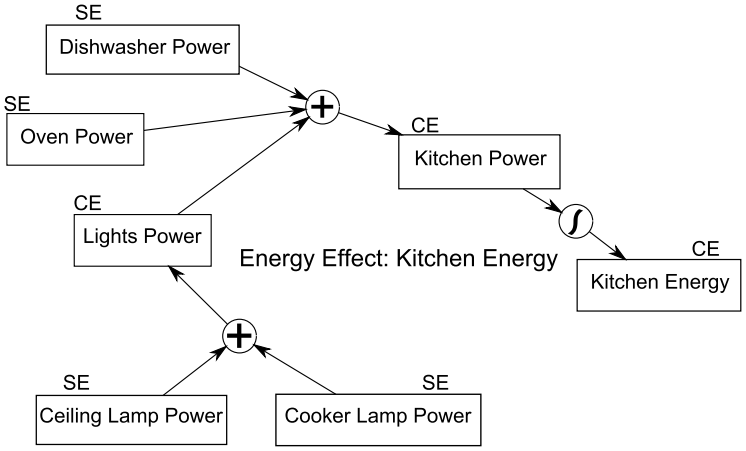


Figure 5.2. SE and CE effects in the energy domain

Simple and complex effects must fulfill well defined requirements to effectively

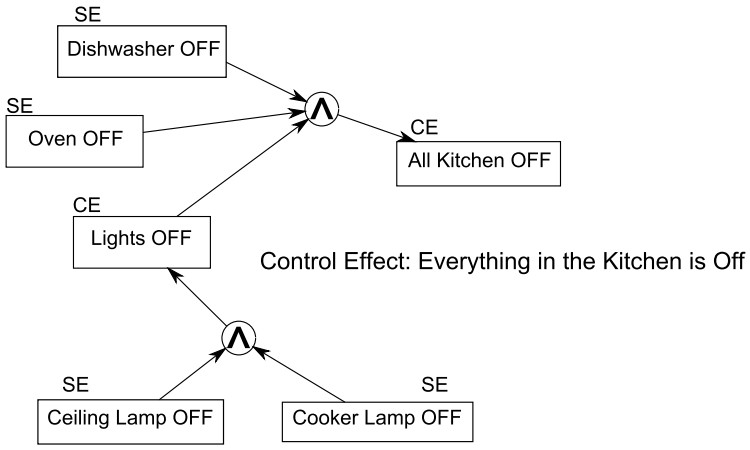


Figure 5.3. SE and CE effects in the control domain

address the representation issues discussed in Chapter 6. Such requirements encompass:

- **Formal definition**. Effects must have a formal, machine understandable specification thus enabling automatic elaboration and mapping into set of device states/activations; the same applies to operators, which are required to be machine processable.

- **Domain dependency**. Effects must be composable in different ways depending on the knowledge and/or application domain in which they are defined. As an example, real-valued effects (e.g., energy or power effects) will undergo operations such as aggregation of values (sum, average, last in a time window), derivation/integration (to convert energy to power and back), etc, as in Figure 5.2. Instead, boolean-valued effects (e.g., active/not-active) might be aggregated by straightforward boolean operators (and, or, not, etc.), as in Figure 5.3.

- **Modularity**. Whilst a restricted core set of modeling primitives is required to set up a sound representation framework, modeling shall be adaptable to different needs, i.e., knowledge domains, and to different actors such as AmI designers and home inhabitants.

- **Evaluation support**. Evaluating effects means computing their amount starting from the values of each device state and sensor value, according to the definition of the effects (SEs) and the semantics of the operators (that might change depending on the domain). The effect values must be updated whenever a device/sensor changes its state/value, therefore evaluation shall be

quick, i.e., comparable with the latency of home automation systems (roughly under one second), and well integrated with the smart environment management system (e.g., an home gateway). Different evaluation algorithms and computational models may be applied depending on the involved physical values and on the modeled operators.

- **Enforcement support**. Human-defined effects should be translated automatically into sequences of device activations (commands). This requires the inverse computability of the effect definition, which typically results in many, alternative solutions. Exploration of the solutions space may also enable optimization of second-level metrics, such as reducing the number of device switches or the energy consumption.

- **Advanced intelligence support**. A new breed of intelligent applications will generate and/or exploit domotic effects. For example, advanced in-home interfaces might exploit effects to offer "intelligent widgets" bound to effects instead of devices. Semantic reasoning might exploit effects to infer common automation recipes applicable to different smart environments. Hybrid and probabilistic reasoning might empower (semi-)automatic selection of alternative home configurations, e.g., raising shutters instead of switching on lamps, or vice-versa, depending on the values of defined effects. Context information might be defined on top of effects instead of devices, supporting higher level policies and behaviors.

- **Human-intelligibility**. While effects shall be uniquely identifiable (e.g., by using URIs), final users shall be able to give meaningful personalized names, to easily identify and understand the represented configurations. Moreover their formal definition shall be informative, i.e., the mechanisms with which SEs are composed into CEs must be easy to understand, even for unskilled users.

- **End user accessibility**. Effects shall be easy to define and to understand by end users, e.g., home inhabitants, through intuitive GUIs [58]. This allows bridging the gap between device-centric and interaction-centric modeling providing means for users to explicitly inform AmI about the sequences of effects (or goals) that activities require.

5.2 Formalism

Given an intelligent environment, we define as $\mathcal{D}$ the set of installed *controllable devices* $d \in \mathcal{D}$. Each device is characterized by a *device category* that, among the

other things, defines the allowed *sub-states* for a device. Depending on the device category, for each device d we define the set of allowed sub-states $S(d)$; this set may be discrete (e.g., {On, Off} for a lamp) or continuous (e.g., [0, 100] for a volume knob). During system evolution, the actual state of each device is a time-dependent function $s(d,t) \in S(d)$. The whole environment therefore possesses a *global state space* $\mathcal{G}$, represented by the Cartesian product of all device state spaces: $\mathcal{G} = \prod_{d \in \mathcal{D}} S(d)$, thus defining a global environment state $g \in \mathcal{G}$.

Formally, a Domotic Effect *DE* is defined as a function of the global state space: $DE : \mathcal{G} \rightarrow \mathcal{V}$, where $\mathcal{V}$ is an application-dependent value space. For example, for control applications, $\mathcal{V} = \{0, 1\}$ since each Domotic Effect represents the activation of a given state configuration; conversely, when dealing with energy savings, $\mathcal{V} = \Re^+$ since Domotic Effects may be used to represent consumed power.

AmI designers and end users may define custom Domotic Effects by working with a domain-specific set of operators. Such operators work on the value space $\mathcal{V}$ relevant to the specific application domain[1]. The specification of a *DE* function requires three levels of formalization:

1. defining Simple Effects (SE), to extract a $\mathcal{V}$-valued quantity from a single device state. Formally, SE is a function that considers the state on only one device, $SE : S(d) \rightarrow \mathcal{V}$; such function is also time-dependent since it depends on $s(d,t)$.

2. defining *effect operators* working within $\mathcal{V}$-space algebraic semantics, suitable for composing new functions in the application domain. Formally, an operator op is a function $\mathsf{op} : \mathcal{V}^N \rightarrow \mathcal{V}$, where N represents the number of operands of the specific op.

3. defining Complex Effects (CE), by applying effect operators to the values computed by other SE or CE. Formally, a CE is represented by a couple $(\mathsf{op}, (DE_1 \ldots DE_N))$ composed of an operator name op and a list of Domotic Effects DE_i whose values are used as operands.

For each application domain, there would be a set of pre-defined SE and a set of operators that the users may combine to compute the values of interest. For example, if we consider control applications ($\mathcal{V} = \{0, 1\}$), then the SE functions that may be adopted are:

- for discrete-valued states, a SE detects whether a device currently is in any given state. E.g., $SE_{\mathrm{On}}(d,t) = (s(d,t) == \mathrm{On})$.

[1] for cross-domain applications, $\mathcal{V}$ would be the union of all relevant value spaces

- for real-valued sensors, a SE usually compares the current sensor data with a threshold. E.g., $SE_{\text{Hot}}(d,t) = (s(d,t) > 30^oC)$.

Conversely, if we consider energy savings applications ($\mathcal{V} = \Re^+$), the SE usually just extracts the current energy or power measurements from the real-valued sensor: $SE_{\text{Power}}(d,t) = s(d,t)$.

The instance layer in the "DogEffects" ontology defines a set $\mathcal{I}$ of all defined domotic effects (instances), i.e., $\mathcal{I} = \{DE_1, DE_2 \dots DE_N\}$.

Chapter 6

Modeling

Modeling formalisms and techniques play a crucial role in smart environments research. Human activities are modeled to better interpret human-home interaction detected by pervasive sensors. Human behaviors are analyzed and related to activities and to temporal evolutions with the aim of learning recurrent patterns, and possibly inferring user wills or goals. The context in which interactions occur, i.e., the environment state, the time, the weather, the season or the users' mood, are modeled to account for all possible factors influencing human behaviors. Together with home inhabitants, environment structure and components are modeled to support better design/operation of environments, e.g. detecting problems or design choices that might hamper usability or accessibility to impaired people. Moreover, device modeling is exploited to overcome integration issues and to offer a shared knowledge on how the environment works and on how it can be controlled.

All these modeling aspects can be roughly divided in two main categories: modeling of *environment interactions*, e.g., activity, behavior and context modeling [59–61], and modeling of *environment set-up*, including home fixtures and devices [35, 62]. Modeling of *environment interactions* treats people living in the environment as hidden targets: users are modeled with the aim of learning typical behaviors and understanding the underlying semantics, i.e., the carried activities. *Environment set-up*, instead, completely ignores the human presence and adopts a device-centric modeling approach addressing integration issues rising from the proliferation of automation devices and communication protocols, which are often incompatible with each other. These two last categories increasingly exploit semantics, i.e., ontologies and reasoning facilities stemming from the Semantic Web community (see Section 6.2 for more details).

Despite this intense modeling activity, a relevant aspect is still quite neglected, although needed to correlate human activities and device-centric descriptions: human intelligible state and goal modeling. Intelligible states and goals may relate to environmental variables (illumination, temperature, ...) or to more abstract

conditions such as security and energy saving. *Domotic Effects* represent such a human-understandable vision, i.e., how the home inhabitants perceive/represent the current environment configuration (state) and how they plan the next environment state (goals).

In this paper we introduce explicit and semantic modeling of such *Domotic Effects*, by empowering the AmI designer, and the end user, to define and specify its own vision of the ambient state, at a human-intelligible level of abstraction. *Domotic Effects* are defined as named sets of states for environment devices, constructed through domain-dependent functional operators (e.g., boolean operators in the direct control domain). They are formally defined through a three tiered modeling approach based on the newly designed DogEffects ontology. For each application domain, the AmI designer (or the end user, in second instance) defines the most suitable algebraic representation of domotic effects and the corresponding functional operators.

The proposed formal model is complemented by a DomoticEffects software framework that links it with the current state of a real smart environment, bidirectionally (Domotic Effects to device states, and device states to Domotic Effects).

Advantages are clear: AmI designers can work on top of an abstraction layer enabling the definition of generic, device-independent goals in a declarative way. Intelligent applications and operations inside the house can therefore be explicitly related to abstract effects, and their actual implementation may vary depending on the underlying infrastructure. Interaction designers can rely on such a shared knowledge, enabling a more natural human-home interaction. For example, user interfaces might concentrate on high-level abstract interactions, instead of single activations, and the *DE* framework would take care of mapping such goals into desired device states and environment configurations.

The main focus of this chapter is the definition of the formal modeling framework and the adopted DogEffects ontology.

The reminder of this chapter is organized as follows: The underlying ontology for the *DE* framework is discussed in Section 6.1, while Section 6.2 compares the proposed approach with the current state-of-the-art.

6.1 Modeling: DogEffects Ontology

In order to provide the formal knowledge base for the *DE* framework, a 3-tiered ontology-based approach has been adopted (Figure 5.1). The ontology is called the *DogEffects* ontology[1] and it is formalized by using the OWL Web Ontology

[1]The ontology can be reached at: http://elite.polito.it/ontologies/effects.owl

Language [29]. The *DogEffects* ontology is based on the modularity pattern, which allows it to be easily attached to other ambient ontologies that model the devices installed in an environment and their properties (especially the concept of state). Three modeling layers are defined, with decreasing usage complexity: a core layer, designed to establish the basic semantics of effects, a middle-layer allowing AmI designers to define SEs and domain-dependent operators for combining them into complex effects, and finally an instance layer where specific effects can either be defined by AmI designers or by final users, through suitable user interfaces.

6.1.1 Core layer

The core layer contains the basic class definitions for expressing *Domotic Effects* (Figures 6.1, 6.2, upper side); this layer is not meant to be modified by AmI designers nor by final users. Every *Domotic Effect* is formally organized into a concept hierarchy inheriting from the `dogEffects:Effect` class. Effects can either be simple (`dogEffects:SimpleEffect`) or complex (`dogEffects:ComplexDeviceEffect`). For both kinds of effects, domain-dependent subclasses are defined at the middle layer.

Simple Effects (SEs) are the terminal nodes of the representation and compute a value depending on a device state or sensor value. SEs act as interface points between the DogEffects ontology and some device description ontology (e.g., DogOnt [35]). The `dogEffects:effectOf` and `dogEffects:functionOf` open relations (i.e., relations without range restrictions) permit to identify the device and the device state for which a given SE is computed, respectively.

Every Complex Effect (CE) represents a functional expression of SEs and other CEs declared by using domain-dependent operators defined at the middle-layer of the *DE* framework. Effect operators take either simple or complex effects as operands (through the `dogEffects:hasOperand` relation) and generate new CEs as result, identified by means of `dogEffects:hasResult` relation.

Two main disjoint families of operators are modeled: unary operators (`dogEffects:UnaryOperator`) and non-unary operators (`dogEffects:NonUnaryOperator`).Unary operators only involve one `dogEffects:Operand` (cardinality restriction on the `dogEffects:hasOperand` relation), which can either map to a SE or a CE (by the `dogEffects:operandEffect` relation). Typical examples of unary operators are the NOT operator (in the boolean control domain) or the Integration operator (in the energy domain). Non-unary operators are further specialized (disjoint union) into commutative (`dogEffects:CommutativeOperator`) and not-commutative (`dogEffects:NotCommutativeOperator`) operators. According to the mathematical definition of commutative (not-commutative) operators, the result produced by the former (`dogEffects:CommutativeOperator`) is independent on the order in which operands are evaluated while, the latter operator provides

a results depending on the order of the effect operands. In such a case the `dogEffects:OrderedOperand` subclass of operands shall be used to account for the operand order, expressed as an ascending integer number by the `dogEffects:hasPositionN` property. Typical examples of non-unary effects are the AND, OR and EX-OR logic operators, in the boolean control domain, or the SUM and the DIFFERENCE operators in the continuous energy domain. The latter, in particular, is also not commutative as the result of a mathematical difference changes depending on the order of the operands.

6.1.2 Middle layer

The middle layer encodes domain-dependent operators typically defined by AmI designers. Every application domain will define different operator classes for this layer by sub-classing the general operator classes defined in the core layer. Two sample middle layers dealing with control (Boolean application domain) and with energy saving are defined below.

Boolean Application Domain

In the control domain, simple effects correspond to devices (sensors) being in specific states (measuring specific range of values). SEs and CEs can only evaluate to true or false, and Boolean logic is sufficient to compute CEs and to implement rather advanced activation scenarios. From the modeling point of view, mapping operators to Boolean logic requires a minimum set of logic operators, e.g., AND ($\wedge$), OR ($\vee$) and NOT ($\neg$) (Figure 6.1); however, AmI designers may choose to define more complex and user-intelligible Boolean operators such as:

1. *ImpliedOperator:* This operator represents the "logical implication" relationship. As it requires only a single operand, it is modeled as a unary operator.
2. *AlternateOperator:* This operator represents a function whose value is active when *exactly one* of its operands is active. It is commutative and non-unary. Mathematically, the Alternate effect operator can be defined as: $Alt(x_1 \dots x_n) = \sum_i \left(x_i \cdot \prod_{j \neq i} \overline{x_j}\right)$.
3. *ExactlyMOperator:* This non-unary operator represents a function whose value is active when *exactly M* number of its operands are active. Suppose there are n operands, i.e., $OP = \{1,2,\dots n\}$. Then the *ExactlyMOperator* effect operator can be defined as:

$$Exactly_M(x_1 \dots x_n) = \sum_{O \subseteq OP, |P|=M} \left[\prod_{i \in O} x_i \cdot \prod_{j \notin O} \overline{x_j} \right]$$

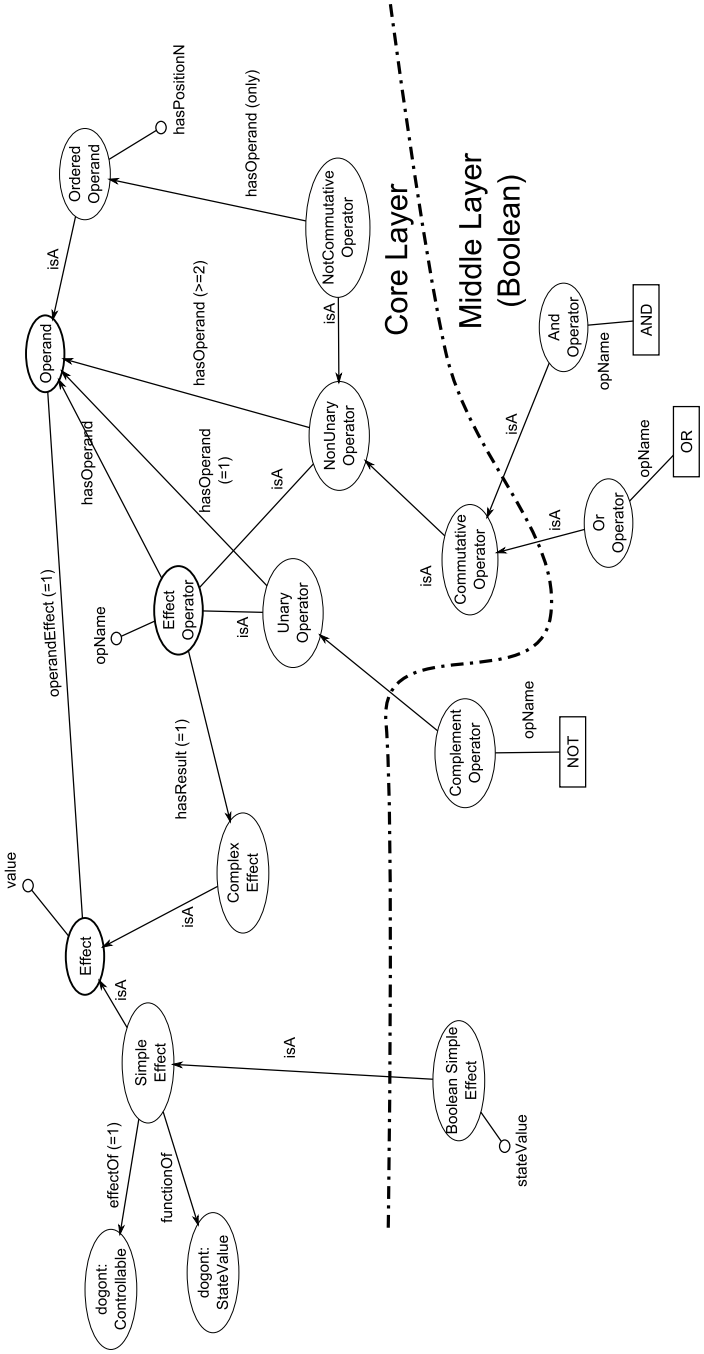


Figure 6.1. The DogEffects middle layer - Boolean Control Domain

Energy saving

In the energy saving domain typical elaborations involve continuous data such as power or energy measures coming from meters located in the smart environment. While simple effects are still defined as sensor-value couples, complex effects require operators such as integration for translating power measures into energy measures (and derivation for doing the inverse), thresholding for detecting overloads or anomalies, average for comparing consumption in a period, etc. With respect to the Boolean control domain, operator modeling is much more complex in this domain, and many different operators can be defined depending on the AmI designer's focus and on the application scope. Figure 6.2 shows a sample middle layer for energy where integration/derivation, average, sum, difference and threshold are defined.

Although this sample energy operator modeling might be somewhat simplistic to be used in full-featured energy aware applications, it's important to notice how the framework easily supports domain-dependent operators. Moreover, modeling accuracy and granularity can easily be adapted to the problem under examination, thus providing AmI designers a powerful tool for defining abstract effects on which more advanced policies can be built.

6.1.3 Instance layer

The instance layer of the Domotic Effects framework represents specific *Domotic Effects* defined in a given smart environment. They are modeled as instances of the classes defined in the core or middle layer and they relate to each other (functional composition) by means of domain dependent operators defined at the middle layer. Effect instances are typically defined by AmI designers or by final users. Following paragraphs better detail this layer by providing sample effect instances for both the Boolean control and the energy saving domain.

Boolean control

Since the current focus is geared towards the applicability of the *DE* framework to Boolean application domains, this section illustrates two use cases which are used through out this thesis.

Figure 6.3 illustrates a sample "Illumination" use case corresponding to the generic goal of lightening up the room. The illumination can be artificial by switching on the mirror lamps or ceiling lamp in different combinations, or illumination can be natural by opening the shutter of the window.

Figure 6.4 illustrates a simplified " Dining@Lunch" use case corresponding to the overall state of a dining room, in a house, during lunch hours. The "Dining@Lunch" use case includes isolating the kitchen, illuminating the dining room and switching

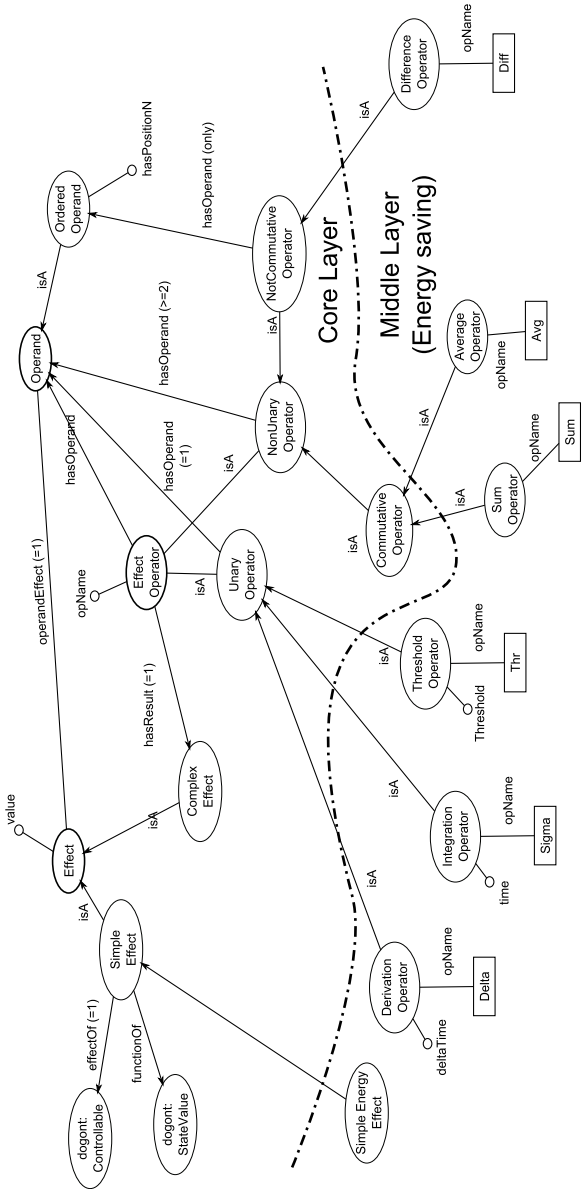


Figure 6.2. The DogEffects middle layer - Energy Saving Domain

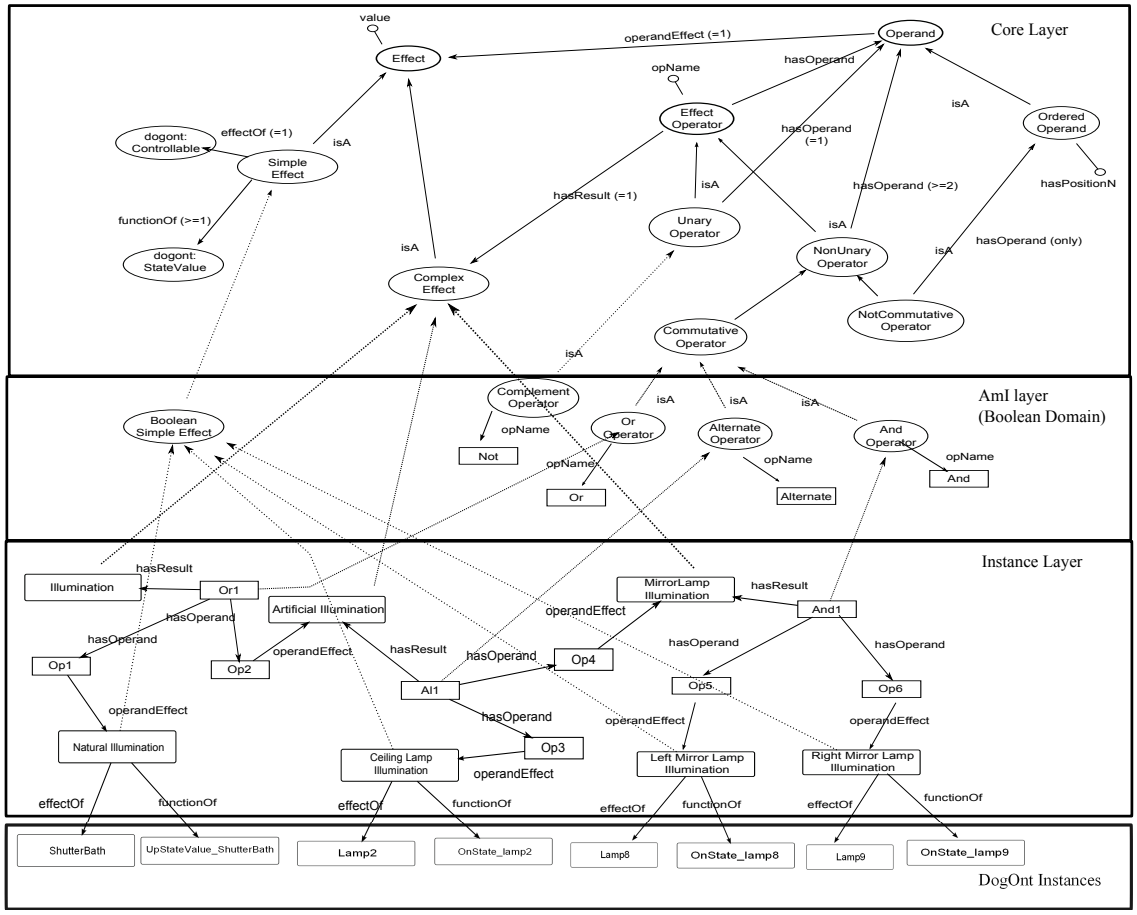


Figure 6.3. "Illumination" use case (DogEffects Ontology)

on the television for entertainment. The functional representation of the use case is outlined in Table 6.1.

Table 6.1. Dining@Lunch functional form

$$Dining@Lunch = \mathbf{And}(IsolateKitchen, TvEntertainment, LampIllumination)$$
$$IsolateKitchen = \mathbf{And}(Door\_Kitchen\_Isolate, Fan\_Off, Kitchen\_Flow)$$
$$LampIllumination = \mathbf{And}(LeftWallLampIllumination, RightWallLampIllumination)$$
$$RightWallLampIllumination = \mathbf{SE}(Lamp5, OnState\_lamp5)$$
$$LeftWallLampIllumination = \mathbf{SE}(Lamp4, OnState\_lamp4)$$
$$Door\_Kitchen\_Isolate = \mathbf{SE}(Door\_Kitchen, OffState\_Doorkitchen)$$
$$Fan\_Off = \mathbf{SE}(FAN\_Kitchen, OffState\_Fankitchen)$$
$$Kitchen\_Flow = \mathbf{SE}(ShutterActuator\_Kitchen, UpState\_SAKitchen)$$
$$TvEntertainment = \mathbf{SE}(Tv1, OnState\_Tv1, Volume\_40)$$

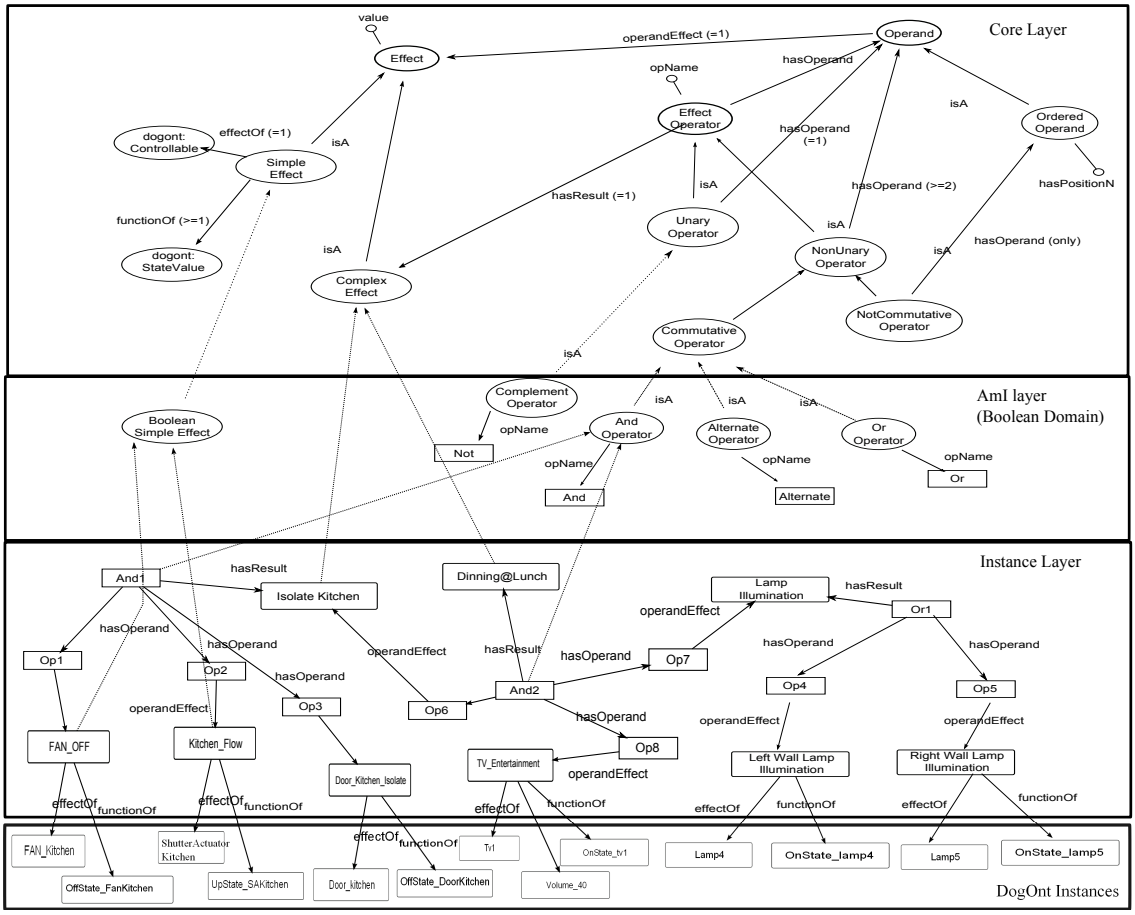


Figure 6.4. " Dining@Lunch" use case (DogEffects Ontology)

Modeling is clearly not restricted to a single knowledge domain. If, for example, we model both boolean control and energy saving at the middle layer, the instance layer can be exploited to define complex effects accounting for the consumption of each device involved in boolean activations, thus enabling easy definition of automatic energy saving policies.

6.2 Related Works

In the literature, few works tackle the modeling gap between interaction and environment modeling, and AmI designers and end-user concerns are mostly addressed separately.

In their recent work [63], Juan Ye et al. start tackling this gap with a top-down

approach based on upper-level ontologies. They define a top-level ontology capturing in a uniform manner the inherent semantics that different types of domain knowledge share, model such semantics across information at different levels of abstraction, and reason on it using a sound and proved reasoning schema. Thanks to this conceptual model, developers can use the provided generic rules and define their own specific rules to facilitate performing system level tasks such as checking the consistency of context and derivation rules that describe activities, integrating the model with statistical information, etc. Although the approach is sound and might actually improve modeling of smart environments, it mainly focuses on AmI developer tasks and almost ignores user-intelligible effects of home/building automation. On the converse, DogEffects addresses at the same time users and AmI designers, by exploiting *Domotic Effects*. The approach of Juan Ye et al. and the one presented in this chapter are not in contrast, instead they can successfully complement each other: while DogEffects provides user-intelligible effect and goal definition, the Juan Ye et al. work can be exploited to link effects to context and activity models and to reason on the consistency of information sensed from the environment and framed in the joint conceptual model.

The Amigo project [64] adopts an ontology-based representation model similar to DogEffects. Amigo models devices, and by extension, aggregations of devices as services handling specific inputs and providing well-defined outputs. A service composition approach allows combining functionalities offered by the smart environment and a set of vocabulary ontologies permit to relate such functionalities to contextual information. The main shortcoming of the Amigo approach is the device-centric vision, which neglects users and how they perceive/conceive automation effects/goals. In this sense, DogEffects overcomes the Amigo approach by explicitly allowing domain-dependent effects (and related operators). On the other hand, however, the open and modular design of DogEffects supports the integration of the two approaches, enriching the set of Amigo vocabularies with a new, more abstract, modeling of device/automation effects.

In [43, 45] complete independent control solutions to manage domotic environments based on the learning pattern of a user's activity are proposed, while in [39, 42, 56] the authors advocate enabling non-programmers to create and manage their smart environments according to their wishes. Providing a complete independent control extracted from a user's activity as the only mechanism might not be a good idea as it diminishes the influence residents have on their personal spaces. It also raises a new set of problems like privacy and security issues. Moreover, instead of focusing on user's goal or intention, it focuses on a set of devices and their actions. It can be said that learning algorithms discover patterns of device activity instead of user's intended goals. The latter group of papers, on the other hand provides end user programming environment but the underlying structure of organizing goals

and their different courses of actions is missing. Finally, [56] and [41] provide organization mechanisms, but the translation from goals to their individual tasks is static and does not allow flexibility. Our proposed modeling approach provides AmI developers with an abstract view that they can change, update or build atop depending on the smart environment, and residents can define their own combinations of devices, according to their wishes.

Chapter 7

Evaluation

This chapter expounds on the ability of the *DE* framework to monitor the overall state of the environment, in real time called *Effect Evaluation*. The monitoring amounts to correctly mapping the devices and their states in terms of user goals or intentions (Figure 7.1). The discussion focuses on the applicability, of the DE framework, to Boolean application domains, i.e., domains in which the outcome of a user goal or intention can be either active (true) or inactive (false). This covers most monitoring use cases in smart homes, offices and industrial plants.

Formally, *Effect Evaluation* means computing the value of all domotic effects starting from the values of each device state and sensor value, according to the definition of the domotic effects and the semantics of the functional operators (that might change depending on the application domain). The computation should be in real time, i.e., comparable with the latency of home automation systems (roughly under one second), and well integrated with the smart environment management system (e.g., an home gateway). The conception, architecture and implementation of *Effect Evaluation* are discussed in this chapter.

The chapter is divided into seven sections. The problem addressed in this chapter is formally defined in Section 7.1, while the general approach adopted for effect evaluation is described in Section 7.2, and later Section 7.3 defines the adopted architecture and implementation. Section 7.4 shows results of the experiments and Section 7.5 provides overview of existing literature. Section 7.6 concludes the chapter.

7.1 Problem Statement

Given the definitions in the previous sections, the problem of "Effects Evaluation" deals with finding the value $\mathcal{V}$ for all the Domotic Effects DE_i defined in the instance layer of the DogEffects ontology, i.e., $\mathcal{I}$. The value of a domotic effect depends upon

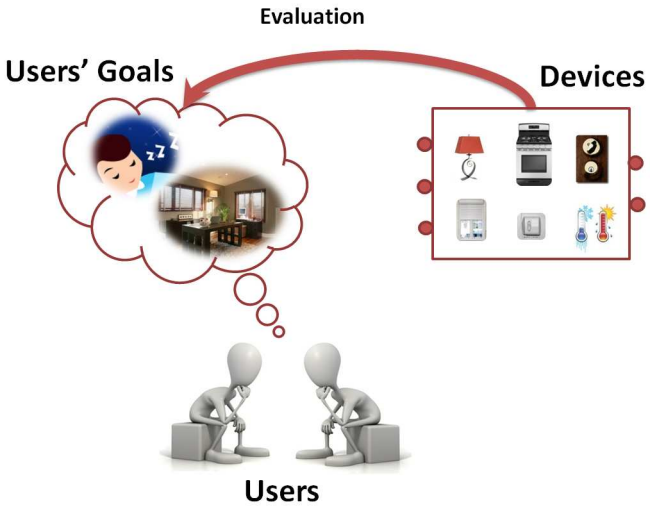


Figure 7.1. Evaluation

the application domain. This chapter addresses "Effects Evaluation" restricted to Boolean application domains, i.e., application scenarios where the AMI layer is defined in terms of Boolean values and Boolean operators. This covers most control applications and many monitoring use cases in smart homes, offices, and automated industrial plants. In this domain, the values of the domotic effects can be either active (true) or inactive (false).

For simple domotic effects, the value is active when the corresponding device is in the state defined for the domotic effect. A complex domotic effect is a combination of other domotic effects (both simple and complex) and the type of combination is governed by an effect operator defined in the instance layer of the DogEffects ontology. Therefore, the value is dependent upon the values of its children and the effect operator associated with the complex domotic effect.

The primary objective of the chapter is to design, develop and verify a module that performs the process of *Effect Evaluation* in the context of *DE* framework. Since the module needs to be verified inside an AMI system, the *Effect Evaluation* module should meet the following set of requirements:

1. The module should be able to monitor the change of state of all the devices installed in the environment.

2. Different third-party applications should be able to register themselves to this module, to listen to any change in the values of the domotic effects.

3. During the "Effect Evaluation" it should asynchronously send out notifications to the registered applications.

4. The process of *Effect Evaluation* should be robust, i.e., to respond in near real-time (NRT).

5. The module should provide integration with Dog [65].

7.2 Approach

In order to meet the objectives of the chapter a module named "Domotic Effects Evaluation" was developed. All the domotic effects are organized in a hierarchical structure that corresponds to a simplified representation of the logical structure defined in the Instance layer of the DogEffects ontology. The structure is identified as Effect Node Network (ENN). The ENN for the *Illumination* use case defined in Section 6.1.3 is shown in Figure 7.2. The module listens to any change in state of devices and if any change occurs, it performs the evaluation of all the domotic effects. The classical Zero Delay Event Driven Logic Simulation algorithm [66] is chosen to perform the evaluation. In the *DE* framework, the effect operators defined in the AmI layer represent the evaluation criteria. Therefore, a proper implementation of each effect operator in the Boolean application domain is provided ,i.e., Complement, And, Or and Alternate.

When a device changes its state, the values of the simple domotic effects associated with the device may change. If the values of simple domotic effects are changed, the evaluation of the complex domotic effects dependent on them are scheduled and recursively evaluated. During the evaluation, if a change in the value of a domotic effect (simple or complex) is observed, an asynchronous notification is sent out to third-party applications. This whole process is called "Effect Evaluation".

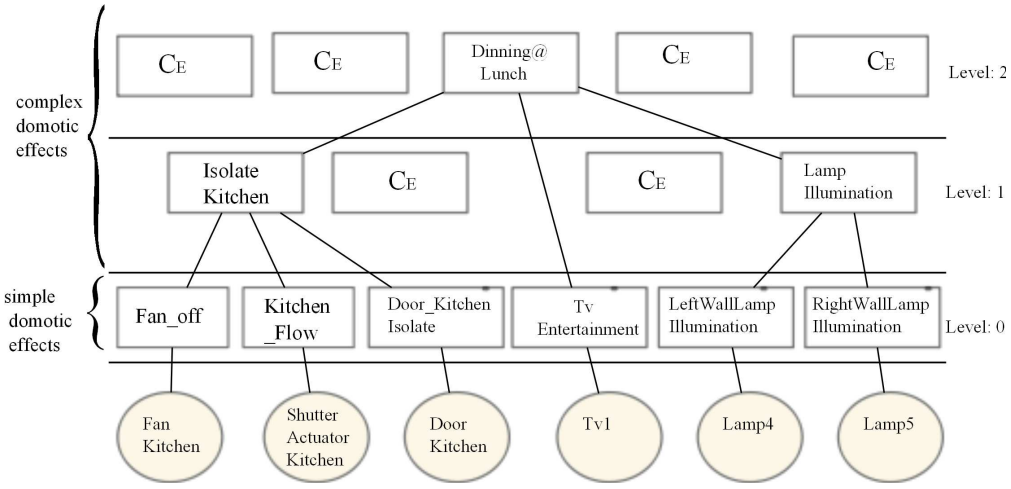


Figure 7.2. ENN for the Dining@Lunch use case

7.3 Solution

7.3.1 Architecture

Dog [65] architecture was extended to integrate the DE based approach in smart environments. Dog has a modular architecture composed of 12 core bundles that is able to expose different domotic networks as a single, technology neutral, home automation system. The DogEffects ontology and its management are handled by the *HouseModel* bundle. The bundle provides information about all the domotic effects defined in the environment. A *Domotic Effects Evaluation* bundle has been defined, which perform the organization of all the domotic effects defined for the environment in the ENN and carries out the effect evaluation process. The functionalities of these two bundles are explained below:

HouseModel Bundle

The HouseModel bundle exploits standard DogOnt classes and DogOnt instances referred by a specific environment to provide knowledge-rich access to the environment properties and capabilities. It accesses the DogEffects ontology classes and their instances and provides all the management operations related to the DogEffects ontology. Inference and reasoning tasks are carried out at this level, both at the Dog startup, for computing consistency checking and transitive closure, at the runtime. In particular, runtime reasoning is adopted for generating inter-operation rules. In the context of this chapter it provides the following services at Dog startup.

Effect Data Extraction This functionality gives the information associated with all the domotic effects defined in the environment. In case of a complex domotic effect, the associated information includes the name of the effect, the name and number of children, the type of the domotic effect (defined using the Effect operator). In case of a simple domotic effect, the name of the effect and its associated device in a given state is mentioned. The generic information template is shown in Figure 7.3.

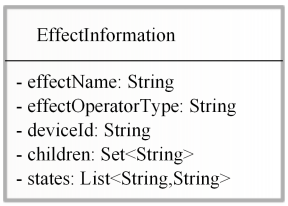


Figure 7.3. Information template

Domotic Effects Evaluation

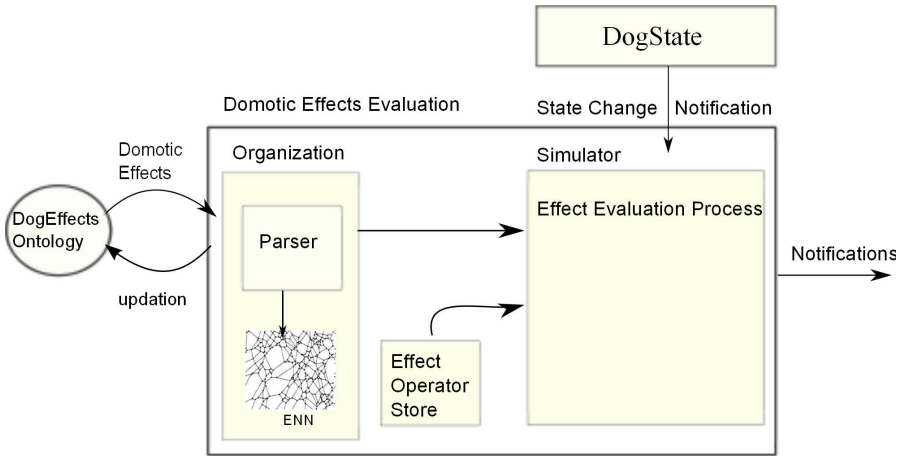


Figure 7.4. DogEffects bundle

This bundle is responsible for performing the effect evaluation process. The logical architecture of this bundle is shown in Figure 7.4.

The *DogState* bundle is responsible for providing the *Domotic Effects Evaluation* bundle information about the current states of the devices connected to the Dog platform. It sends out device state change notifications containing the name of the device and its current state. During Dog startup, the *Organization* component requests and receives all the domotic effects defined in the instance layer for a particular environment. It includes a parser which parses the received response from the *HouseModel* and organizes all the domotic effects in a hierarchical data structure (called Effect Node Network, ENN).

As defined in Section 7.2, to evaluate different types of domotic effects, proper evaluation algorithms (to achieve the activation criteria) for the effect operators defined in the AmI layer are defined. The Effect Operator store contains the Java code for each effect operator to provide evaluation.

The effect evaluation process is managed by the Simulator component and comprises receiving notifications of device state change from the *DogState* bundle, evaluating all the domotic effects and sending out notifications if a change in values of the domotic effects is observed.

- When a device state change notification is received, the simulator matches the received notification to the simple domotic effects associated with the device to check their activation criteria. If the evaluation criteria matches, it activates the simple domotic effect and deactivates other simple domotic effects associated with the device. Afterwards, the complex domotics effect are evaluated.

- For each complex domotic effect, the values of its children is determined and then the type of the effect operator. Once the effect operator is determined, the appropriate Java implementation in the effect operator store takes the values of its children and determines the value of the current complex domotic effect.
- The classical Zero Delay Event Driven Logic Simulator Algorithm has been chosen to perform the effect evaluation process. The transformed algorithm for our proposed approach, in pseudo-code is reported in Algorithm 1. It uses the schedule method, whose pseudo-code is reported in Algorithm 2.
- As soon as, a change in value of a domotic effect is observed, a notification is sent out for that domotic effect.

Algorithm 1 Zero Delay Event Driven Logic Simulator Algorithm

```
for all queues as currentqueue do
    for all currentqueue as node do
        oldValue=(node).isValue();
        newValue = (node).evaluate(); → Evaluation Criteria for the node
        if oldValue NOTEQUAL newValue then
            node.setValue(newValue);
            node.sendNotification();
            node.schedule();
        end if
    end for
end for
```

Algorithm 2 schedule method Algorithm

```
parents=this.getParents();
for all parents as node do
    p=(node).getLevel();
    queue=(queues).get(p);
    if queue.contains(node) == false then
        queue.add(node);
    end if
end for
```

7.3.2 Extensibility

The system designers have the ability to define different Boolean effect operators depending on the environment and for monitoring purposes appropriate implementation for determining the value of the corresponding domotic effect should be included in the *effect operator store*. Figure 7.5 shows a class template to implement an effect operator declared in the AmI layer of the DogEffects ontology.

When a new effect operator is defined in the AmI layer of the "DogEffects" ontology, the Effect Evaluation module can be extended as follows:

```
public class NewOperatorName extends EffectNode {

    public NewOperatorName() {

       super();
       /*
       The passed operator name should be
       equal to the effect operator name
       defined in the DogEffects ontology.
       */
      super.setOperator("NewOperatorName");
    }

    @Override
                public boolean evaluate() {

          }
}
```

Figure 7.5. Procedure to define a new effect operator.

1. A new class is defined that extends the *EffectNode* class (Figure 7.6).

2. The parent class gives the ability access the value of the children nodes and based on it, a proper condition for activation (true) or deactivation (false) can be provided inside the *evaluate* function.

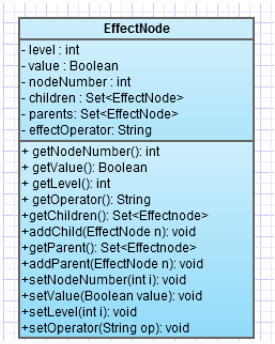


Figure 7.6. Template of abstract EffectNode Class

Currently, the *effect operator store* includes the Complement, And, Or and Alternate operators. The Complement effect operator represents an invert relationship, and its evaluation algorithm is given in Algorithm 3.

Algorithm 3 Complement evaluation algorithm

child=*node*.getChildren();
return !(*childNode*.isValue());

Algorithms 4 and 5 show the evaluation algorithms for the "And" and "Or" effect operators. The Alternate effect (Algorithm 6) operator represents a function which

is active when only one of its children is active. Mathematically, the Alternate effect operator can be defined as:

$$\sum_i \left(OP_i \cdot \prod_{j \neq i} \overline{OP_j} \right)$$

Algorithm 4 And evaluation algorithm

```
children=node.getChildren();
for all children as childNode do
    if childNode.isValue() == false then
        return false;
    end if
end for
return true;
```

Algorithm 5 Or evaluation algorithm

```
children=node.getChildren();
for all children as childNode do
    if childNode.isValue() == true then
        return true;
    end if
end for
return false;
```

Algorithm 6 Alternate evaluation algorithm

```
children=node.getChildren();
counter=0;
for all children as childNode do
    if childNode.isValue() == true then
        counter= counter + 1 ;
        if counter > 1 then
            return false;
        end if
    end if
end for
if counter == 1 then
    return true;
else
    return false;
end if
```

7.4 Experimental Study

To study the feasibility of the proposed approach, measure performance parameters of the modified *HouseModel* bundle and the newly developed *Domotic Effect Evaluation* bundle two sets of experiments were carried out. In the first set of experiments

the integration of the proposed approach inside the Dog was assessed. Conversely, in the second set of experiments performance parameters were measured. These parameters include the time taken by the *Domotic Effect Evaluation* bundle to request, receive and organize the domotic effects in the ENN during Dog startup, the average time taken to completely perform the effect evaluation process.

7.4.1 Feasibility Testing

The *Domotic Effects Evaluation* bundle was built using the eclipse equinox OSGi framework[1], so that it can be integrated inside Dog. A set of standalone applications, based on the Publisher-Subscriber pattern [24], were developed to test the correctness of integration and to witness the overall effect evaluation process.

After modifying the *HouseModel* bundle and integrating the *Domotic Effects Evaluation* bundle, in order to realize the Publisher-Subscriber pattern over the web the *LO(D)D* architecture was used. LO(D)D is a distributed framework that provides a systematic way to publish environment data which is being updated continuously; such updates might be issued at specific time intervals or bound to some environment specific event (Chapter 11). In our case, these events are changes in the values of domotic effects, defined in the environment.

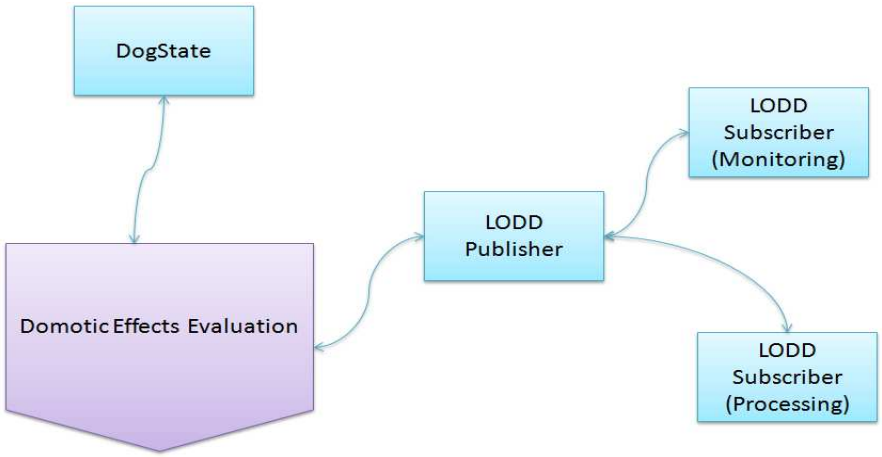


Figure 7.7. Action Sequence of Feasibility Testing.

The architecture of the feasibility testing is shown in Figure 7.7. The testing was conducted over a simulated environment of a house whose structure is shown in Figure 7.8. The house is composed of a bed room, a living room, a lobby, a bath room, a store and a kitchen, and is equipped with with several automatic

[1] http://www.osgi.org

devices/appliances like lamps, oven, television, door actuators, window actuators, shutter actuators, gas heaters etc. For each device, a certain number of simple domotic effects have been defined in the instance layer that correspond to the number of controllable states that the device can achieve. Based on these simple domotic effects and using the set of Boolean operators encoded in the AmI layer, several complex domotic effects were defined. Some of them are explained in the following section.

Use cases

This section describes six top level use cases defined, over the simulated house environment, using simple and complex domotic effects. The number of total domotic effects defined to map all six use cases were 190, and the number of involved devices were 57. To check the integration, several iterations were performed. In each iteration, the states of random number of devices were changed and the corresponding changes in the values of domotic effects were monitored. Table 7.1 summarizes the feasibility testing. It defines, for each iteration, the number of devices whose states were changes, the corresponding number of active and inactive domotic effects (after the change) and the time, in milliseconds, taken for the complete operation.

To perform the experiment, a new test *LO(D)D Publisher* bundle was built which registers to the *Domotic Effects Evaluation* bundle for receiving notifications when the value of any domotic effect is changed. Another web application named *LO(D)D Subscriber* was developed, which subscribed to the *LO(D)D Publisher* bundle. A separate bundle *StateChange* bundle sent commands to change the states of different devices.

The *Domotic Effects Evaluation* bundle listened for any change in state of devices installed in the environment (from the DogState bundle). As it received the state change notification from the *DogState* bundle the effect evaluation process is performed and new values of all the domotic effects are computed. During the evaluation process, when it encounters a change in the value of a domotic effect, a notification is asynchronously sent to the registered applications, i.e., in our case *LO(D)D Publisher* bundle. The *LO(D)D Publisher* bundle forwards the notifications to different monitoring applications (LO(D)D Subscriber in our case).

Secure Home The "Secure Home" use case (CE_A) secures all the exit points of the house, i.e., by closing all the exit doors and shutting all the windows of the house. This use case comprises many DEs providing the ability to secure different rooms of the house. This can be used in case of emergency, theft, robbery or fire etc. The functional representation is shown in Appendix A.

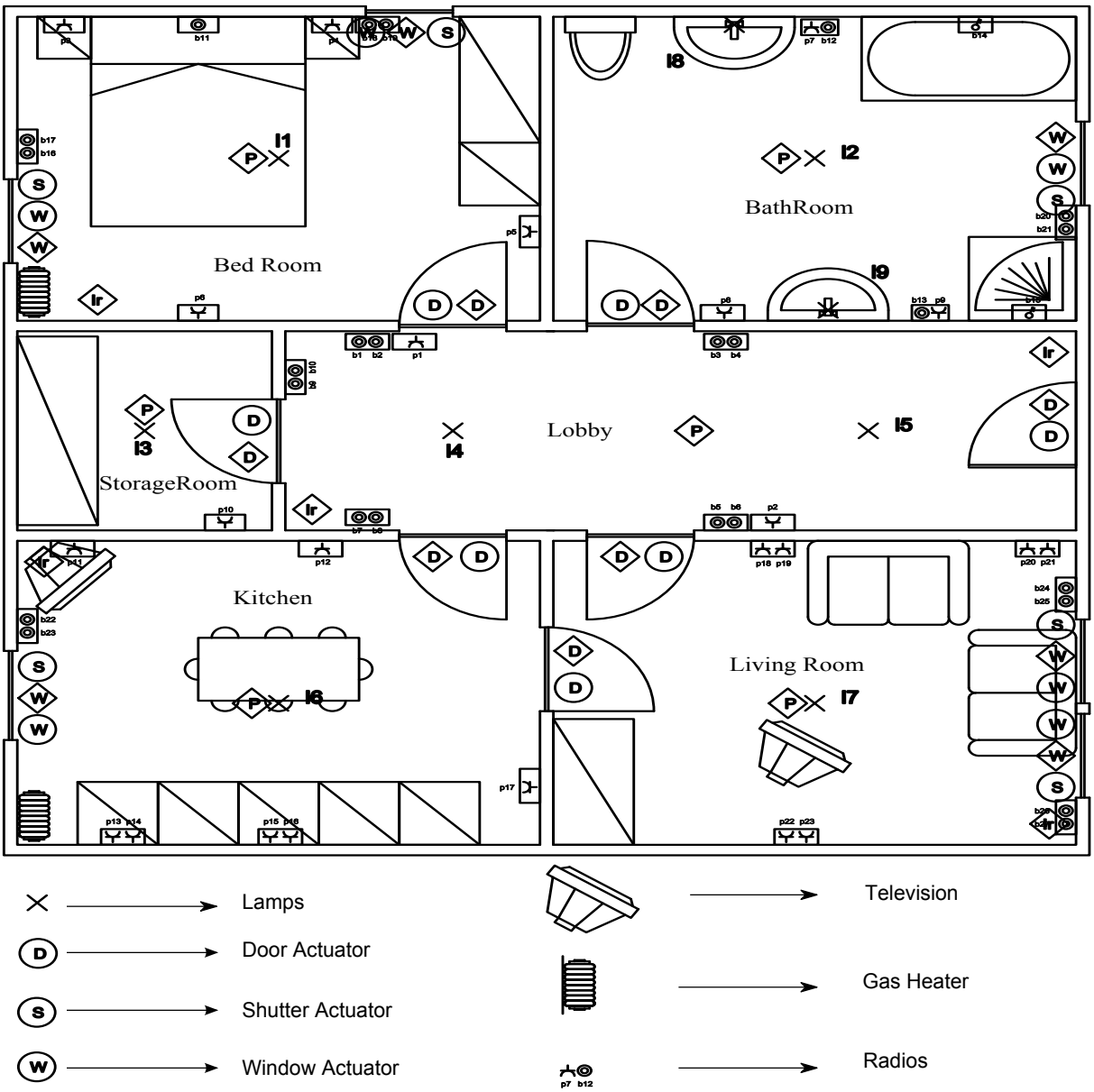


Figure 7.8. A Sample Structure of a house.

BathRoom Illumination The "BathRoom Illumination" (CE_B) combines small use cases that illuminate the bathroom. The illumination can be artificial by switching on the mirror lamps or ceiling lamp in different combinations, or illumination can be natural by opening the shutter of the window during morning and afternoon hours. The functional representation is given in Appendix A.

Home Illumination The "Home Illumination" (CE_C) requires that all the rooms of the house are illuminated. Illumination can be both natural or artificial in nature. The functional representation is shown Appendix A.

Afternoon Lunch The "Afternoon Lunch" (CE_D) deals with the daily routine of cooking lunch inside the kitchen. The resident desires the kitchen's oven to be heated, the television to be switched on and the kitchen to be closed so that the aroma of cooking does not spread to other rooms of the house. The functional representation is given inAppendix A.

Air Passage inside the house The "Air Passage" use case (CE_E) manages the natural air flow inside the house or its different rooms. It combines different DEs that open windows of opposite sides for the flow of air between different rooms of the house. The functional representation is shown in Appendix A.

Morning WakeUp The "Morning WakeUp" use case (CE_F) maps a typical scenario when a resident wants to perform a sequence of activities after waking up in morning, like illuminating the bedroom, the kitchen and the bathroom, switching off the gas heater inside the bedroom, switching on the television in the kitchen and the radio inside the bathroom. The functional representation is outlined in Appendix A.

Comments

Table 7.1 illustrates the number of active and inactive effects when the states of devices were changed randomly, over the defined use cases. All the changes were detected and propagated in the ENN. The list of active domotic effects was constantly updated and published as linked data using the LO(D)D Publisher.

Table 7.1. Results of the feasibility testing

No. of Devices changed	No. of Active Effects	No. of Inactive Effects
2	3	187
4	7	183
5	6	184
2	3	187
7	14	176
2	4	186
1	2	188
3	3	187

7.4.2 Performance Evaluation

To measure the performance parameters of the implemented *Domotic Effects Evaluation* and the modified *HouseModel* bundles, two experiments were conducted. The performance parameters include the total number of domotic effects (simple and complex), during Dog startup the time taken by the *HouseModel* bundle to extract domotic effects information from the ontologies (called Effect Extraction time), during Dog startup the time taken by the organization component to create the ENN (called Organization time), maximum level of the ENN, and the average evaluation time over a number of iterations. The effect data extraction time includes loading the ontologies (DogOnt and DogEffects), checking consistency and performing realization. The organization time represents receiving all the domotic effects and organizing them in the ENN. Both effect data extraction and organization time is taken by the Dog during the startup and it happens only once. The average evaluation time represents evaluating all the domotic effects and sending out notifications over a number of iterations and is calculated in milliseconds.

For the experiments a house environment with 50 devices was simulated, whose domotic structure was defined using the DogOnt ontology. For more details please refer to [35]. The experiments ran on a standard personal computer with a quad-core Intel i5 processor and 4GB of RAM. A *TestDogEffect* bundle was created to carry out the experiments and measure the performance parameters.

Experiment 1: Daily Chores Scenario

This experiment simulated a scenario that encapsulates daily chores occurring in a house using the "Domotic Effects". 12 test iterations were created with each iteration simulating 50 devices in the house. During each iteration a random number of simple and complex domotic effects were generated in a range from 100 to 1500. The type of the effect operator between complex domotic effects, the number of children and parent, and the inter-dependency of all the domotic effects (level) among themselves were generated randomly. To measure the performance parameters, the *TestDogEffect* bundle randomly chose devices and changed their respective states. For each iteration, the process of changing device states was repeated at least 150 times and then the evaluation time was averaged for each iteration.

The performance parameters of this experiment are depicted in Table 7.2 and the average effect evaluation time taken is shown in Figure 7.9.

Experiment 2: Maximal Propagation Scenario

This experiment was an attempt to simulate a scenario in which a change in state of a single device will initiate the evaluation of a significant number of nodes in the ENN. As defined in Section 7.2 the classical Zero Delay Event Driven Logic Simulation

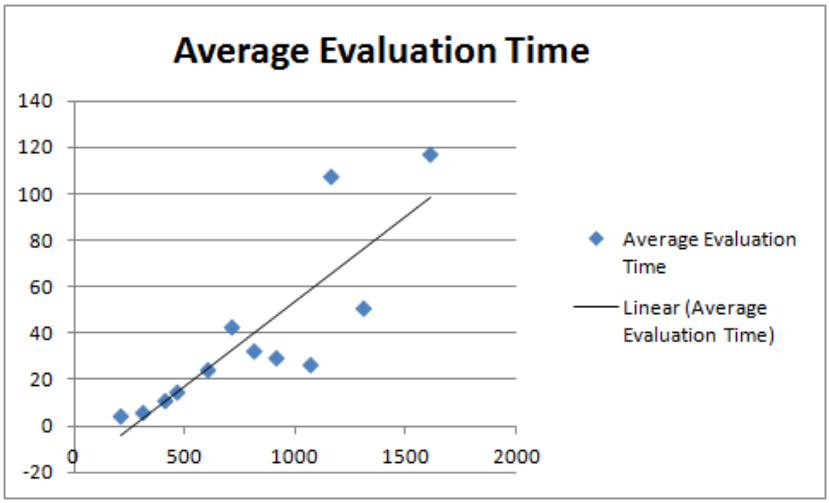


Figure 7.9. Relationship b/w Average Evaluation Time & Total No. of DEs

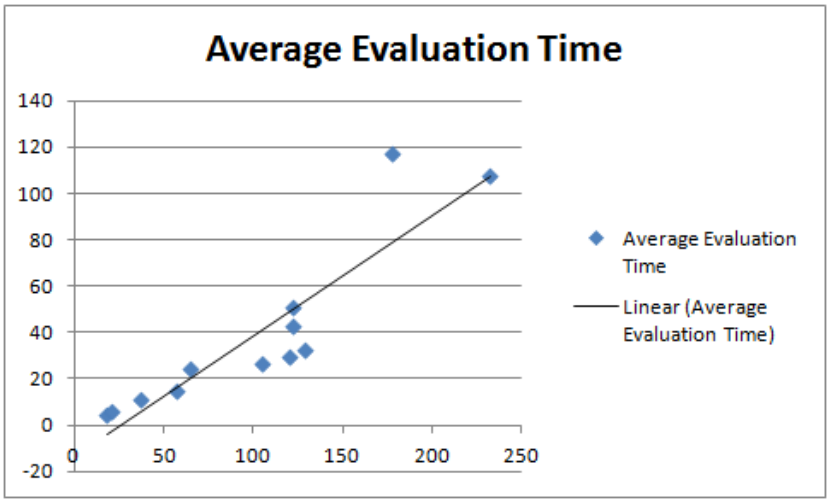


Figure 7.10. Relationship b/w Average Evaluation Time & Maximum level of ENN

performs the effect evaluation process. The semantics of the simulation is such that if a device changes its state all the domotic effects dependent on that particular device may need to be evaluated and the value of other domotic effects remain unchanged (Figure 7.11). In Section 7.4.2, the described test iterations generated random simple and complex effects on top of 50 devices in the house. However, in this experiment though the generation of complex domotic effects was random but all the domotic effects were dependent on a single unique device (identified by *UD*) plus any other devices. Therefore, when this unique device will change state all the complex domotic effects might need to be evaluated (Figure 7.12). 10 iterations

were performed with random number of complex domotic effects generated, the type of the effect operator between complex domotic effects, the number of children and parent and the inter-dependency of all the domotic effects (level) were generated randomly.

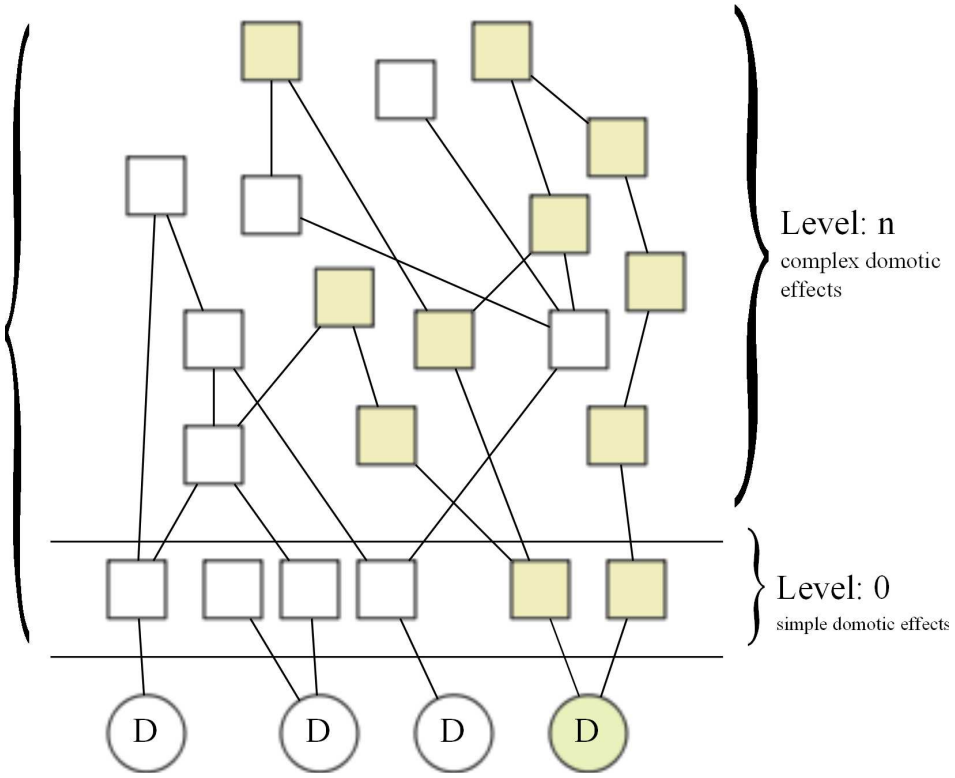


Figure 7.11. Semantics of Effect Evaluation process in Experiment 1.

For each iteration the state of the unique device was changed. After that the states of other randomly chosen devices in the house were changed and in the end again the state of the unique device was changed. This process was repeated at least 20 times for each iteration. The performance parameters of this experiment are depicted in Table 7.3 and the average effect evaluation time is shown in Figure 7.13.

7.4.3 Discussion

The developed set of applications presented in Section 7.4.1 proves that the "Domotic Effects" based approach can be integrated inside smart environment systems and is relatively flexible and easy to integrate with third party applications and services. Different monitoring applications (both web and desktop based) can be developed to monitor the state of the overall environment using the "Domotic Effects" approach.

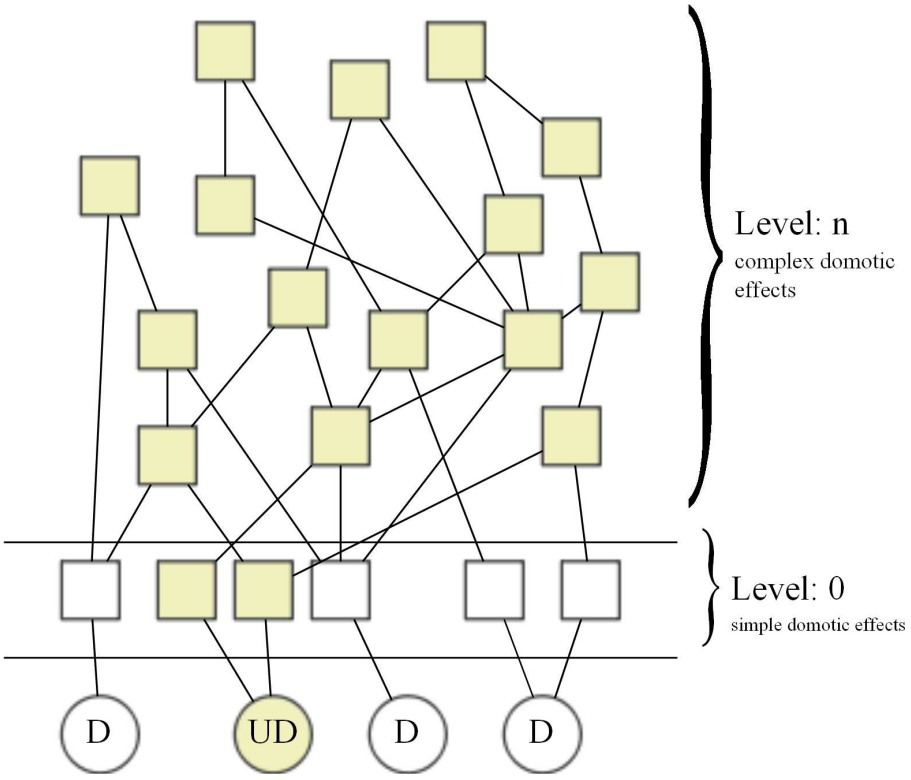


Figure 7.12. Semantics of Effect Evaluation process in Experiment 2.

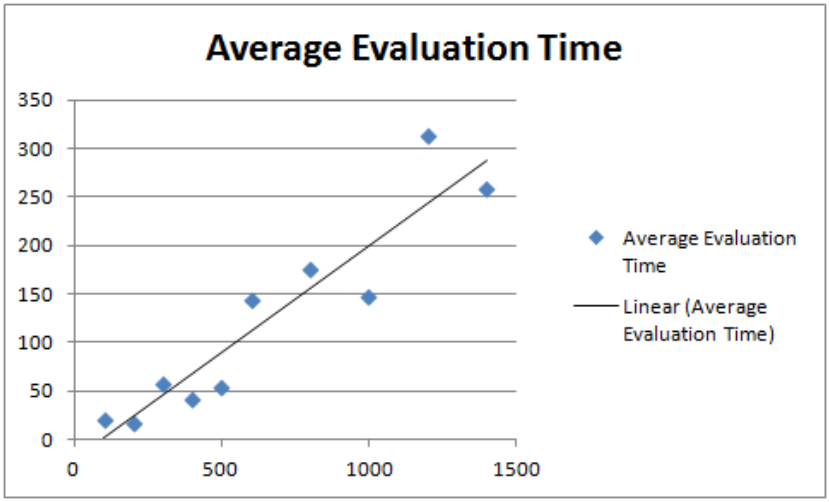


Figure 7.13. Relationship between Average Evaluation Time & Total No. of DEs

The observations regarding the measured performance parameters presented in Section 7.4.2 are pointed below.

- In the daily chores scenario (Table 7.2), the total domotic effects depend upon all the devices in the house. The effect evaluation process takes a maximum of around 108 milliseconds to complete the effect evaluation process and send out notifications. It can be seen that the "Domotic Effects Evaluation" bundle is quite responsive and responds in near real-time. In most of the cases the time for evaluation and sending out notification is less than 150 ms.
- In the maximal propagation scenario (Table 7.3), all complex domotic effects are dependent directly or indirectly on a single unique device. The effect evaluation process takes an average of around 313 milliseconds to complete the effect evaluation process and send out notifications. Considering a unique device having 1000 domotic effects in a house means that a house with 50 devices will have 50,000 effects. Even in such rare occurrence, the average evaluation time is less than a second and takes on average 312.86 ms for 1200 CEs to be evaluated and therefore indicates the responsiveness of the approach.
- Figure 7.10 shows that relationship between the average effect evaluation time and the maximum level of the ENN is linearly increasing (for daily chores scenario).
- Figures 7.9 and 7.13 show that in both experiments (daily chores and maximal propagation scenarios) the average evaluation time increases with the number of domotic effects that are effected by a change in the state of a device and is directly proportional.
- Figure 7.14 shows the comparsion between the average evaluation time, height of the ENN and the total number of domotic effects. It can be observed that the evaluation time is more dependent upon the height of the ENN as compared to the total number of domotic effects. However, this observation needs more investigation in the future.

7.5 Related Works

A neuro-cognitive model for Environment Recognition, Decision-making, and Action execution inside automated buildings is proposed in [53–55]. They introduce separate models for perception known as Artificial Recognition System PerCeption (ARS-PC) and decision making, identified as Artificial Recognition System Psycho-Analysis (ARS-PA). The perception models provide a three layered architecture, i.e., Micro Symbol Layer, Snapshot Symbol layer and Representation layer. Micro symbol layer are formed from sensor input data, which are combined to create snapshot symbols in the Snapshot symbol layer and then representation symbols are created

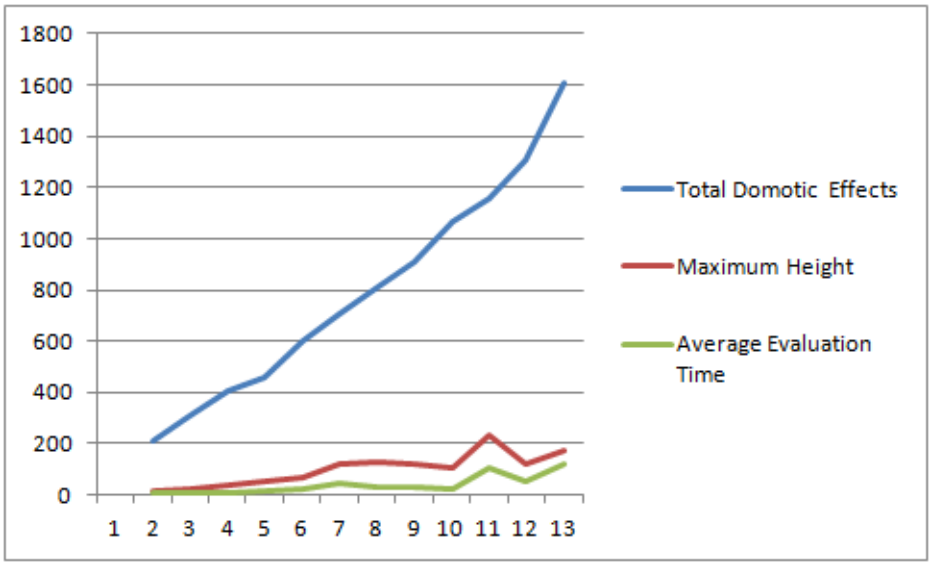


Figure 7.14. Average evaluation time, ENN levels & No. of DEs comparison

corresponding to the perception of the system in the Representation layer by combining snapshot symbols. The representation symbols are then fed to the ARS-PA to perform a decision making process. In comparison, the "Domotic Effects" approach provides modeling of generic goals and their achievement criteria (DogEffects ontology). Besides perceiving and monitoring the environment, it has the ability to enforce and optimize those goals based on a possible set of paths (Chapter 8). Moreover, the "Domotic Effects" provide separate views for system designers and users of the environment. System designers are allowed to define several different types of combinations for different application domains and users can defined their own achievement criterias. This flexibility is missing from the neurocognitive model.

Hemrik Dibowski et al. [52] proposes an automatic design approach for large building automation systems (BAS). The top-down approach initiates by defining the structure of the building, then the system integrators define requirements using ontologies. The next step is to define abstract designs and required functionalities which are then transformed to detailed designs of the specific BAS. As indicated, the complexity of this transformation task can be very high, making the approach complicated and lacking flexibility to achieve detailed designs. On the other hand, the "Domotic Effects" approach is flexible in providing system designers separate working space independent of lower level details and allowing them to focus on general characteristics of the environment, which can later easily be extended or changed (AmI layer). The users have the ability to play a key role in programming their personal spaces.

While techniques addressing both the user and the system designer concerns

are rare in literature, several papers have documented them separately. Rashidi et al. [43] proposes a software architecture which incorporates learning techniques to discover patterns in user's daily activities. While the patterns of user's activities are observed and stored, the user can also define their own activity patterns as well. The activity pattern are observed as changes in states of devices occur in the house. Hierarchical activity model (HAM) is used to store discovered activity patterns and their temporal knowledge. The activities are discovered based on a device centric vision, which does not allow users to view the bigger picture about the state of the environment, in a concise way. On the other hand, the "Domotic Effects" can provide the bigger picture about the environment concisely and in a manner understood by the user.

Cheng et al. [45] proposes a smart home reasoning system called ASBR system. The system learns user's preferences by adaptive history scenarios and put forwards a way to rebuild reasoned knowledge in other smart homes. They propose that contextual information can be extracted and reasoned as a set of scenarios. In addition, the system can derive personalize habits and store them in OWL files. Though it does provide an organization mechanism but here the concept of scenario is different from our proposed effect. "Domotic Effects" are different from scenarios as it is not a storage of historical events or repetitive tasks, but a modeling of the environment as envisioned by the user. Therefore, during the process of monitoring the environment, the user has enhanced comprehension about his/her environment.

Salomons et al. [44] introduces a generic model for intelligent homes that describe the current state. the target state and the transition. In the context of monitoring the current state, a persona model is put forward. A persona is a model of individuals that share preferences. It stores the preferences in a second layer. The detailed description, type and frequency of the stored preferences are missing. Moreover, the approach seems to be based on storing preferences on individual devices which is contrary to the approach presented in this chapter.

Personal spaces play an important role in group or individual self-definition: rather than just using them for a specific purpose, users pour their personalities and lives in the way they transform their environments [38]. "Domotic Effects" is one such framework that can enable the monitoring of the environment as the user wishes to perceive it, i.e., in terms of user-defined goals. Several other monitoring techniques have been proposed in the literature [63, 67], but the "Domotic Effects" offer several advantages. First, being an ontology based approach offers large scale adoption, application development, system prototyping and a solid technological infrastructure [15]. Second, it offers a unified framework for modeling, controlling and monitoring the environment. The modeling and controlling aspects were discussed in Chapter 6 and Chapter 8, respectively. This chapter discusses the monitoring aspect. Third, the approach is robust (see Section 7.4) and scalable.

7.6 Synopsis

This chapter presented a high level approach, based on the concept of Domotic Effects for modeling and interpreting a complex smart environments. The Domotic Effects framework, based on the DogEffects ontology, is general and extensible, and is easy to customize to specific application requirements. In particular, this chapter focuses on monitoring applications, where high level effects may be described resorting to Boolean expressions operating on device states.

The chapter presented extensive examples of Simple and Complex Effects over a sample home environment, and shows experimental results that prove that the complete state of the environment can be monitored using the Domotic Effects with a latency under 150 ms.

Table 7.2. Daily chores scenario performance parameters

Total DEs	Complex Effects	Maximum Level (ENN)	Effect Data Extraction Time	Organization Time	Average Evaluation Time (ms)
209	100	18	22506	182	5
309	200	21	44929	289	7
409	300	37	22107	363	11
459	350	57	30400	482	15
599	490	65	24170	662	24
709	600	122	36596	819	43
809	700	129	28261	887	33
909	800	120	33815	1017	30
1069	960	105	45106	1143	27
1159	1050	232	49721	1280	108
1309	1200	122	70404	1450	52
1609	1500	177	67311	1994	118

Table 7.3. Maximal Propagation Scenario Statistics

Device Complex Effects	Maximum Level (ENN)	Average Evaluation Time (ms)
100	38	21
200	124	18
300	117	59
400	178	41
500	270	55
600	100	144
800	265	176
1000	314	148
1200	201	313
1400	272	258

Chapter 8

Enforcement

This chapter discusses the control aspect of the DE framework. The control aspect involves the ability of users to manage and control their environments with the help of user-defined intentions or goals. This amounts to correctly mapping user goals in terms of a combination of devices having particular states (Figure 8.1).

For discussion in this chapter, the applicability of the DE framework to Boolean application domains is considered, i.e., domains in which user goals (effects) can either be true (active) or false (inactive) depending on the value(s) of the involved states and sub-states, that may be Boolean, discretely enumerate or real-valued. This covers most control applications and many monitoring use cases in smart homes, offices and industrial plants.

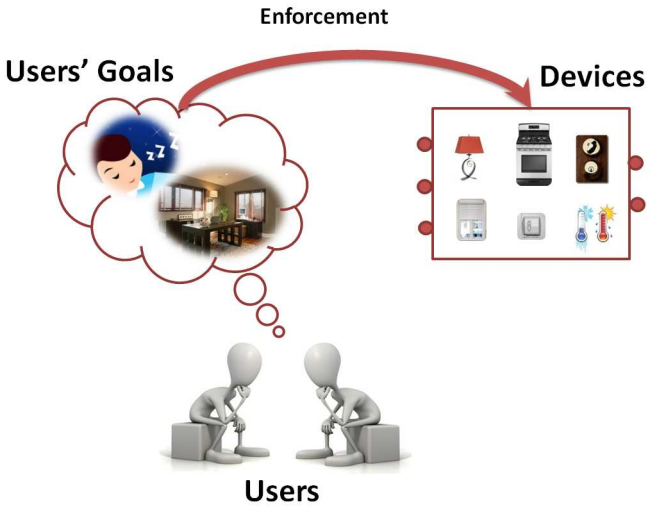


Figure 8.1. Enforcement

The chapter is divided into seven sections. The problem of effect enforcement

is formally defined in Section 8.1. The general approach adopted for enforcement is described in Section 8.2, and later Section 8.3 defines its architecture and implementation. Section 8.4 shows results of the experiments carried out on effects enforcement. Section 8.5 compares our approach to some related works and Section 8.6 concludes the chapter and highlights future work.

8.1 Problem Statement

Consider a smart environment with an AmI system managing it. A user can define several domotic effects (simple and complex) on top of the domotic structure, based on the effect operators defined for the environment. At any instant, each domotic effect has a value associated with it. The user has the ability to request $\mathcal{R}$ the AMI system to enforce a set of domotic effects on the environment. "Effect Enforcement" addresses the problem of finding at least one configuration that satisfies the user's request $\mathcal{R}$. The configuration refers to the combination of devices having particular states and sub-states.

The user request $\mathcal{R}$ is defined as a subset of the domotic effects present in the instance layer: $\mathcal{R} \subseteq \mathcal{I}$. In simple terms, the user request $\mathcal{R}$ is the subset of DE_i that the user wants to be active (true) at a given instant.

Given $\mathcal{R}$, effect enforcement tries to find a global domotic state $g \in \mathcal{G}$ where *all* the domotic effects $DE_i \in \mathcal{R}$ are true. This is equivalent to computing the *satisfiability* of the function

$$F_{\mathcal{R}}(g) : \prod_{DE_i \in \mathcal{R}} DE_i$$

8.2 Approach

In order to enable the user to enforce particular values of domotic effects on the environment, at least a configuration needs to be found which fulfills the user request $\mathcal{R}$, as defined in Section 8.1. To solve this problem the chapter proposes to transform the user's request into a Boolean satisfiability problem (SAT). In complexity theory, the satisfiability problem (SAT) is a decision problem, whose instances are Boolean expressions written using variables and basic Boolean operators, i.e., AND, OR, NOT.

To transform the user's request into a SAT problem, each domotic effect defined in the instance layer is mapped as a Boolean variable. The functionality of each effect operator defined in the AmI layer is mapped in terms of a Boolean sub-expression in the SAT problem. The value of the variable corresponding to the Simple Effect is true (active) if and only if the device is in a particular sub-state(s). Meanwhile, complex domotic effects can depend upon the values computed by multiple simple

or complex domotic effects and therefore the value corresponding to their variables are dependent on the values of their operands. As a consequence, the Boolean expressions for a complex domotic effect are constructed over its dependent domotic effects using the effect operator defined for it. The process is recursive, as the Boolean expressions for all the operands are constructed and conjuncted.

For example, consider a trivial user request $\mathcal{R}$ to enforce the *Illumination* use case on the environment. The *Illumination* use case will be represented as an "Illumination" CE inside the *DogEffects ontology*. Table 8.1 illustrates the functional representation of the use case. A SE is represented as SE(device, sub-state(s)). For instance, the representation of *CeilingLampIllumination* SE $CeilingLampIllumination = \mathbf{SE}(l2, OnState\_lamp2)$ depicts a lamp *l2* having *OnState_lamp2* sub-state. A CE is represented as $Operator(DE_1, DE_2 \ldots)$. For example, the *Illumination* CE is represented as *OR* operator applied over *ArtificialIllumination* and *NaturalIllumination* DEs.

In order to build the Boolean expressions for $\mathcal{R}$, all domotic effects are represented as Boolean variables. Then, the effect operator (and its type) attached with the "Illumination" CE is extracted, i.e., Or1 has type *OR*, which is followed by the extraction of operands (domotic effects) attached with the operator, i.e, *Natural illumination* and *Artificial Illumination*. After the extraction of operator and operands, the first Boolean expression becomes $Illumination = \mathbf{OR}(NaturalIllumination, ArtificialIllumination)$. Then, the Boolean expressions for *Natural illumination* and *Artificial Illumination* CEs are constructed iteratively, until SEs are reached. The Boolean expressions for the "Illumination" CE are shown in Table 8.1. All the Boolean expressions are then conjuncted and the value of Boolean variable associated with *Illumination* CE is set to true.

Table 8.1. Illumination functional form (CE_B)

$Illumination = \mathbf{Or}(ArtificialIllumination, NaturalIllumination)$
$ArtificialIllumination = \mathbf{Alternate}(CeilingLampIllumination, \quad MirrorLampIllumination)$
$MirrorLampIllumination = \mathbf{And}(LeftMirrorLampIllumination, RightMirrorLampIllumination)$
$RightMirrorLampIllumination = \mathbf{SE}(l9, OnState\_lamp9)$
$LeftMirrorLampIllumination = \mathbf{SE}(l8, OnState\_lamp8)$
$CeilingLampIllumination = \mathbf{SE}(l2, OnState\_lamp2)$
$NaturalIllumination = \mathbf{SE}(ShutterBath, UpStateValue\_ShutterBath)$

To put it concisely, the user can request $\mathcal{R}$ several domotic effects DE_i, to be enforced on the environment. The Boolean expressions for all domotic effects DE_i present in the user request $\mathcal{R}$ are constructed and conjuncted. The process is recursive, as the Boolean expressions for all the operands are constructed too. After getting all the Boolean expressions, the ones corresponding to the user request are enforced as SAT constraints, i.e., the values of variables corresponding to DE_i are set to true.

Once the Boolean expressions are constructed, conjuncted and the values of the

variables corresponding to DEs in $\mathcal{R}$ are set, they are fed to a SAT solver to determine values of other variables (corresponding to other DEs) under which the values of the DEs in $\mathcal{R}$ will hold. Since SEs represent terminal nodes of the expressions, the values of the variables corresponding to SEs will give us a combination of devices and their particular states and sub-states fulfilling the user's request $\mathcal{R}$. In short, bringing the combination devices into particular states and sub-states would fulfill $\mathcal{R}$. In case, the user request $\mathcal{R}$ is not satisfiable, the enforcement procedure is canceled and the user is informed. Additionally, it is likely that several configurations satisfy $\mathcal{R}$ which gives system designers an option to find an optimal configuration based on some constraints. For example, a configuration that minimizes energy consumption (Chapter 9).

8.3 Architecture

This section describes a generic, modular and extensible architecture for the implementation of the effect enforcement approach (defined in Section 8.2) inside the smart environments and highlights the procedure to extend the architecture to define new effect operators. The architecture consists of a *Domotic Effect Enforcement* module and the *DogEffects* ontology containing all the domotic effects defined for the environment.

8.3.1 Domotic Effect Enforcement

Given a user's request $\mathcal{R}$, the *Domotic Effect Enforcement* module finds a configuration to fulfill $\mathcal{R}$. It is responsible for extracting all domotic effects from the *DogEffects* ontology, receiving user's request for enforcing particular values of for a set of domotic effects, transforming the user request into a SAT problem, and finding at least one configuration that fulfills the user's request (or otherwise finding conflicts).

Figure 8.2 presents the logical architecture. The module comprises querying, solver, library components and an effect operator store.

The querying component queries the *DogEffects* ontology for all the domotic effects and then it organizes all the domotic effects in a hierarchical internal data structure which is similar to the organization of domotic effects in the *DogEffects* ontology. Whenever any addition or editing in the *DogEffects* ontology occurs, the querying component reconstructs the data structure.

As defined in Section 8.2, to transform the user's request into a SAT problem, the effect operators defined in the AmI layer should be defined in terms of Boolean sub-expressions. The Effect Operator store contains the Java code for each defined

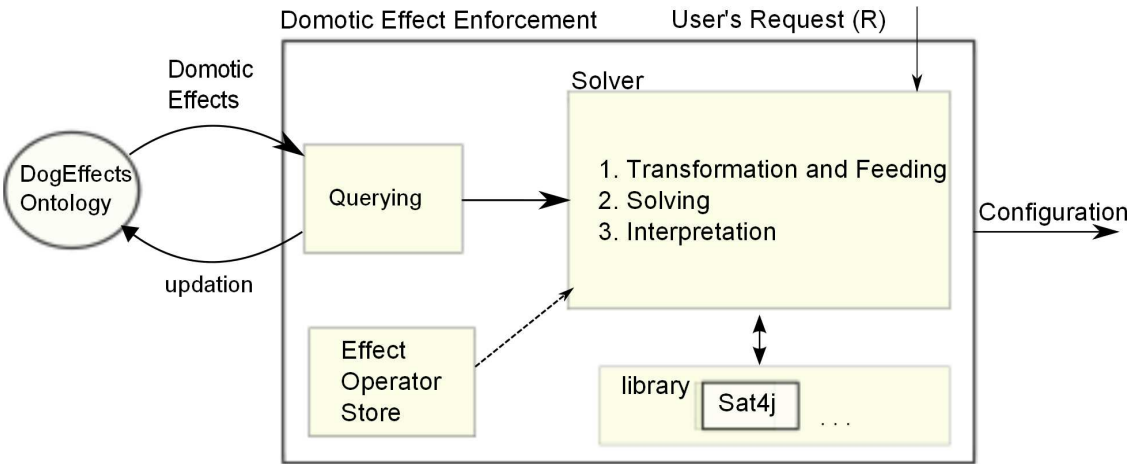


Figure 8.2. Domotic Effect Enforcement Architecture

effect operator providing methods to create the corresponding sub-expressions in terms of basic Boolean operators.

The user's request is handled by the Solver component. The Solver component transforms the user request into a SAT problem, finds a configuration that satisfies the user's request and then it enforces the configuration on the environment. For each $\mathcal{R}$ the steps are detailed below:

- *Transformation and Feeding:* It comprises transforming the user's request for particular values of domotic effects in to a correct set of Boolean equations and applying constraints over them. Then, these Boolean equations and constraints are fed to the SAT solver. Currently, the Sat4j solver [68] is used.
- *Solving:* Based on the set of Boolean equations, the Sat4j solver determines (if possible) the values of all the variables inside the Boolean equations. There may be cases in which the values of domotic effects requested by the user, i.e., $\mathcal{R}$, can not be satisfied at all.
- *Interpretation:* It comprises finding the values of the variables corresponding to SEs and interpreting them in terms of devices and their states and substates.

The Sat4j library requires that the input is in the Conjunctive Normal Form and each variable in the SAT problem is represented by an integer positive number (negative numbers represent complemented variables). Therefore, the querying component assigns a unique integer to each domotic effect. The transformation begins by taking each user requested domotic effect and determining the effect operator that acts among its children. Once the effect operator type of a domotic effect is

```
public class {NewOperatorName} extends EffectNode {

    public NewOperatorName() {

       super();
       /*
       The passed operator name should be
       equal to the effect operator name
       defined in the DogEffects ontology.
       */
      super.setOperator("NewOperatorName");
    }

    @Override
    public void setEquation(GateTranslator gator) {
       ...
    }
}
```

Figure 8.3. Procedure to define a new effect operator.

determined, the corresponding Java class in the "Effect Operator Store" creates its sub-expression in terms of basic Boolean operators and appropriate Boolean equations are constructed for the domotic effect and its children domotic effects. These Boolean equations are fed to the Sat4j solver to determine a configuration satisfying them.

8.3.2 Extensibility

The Domotic Effects approach is extensible, and AmI designers have the ability to define new and different Boolean operators depending on the environment. Appropriate implementation of new operators should be included in the Effect Operator store for constructing Boolean equations. Figure 8.3 shows a class template to implement an effect operator declared in the AmI layer inside the effect operator store.

When a new effect operator is defined in the AmI layer of the "DogEffects" ontology, the Effect Enforcement module is easily extended as follows:

1. A new class is defined that extends the *EffectNode* class, which is an abstract class representing the general properties of Domotic Effects;

2. For the construction of Boolean equations, the mapping of the effect operator using basic Boolean operators is provided inside the *setEquation()* method. The *setEquation()* method receives a parameter (*gator* of type *GateTranslator*) that represents all the Boolean expressions. The "GateTranslator" is a Sat4j library class which provides functionalities of the SAT's basic Boolean operators like Not, And and Or. One can define any kind of effect operator in the AmI layer as long as it can be defined in terms of basic Boolean operators.

Currently, the effect operator store includes the Complement, And, Or and Alternate operators. The Complement effect operator represents an invert relationship, and is mapped as a Not Boolean operator in SAT (Algorithm 7).

Algorithm 7 Complement effect operator

```
nodeNumber = node.getNodeNumber();
child=node.getFirstChild();
literals= new VecInt();
literals.push( child ) ;
gator.not(nodeNumber,literals);
return true;
```

Algorithm 8 shows the mapping of the And effect operator in terms of basic Boolean operators. The Or effect operator algorithm is similar, but it is mapped as an Or Boolean operator in SAT.

Algorithm 8 And effect operator

```
nodeNumber = node.getNodeNumber();
children=node.getChildren();
literals= new VecInt();
for all children as child do
    literals.push( child ) ;
end for
gator.and(nodeNumber,literals);
return true;
```

The Alternate effect (Algorithm 9) operator represents a function which is true when only one of its children is active.

Algorithm 9 Alternate effect operator

```
nodeNumber = node.getNodeNumber();
children=node.getChildren();
literals= new VecInt();
globalNumber → Counter for temporary variables;
globalCounter → list of temporary variables;
literals.clear();
int_List = newList();
int_List.addAll( children ) ;
for count:=0 ; count < int_List.size(); count++ do
    literals.clear();
    for innercount:=0 ; innercount < int_List.size(); innercount++ do
        if count =innercount then
            literals.push( int_List.get(innercount) ) ;
        else
            literals.push( - int_List.get(innercount) ) ;
        end if
    end for
    globalCounter.add(globalNumber++);
    gator.and(globalNumber, literals);
end for
literals.clear();
for all globalCounter as each do
    literals.push( each ) ;
end for
gator.or(nodeNumber,literals);
return  true;
```

8.4 Experimental evaluation

The "Domotic Effect" modeling framework was developed to be integrated with the Dog2.0 [65] smart home gateway and therefore two modules, i.e., Ontology Loader and Domotic Effect Enforcement, were built using Eclipse Equinox, which is an

implementation of the OSGi framework [37]. The OSGi framework brings versatility and modularity by providing each module as a service called a bundle. Experiments were conducted to measure different performance parameters of the "Domotic Effect Enforcement" module. These performance parameters include the time needed to transform a user's request $\mathcal{R}$ into a SAT problem, and if possible, to find at least a configuration that satisfies the user's request.

A complete house environment was simulated. The domotic structure was modeled as an instance of the DogOnt ontology. A new *TestDogEffectSolution* bundle was developed to perform the experiments and to measure performance parameters for each experiment. Six use cases were defined $\{CE_A \dots CE_F\}$ (see Section 8.4.1). In order to define the use cases in the instance layer 190 intermediate domotic effects (CEs and SEs) were declared. A number of iterations were performed enforcing different user requests $\mathcal{R} \subseteq \mathcal{I}$. In the experiments, a total of 63 iterations were performed, corresponding to each possible $\mathcal{R}$ over 6 use cases (omitting the trivial $\mathcal{R} = \emptyset$). The experiments were conducted on an Intel Core i5 CPU running at 2.6 GHz with 4GB of RAM.

8.4.1 Use cases

In order to carry out the experiments some use cases defined over a home. Figure 7.8 shows the structure of the house: The house has a bed room, a living room, a lobby, a bath room, a store and a kitchen, and is equipped with with several automatic devices/appliances like lamps, oven, television, door actuators, window actuators, shutter actuators, gas heaters etc. Based on simple domotic effects and using the set of Boolean operators encoded in the AmI layer, several complex domotic effects (CE_A through CE_F) have been defined. The use cases are provided in Chapter 7 (Section 7.4.1).

8.4.2 Results and Discussion

In the experiments, a total of 63 iterations were performed, corresponding to each possible $\mathcal{R}$ over the 6 use cases. In the first experiment, two performance parameters were measured:

- the time taken by the "Domotic Effect Enforcement" module to construct the set of Boolean equations and to feed them to the Sat4j solver (construction time, CT);
- the time to find at least one configuration that satisfies the set of Boolean equations (solution time, ST).

Both time measurements were taken at the milliseconds level, and Figure 8.4 shows the performance measures for all the 63 iterations. Each cell contains a combination expressed as $CT + ST$. The results are represented as a Karnaugh map for easier reading and identification of the simultaneously enforced domotic effects.

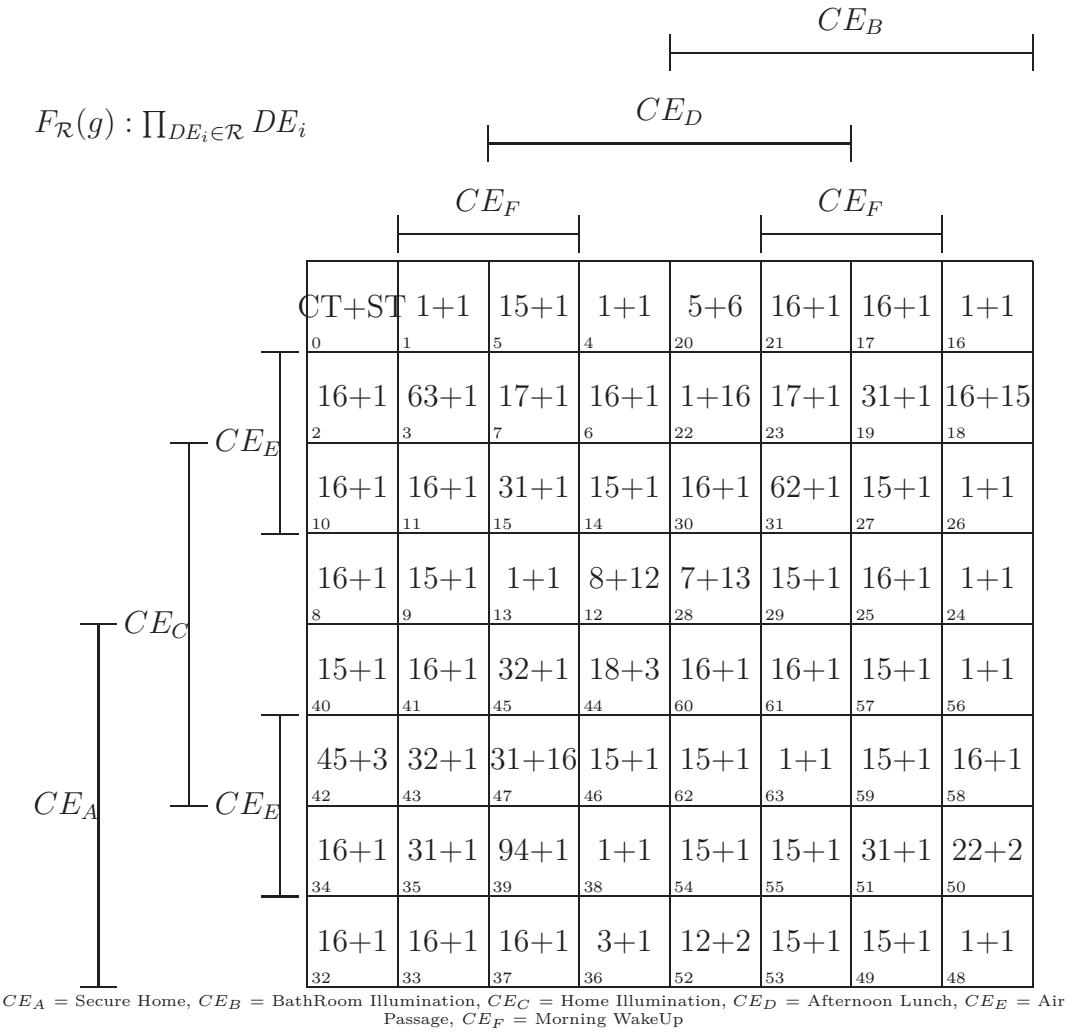


Figure 8.4. CPU time measurements (in ms, CT+ST)

It can be seen that the "Domotic Effect Enforcement" module is quite responsive and in all cases the time for construction of Boolean equations and determining configuration is less than 100 ms. The module was developed to be used in real world applications and therefore completing the user's requests in few milliseconds shows that the proposed approach is promising. On the other hand, finding at least a configuration to satisfy user's request depends upon the number of variables involved

in the Boolean expression. Though the measured time is in few milliseconds for this experiment, the time may vary according to the number of variables involved in the Boolean expression (more discussion in Section 8.4.3).

The second experiment is highlighted in Figure 8.5. For each iteration, Figure 8.5 shows the total number of configurations that can satisfy a set of Boolean equations (total configurations, TC) and the number of devices involved in the construction of Boolean equations (Dev). Each cell contains a combination like $TC_{\{Dev\}}$. The clusters of unsolvable problems ($TC = 0$) correspond to incompatible user requests, such as air flow and security.

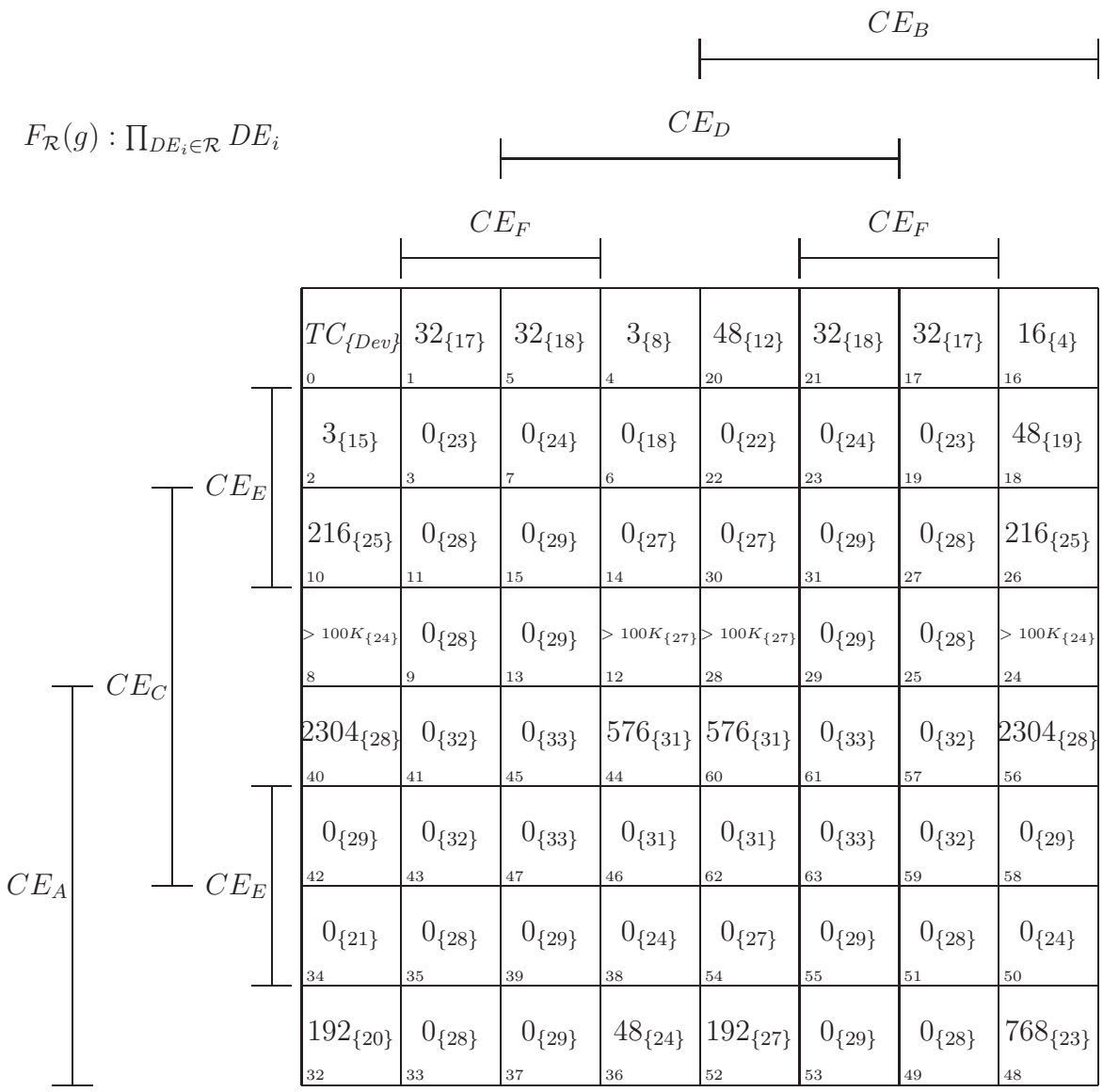
8.4.3 Extensibility and Scalability

In order to become a potential candidate for a wider adoption, an approach should at least have two characteristics, i.e., extensibility and scalability. The "Domotic Effects" framework should also demonstrate such characteristics. In the framework the question of *extensibility* can be raised at two levels. First, whether the modeling (DogEffects ontology) can be extended to other domains and second, whether the approach proposed in this chapter, to provide control over Boolean application domain, can be extended for more operators.

The extensibility of the framework depends upon the correct semantic modeling of the framework and the applicability of the framework to several domains. The semantic modeling of the framework allows the framework to be expendable across several application domains, i.e., Boolean domain, Energy Saving Domain (Chapter 9). In fact, this chapter discusses the specialized case of the "Domotic Effects" modeling framework in the Boolean application domain. The framework is based on an Ontology-Based approach and therefore it has an advantage of large-scale adoption, application development, system prototyping, solid technological infrastructure as acknowledged in [15].

In the Boolean application domain, the extensibility of the proposed approach is achieved by providing the AmI designers control over defining and implementing their own Boolean operators (see Section 8.3). This chapter defines the fundamental Boolean operators and their implementations, but the designers are free to define any operator that can be translated into a Boolean expression.

Meanwhile, the scalability of the proposed approach depends upon the robustness of the Sat4j solver. Sat4j is a mature, open-source library providing access to SAT-related technologies to Java programmers. While the core SAT engine may not be competitive against commercial SAT solvers, the results of the library on pseudo-boolean problems are reasonable [68]. From the experiments it can be observed that the solver can handle hundreds of domotic effects in few milliseconds. In fact, the number of domotic effects needed for homes and small buildings will be in hundreds and the Sat4j solver will be robust enough to solve Boolean expressions in near



$TC_{\{Dev\}}$ 0	$32_{\{17\}}$ 1	$32_{\{18\}}$ 5	$3_{\{8\}}$ 4	$48_{\{12\}}$ 20	$32_{\{18\}}$ 21	$32_{\{17\}}$ 17	$16_{\{4\}}$ 16
$3_{\{15\}}$ 2	$0_{\{23\}}$ 3	$0_{\{24\}}$ 7	$0_{\{18\}}$ 6	$0_{\{22\}}$ 22	$0_{\{24\}}$ 23	$0_{\{23\}}$ 19	$48_{\{19\}}$ 18
$216_{\{25\}}$ 10	$0_{\{28\}}$ 11	$0_{\{29\}}$ 15	$0_{\{27\}}$ 14	$0_{\{27\}}$ 30	$0_{\{29\}}$ 31	$0_{\{28\}}$ 27	$216_{\{25\}}$ 26
$> 100K_{\{24\}}$ 8	$0_{\{28\}}$ 9	$0_{\{29\}}$ 13	$> 100K_{\{27\}}$ 12	$> 100K_{\{27\}}$ 28	$0_{\{29\}}$ 29	$0_{\{28\}}$ 25	$> 100K_{\{24\}}$ 24
$2304_{\{28\}}$ 40	$0_{\{32\}}$ 41	$0_{\{33\}}$ 45	$576_{\{31\}}$ 44	$576_{\{31\}}$ 60	$0_{\{33\}}$ 61	$0_{\{32\}}$ 57	$2304_{\{28\}}$ 56
$0_{\{29\}}$ 42	$0_{\{32\}}$ 43	$0_{\{33\}}$ 47	$0_{\{31\}}$ 46	$0_{\{31\}}$ 62	$0_{\{33\}}$ 63	$0_{\{32\}}$ 59	$0_{\{29\}}$ 58
$0_{\{21\}}$ 34	$0_{\{28\}}$ 35	$0_{\{29\}}$ 39	$0_{\{24\}}$ 38	$0_{\{27\}}$ 54	$0_{\{29\}}$ 55	$0_{\{28\}}$ 51	$0_{\{24\}}$ 50
$192_{\{20\}}$ 32	$0_{\{28\}}$ 33	$0_{\{29\}}$ 37	$48_{\{24\}}$ 36	$192_{\{27\}}$ 52	$0_{\{29\}}$ 53	$0_{\{28\}}$ 49	$768_{\{23\}}$ 48

CE_A = Secure Home, CE_B = BathRoom Illumination, CE_C = Home Illumination, CE_D = Afternoon Lunch, CE_E = Air Passage, CE_F = Morning WakeUp

Figure 8.5. Number of solutions TC and involved devices Dev

real-time. However, for large industries the "Domotic Effects" modeling framework may require commercial SAT solvers (since domotic effects may be in thousands) or other approaches such as problem partitioning. This aspect needs to be investigated further.

8.5 Related Works

Garcia-Herranz et al. [39] proposes an application-independent indirect control programming system to program complex behaviors with the simplicity required to allow novice users to program their smart environments. The objective is to allow users to create powerful and personal behavior without expert assistance. They developed a rule-based language for a modular agent system [40]. The rules allow expressing behaviors of type "When *triggers*, if *conditions*, then *action*". The rule language lacks the flexibility of providing different courses of action to achieve a solution. Moreover, it does not provide abstraction to allow AmI designers to develop techniques independent of devices.

Katasonov [42] motivates to build *Digital fluency* in smart environments by enabling the non programmers to design, create and modify their smart environments. The chapter proposes a higher level of abstraction in application design, on-the-fly development, flexibility with respect to adding new devices and software components. To build higher level of abstraction, an ontology that contains the hierarchy of tasks at the higher level is needed. The chapter mentions defining tasks and their corresponding subtasks, without providing the organization of tasks in ontologies and the mechanism to achieve tasks. Our proposed solution in this chapter not only provides details of organizing abstract goals but also provides mechanism to achieving those goals.

D-HTN [56] is a planning system for AmI applications, based on the hierarchical task network (HTN) approach, that is able to find courses of actions to address given goals. It combines concepts of both centralized planning and distributed planning in agent theory but the language (*task network* [69]) that is used to store goals and their courses of actions is static. Our proposed solution also provides a hierarchical structure to store goals and their courses of actions but allows AmI designers to define their own language of translation.

In [41] an Artefact framework is proposed which allows end users to deploy ubicomp systems easily in a Do-it-yourself fashion. Secondly, it allows developers to write applications and to build augmented artefacts in a generic manner. The Artefact framework provides a layered architecture where basic artefact functionalities are combined in a core component. Additional augmented features can be added as plugins into the core. Each augmented feature is called a profile. Each profile defines a specific functionality and implements the underlying logic of the functions,

e.g., room temperature, lamp brightness. Though the profiles provide abstraction to hide the heterogeneity of the underlying devices, their functionality corresponds to the functionalities of devices and lacks the focus of providing a more generic goal that the user might wish to achieve.

Rashidi et al. [43] proposes a software architecture which incorporates learning techniques to discover patterns in resident's daily activities. The activity pattern are observed by monitoring the changes of states in different devices around the house. After discovering an activity pattern, it stores the activity pattern and its related temporal knowledge in a Hierarchical activity model (HAM). HAM captures the temporal relationships between events in an activity by explicitly representing sequence orders in a tree structure containing Markov chains at the bottom level. The activities are stored based on individual devices in the house which does not allow observers to see the bigger picture at higher level of user goal. Though currently our solution does not employ learning patterns to automatically extract repetitive tasks but it can easily be employed in our proposed organization of Domotic Effects. Moreover, observing patterns at an abstract level can give a more clear picture of user's intentions instead of focusing on individual device or chain of devices.

Cheng et al. [45] proposes a smart homes reasoning system called ASBR system. The system learns user's preferences by adaptive history scenarios and put forwards a way to rebuild reasoned knowledge in other smart homes. They proposed that contextual information can be extracted and reasoned as a set of scenarios. In addition, the system can derive personalized habits and store them in OWL files. They do not provide an organization mechanism and the concept of scenario is different from our proposed effect. Effects are different from scenarios as it is not a storage of historical events or repetitive tasks. Though repetitive tasks can be mapped onto effects, our approach provides complete control to the residents to define their own abstract level control which are ultimately resolvable to a set of devices in certain states. The effect based approach is designed to be extensible to smart environments in general.

Dey et al. [46] proposes a software infrastructure solution to detect the current states of the environment (called Context) and take action based on it. The infrastructure is focused on developing context aware applications. Though the concept of Domotic Effect can be used to monitor the current state of the environment (Effect Evaluation), the focus of this chapter in particular is to enforce generic goals on the environment. Our modeling allows to handle both tasks in a more simple manner. Moreover, currently the enforcement implementation focuses on Boolean application domain states, but the DogEffects ontology can be used to monitor devices with continuous states, which was described as the limitation of the infrastructure in [46]. The Effect Enforcement implementation is designed with the extensibility in mind, which is missing in [46]. Generally, context represents the knowledge of external conditions and their complexities in the environment. This knowledge is used in

some way to make particular action(s) choice. The question of how the action(s) are actually performed is not part of the context but rather is a characteristic of the system that handles the environment. Our current chapter focuses on this internal characteristic of the system rather than collecting conditions that triggered those actions.

Reference [57] proposes a middleware architecture for smart home systems. The architecture has pervasive, composition and user layers. The composition layer contains a CSP based planner which computes a plan, that is, a sequence of actions that need to be applied in order to satisfy a user's goal. The goals are pre-defined in a declarative manner. Our approach does employ declarative manner to describe goals called domotic effects but it is more flexible as it allows the AmI designers to define operators to manage combinations according to environments and the user's can define their own domotic effects based on those operator. Moreover, the planner in [57] takes time in seconds to construct a problem and determine results, whereas the effect enforcement module takes time in milliseconds for both construction of set of Boolean equations and finding a solution.

In [49, 50] a goal based interaction has been proposed, and extended in [51], that takes a user's goal and finds a path achieving the goal. The use of propositional calculus is advocate, however, unlike our chapter, [51] lacks implementation details, i.e., a proper mapping from user's goal to propositional calculus and its interpretation. To support goal based interaction [49–51] advocate that each device in the environment implements an event processing pipelines consisting of user interface, control application and actuators. Moreover, they make a huge assumption that each device shares data inside event processing pipelines across all the devices present in the environment, creating a SODAPOP infrastructure. In real world, an environment comprises devices from several different and competing vendors which may not be willing to expose to other vendors internally stored information of their devices, or may not have enough computing capabilities. Domotic Effects are a more centralized approach, where all relevant information about devices is available in the automation gateway and no requirements are imposed onto the devices, thus providing easy and immediate interoperability with existing devices from different vendors.

Domotic Effects provides the end user, the ability to personalize a smart environment, as well as allows AMI application designers to design, develop and manage based on a higher level of abstraction. While [43, 45] proposes a complete independent control solution based on the learning pattern of a user's activity, the [39,42,56] advocates enabling non programmers to create and manage their smart environments according to their wishes. Providing a complete independent control extracted from a user's activity might not be a very good idea as it does not allow people to program their personal spaces. It also raises a new set of problems like privacy and security issues. Instead of focusing on user's goal or intention, it focuses on a set of

devices and their activity. It can said be that learning algorithm discover pattern of device activity instead of user's intention and activity. The latter, on the other hand provide end user programming environment but underlying structure of organizing goals and their different courses of actions is missing. Though [41, 49, 56] provide goal based interaction mechanism, they lacks the flexibility, separate views of development for system designers and users and applicability to different application domains.

8.6 Synopsis

This chapter presented a high level approach, based on the concept of Domotic Effects for modeling and satisfying user requests in complex smart environments. The Domotic Effects framework, based on the DogEffects ontology, is general and extensible, and is easy to customize to specific application requirements. In particular, this chapter focuses on control and monitoring applications, where high level effects may be described resorting to Boolean expressions operating on device states.

The chapter presented extensive examples of Simple and Complex Effects over a sample home environment, and shows experimental results that prove that high-level user requests are satisfied in less than 100 ms, thanks to the mapping of the request into a SAT problem that may be efficiently solved.

Future work will include extending the approach in several directions: allowing real-time evaluation and monitoring of Domotic Effects (in the Boolean or Real domain).

Chapter 9

Optimization

In the last decade, intelligence emerged as the basic component to design modern home and building automation systems. The term "intelligence" implies a provision of automated control over the buildings to solve interoperability issues among devices from different vendors, to sense the environment, to provide context-aware services to the residents and to manage safety and security issues. Regardless of how ambitious and diverse the notions might seem, the research community has demonstrated the ability to achieve such goals using pilot projects [65, 70–72]. In the past few years, energy efficiency has become a key requirement for designing modern buildings and industries. The approaches in this regard not only rely on improving building structures and adopting more efficient appliances but also aim at increasing user awareness towards their energy usage.

Energy efficiency has become one of the major concerns in today's life, impacting almost all human activities, from industrial and commercial, to leisure and vacation. According to the statistics from the US Department of Energy and the European Union Energy Commission, global energy consumption is likely to increase in the next decade, with residential and commercial buildings raising their aggregate figure to 20-40% of the total yearly consumption. If only electricity is considered, the consumption share allocated to buildings is suppose to increase up to 73%, evenly distributed between residential and commercial buildings [73].

To cope with increasing energy needs the smart grid is a promising infrastructure [74] which focuses on demand side management. It provides customers an ability to make informed decisions about their energy consumption by adjusting timing and quantity of their electrical usage [16, 17]. This flexibility is enabled by pricing policies for electrical usage over time [18, 19] and/or by dynamic demand scheduling algorithms to optimize energy services in buildings [20, 75]. The smart grid infrastructure requires a two-way communication through which appliances can be monitored and controlled by a control center installed on the premises of the energy provider which may lead to privacy and security issues [76].

A complementary approach to energy management is the local optimization of energy consumption using a locally installed Energy Management System (EMS) on the building premises. Most EMS focus on making the consumer more aware of their electrical power usage and/or providing methods to share this information with energy providers or third party application developers [23,77,78]. The research focuses on different graphical illustrations of data related to consumed energy to ease consumer comprehension [21,22] and on different tools and methodologies to share this data over the web (Chapter 12). All these approaches need active user participation in order to implement energy management strategies.

This chapter proposes a more automated approach, where the EMS may automatically act on appliances and control their power consumption, while satisfying user requirements about the current environment state. The presented approach is based on two pillars: the availability of an explicit model for the smart home (such as provided by intelligent home gateways), and the expression of user needs in a more abstract way. The environment should be controlled by "user intelligible goals" that represent the state of the environment perceived by the user, on an abstract level. For example, the user may wish to illuminate a room and this may be done by acting on lamps, curtains and shutters in different ways. Therefore, the user achieves the effect of illuminating the room on an abstract level.

The chapter describes a novel approach to optimize the energy usage in a building while achieving user intelligible goals. The main contributions of the chapter are: adopting an explicit formal modeling for user goals (based on the Domotic Effects modeling framework); proposing an architecture that is compatible with existing ambient intelligence solutions; describing an algorithm based on Boolean satisfiability (SAT) for computing the optimal solution and integrating the SAT algorithm with a suitable heuristic in order to tackle combinatorial explosion.

The chapter is divided into seven sections. Section 9.1 describes the theoretical framework of the chapter. The problem tackled in the chapter is then formalized in Section 9.2, while the approach adopted for optimization is described in Section 9.3. Section 9.4 shows detailed results of a preliminary experiment. A literature overview is provided in Section 9.5. Finally, Section 9.6 concludes the chapter.

9.1 Formalism

9.1.1 Representing Power with Domotic Effects

Each device, in each operating state, consumes some amount of electrical power[1], thas is represented as a real-valued Simple Effect
$P(s)$, $P : S(d) \rightarrow \Re^+$.

The instantaneous power consumed by the whole environment is therefore represented as a Complex Effect
$\mathcal{P} : \mathcal{G} \rightarrow \Re^+$ aggregating all individual power measurements:

$$\mathcal{P}(g) = \sum_{d \in \mathcal{D}} P(s(d)) \tag{9.1}$$

9.1.2 Domotic Effect Enforcement

As discussed in Chapter 8, for Boolean valued domotic effects the user can request the system to enforce particular domotic effects. Satisfying user requests amounts to solving the Boolean function $F_{\mathcal{R}}(g)$ defined as:

$$F_{\mathcal{R}}(g) = \prod_{DE \in \mathcal{R}} DE \tag{9.2}$$

The problem was addressed by transforming the user request $\mathcal{R}$ into a Boolean satisfiability problem (SAT).

9.2 Problem Statement

Given the definitions in the previous sections, the goal of in this chapter is to compute the minimum value of $\mathcal{P}(g)$, while satisfying the user request $\mathcal{R}$. This correspond to a constrained optimization of $\mathcal{P}(g)$ subject to the Boolean constraint $F_{\mathcal{R}}(g)$. In this chapter, the basic SAT-based approach for effect enforcement has been extended to find a solution with minimum power consumption. Since the set of possible solutions may be extremely large, a suitable heuristic is proposed to get a satisfactory low-power solution in acceptable CPU times.

Energy management techniques should in fact respond in near real-time (NRT), by acting on a time scale comparable with user requests and device state change frequencies. Normally, the computational delay should be less than a few seconds.

[1] in this chapter active instantaneous power is considered, although the modeling approach can be trivially extended to other electrical properties

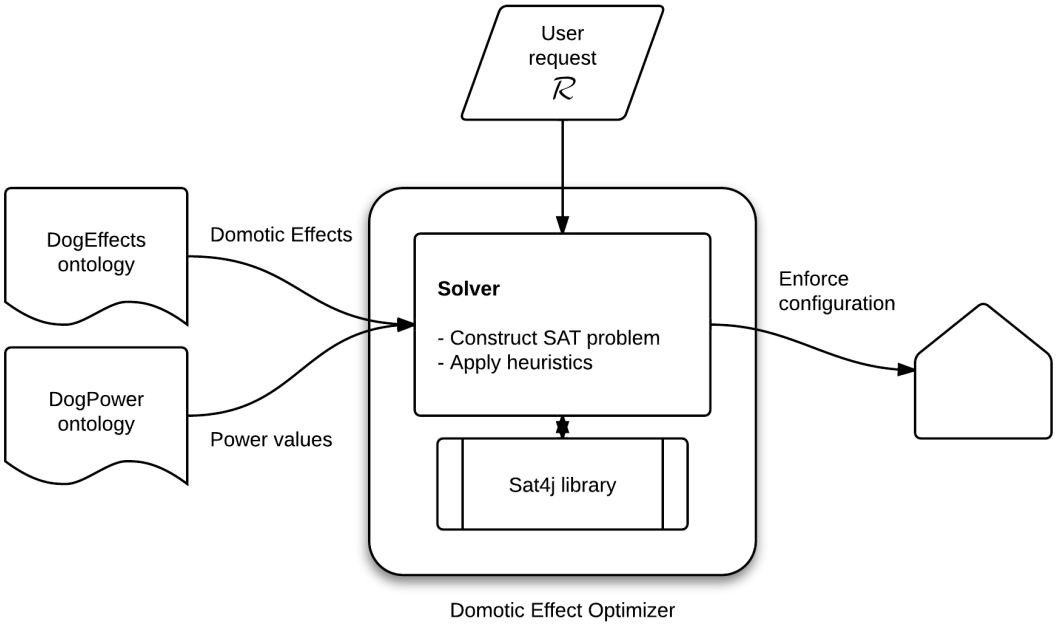


Figure 9.1. Architecture of proposed approach

9.3 Proposed Approach

To minimize power $\mathcal{P}(g)$ subject to user-requested domotic effects $F_{\mathcal{R}}(g)$, a *Domotic Effect Optimizer* module is developed (Figure 9.1). The *Domotic Effect Optimizer* receives a user request and transforms it into a SAT problem, that is solved to find valid configuration(s). The number of configurations may be zero or more. If zero, the user request is not satisfiable. Otherwise, a configuration with minimum power consumption needs to be determined.

An exhaustive enumeration approach can be adopted, in which each valid configuration is checked for its total power consumption value $\mathcal{P}(g)$ and the one with minimum value is enforced on the environment. However, the enumerated approach becomes computationally expensive and practically infeasible if the number of configurations is too large.

To guarantee near real-time (NRT) execution, the number of configurations returned by the SAT solver is compared with an experimentally-tuned configurable threshold T_c that roughly corresponds to the number of configurations that may be enumerated in one second. If the number of configurations is lower than T_c, then exhaustive enumeration is fast enough to achieve NRT responsiveness. Otherwise, a heuristic is applied to guarantee results in NRT, even if the absolute optimum is

no longer guaranteed.

The complete approach is highlighted in Figure 9.1. At startup, the *Domotic Effect Optimizer* queries the DogEffects and DogPower ontologies to get all the domotic effects and their associated (device and power) information. The *Domotic Effect Optimizer* transforms the user request for particular values of domotic effects in to a correct set of Boolean equations and constraints, constructing a SAT problem. Then it feeds the SAT problem to a SAT solver. For our current implementation, the Sat4j [68] solver is used. Based on the set of Boolean equations, the Sat4j solver determines (if possible) the total number of configurations that satisfy the set of Boolean equations.

9.3.1 Heuristic

A novel power minimizing heuristic is proposed to determine in near real-time a configuration that consumes minimal electrical power and satisfies the user's request. Since the heuristic is called only when the solution space is large ($> T_c$), this degree of freedom is exploited by trying to switch off appliances that have the highest electrical power consumption.

Forcing a device to be switched off reduces the size of the solution space, but it might render the SAT problem infeasible. Therefore a greedy approach was adopted which tries to force all the involved devices off, one by one, starting from the highest-consuming SE. Those SE that render the problem infeasible are kept free in the SAT problem. The others are forced off. There is no guarantee that the configuration received after applying the heuristic has the minimum power consumption. There might be cases in which the combination of small power consuming devices in total consumes more than the device with high power consumption, but such conditions are rare and the experiments (Section 9.4) prove the configuration with minimum power value is usually achieved. Algorithm 10 shows the overall steps taken to find the optimized configuration, and Algorithm 11 details the greedy procedure used to simplify the SAT problem.

Algorithm 10 Overall approach

```
SAT = SAT problem derived from F_R(g)
if (solvable(SAT)) then
   if (num_solutions(SAT)> T_c) then
      SAT = Heuristic Algorithm (SAT)
   end if
   device states = solve (SAT)
end if
```

Algorithm 11 Heuristic Algorithm (SAT)

```
sorted_SE = sort( all_SE, decreasing_power )
for all ( SE in sorted_SE ) do
   SAT' = SAT ∩ ( SE=false )
   if (solvable(SAT')) then
      SAT = SAT'
   end if
end for
return SAT
```

9.4 Experimental evaluation

To prove the validity of the proposed approach and measure the performance of the proposed heuristic, a set of experiments were carried out. The "Domotic Effect" modeling framework was developed and integrated with Dog2.1 [65] as a new *Domotic Effect Optimizer* bundle running in the Dog OSGi framework.

A complete house environment was simulated, whose domotic structure was modeled as an instance of DogOnt ontology. A new test bundle was developed to test the approach and the proposed heuristic. The house environment contains 1500 user-defined Domotic Effects. These DE correspond to generic goals like securing or illuminating the house.

The experiments have been run on a standard personal computer with a quad-core Intel i5 processor and 4GB of RAM.

9.4.1 Use Cases

In the experiments, all possible combinations of six use cases were enforced on the environment one after another. These use cases $\mathcal{I}$ were *Secure Home*, *Bath Room Illumination*, *Home Illumination*, *Afternoon Lunch*, *Isolated Kitchen* and *Morning Wakeup* scenarios. The "Secure Home" use case secures all the exit points of the house, i.e., all exit doors and windows. This use case comprises many DEs providing the ability to secure different rooms of the house. This can be used in case of emergency, theft, robbery or fire etc. The "Bath Room Illumination" combines small use cases that represent alternative ways to illuminate the bathroom. The "Home Illumination" requires that all the rooms of the house are illuminated. Illumination can be either natural or artificial. The "Afternoon Lunch" deals with the daily routine of cooking lunch inside the kitchen. The "Isolated Kitchen" use case represents isolating the kitchen from the rest of the house during cooking hours; this scenario does not consider the energy spent for cooking, since that action is not automated. The "Morning WakeUp" use case maps a typical scenario when a resident wants to perform a sequence of activities after waking up in morning, like illuminating the bedroom, the kitchen and the bathroom, switching off the gas heater inside the bedroom, switching on the kitchen television and the bathroom radio.

Since $|\mathcal{I}| = 6$, there were $2^6 = 64$ possible user requests, or 63 if the trivial $\mathcal{R} = \emptyset$ is omitted, where no domotic effect is enforced.

Table 9.1 and Table 9.2 show the total number of configurations and the time taken by the *exhaustive enumeration* approach to find the total number of configurations, compute their power consumption and determine the configuration with minimum power consumption. The first 6 columns report which use cases are enforced (1) or not (0) by the user. The time is calculated in milliseconds. When the number of configurations were very large, the enumeration was stopped at 100,000.

For the application of the heuristic optimization, the problems that require more that one second to be enumerated were selected. From the analysis of the computation times in Table 9.1 and Table 9.2, it is evident that these cases can be selected by choosing a threshold value T_c equal to 150.

Table 9.1. Enumeration approach statistics (a)

Secure Home	Bath Room Illumination	Home Illumination	Afternoon Lunch	Isolated Kitchen	Morning Wake Up	No. Of Configurations	Time (ms)
0	0	0	0	0	1	32	220
0	0	0	0	1	0	3	16
0	0	0	0	1	1	32	56
0	0	0	1	0	0	3	18
0	0	0	1	0	1	32	65
0	0	0	1	1	0	3	13
0	0	0	1	1	1	32	73
0	0	1	0	0	0	>100000	unknown
0	0	1	0	0	1	0	10
0	0	1	0	1	0	>100000	unknown
0	0	1	0	1	1	0	14
0	0	1	1	0	0	>100000	unknown
0	0	1	1	0	1	0	15
0	0	1	1	1	0	>100000	unknown
0	0	1	1	1	1	0	13
0	1	0	0	0	0	16	94
0	1	0	0	0	1	32	111
0	1	0	0	1	0	48	65
0	1	0	0	1	1	32	67
0	1	0	1	0	0	48	74
0	1	0	1	0	1	32	86
0	1	0	1	1	0	48	84
0	1	0	1	1	1	32	58
0	1	1	0	0	0	>100000	unknown
0	1	1	0	0	1	0	11
0	1	1	0	1	0	>100000	unknown
0	1	1	0	1	1	0	11
0	1	1	1	0	0	>100000	unknown
0	1	1	1	0	1	0	11
0	1	1	1	1	0	>100000	unknown
0	1	1	1	1	1	0	9

From Table 9.1 and Table 9.2, three types of cases are observed. They are:

Table 9.2. Enumeration approach statistics (b)

Secure Home	Bath Room Illumination	Home Illumination	Afternoon Lunch	Isolated Kitchen	Morning Wake Up	No. Of Configurations	Time (ms)
1	0	0	0	0	0	192	534
1	0	0	0	0	1	0	12
1	0	0	0	1	0	48	97
1	0	0	0	1	1	0	16
1	0	0	1	0	0	48	95
1	0	0	1	0	1	0	19
1	0	0	1	1	0	48	113
1	0	0	1	1	1	0	14
1	0	1	0	0	0	2304	5100
1	0	1	0	0	1	0	18
1	0	1	0	1	0	576	1285
1	0	1	0	1	1	0	16
1	0	1	1	0	0	576	1424
1	0	1	1	0	1	0	15
1	0	1	1	1	0	576	1392
1	0	1	1	1	1	0	12
1	1	0	0	0	0	768	1421
1	1	0	0	0	1	0	13
1	1	0	0	1	0	192	394
1	1	0	0	1	1	0	13
1	1	0	1	0	0	192	416
1	1	0	1	0	1	0	50
1	1	0	1	1	0	192	417
1	1	0	1	1	1	0	13
1	1	1	0	0	0	2304	4650
1	1	1	0	0	1	0	39
1	1	1	0	1	0	576	1215
1	1	1	0	1	1	0	30
1	1	1	1	0	0	576	1195
1	1	1	1	0	1	0	9
1	1	1	1	1	0	576	1387
1	1	1	1	1	1	0	13

1. **Zero Configurations:** It refers to the case when the Sat4j solver cannot find a configuration satisfying the user's request, which means that the current combination of use cases can not be enforced together.

2. **Below Threshold:** It refers to the cases when the total number of configurations satisfying the user's request are less than T_c. In such cases, the enumeration approach is sufficient to determine in NRT a configuration with minimum power consumption and enforce it.

3. **Above Threshold:** It refers to the case when the total number of configurations satisfying the user's request exceeds the configuration threshold T_c. For such cases, the time to determine a configuration with minimum power consumption exceeds the NRT requirements, or is marked as *unknown*. Unknown refers to cases in which the number of configurations exceed 100,000. The enumeration approach is practically infeasible in such cases and the proposed heuristic must be applied.

9.4.2 Results

To demonstrate the applicability and results of the proposed heuristic the*Above Threshold* cases were focused, only, since the exhaustive enumeration approach is sufficient for the *Zero Configuration* and *Below Threshold* cases, that have been dropped from the subsequent tables.

Table 9.3 compares the time taken by the exhaustive enumeration approach against the time taken by the heuristic described in Section 9.3.1 to determine a configuration with minimal power usage. The *Enumeration Solution Time* column represents the time (in milliseconds) taken by the enumeration approach. The *Heuristic Solution Time* column represents the time (in milliseconds) taken by the heuristic. The *Result* column reports the comparison, i.e., Solved, Good, or Responsive. The case is "Solved" when the heuristic is able to find a configuration with minimal power consumption in NRT while the enumeration approach is infeasible. The "Good" cases mean that the heuristic solution is faster than enumeration, while the "Responsive" label means that the heuristic solution is slower but still well inside NRT.

Table 9.4 shows the comparison of the computed power consumption values between the enumeration and the heuristic approaches. The *Enumeration Power Value* column shows the minimum electrical power (Watt), when it can be exhaustively computed. The *Enumeration Est. Power Value* column shows the estimated minimum electrical power (Watt) found after 100,000 iterations; this value is useful only as a comparison, since the involved CPU time is unrealistic. The column *Heuristic Power Value* shows the power value (Watt) of the configuration found by the heuristic. The *Result* column shows our observations, i.e., Better, Poor, or Equal. In the "Better" cases the heuristic was able to find a configuration that consumes less than the configuration found by the enumeration approach. The "Equal" label shows cases in which the heuristic was able to find the configuration that consumes

Table 9.3. Time comparison between enumeration & heuristic approaches

Secure Home	Bath Room Illumination	Home Illumination	Afternoon Lunch	Isolated Kitchen	Morning Wake Up	No. Of Solution	Enumeration Solution Time (ms)	Heuristic Solution Time	Result
0	0	1	0	0	0	>100000	unknown	556	Solved
0	0	1	0	1	0	>100000	unknown	743	Solved
0	0	1	1	0	0	>100000	unknown	689	Solved
0	0	1	1	1	0	>100000	unknown	1123	Solved
0	1	1	0	0	0	>100000	unknown	829	Solved
0	1	1	0	1	0	>100000	unknown	463	Solved
0	1	1	1	0	0	>100000	unknown	760	Solved
0	1	1	1	1	0	>100000	unknown	1172	Solved
1	0	0	0	0	0	192	534	506	Good
1	0	1	0	0	0	2304	5100	708	Good
1	0	1	0	1	0	576	1285	662	Good
1	0	1	1	0	0	576	1424	1299	Good
1	0	1	1	1	0	576	1392	907	Good
1	1	0	0	0	0	768	1421	443	Good
1	1	0	0	1	0	192	394	972	Responsive
1	1	0	1	0	0	192	416	578	Responsive
1	1	0	1	1	0	192	417	1252	Responsive
1	1	1	0	0	0	2304	4650	2358	Good
1	1	1	0	1	0	576	1215	1296	Responsive
1	1	1	1	0	0	576	1195	1018	Good
1	1	1	1	1	0	576	1387	1371	Good

minimum electrical power in a shorter time than the enumeration approach. Only two cases are marked with “Poor”, where the heuristic was not able to find the minimum power, but this happened for infeasible cases, only, where no practical alternative approach is available.

The size of the search space seems also to influence the effectiveness of the

heuristic procedure: for example, the first row in Table 9.4 puts very few constraints over device states, and the heuristic is usable to find a good solution, while the second row adds some constraints (i.e., Isolated Kitchen), and the narrower search space allows to find a better solution. The same applies to rows 5 and 6.

Table 9.4. Power value comparison between enumeration & heuristic approaches

Secure Home	Bath Room Illumination	Home Illumination	Afternoon Lunch	Isolated Kitchen	Morning Wake Up	No. Of Solution	Enumeration Power Value	Enumeration Est. Power Value	Heuristic Power Value	Result
0	0	1	0	0	0	>100000	N/A	4047.02	5411.29	Poor
0	0	1	0	1	0	>100000	N/A	3355.93	2763.87	Better
0	0	1	1	0	0	>100000	N/A	4728.43	4136.37	Better
0	0	1	1	1	0	>100000	N/A	4728.43	4136.37	Better
0	1	1	0	0	0	>100000	N/A	3408.39	5411.29	Poor
0	1	1	0	1	0	>100000	N/A	2961.98	2763.87	Better
0	1	1	1	0	0	>100000	N/A	4334.48	4136.37	Better
0	1	1	1	1	0	>100000	N/A	4334.48	4136.37	Better
1	0	0	0	0	0	192	0	N/A	0	Equal
1	0	1	0	0	0	2304	2583.16	N/A	2583.16	Equal
1	0	1	0	1	0	576	2763.87	N/A	2763.87	Equal
1	0	1	1	0	0	576	4136.37	N/A	4136.37	Equal
1	0	1	1	1	0	576	4136.37	N/A	4136.37	Equal
1	1	0	0	0	0	768	175.88	N/A	175.88	Equal
1	1	0	0	1	0	192	1146.36	N/A	1146.36	Equal
1	1	0	1	0	0	192	2518.86	N/A	2518.86	Equal
1	1	0	1	1	0	192	2518.86	N/A	2518.86	Equal
1	1	1	0	0	0	2304	2583.16	N/A	2583.16	Equal
1	1	1	0	1	0	576	2763.87	N/A	2763.87	Equal
1	1	1	1	0	0	576	4136.37	N/A	4136.37	Equal
1	1	1	1	1	0	576	4136.37	N/A	4136.37	Equal

9.4.3 Discussion

In our experiments, a total of 63 iterations were performed, corresponding to each possible $\mathcal{R}$ defined over an environment with over 1500 declared DEs. Two performance comparisons were measured between the proposed heuristic and the enumeration approach.

- the comparison of time taken by the approaches to compute the best solution to the user's request (Table 9.3).
- the consumed electrical power by the enforced settings of domotic effects (Table 9.4).

From Table 9.3, it can be seen that our proposed heuristic was able to solve all cases in NRT, even where the total number of configurations made the enumeration approach infeasible. Most cases took around 1 second to be solved by the heuristic. By observing the results, it can be stated that the proposed approach is feasible for integration with intelligent building systems.

Table 9.4 compares the power values of the configuration obtained using the enumeration and the heuristic approach. In cases where the total number of configurations were less than 100,000 it can be seen that the proposed heuristic always finds the configuration with the absolute minimum electrical power value. On the other hand, the cases in which the number of configurations exceeds 100,000, the heuristic was able to quickly solve all of them, and in most of the cases it was able to find a configuration that consumed less electrical power, compared to an (inapplicable) enumeration approach. Hence the experiments prove the feasibility of the complete approach as well as highlighting the robustness of the proposed optimization heuristic.

9.5 Related Works

Hubert et al. [79] outlined that in order to realize the potential of the smart grid, optimization of energy usage is required at different consumer levels, i.e., residential, commercial, industrial. In the domain of EMS, the literature on optimizing the electrical power usage while achieving user intelligible goals (in real time) is scarce but several researchers have addressed the energy optimization issue at different consumer levels.

Reference [80] advocates the need to build an intelligent decision support system which takes into account user preferences and behavior, and then tries to assist the user in reducing the energy consumption according to a dynamic notion of price. A model is proposed that learns user preferences and characteristics over time, and provides different alternatives for efficient energy usage. However, the practical

implementation of such model, i.e., how to integrate it with a home automation system and its feasibility was not discussed. Moreover, the model focuses on user's preferences over devices rather than on higher level intelligible goal. A similar approach is proposed in [81]. Dynamic pricing and incentive pricing policies are adopted and advocated by many in the smart grid community to optimize the energy usage [18, 82] but the user perspective is often missing.

Amir-Hamed et al. [83] propose optimization of residential load control with price prediction in a Real-Time electricity pricing environment. It minimizes the householder's electricity costs by scheduling the operations of each appliance, subject to special needs of the user. The user perspective is modeled as a waiting parameter in the scheduling problem, whose cost increases with time. Therefore, each appliance operation is scheduled based on price of electricity and the value of the waiting parameter.

Reference [84] proposed a system model that uses game theory to design a energy consumption scheduling game among consumers to address demand side management. It considers a single energy source and multiple consumers. The consumers automatically coordinate among each other to find optimal energy consumption on an hourly basis. The scheduling problem is modeled over a set of consumers and could face scalability issues when the number of consumers increases. This technique also lacks the description of modeling consumer requirements.

One potential weakness of all above proposals is that they focus entirely on minimizing energy consumption and ignore other environment aspects, especially the user's perspective. The Ambient Intelligence (AmI) community has addressed such aspects in the domain of smart environments. Often missed is the point that the EMS inside a building will be part of a larger smart environment system, providing sensing, actuation and user interaction.

9.6 Synopsis

This chapter tackles the minimization of power consumption from the point of view of individual buildings or homes. Smart environments may be equipped with an energy management system that is able to intelligently control the activation of devices and to minimize power accordingly, taking into account the varying requirements of the users. The approach exploits the degrees of freedom that are available when the users express their requirements at a higher level, in a user-intelligible way, rather than directly controlling the state of each device.

The Domotic Effects modeling framework that has been presented effectively enables users to easily express their needs at a higher level, by means of a Boolean formalization of the Domotic Effects enforcement and a SAT problem. The Boolean problem has been extended to minimize power consumption, in near real-time, while

satisfying user requirements, and a heuristic algorithm has been proposed to find satisfactory power results while respecting timing constraints.

The extensive results reported on a case study show the feasibility and the robustness of the approach, making it suitable for adoption in smart environments. The proposed approach can be extended to include further constraints like reducing the number of state changes to conserve the life-time of the appliances, or taking into account the energy needed to switch between states (e.g., for mechanically actuated devices).

Currently the work is being done towards a better integration of the approach in the Dog gateway open source distribution, and on devising intuitive user interfaces to monitor and control the environments through the Domotic Effects paradigm.

Part III

Semantic Data Exchange

Chapter 10

Motivation and Scenarios

Energy Conservation is a rising concern for many countries around the world. The resources used to generate energy, their scarcity and the rising impact of those resources on the global environment have made energy conservation a top agenda on the tables of high government officials around the world. In USA, the Department of Energy (DOE) launched a Weatherization Assistance Program that enables low-income families to permanently reduce their energy bills by making their homes more energy efficient[1]. The US Environmental Protection Agency defined an Energy Conservation Action Plan which addresses opportunities for energy conservation in homes, schools, offices and industrial environments through the use of energy-saving innovation[2]. China introduced a medium and long term energy conservation plan to push the whole society towards energy conservation and energy intensity reduction, to remove energy bottlenecks, to build an energy saving society, and to promote sustainable social and economic development[3]. The International Energy Association published statistics of energy consumption by sector [85], according to which China uses 38.2% and 40.0% of its total energy on the residential and industrial sectors, respectively. Europe uses 26.6% and 32.2% of its total energy on the residential and industrial sectors. Currently, a trend can be seen that developing countries with growing population use a major portion of their energy in the residential and industrial sectors.

Ambient, Ubiquitous and Intelligent computing have provided stimulus to the research of a number of residential gateways [86–89] which provide control of appliances in a house and access to the general appliance properties. Access to this

[1] http://www1.eere.energy.gov/wip/wap.html

[2] http://www.epa.gov

[3] http://www.chinaenvironmentallaw.com/wp-content/uploads/2008/04/china-medium-and-long-term-energy-conservation-plan.doc

information can be provided locally through a software application or remotely over the web. Due to the variety of approaches proposed or adopted in Smart Home research, we rely on a somewhat restricted definition, that focuses on the current applicability of Smart Home technologies. This chapter targets Intelligent Domotic Environments (IDE), defined as "environments where commercial domotic systems[4] are extended with a low cost device (embedded PC) allowing integration and interoperation with other appliances, and supporting more sophisticated automation scenarios" [35,36], as they currently achieve advanced intelligence at a relatively low cost, enabling the creation of new building automation scenarios, with much more complex behavior and functionality.

If residential gateways provide energy consumption information then energy providers or 3rd party players could provide applications to increase energy awareness among consumers. Most of these residential gateways [71,86–89] do not provide the energy consumption information about different appliances in the house. To support different types of applications and services the energy consumption information should be exposed in an open and machine understandable format, so that different applications can use the data according to their own diverse goals. Chapter 11 and Chapter 12 discusses two techniques that enable residential gateways to expose power consumed by different appliances installed in a house, in a machine understandable format, to support the development of external applications. In the home environment we are interested in active power only, since reactive power is much smaller and is not billed by most energy providers to residential users; therefore throughout all of this chapter, we always refer to active power, only[5]. Such external applications, starting from data published by the gateway, can provide visualization of energy information (either locally or on the web), can provide statistics and analysis of the energy data, and in the near future may aim at achieving intelligent negotiation and consumption coordination. Exposing energy consumption in a neutral and machine understandable format will allow multiple services to use the energy and power consumption information according to their own application goals.

This chapter discusses two common scenarios where publishing energy consumption information can be helpful.

[4]the word "domotics" is a contraction of the latin word *domus*, for house, with informatics, and represents the residential extension of "building automation"

[5]Incidentally, low-cost power meters compatible with affordable domotic systems are usually not capable of measuring reactive power

10.1 Scenario 1: Home Energy Management System (HEMS)

Consider an energy provider, which provides its residential consumers a home energy management system. The system is connected through a home network to a smart utility meter and electrical appliances in the home. To reach goals of energy awareness and efficiency, the system provides different applications to track the power consumption of different appliances inside the house. It provides tools to monitor current energy needs, delivering an analysis on the power consumed over time and suggestions on better energy management plans. The system is a plug and play management system in which third-party applications can be installed to provide consulting services. These services can give suggestions, such as the vendors that provide more energy efficient devices, or plans to save money by saving energy. Shifting the use of major appliances such as dishwashers and clothes dryers to hours with a lower overall electricity usage can help utilities meet the energy demands and help consumers save the energy under demand-based pricing plans. Providing real-time information linked to such dynamic pricing may be a winning combination for consumers who want to cut energy costs.

In order to build HEMS following issues should be addressed.

1. The gathering and representation of information related to the power consumed by appliances in the environment.

2. Publishing the power consumption information in an open and standard format.

10.2 Scenario 2: 2020 Intelligent Energy Grids

Consider the energy delivery and consumption landscape in 2020 (or even before). In 2020 the world energy demand has grown by 76% with respect to 2007, requiring 4 800 GW of capacity additions, almost five times the 2009 capacity of the US [90]. In this scenario, energy production and delivery dramaticaly relies on smart grid solutions to effectively distribute the available energy (mostly electrical) and to coordinate with consumption demands to avoid peaks and abnormalities that today require oversizing of distribution and production systems.

One of the main contributions to the future ability to cope with such a high energy demand is the improved and automated cooperation between consumption centers, be they residential houses (around 30% of the current consumption) or industrial and commercial facilities. The Internet growth and the take off of Information and Communication technology and Artificial Intelligence based techniques

has driven the energy distribution scenario to the current state where consumers and producers continuously and autonomously negotiate the best trade-off between energy needs and availability. Take 2020 homes as an example, they are automatically communicating and coordinating their energy needs. In every city district, single homes interact and communicate with neighbors to shape the global district consumption, to activate home level energy transformation and to coordinate local energy production, thus reducing the cumulative amount of energy required from the main electric distribution network and almost eliminating consumption peaks. This amazing capability of coordinating different homes together can be observed every day: at each day hour some houses are producing energy thanks to solar cells installed on their roofs or to thermal co-generation of their heating systems. The homes that are not generating, or that require additional energy, negotiate with neighbors energy transfers, minimizing the need of 'external supplies' through the main power delivery line.

Even countries are coordinating and collaborating in the same way, while one hemisphere of the world is sleeping, energy production is mainly routed on the illuminated side of the earth, supporting the higher day-light consumption request. Everything happens seamlessly, and if observed from a distant energy point of view the whole globe is traversed by a steady wave of energy, that regularly feeds human activities, 24/7.

This futuristic, but still realizable, scenario involves many subtle issues, that need to be unveiled in order to guide research on the technology infrastructure needed to support it. At the basis of the depicted scenario, the Internet acts as a connective tissue, flowing energy related information between different involved entities such as homes, industries, offices, power delivery and power production plants. On top of this connection network, data exchange needs common, machine understandable formats to enable all the above intelligent negotiations and consumption coordinations. We do not go further in the analysis of the issues raised by the 2020 scenario, instead we focus on this information exchange infrastructure on which every advanced consumption policy is rooted.

In particular, we must acknowledge that large-scale coordination may not rely solely on the efforts of the utility providers, that often lack the details about how energy is consumed by their customers, and that can't take into account increasing self-production of power by end users. In this context, users must be willing to share part of their consumption (and production) information, in real time, for the benefit of advanced monitoring and forecasting applications: this implies that end-users should directly publish their information and rust the utility providers and other service providers to use it an provide added-value services. However, the intelligence of coordination application is expected to expand over the next years, and to be able to encompass more and more sources of information. For this reason the published data should be application-agnostic, and available in an open and

interoperable way. As Tim Berners-Lee literally shouted at the audience [9], we need “Raw Data Now” to enable future intelligent applications.

Chapter 11

Linked Open (Dynamic) Data

The last decade saw the emergence of economically viable and efficient sensor technology which can be integrated with appliances across environments, enabling them to sense and measure features around them, i.e., proximity, temperature, luminosity, pressure, electricity, gas, water, etc. This enabled system designers to construct smart sensing and measuring environments [12] and gave rise to computing models in which networks of devices or sensors may interact with each other and with their environments on regular or sporadic moments to reach some predefined goals. These goals may be managing the comfort of residents, providing feedback to the residents over their daily routines, suggesting possible alternatives to the resident's routine, managing efficiently different operations across the environment, etc.

The potential of building diverse applications over the real-time data generated by devices or sensors inside smart environments is huge. For example, in smart homes, device activation can provide the ability to monitor the current state of the system in real time and at the same time allow estimating the energy usage of the environment. In smart metering systems, the real time measurement of electricity in an environment can provide the consumers with a graphical feedback enabling them to follow better and efficient consumption patterns or with computing models that provide suggestions for efficient consumption. Alternatively, it may provide the energy utility the information to make better energy consumption forecasts by taking into account the consumers' needs. The fundamental property of such systems is the potential of supporting various applications over the same set or subset of data generated by the environment.

This potential can be achieved, if the data gathered and generated by networks of devices or sensors follows an open and standard encoding structure for representation, i.e., the data should be machine understandable and processable. Usually the structure of the environment which models the controllable and uncontrollable elements of the environment (e.g., house plants, walls, floors, rooms etc) does not evolve over time. In contrast, the data gathered and generated by the installed devices or

sensors does evolve over time depicting a new overall picture of the environment at each update or change.

Many researchers have used semantic web tools (ontologies) to describe smart environments [35,91]. An ontology acts as a knowledge base which models the organization and semantics of elements in an environment. While existing semantic tools and reasoning engines deal with time invariant ontological knowledge, supporting rapidly changing information become critical in last couple of years.

This chapter proposes a framework providing smart sensing and measuring environments the ability to expose semantically annotated sensor's or device's data being continuously updated; such updates might be issued at specific time intervals or be bound to some environment-specific event. The framework is based upon the publisher-subscriber pattern [24] and is designed to be integrated with an environment, wishing to expose machine understandable data over a unique interface for supporting the development of applications with diverse goals.

The chapter is divided into six sections. Section 11.1 defines the addressed problem and outlines general characteristics of the environment where the framework can be employed. Section 11.2 explains the proposed framework and outlines an architecture for its implementation. To demonstrate the ability of the proposed framework, Section 11.3 defines a realistic example, using real-world data. Section 11.4 compares our technique with some existing techniques proposed in the literature and section 11.5 concludes the chapter and provides future directions.

11.1 Problem Definition

To better understand the problem and its applicability to smart environments, let us consider a future industrial plant, equipped with a state-of-the-art Energy Management System (EMS). The plant has several energy meters measuring the consumption of electricity, gas and water. The plant has its own electricity generation facility fulfilling its own electricity requirements. The plant consumes the needed electricity and may sell the remaining electricity to the National Energy Grid. The scenario contains some unique characteristics; there are many variables changing over time (representing dynamic data), i.e., electrical generation capacity, local electrical usage, current purchasing price for electrical power, gas consumption, water consumption, etc. The structure of the plant remains constant for a long period (representing static data), i.e., the power generation facility, devices, sensors, location, building structure, etc. If the plant is able to expose its requirements and consumptions in a systematic way, a huge number of potential applications can be build on top. For example, feedback applications can be provided with real time

data, which in turn could provide real time analysis for efficient energy usage and future pricing predictions to the user, energy brokers could collect data about the extra electricity available at the plant and the current purchasing price and in turn provide recommendations in real time.

In the above described scenario, some systems (industrial plant, energy broker) are publishing pieces of information being continuously updated and the information is subscribed by some other systems (energy broker, energy provider) to support different applications performing variety of objectives. To differentiate between systems, we describe the former systems as "Publishers" and the latter as "Subscribers". Formally, a "Publisher" is a software application that wishes to expose data (being continuously updated) in an open and standard format having formal semantics. A "Subscriber" represents an application that consumes the data exposed by publishers to achieve some objectives or provide third-party services to users. It can re-publish the data after processing, acting as a publisher, too.

The proposed framework targets scenarios like the one described above, that will enable publishers to publish both the static data and dynamic data of a smart environment to interested subscribers. The framework should be generic in nature and should provide following features:

1. The framework should provide publishers the ability to expose the data being continuously updated; such updates might be issued at specific time intervals (typical update cycles range from a few seconds for device states, to few minutes for energy related information) or might be bound to some environment-specific event.

2. The framework should provide publishers the ability to expose the dynamic aspects of the environment (sensor or device data) separately from the static aspect (the structure of the environment).

3. In order to allow other systems, i.e., automated agents or machines, to understand and process the system on the fly, data should be encoded in an open and standard format with attached semantics.

4. The framework should provide publishers the ability to expose several different streams of data (channels).

5. The framework should provide subscribers the ability to discover different channels exposed by publishers and the structure of the data carried by them.

6. The framework provides subscribers the ability to consume data originating from different publishers.

7. The increase in the number of subscribers should not affect the performance of the publisher.

8. Besides providing the publisher the ability to expose data (corresponding to updates) in an open and standard format, the framework should allow the publisher to define structure of the data carried by channels. For example, the data could be sensor measurement, time of measurement etc.

11.2 Proposed Framework

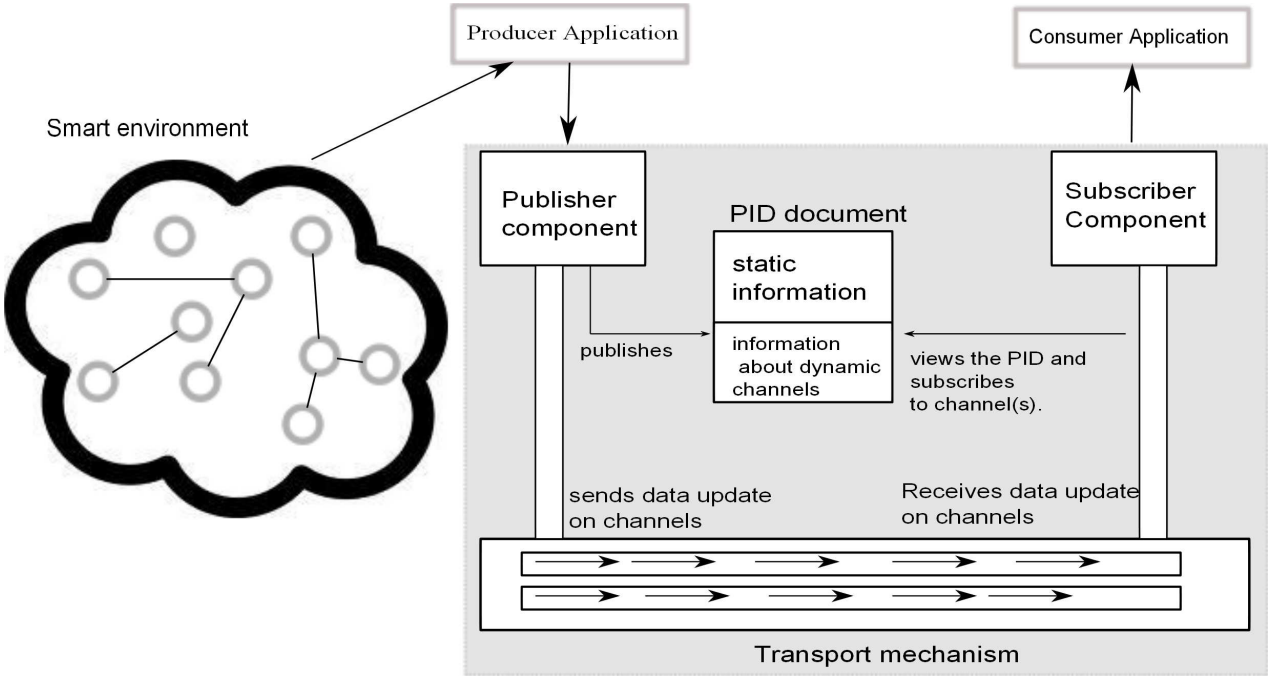


Figure 11.1. Architecture of the Framework

Figure 11.1 depicts the general architecture of the proposed framework. It consists of a publisher, a subscriber components and a transport mechanism. The framework is used by a smart environment willing to publish some data and a consumer application interested in the data of the environment.

The publisher component allows a smart environment (having network of devices or sensors) to publish the data. The environment can use the publisher component to expose the description of the environment which contains the structure of the environment, its different elements (sensor or devices) and their properties, and a description of channels created to carry updated sensor values or device states,

in form of a "Publisher Information Document (PID)". The publisher component allows to create channels, which stream updates to interested subscribers in near real time. A channel carries an update in form of a RDF [92] fragment with a set of properties. At each update, the values of the properties in the RDF fragment are updated and the RDF fragment is published on the channel. The set of properties, i.e., measurement value, measurement time, of the RDF fragment and the method to subscribe to the channel is also included in the *PID*. The *PID* is described using a "Publisher" ontology defined latter in Section 11.2.1.

The subscriber component allows a consumer application to consume published data originating from the publisher. It accesses the PID document to know the structure of the environment and the list of exposed channels. It then allows consumer applications to subscribe to different channels and when updated data arrives at the channel, the subscriber component provides it to the consumer application.

In order to avoid additional resource consumption on the publisher end, as the number of subscribers increases the proposed architecture favors the use of a third-party transport mechanism that keeps the list of subscribers and the logic of streaming data outside the publisher component. In this regard, we propose the use of cloud based publisher-subscriber services like PubNub[1] and/or Pusher[2].

The underlying theme of the proposed framework is that the producer applications use a publisher component to publish the data using PID and Channels. The consumer application subscribes to the channel(s) and processes the published data, and it could range from being a simple feedback application to an advanced complex event processing system. The tasks of keeping track of subscribers and streaming data to them is handled by the transport mechanism, setting the publisher free of such tasks. Besides being easy to integrate, the publisher is light-weight as the logic of keeping track of subscribers and streaming updates is outside the publisher component.

The details of each component and their working are given below.

11.2.1 Publisher Component

This component allows a *producer application* to publish structured data. The basic tool to publish data is by defining a PID document, which describes the structure of the environment and the created channels. The structure of the environment can be defined by means of a domain-dependent ontology description, that uses one of the many available ontologies (e.g., for smart homes, it will be structure of the house; for energy management system, it will be installed metering system). The

[1] http://www.pubnub.com

[2] http://pusher.com

created channels will carry updates. Each update is a RDF fragment having a set of properties, which are dependent on the update being carried by the channel and represent any data which changes over time.

Modeling: Publisher Ontology

LO(D)D is an ontology driven framework and it is driven by a PID document. In order to provide the formal structure of the PID document, a 3-tiered ontology has been developed. The ontology is called *Publisher* ontology and it is formalized by using the OWL Web Ontology Language [29]. The encoded ontology will provide both publisher and subscriber components a common structure and agreed upon formal semantics. Three modeling layers are defined, with decreasing usage complexity: a core layer, designed to outline the basic elements of the LO(D)D architecture, a semantics layer allowing a producer application to specifically define artifacts regarding the environment. For instance, description of the environment, type of dynamic channels, type of data events carried and the type of transport mechanism. Inheriting from the previous two layers, the operation layer allows the actual description of the environment, the number of channels, the data events being carried by the channel etc.

[Core layer:] contains the basic class definitions for expressing a *producer application* (Figure 11.2); this layer is concrete and not meant to be modified by the designers of producer applications. It provides a common structural foundation for diverse producer applications that can be utilized by a consumer application to retrieve generic information regarding the producer applications.

Every *producer application* is formally organized into a concept hierarchy inheriting from the `Publisher` class. As mentioned before, each publisher has a structural aspect (which remains static over a long period of time) and a dynamic aspect. The former static aspect is represented by the *StaticContent* class (using *hasContent* property of *Publisher* class). The *StaticContent* class can be used to define/attach a domain-dependent ontology (using *pointsToResource* property). For instance, in the smart home environment the domain-dependent ontology may be DogOnt [35], whereas, in a personal environment the domain-dependent ontology may be FOAF[3]. In order to handle the dynamic aspect of the *producer application*, LO(D)D uses the concept of channel and therefore, each *Publisher* class can have a number of dynamic channels, each represented as a *Channel* class (using *hasChannel* property). A channel streams data events of specific type, which are represented as *DataEvent* class (pointed as *streams* property. The transport mechanism used by a channel to carry data events in real time is inherited from the *TransportProtocol* class.

[3]http://www.foaf-project.org

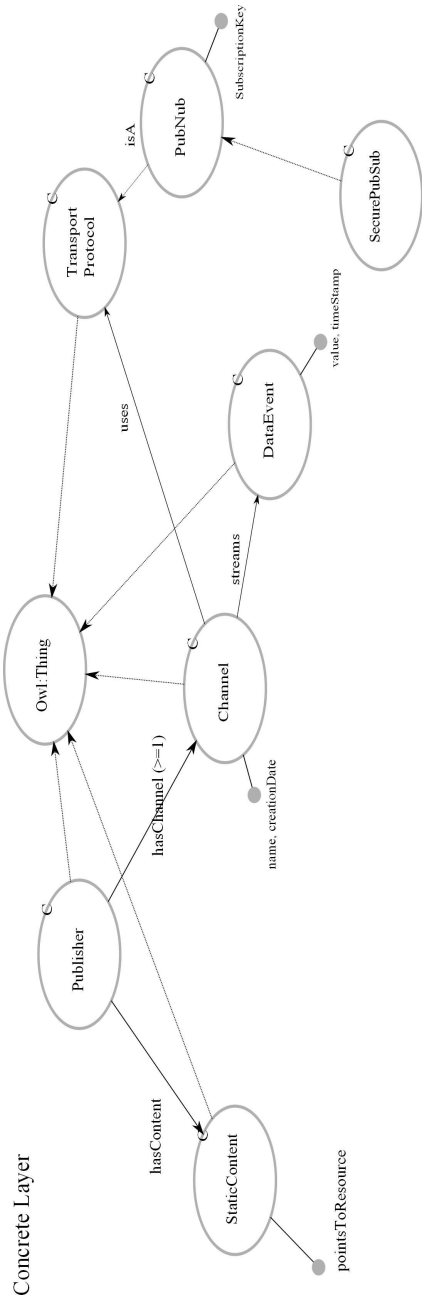


Figure 11.2. Publisher Ontology: Core layer

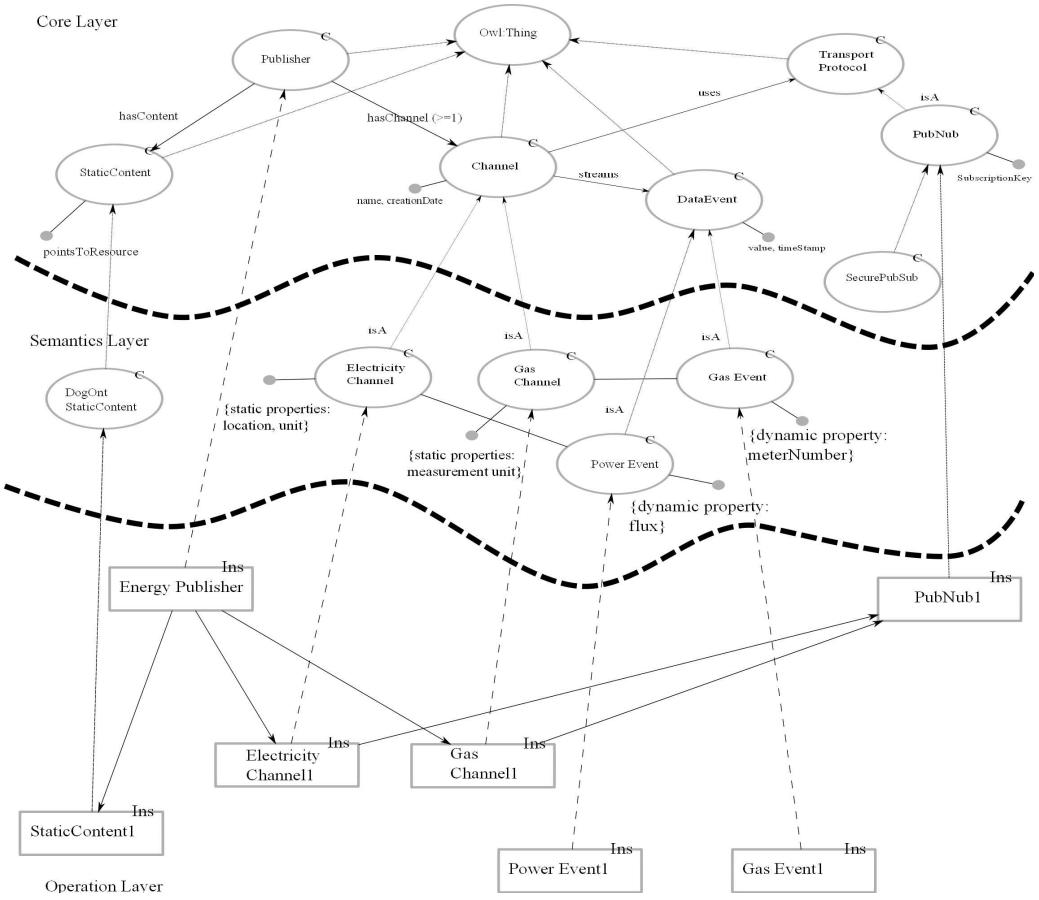


Figure 11.3. Publisher Ontology: Energy Management domain

[**Semantics layer:**] Every application domain will define different classes inheriting from the core layer by sub-classing the general classes defined in the core layer. A sample semantics layer for an energy management domain is shown in Figure 11.3. In the energy management domain typical elaborations involve continuous data such as power or gas measures coming from meters located in the smart environment. *Power* and *Gas* events are defined that will be streamed over two channels, i.e., *Electricity* and *Gas* channels. One can define additional properties dependent upon the domain. For example, the *Power* event contains the meter number from where the power event is being generated. The *Gas* channel contains the unit of measurement for the data events being transported.

The LO(D)D architecture is generic in nature, although Figure 11.3 models a simple environment for a energy aware producer application, it's important to notice how the framework easily supports domain-dependent definition of semantics layer. Moreover, modeling accuracy and granularity can easily be adapted to the problem under examination, thus providing designers a powerful tool for defining abstract effects on which more advanced policies can be built.

[**Operation layer:**] The operation layer of the Publisher ontology represents a specific *Publisher* defined in a given smart environment. They are modeled as instances of the classes defined in the core or semantics layer. Figure 11.3 shows a publisher named "Energy Publisher" that has two channels. The *ElectricityChannel1* and *GasChannel1* uses PubNub transport mechanism and its specificities (secret, public and private keys) are defined using the instance *PubNub1*. Now whenever, an update occurs it is passed on a specific channel as an instance of the *Power* or the *Gas* event.

Modeling is clearly not restricted to a single knowledge domain. If, for example, we model try to model domotic effects, then the semantics layer can be created to send updates regarding the activation or deactivation of domotic effects.

11.2.2 Subscriber Component

The subscriber component allows a consumer application to access and consume the data published by the publisher. It has the ability to query the *PID document*, extract the defined structure of the environment along with the number of exposed channels. Afterwards, it can subscribe to relevant channels using the subscription key provided for the relevant channel in *PID*. Whenever the publisher component publishes the data over the channel, the subscriber receives it.

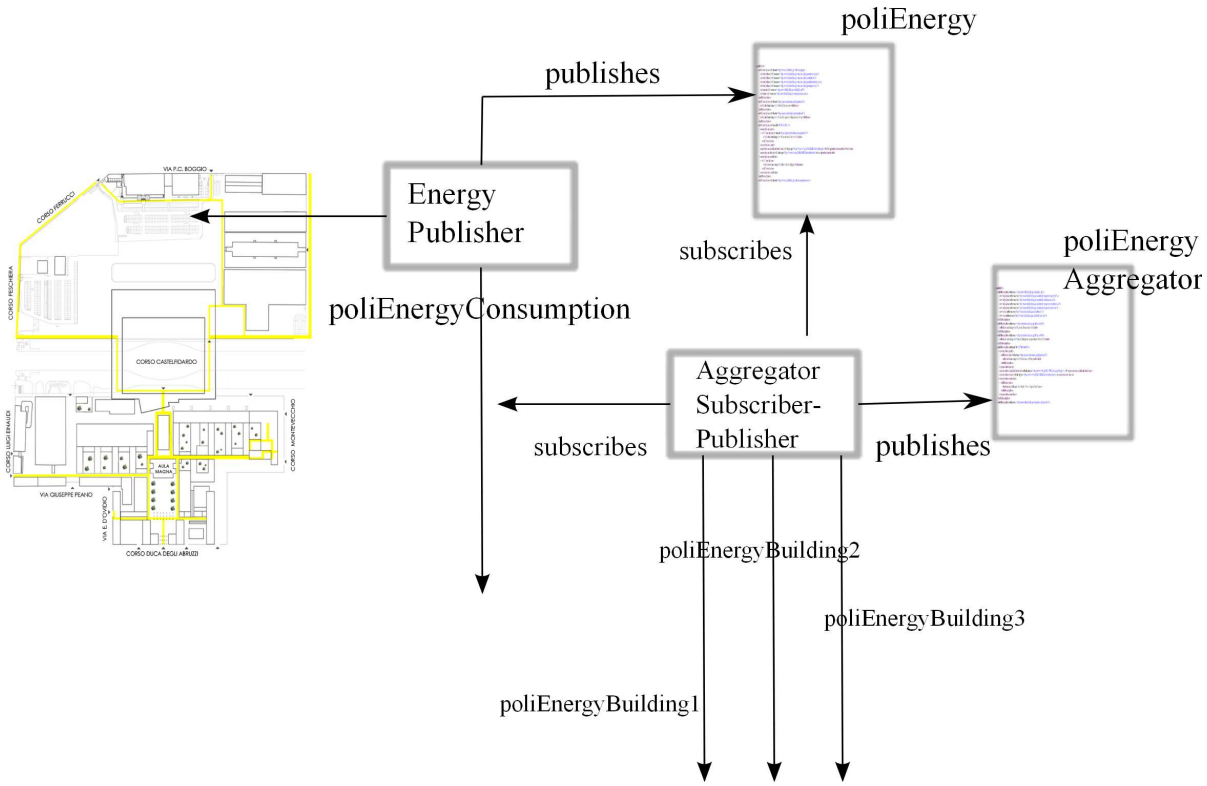


Figure 11.4. University Metering System Use Case software infrastructure

11.3 Use Case: Energy Management Domain

To evaluate the feasibility of the proposed framework, the framework has been implemented as a Java library that includes a publisher and a subscriber component. The publisher component allows to publish the PID document and to create one or more channels. The PID is structured corresponding to the *publisher ontology* and it is published as a web page using the embedded Jetty web server[4]. Channels are created by defining the elements of data (as Channel properties). The elements may be measuring unit, measuring value, time stamp etc, depending on the environment publishing the data. For our current implementation, the publisher component uses PubNub as the transport mechanism, therefore, a *subscription key* is defined for each channel, which will allow the subscriber component to subscribe to a particular channel. All RDF and ontology related operations are handled using the Jena library[5].

[4]http://jetty.codehaus.org/jetty

[5]http://incubator.apache.org/jena

To evaluate the feasibility of the proposed framework and its implementation, the main metering infrastructure installed at our university is considered. It is composed of 126 electrical meters. The electrical meters measure the consumption of electrical power (active/reactive) at different locations within the university. Each meter takes the measurements every 15 minutes and stores them in a central database for further analysis. The database contains measurements for over three years and is continuously being updated every 15 minutes. In order to provide real time monitoring of all the meters installed and to analyze measurements from different buildings within the university, we developed a software infrastructure (Figure 11.4) on top of the installed university metering system, using our proposed framework. The components of the infrastructure are explained below:

PID Snippet	Channel Snippet
<rdf:RDF xmlns:RDF="http://www.w3.org/1999/02/22-RDF-syntax-ns#" xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:publisher="http://elite.polito.it/ontologies/Publisher#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:rdfs="http://www.w3.org/2000/01/RDF-schema#" > <!-- Channel properties --!> <RDF:Description RDF:about="&Publisher;Unit"> <rdfs:range RDF:resource="http://purl.oclc.org/NET/muo/ucum/unit/"/> <rdfs:domain RDF:resource="&Publisher;Channel"/> <RDF:type RDF:resource="&owl;DatatypeProperty"/> </RDF:Description> <RDF:Description RDF:about="&Publisher;MeterNumber"> <rdfs:range RDF:resource="&xsd;integer"/> <rdfs:domain RDF:resource="&Publisher;Channel"/> <RDF:type RDF:resource="&owl;DatatypeProperty"/> </RDF:Description> <RDF:Description RDF:about= "&Publisher;hasCurrentValue"> <rdfs:subPropertyOf RDF:resource="&owl;topDataProperty"/> <rdfs:range RDF:resource="&owl;real"/> <rdfs:domain RDF:resource="&Publisher;Channel"/> <RDF:type RDF:resource="&owl;DatatypeProperty"/> </RDF:Description> <!-- Channel Name and Static properties --!> <RDF:Description RDF:about= "&Publisher;poliEnergy_communication"> <publisher:Location RDF:datatype="&xsd;string"> Torino, Italia</publisher:Location> <publisher:subscribekey>sub-xxxxxx-4290-yyyy-a138-4d46dEEEEE </publisher:subscribekey> <publisher:channelName>poliEnergy_communication </publisher:channelName> <RDF:type RDF:resource="&Publisher;Channel"/> </RDF:Description>	<rdf:RDF xmlns:RDF="http://www.w3.org/1999/02/22-RDF-syntax-ns#" xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:publisher="http://elite.polito.it/ontologies/Publisher#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:rdfs="http://www.w3.org/2000/01/RDF-schema#" > <rdf:Description RDF:about=" &publisher;poliEnergy_communication"> <publisher:MeterNumber RDF:datatype="&xsd;int"> **231** </publisher:MeterNumber> <publisher:Unit RDF:datatype="&xsd;string"> **http://purl.oclc.org/NET/muo/ucum/unit/power-level/bel-kilowatt** </publisher:Unit> <publisher:hasTimeStamp RDF:datatype="&xsd;dateTime"> **2012-02-02T13:06:41.056Z** </publisher:hasTimeStamp> <publisher:hasCurrentValue RDF:datatype="&xsd;double"> **0.3** </publisher:hasCurrentValue> </RDF:Description> </rdf:RDF>

Table 11.1. Energy Publisher Information

1. **Energy Publisher:** Its objective is to provide real time measurement updates from all the electrical meters installed in the university. It initially publishes a *PID document* named *poliEnergy* accessible over HTTP. The metering structure is defined using the DogOnt ontology [35]. It exposes a single

data channel, i.e., poliEnergyConsumption, which carries the updated measurements from all the meters every 15 minutes. A message is sent on the channel for each meter measurement. The defined properties for the *poliEnergyConsumption* channel are meter number, current value, unit and time stamp and the subscription key to subscribe to the channel.

Table 11.1 in column *"PID Snippet"* shows a snippet of the *PID* describing the publishing environment, the channel and its elements (properties) describing the information carried by the channel. The column *"Channel snippet"* shows a single message being sent for one electricity meter.

2. **Aggregator Subscriber-Publisher:** Its objective is to combine the measurements of all the meters coming from the *Energy Publisher* and to provide aggregated measurements for three different buildings within the university. It subscribes to the "poliEnergyConsumption" channel. Every 15 minutes, it is informed about the updated measurements of all the electrical meters, i.e., 126 electrical meters. It aggregates the measurements from electrical meters corresponding to three main buildings and publishes the aggregated electrical measurements using three separate channels, i.e., poliEnergyBuilding1, poliEnergyBuilding2 and poliEnergyBuilding3. The information carried by these channels includes current aggregated measurement and the time stamp.

3. **Desktop Monitoring (Subscriber):** It provides the ability to monitor the aggregated measurements of three buildings coming from the *Aggregator Subscriber-Publisher* in real time. It retrieves the PID of the *Aggregator Subscriber-Publisher* and then subscribes to all three channels. A separate real time graph is shown for each building. Figure 11.5 shows a snapshot of the application.

11.4 Related Works

A. Passant et al. [93] proposed sparqlPuSH, a proactive notification system of data updates in RDF stores using PubSubHubbub. The publisher consists of a RDF triple store and subscribers can register different queries on the publisher. Whenever there is an update in the triple store, all the registered queries are evaluated and subscribers are informed about the new or updated results. The list of subscribers is maintained at the publisher end. Our technique is clearly distinct as it advocates a light-weight publisher providing the ability to expose the data being continuously updated, no specific processing is performed at publisher end and most importantly, the list of subscribers is not kept and managed by the publisher but by a third-party cloud based service. In sparqlPuSH [93], as the number of registered queries

becomes large, for each triple store update, the time to evaluate all queries and send notification using feeds will become large. In our approach the usage and processing of the data is left at the subscriber end. Although it increases network traffic.

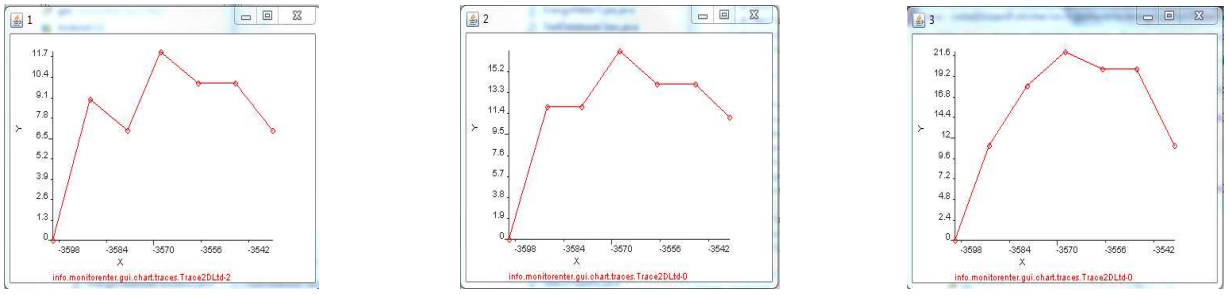


Figure 11.5. A snapshot of the Desktop Monitoring Application

A platform (Sense2Web) to publish Linked Sensor Data for the sensor network community is presented in [94]. It enables users to publish their sensor description data as RDF triples, associate any other existing RDF sensor description data, link to the existing resources on publicly available linked data repositories and make it available to consumers using SPARQL endpoint [8]. The main focus of Sense2Web is on defining an approach for enriching the sensor description data. Our technique is not an alternate to [94] but is a complement as it focuses more on publishing the updated data, i.e., sensor data. Instead of using the SPARQL endpoint to expose sensor data, which would need to be continuously queried by the consumer to get updates. Our approach will keep subscribers up-to-date.

SEIPF (Chapter 12) is a semantic energy information publishing framework, which provides the ability to query energy consumption information from residential gateways in a machine understandable format, to achieve consumption coordination and intelligent negotiation. It is based on the client-server model and if queried by the client, gives the consumption of the house based on an Energy Profile ontology. The framework proposed in this chapter incorporates the lesson learned from the development of SEIPF. It allows us to move from a client-server model to a publisher-subscriber pattern and allows separation of static information about the environment (using PID document) and dynamic updates (using channels). Moreover, the framework proposed allows easy integration with a smart environment, having characteristics defined in Section 11.1.

RDF Streaming Engines (EP-SPARQL, SPARQLStream, C-SPARQL, Unified processing of Linked Streams and Linked Data) [95–98] focus mainly on handling the RDF streaming data from a source. To handle the streaming data, they provide different querying protocols (based on SPARQL) to handle data changes over time. All these techniques mostly focus on the subscriber end. Our proposed technique focuses more on the publisher end, providing a light-weight integration with any

source to describe the source and stream its RDF data.

Pachube[6] acts as a central repository where sensors distributed across several locations can upload their data. Data can be uploaded using continuous or non-continuous feeds. A client can download the uploaded data (given access) and use it to achieve its goals. Pachube provides APIs to upload and download data. Our chapter also proposes the use of standard APIs to publish and subscribe data, however there is no central repository which stores the data. The transport layer just acts as a transport mechanism. Our model publishes the data in RDF format, so it is easier to build automated subscribers on large scale as the semantics of data are much clearer. The overhead on the client end to continuously query to check latest update and download them in Pachube is overcome by the publisher-subscriber pattern.

11.5 Synopsis

This chapter proposed a framework that provides a systematic way to publish sensor rich data from environments. The framework follows general characteristics defined in Section 11.1. It provides separation of views for the data contained in an environment using the concepts of *PID document* and *Channel*. The *PID document* allows publishers to describe their environments and also provides a systematic way to describe and subscribe to the dynamic channels exposed by the environment for the interested subscribers. The *Channel* carries the data being updated in real time to subscribers. The proposed framework is based on the concept of light-weight publishers, not concerned with keeping track of subscribers. The track of subscribers is kept by a third party cloud based system. The chapter also describes the implementation of the framework and a preliminary use case is also detailed in the chapter to prove the feasibility of the framework. In future, we plan to evaluate the proposed framework on a large network of nodes subscribed to multiple data channels, processing the data in real time and re-publishing the data.

[6] https://pachube.com/

Chapter 12

RDF Publishing

This chapter proposes a Semantic Energy Information Publishing Framework (SEIPF) which publishes, for different appliances in the house, their power consumption information and other appliance properties, in a machine understandable format. Data is published according to the Semantic Web standards and best practices, to ensure application neutrality and intelligent machine processing. While appliance properties are exposed according to the existing semantic modeling supported by home gateways, power consumption is modeled by introducing a new modular Energy Profile (E.P) ontology.

The proposed framework is consistent with publication of information at different granularity levels (e.g., by aggregating over device groups) and respecting different authorization levels. Depending on the type of application, different complexity levels require different complexity in data representation, so that the SEIPF framework is able to expose the data both as simple RDF triples (according to Linked Data requirements) and as full ontology instances, for the benefits of applications needing intelligent processing.

A different complexity dimension is time, since energy consumption is a real-time process showing variable speed, and measuring and publishing power information must take into account the time instants and the time intervals corresponding to the published figures. The SEIPF framework proposed in this chapter, being based on semantic web and linked data standards, is easily extensible by referring to standard ontologies, and can describe arbitrary complexity levels in information structure.

Today the ability to collect and share instantaneous power consumed by different devices in a house can enable the creation of many applications that can lead towards a more energy efficient society. These applications will be making consumers aware of their active power consumption through power meters, sharing their data over the web to create mash up applications, and/or consulting services to provide feedback on different appliances power requirements. In the future, intelligent negotiation and consumption coordination will allow third-party service providers

to build intelligent and automated services that use the energy consumption information to build dynamic services, such as automatic load transfer over intelligent energy grids. A basic requirement to fulfill the scope of aforementioned applications is a standard, open and semantic representation of a device's power consumption information, so that applications may use this information according to their own goals.

The remainder of this chapter is divided into eight sections. In Section 12.1, the main issues to achieve target scenarios are outlined. Section 12.2 explains the basic principle on which the envisioned solution is founded. In Section 12.3 a possible solution is presented, which is further detailed in Section 12.4. Section 12.5 provides implementation details of the proposed framework along with the experimental results by showing samples applications built to prove the feasibility of the solution. Section 12.6 presents related works and Section 12.7 concludes the chapter and discusses possible extensions.

12.1 Design Issues

The core requirement of the defined scenarios is the ability to gather the current power consumption information of an appliance or a group of appliances in the house and exposing that information over the Internet. To accomplish scenarios mentioned in Chapter 10 three main issues need to be addressed: gathering the power consumption information, publishing this information, and making the information usable by machines. Gathering power consumption information means measuring the active power consumption of home devices at a given time instant. Such a measure is greatly facilitated if a smart or home automation plant is available in the home. Publishing information means deciding, and most importantly, enabling the householders to decide which information to expose and at what granularity. Finally, distributing machine understandable information implies the adoption of an open and effective data format that enables machines to interpret them; the Semantic Web and Linked Data research communities have already paved the way in this direction. By going a little further on these three main issues, we can identify the following related needs.

12.1.1 Energy Consumption Information

The power consumption information of an appliance or group of appliances is exchanged between smart homes, energy providers, and/or any third-party applications. Homes will be equipped with appliances ranging from a lamp to a fully automated heating control system. An appliance during its operation can have different operating states like on, off, stand-by, up, down, etc. In different states the

appliance can have different power consumption levels: we need a mechanism to encode this state-dependent power consumption information. The residential gateway should provide the power consumption of appliances according to their current operating states. The availability of a house automation system, coupled with the knowledge of device characteristics, allows us to estimate and couple the power consumption information at a much finer level, with less expensive means compared to installing power meters.

12.1.2 Publishing in a Machine Understandable format

Power consumption information should be distributed in an open, machine understandable, semantic enabled and effective data format, so that the residential gateway can act as a single power consumption information point. This information point can provide inputs for multiple and diverse third-party applications, services and potentially automated agents as well.

12.1.3 Information Publishing Control

Residential gateways are designed to provide automated control to the house and thus they have information about every appliance in the house. The information may comprise the appliance properties, its power consumption and procedures to automatically control the appliance. This information, if utilized by third party applications, services or automated agents, has the potential to provide the consumer better services. However, this information sharing should be governed by a sound access control mechanism so that the basic consumer rights and privacy issues are addressed. Privacy is a subjective issue, and different consumers or even people living in a single home might perceive it differently from others. Several case studies in relation to the issue of privacy have been carried out [99, 100] and the research is still on going.

12.2 Web Of Domotics (WoD)

WoD [101] is an Internet based architecture derived by combining features of three main concepts, namely: Internet of Things, Ubiquitous Computing and Domotics. It enables mobile users to access device information and operate them in a ubiquitous manner, independent of network-specific location dependence. It addresses issues like proximity-based device identification (e.g., visual tags), network independent detection of service access points (through DNS-based device de-referencing), user identification through OpenID, open data exchange, service/device description through Linked Data formats and device operation through REST-based interaction.

WoD offers ubiquitous and mobile access to actionable tagged objects. The underlying interaction paradigms are compatible with web-based remote control of smart environments, but WoD enhances such solutions by specifically tackling co-location of users and environments and by supporting on-the-move users. The resulting architecture is shown in Figure 12.1 and includes Dog enriched with a new WoD layer, the mobility access provider application running on top of Dog, the remote WoD Dynamic DNS service and the user's mobile terminal hosting a WoD application.

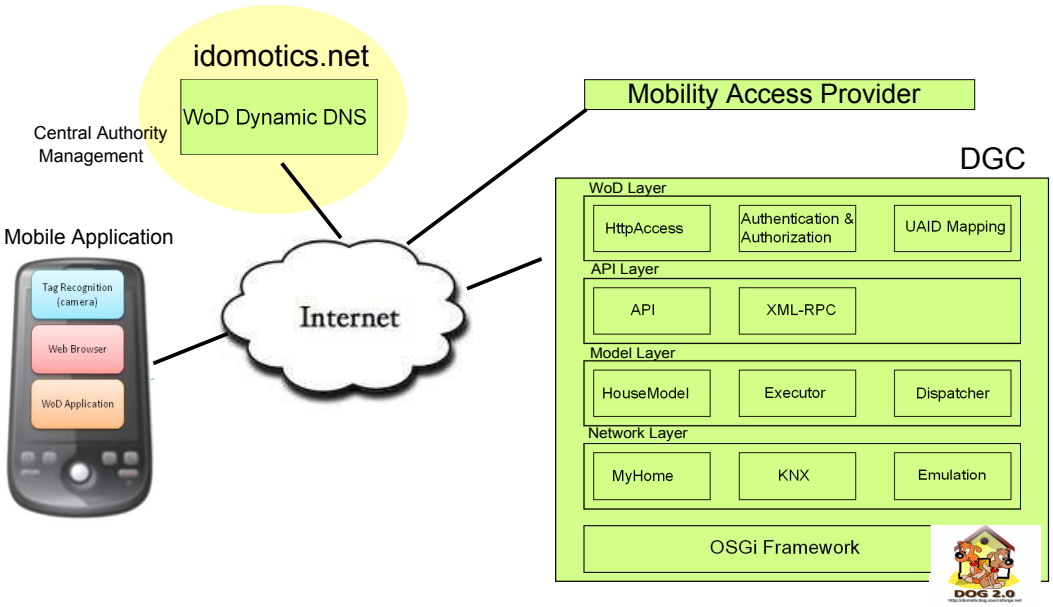


Figure 12.1. The WoD reference architecture.

12.2.1 Domotics Gateway Controller (DGC)

The DGC has been implemented by extending the architecture of the Dog gateway. A new layer called WoD layer has been added to the architecture to:

- relay user authentication to the main OpenID services (Authentication);
- implement user authorization policies (Authorization);
- provide Dog-extracted environment information through LD (HttpAccess);
- expose the Dog APIs to authorized users exploiting the HttpAccess bundle. These services are available on a particular domain address over the web.

New bundles are implemented as OSGi bundles running inside Dog, which is in turn based on the Eclipse Equinox OSGi framework, and include:

HTTP Access (HA)

The HA bundle provides the Http/Https interface to interact with different functions provided by the DGC. The request from the mobile application is received by the HA bundle and the HA bundle act as a central entry point to DGC over the web. The request contains parameters to query information about the device or to perform operations associated with the device. The communication is based on the REST over HTTP paradigm. The HA by default provides information in RDFa/XHTML format, but depending upon the needs of the application accessing the domotics environment through HA, it can also provide information in plain RDF format as well. Internally, the HA bundle communicates with the API bundle of Dog to get information about the device or to request operation on the device, in the Dog specific XML format. The HA plays a pivot role in coordinating operations once a request is received. Besides being an interface to communicate with the DGC, it forwards authentication credentials to the Authentication and Authorization (AA) bundle. Based on the response from the AA, it provides back the information about the device or directs Dog to perform the operation as requested by the user.

Authentication and Authorization (AA)

The Authentication and Authorization (AA) bundle forwards the request to the OpenIDAuthentication Server, which validates or invalidates the user identifier and communicates that information back to this bundle. Authorization is preliminarily implemented through a simple type based authorization. The implementation of AA depends upon the policies adopted by different organizations to provide access to information. It can be role based, rule based, type based or individual based. The following two operations are provided by the AA bundle to encapsulate the authentication and the authorization logic.

- **validateOpenID**. It acts as a OpenID stub. After receiving the user's OpenID credentials, it determines the correct OpenID service and sends the user credentials to the service. The OpenID service validates or invalidates the user credentials.

- **getAuthorizedOperations**. It provides the list of operations that can be performed by the user, whose credentials are already authenticated through an external OpenID service. The implementation of this function is not specified by the WoD architecture but can vary from one organization to the other, depending upon the adopted authorization policy.

UAID Mapping (UM)

The UAID helps to locate the Dog responsible for controlling the device. On the other hand, within Dog a LocalID is used to identify devices. This separation helps us to use different identification schemes for UAID and LocalID. Every DGC can identify the device using different internal identification schemes. This UM bundle implements the logic to convert the UAID identification scheme to the LocalID identification scheme. The following two functionalities are provided through the UAID Mapping Bundle:

- **getLocalID** It takes the UAID of a device and returns its LocalID.
- **getUAID** It provides the UAID of the device, given its LocalID. It is the reverse function of *getLocalID*.
- **createMapping** It is used to create a new mapping, which associates a UAID with its appropriate LocalID. The mapping is stored inside the DGC.

12.2.2 WoD Dynamic DNS

The WoD Dynamic DNS is simply a standard Dynamic DNS server which is registered as the root DNS for the well-known WoD domain, e.g., idomotics.net. The WoD architecture is ideally deployed with a single, unique, and universal WoD domain for all tags like idomotics.net exist but this is not mandatory. There can be multiple domains based on organizations for example, all the universities in a city can share a single domain, all the hotels in a locality can share a domain. But it must be noted that such an action will decrease the ubiquitous nature of WoD.

12.2.3 Mobility Access Provider

To enable the mobile application to locate the correct DGC controlling the device, during the installation of the architecture or when new tags need to be assigned with devices, the Mobile Access Provider (MAP) registers the UAID associated with the device to the domain address of the DGC that controls the device. This will finally enable the mobile application to determine the correct domain of the DGC using the extracted UAID without any a priori knowledge about the location of the DGC itself. Tags will be easily available to users: they can be bought and attached to any entity (in domotics, those would be devices) or the user can easily generate and print them by using a tag-writing application. A reference domain is pre-assigned to all tags, like *idomotics.net*. The MAP acts on behalf of the owner of the building where the WoD is being deployed to declare tag ownership. It contacts the central authority maintaining the reference domain and gets permission to add, modify or

delete the CNAME entries for its owned tags in the reference domain. It records an entry associating the device-attached UAID with the correct DGC controlling the device in the master DNS server. The MAP operations for adding a CNAME entry are shown in Figure 12.2.

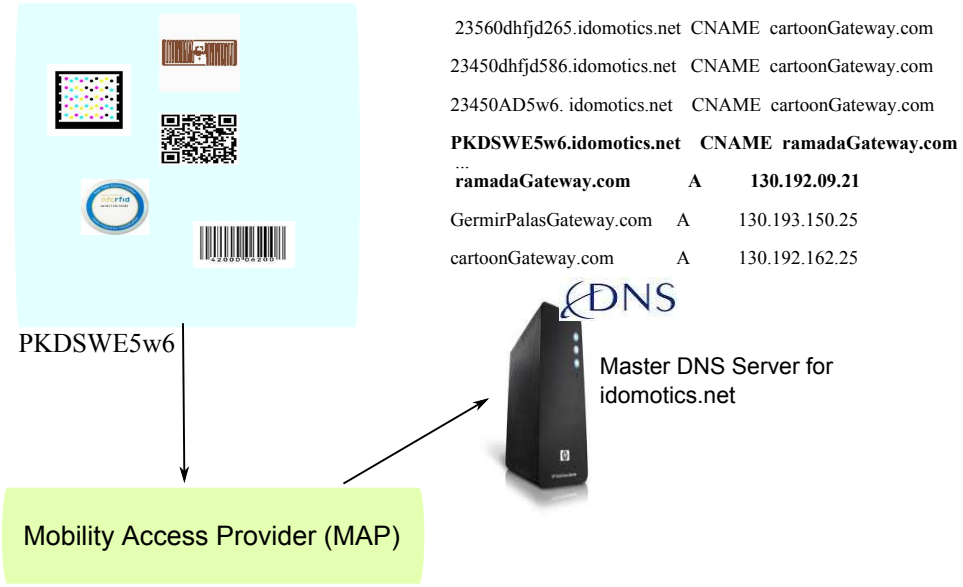


Figure 12.2. Mobile Access Provider

The MAP is a separate application that interacts with the master DNS server controlling *idomotics.net* domain.

1. The MAP decodes the (QR Code) tag to read the UAID of the device. For example, PKDSWE5w6.
2. Then it creates a CNAME entry of [UAID].idomotics.net against the domain name of the DGC, which controls the device.
3. The MAP records the entry in to the master DNS Server.
4. The mapping is stored inside the UAID Mapping bundle of Dog.

In a particular environment the device must be described in the local Dog formalism, as a DogOnt device instance, using the OWL language.

12.2.4 Mobile Application

It is assumed that users' mobiles are already equipped with tag reading software (e.g., Zebra Crossing for 2D tags on Android-based mobiles, or QuickMark (TM)

for other mobile phones) and with a web browser application, able to be run as part of the tag recognition work flow. This minimalist configuration, available on almost every modern mobile terminal, must be integrated by a WoD application for accessing and operating devices. In our experiments, we choose QR Codes to encode the UAID of all devices. QR Code or Quick Response code is a 2D matrix bar code, created by the Japanese corporation Denso-Wave in 1994[1]. We choose QR codes because they were developed by keeping in mind a quick decoding process and many modern mobile phones are by default equipped with software to decode information in QR codes[2]. WoD applications can either provide light XHTML interfaces, or can offer more advanced functionalities by interpreting the RDF device data provided by the Dog Linked Data endpoint.

12.3 Proposed Solution

To provide residential gateways with the ability to expose the power consumption information of an appliance or a set of appliances installed in the house and also do so by addressing the issues raised in Section 12.1, this chapter proposes a Semantic Energy Information Publishing Framework (SEIPF). The SEIPF exposes the power consumption information of appliances along with different appliance properties in RDF format over the web. The *Energy Consumption Information* modeling issue (Section 12.1.1) is addressed by defining a new Energy Profile (E.P) ontology (defined in Section 12.3.1). This ontology is based on the modularity pattern and models the energy consumption information about any appliance that modeled through the underlying domotic ontology, i.e., DogOnt. The modularity pattern provides separation to model different aspects of a system through separate ontologies and may be plugged on top of various ontologies. The *Machine Understandable Format* issue (Section 12.1.2) is addressed by adopting RDF as the standard format to expose information because it provides meaningful representation of information which can be semantically post processed. The complete approach is defined in Section 12.3.3. The *Information Publishing Control* issue (Section 12.1.3) is currently addressed by using the Authentication and Authorization unit available inside the WoD architecture. However, in the future we intend to incorporate an ontology based access control policy. It is further explained in Section 12.3.2.

[1]Other closely related codes are Data matrix, High Capacity Color bar code, Code 39 and Code 128

[2]http://www.mobile-barcodes.com/qr-code-software/nokia/, http://mobilecodes.nokia.com/

12.3.1 Energy Profile Ontology (E.P)

To model the energy consumption information, an Energy Profile (E.P) ontology has been developed. It models the energy consumption information about different appliances in the house. The E.P ontology is developed according to the modularity pattern, so that it can be attached to any ontology that can describe the domotic environment of a building (DogOnt in our case). The basic concepts of the E.P Ontology are DeviceProfile and Consumption (as shown in Figure 12.3).

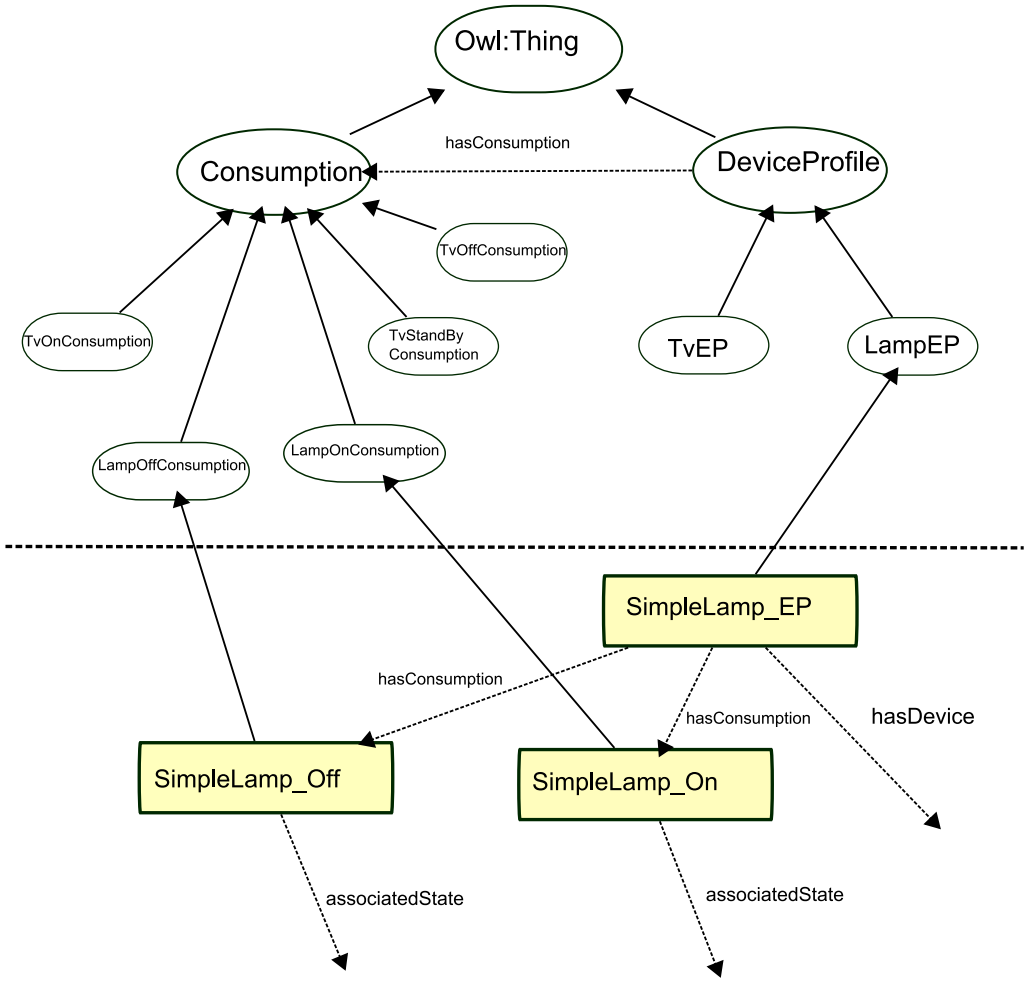


Figure 12.3. Energy Profile Ontology

1. *DeviceProfile:* This class describes energy profiles of all the major device categories in the house. The energy profile information can be related to different appliances, such as lamp, coffeemaker, dishwasher, etc. This class has two properties.

- *hasDevice:* This property specifies the instance of the device to which this *DeviceProfile* applies. The property maps onto the DogOnt device instances (i.e., instances of the Controllable class).
- *hasConsumption:* Every device may have different levels of power consumption, depending on the operating state of the appliance: each *DeviceProfile* collects various power *Consumption* object instances, one for each allowed device state.

2. *Consumption:* This class encodes the power consumed by the appliance in a given state. For each device (e.g., Lamp), different states (e.g., LampOn and LampOff) are described, each corresponding to a different power consumption level. *Consumption* instances have four properties:

   - *associatedState:* This property specifies the state of the appliance whose power consumption we are describing with this instance. The property maps onto the DogOnt ontology, where each device is described in terms of its allowed states.
   - *nominalValue:* This property shows the nominal power consumption of the appliance in the given state. It gives the estimated power consumption of a device in a state.
   - *realValue:* This property is the measured power consumption of an appliance in a given state. This property is used if the device has a power characterization available, otherwise the nominal value is used.
   - *hasUnit:* This property defines the unit of power for the power consumed by the appliance, expressed as one pre-defined instance of the *MetricUnit* class in the Measurement Units Ontology[3].

The E.P ontology defines two extension points through which the ontology defining the domotic system (in our case DogOnt) can be attached. The first is the *hasDevice* property of DeviceProfile class, which attaches a energy consumption information structure to a device or an appliance. For example, in DogOnt it relates to instances of Controllable concept. The second is the *associatedState* property of the Consumption class, which relates a given state of the appliance or device to its consumption level. For example, in DogOnt it relates to the instances of the StateValue concept.

A fragment of E.P instances is shown in Figure 12.4, where we define a single Device Profile instance named SimpleLamp_EP. SimpleLamp_EP is attached to two consumption instances SimpleLamp_On and SimpleLamp_Off, which are instances of LampOnConsumption and LampOffConsumption classes respectively.

[3]http://idi.fundacionctic.org/muo/muo-vocab.html

```
<LampOffConsumption rdf:ID="SimpleLamp_Off"/>

<LampOnConsumption rdf:ID="SimpleLamp_On">
    <hasUnit rdf:resource="http://purl.oclc.org/NET/muo/ucum/unit/power/Watt"/>
    <realValue>35</realValue>
</LampOnConsumption>

<LampEP rdf:ID="SimpleLamp_EP">
    <hasConsumption rdf:resource="#SimpleLamp_On"/>
    <hasConsumption rdf:resource="#SimpleLamp_Off"/>
</LampEP>
```

Figure 12.4. An excerpt of the power consumption information about a device

12.3.2 Information Access Control

The residential gateway houses different chunks of data about a home ranging from appliance properties and operations to sensing the presence of people and their choices. Some of this information can be utilized to provide a better standard of living to the people living inside the house, e.g., informing the people about their current energy consumption can help them to be more energy efficient. Related approaches to provide semantic access control to a system may be found in the literature. Chi-Chun et al. [102] proposed a Semantic Access Control Enabler (SACE), a middleware-based system that has been designed and implemented to enable Semantic Access Control on the Web. Toninelli *et al.* [103] proposed a semantic context aware policy model that adopts ontologies and rules to express context and context-aware access control policies and supports policy adaptation. Ionita *et al.* [104] proposed a model that regulates access control on ontologies defined in the Semantic Web.

Since the main issue of this chapter is on power consumption information sharing, a very basic solution is adopted for our SEIPF implementation: currently the SEIPF uses the Authentication and Authorization mechanism provided by the WoD architecture to control the access to information.

12.3.3 Machine Understandable format

Exposing the power consumption information in an open, neutral and semantic format will allow multiple services to use information according to their application goals and, in future, will allow intelligent negotiation between automated software agents. To achieve the aforementioned task, RDF is adopted as an open format that has embedded semantic information which allows information to be machine understandable. The SEIPF publishes all information pertaining to an appliance,

including its operating states, related current power consumption and general properties, as a pure RDF structure.

To provide reasoning support over the RDF response received from the SEIPF, a basic set of general concepts are defined through a vocabulary. The vocabulary is encoded in the SimpleDomoticData ontology, which is very similar to the E.P ontology but it has been created separately for following reasons:

1. The E.P ontology was developed to model only the power consumption of an appliance or group of appliances. The actual appliance modeling is provided through the DogOnt ontology. Whereas, in the SimpleDomoticData ontology, we model the appliance, its properties and its power consumption in current state. It can be extended to include other appliance properties.

2. Providing the response based on the DogOnt and the E.P ontologies would have required external applications to understand the more complex structure of DogOnt with additional information about the domotic system, which in this case is not always needed. Therefore, to provide easy and simple integration a simplified SimpleDomoticData vocabulary is preferred to encode the response.

The simplified ontology and its general concepts are explained below, and are basically the minimum set of classes and properties, extracted from the E.P and the DogOnt ontologies, that allow semantic publishing of power information:

1. *Device:* This class indicates the appliance for which power consumption is inquired.

    - *hasConsumption:* This property points links this instances of the Device class to the Consumption class (defined below) instances describing its current power consumption value.
    - *hasState:* This property defines the actual state value of the instance of State class, as a string.

2. *Consumption:* This class encodes the power consumption information of the appliance in the current state.

    - *hasUnit:* This property defines the unit of power for the power consumed by the appliance, according to the Measurement Units Ontology.
    - *value:* This property shows the power consumption of a device, encoded as a real number.

An example of such encoding is shown in Figure 12.5.

```
<rdf:RDF
  xmlns:sdd="http://elite.polito.it/ontologies/SimpleDomoticData.owl#"
  ... other namespaces ...  >

  <rdf:Description rdf:about="&sdd;#Computer_faisal">
    <hasConsumption rdf:resource="&sdd;#ComputerStandbyConsumption"/>
    <hasState>Stand-by</hasState>
    <rdf:type rdf:resource="&sdd;#Device"/>
  </rdf:Description>

  <rdf:Description rdf:about="#&sdd;ComputerStandbyConsumption">
    <hasUnit>http://purl.oclc.org/NET/muo/ucum/unit/power/Watt</hasUnit>
    <value>25</value>
    <rdf:type rdf:resource="&sdd;#Consumption"/>
  </rdf:Description>

</rdf:RDF>
```

Figure 12.5. SimpleDomoticData excerpt for a device's energy consumption

12.4 Semantic Energy Information Publishing Framework (SEIPF)

The SEIPF can be installed on any centralized residential gateway that uses the DogOnt ontology to model the domotic structure of an environment. It comprises a core Publishing Unit that provides the power consumption details pertaining to different appliances in the house and other appliance properties and is explained more in Section 12.4.1.

The SEIPF is integrated with the WoD architecture. The WoD provides an interface over the web based on the REST over HTTP interaction paradigm to access the domotic system of an environment. The interface is implemented through an OSGi bundle named HttpAccess. The HttpAccess bundle was extended to add new functionalities to query the SEIPF. The integration of the SEIPF and the WoD archiecture provides functions that enable a requesting entity to acquire information in Linked Data format about the general domotic structure of the environment and identification of different devices installed in the environment. The requesting entity could be a third-party application, a service or an automated agent etc.

The request can be made with certain parameters (using the HTTP GET method) to retrieve the power consumption information from the SEIPF. The parameters and their possible values are shown in Table 12.1.

Access control is provided through the Authentication and Authorization Bundle of WoD. To provide an authentication which is scalable and has a web wide scope, WoD adopts Open ID [105] as an authentication mechanism for the requesting entity. Open ID is a decentralized standard to authenticate the requesting entity. Currently,

Table 12.1. List of Parameters

Parameter	Values	Example	Detail
`command`	`info`	`info`	To request an appliance power consumption
`device`	*device id*	`lamp9`	The identifier of the device
`room`	*room id*	`livingroom`	To request information about all devices in the livingroom
`devicecategory`	*device type*	`Lamp`	To ask the power consumption of Lamp-type devices.
`query`	*SPARQL query*	-	To query the DogOnt and E.P ontologies directly.

Authorization is provided through a type-based policy but as stated earlier, we are working on a separate ontology-based publishing policy that will be independent of WoD.

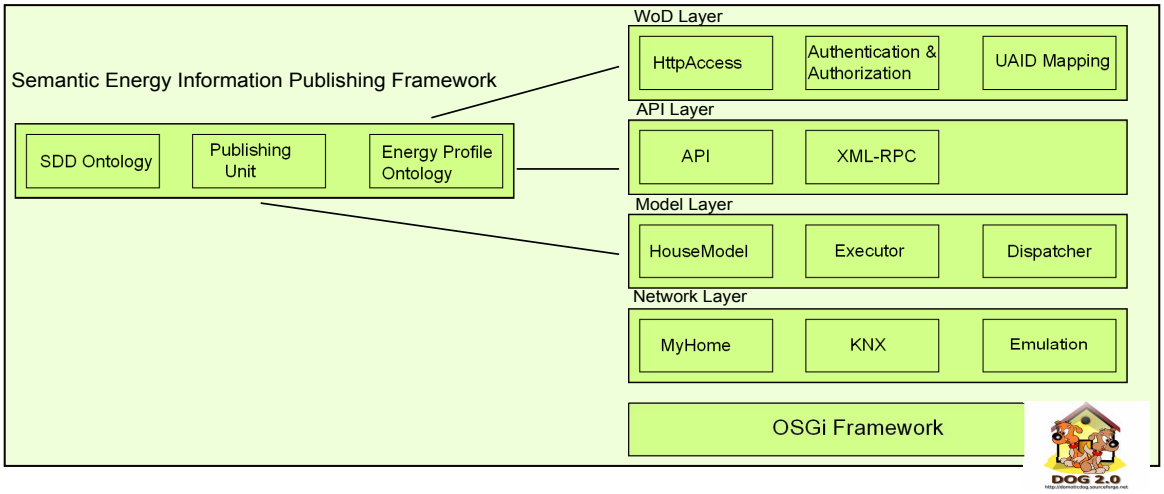


Figure 12.6. Publishing Framework Architecture

12.4.1 Publishing Unit

The Publishing unit is the core logic unit of the SEIPF (Figure 12.6). It accepts the request through the HttpAccess bundle. Based on the request, it queries the XML-RPC bundle of Dog to determine the current state of an appliance or set of appliances. Then it determines the power consumption of appliances by quering the E.P ontology and DogOnt ontology (available in the HouseModel Bundle of Dog). The publishing unit then sends a pure RDF response (based on the SimpleDomoticData ontology) to the requesting entity.

It has three methods, which can be accessed through the HttpAccess bundle by providing different parameters depicted in Table 12.1:

1. **getPowerInfo:** This function provides the current power consumption level of an appliance/device in the house based on the current state of the device. It takes the device identifier and returns the power consumption information about the device as an instance of SimpleDomoticData ontology. The device identifier is mentioned through the **device** parameter and the **command** parameter must be set to value **info**. An excerpt of the response is shown in Figure 12.5.

2. **getRoomPowerInfo:** This function provides the power consumption of all appliances present in a given room inside the house. The room is specified through the **room** parameter.

3. **getDeviceTypePowerInfo:** This function provides the power consumption of all appliances belonging to a single device category. By device category we mean the type of the device. For example, Lamp is the device category for all lamps inside the house. The name of the device category is passed as a **devicecategory** parameter.

SPARQL endpoint

In addition to predefined queries exposed through the WoD interface, the SEIPF also allows direct querying of the underlying ontologies, for applications that need to reason about the whole model, and not just to use the power data. We terefore provide an interface to send arbitrary queries, using the semantic web standard query language, SPARQL. The SPARQL endpoint provides the ontological level access to the requesting entity. To ensure safety, the access to this point is granted to highest levels of authorization, only. SPARQL queries are forwarded to query the DogOnt and the E.P ontologies directly, for extracting general device properties and the power consumption of the device. As defined in Table 12.1, the query parameter can be used to access the SPARQL end-point.

12.5 Implementation and Experiments

The core publishing unit of the SEIPF is implemented as a separate OSGi bundle inside the Dog. We adopted the Eclipse Equinox[4] OSGi framework to implement this bundle. As stated earlier, the SEIPF use the HttpAccess bundle (inside the WoD architecture) to provide access over the web. The HttpAccess bundle uses the jetty web server[5] to expose its services. Access Control is provided through the Authentication and Authorization bundle (inside the WoD architecture). It uses the OpenID4Java[6] library to forward the authentication credentials to the appropriate external OpenID server. Requesting entities are authorized according to different levels, based on the type of entity accessing the framework.

The goal of the SEIPF is to provide residential gateways the ability to expose the power consumption of different devices in an open, effective and semantic format, which in turn enables external applications to consume information according to their own application goals. Applications can be standalone providing the feedback on the current power consumption of devices or they could be mash-up applications that consume data from different data sharing sources to provide feedback.

To demonstrate such use cases, we have implemented two experimental applications. We integrated the SEIPF with the Dog and ran tests in our department Lab, using two demo cases (shown in Figure 12.7) equipped with a BTicino MyOpen and a KNX domotic plant, respectively. In the absence of a real inhabited house, we used the emulation capabilities of the Dog gateway to simulate the behavior of devices configured in the sample houses. In fact, Dog simulates domotic environments thanks to the DogSim [106] library, that exploits a state machine model describing the dynamic behavior of each device class. Thanks to DogSim, we are able either to interact with fully simulated environments, or to environments that include some simulated devices and some real domotic devices: we call "emulation" this last case, in which we may interact with a real plant by emulating some new devices before they are actually installed.

We marked different buttons to emulate the functioning of three devices for our experiment namely: a computer, a coffee maker and a lamp. The experiments were run with arbitrary input conditions, and no special assumption has been made over user behavior; therefore the attained results may be used to validate the measuring and publishing framework, but are not suitable to infer user-related conclusions.

The first experiment exploits a standalone application that provides the current power consumption of the emulated devices. The testing application was built

[4]http://www.eclipse.org/equinox/

[5]http://www.mortbay.org/jetty/

[6]http://code.google.com/p/openid4java/

Figure 12.7. BTicino and KNX Domotic demo cases

as a separate Java application that queries the SEIPF to acquire the current power consumption of emulated devices and uses the Google Chart Tools[7] to provide graphical feedback on the current power consumption of appliances. A snapshot of the response of the application is shown in Figure 12.8.

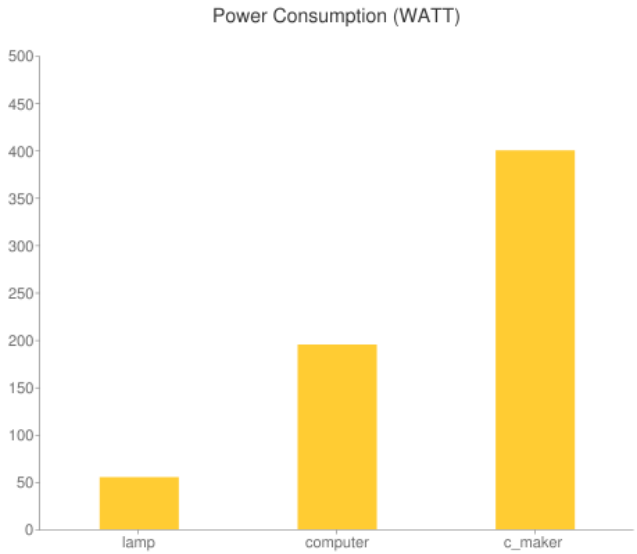


Figure 12.8. Current Power Consumption of Emulated Devices

The second experiment acts as a data sharing application. It accumulates the power consumption of individual emulated device over time. The application shares

[7]http://code.google.com/apis/charttools/

the data with the COSM[8] service. COSM is a convenient, secure and scalable platform that helps applications and services connect to and build the Internet of Things. It stores, shares and discovers real time sensors, energy and environment data from objects, devices and buildings around the world. COSM provides most of its functionality through a REST based API and can be used to send real time sensor, energy and environment data from anywhere around the globe. We built a Java application that periodically queries the SEIPF for polling instantaneous power consumption of appliances at regular time intervals. The application uses polling to acquire data over time and then sends the power consumption data to the COSM server. A snapshot of the power consumption data accumulated over time on COSM is shown in Figure 12.9.

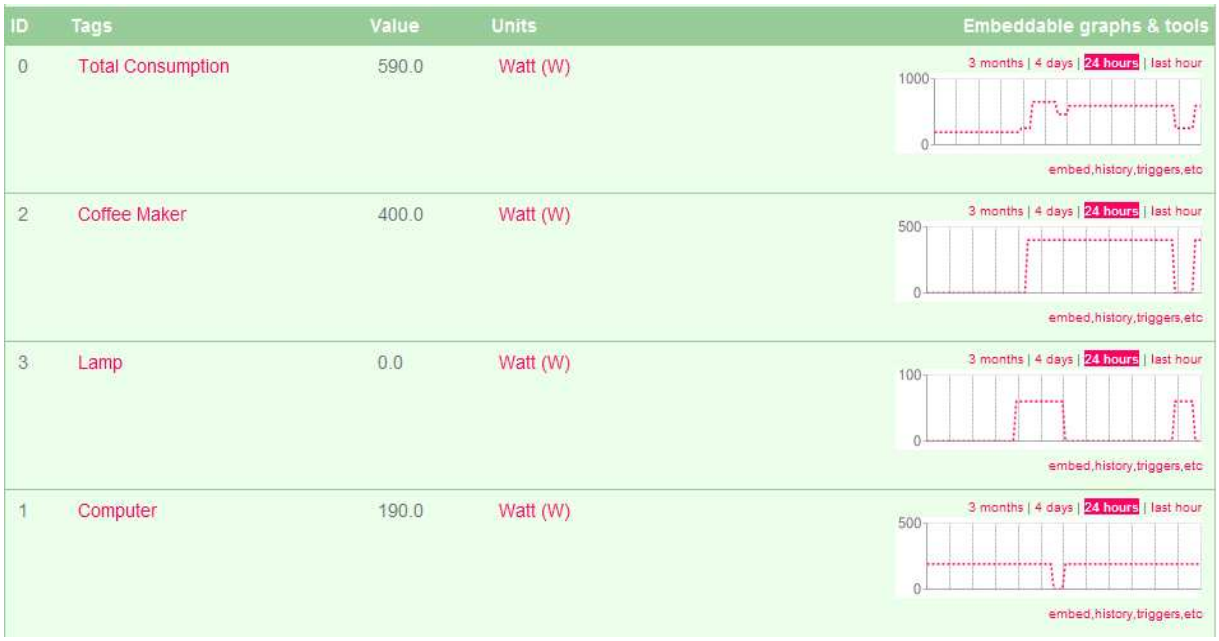


Figure 12.9. Power consumption snapshot obtained on COSM

The two experimental applications prove the feasibility of the framework as well as provide a step towards defining an open, standard and semantic powered format that will allow different applications to use the power consumption data according to their own application specific goals.

[8]https://cosm.com/

12.6 Related Works

Several works can be found in the literature that approach the residential energy consumption problem from many different viewpoints. These different approaches can be classified on the basis of the tackled aspects, e.g., estimation of consumption, gathering of available information, processing of measured consumption data, to cite the most relevant. In the first category, Fumo et al. devised a simplified methodology to estimate hourly electrical and fuel energy consumption of a (residential) building by applying a series of predetermined coefficients to monthly energy consumption data from electrical and fuels utility bills [77]. This approach can exploit the publishing framework introduced in this chapter to gather precise data on consumed energy and fuel and may exploit the easy-to-access information exposed by the SEIPF framework to increase the granularity of the estimation it provides.

Pérez-Lombard et al. worked on a review of available information concerning energy consumption in buildings [78], in particular related to HVAC systems. This work can be easily integrated in the SEIPF approach, contributing to first identify relevant information to be gathered and, second, receiving back as a benefit, more granularity on the same information plus integration with other information sources available in the home for which explicit billing data or statistical consumption data is still lacking.

In the information processing area many approaches can be found, which are more strictly related to the proposed SEIPF framework: Seem, for example, introduced a solution for the detection of abnormal energy consumption values in buildings [107] that exploits intelligent data analysis. In his chapter, Seem describes a novel method for detecting abnormal energy consumption in buildings based on daily readings of energy consumption and peak energy absorption. In this context SEIPF can act as data source allowing for direct collection of measurements and providing the basis for finer analysis based on shorter intervals of the order of minutes or seconds. At the same time, data exposed by the SEIPF framework can be consumed by other detection services allowing for the integration of several analysis toolkits in the same home/building environment.

Google PowerMeter [108] is a free energy monitoring tool that allows you to view your home's energy consumption from anywhere online. It assumes that the device must make an SSL-secured outbound TCP/IP connection to Google so that it can periodically transmit data to Google via HTTPS. Typically, a device should use the device owner's home Internet connection to transmit data. Our approach is different from that of Google Powermeter. Our basic assumption is that all devices in the house are controlled by a centralized residential gateway. The publishing framework exposes device information in a machine understandable RDF format, which can be semantically post-processed by any third party application, service or automated agent. The information is exposed through a proper authentication mechanism and

the resident of the house is provided the complete control over information that is exposed through the framework. Incidentally, a simple application using the SEIPF framework could act as a client of the Google PowerMeter system.

Weiss *et al.* [109] proposed an interactive feedback system that uses a smart electricity meter to provide consumption feedback for different household devices. It also provides a set of API to communicate with the smart electricity meters. The SEIPF approach is different as it can be installed on any residential gateway that uses DogOnt to define the domotic structure of a house. The SEIPF also has the ability to expose different information related to devices. Moreover, the SEIPF exposes information in pure RDF format which allows any application to consume the power consumption information according to its own application requirements. By contrast, [109] provides a predefined custom format, and exposes consumption information, only.

Sheth *et al.* [110] proposed that the sensor data retrieved from sensor networks is annotated with semantic meta-data to increase interoperability as well as provide contextual information essential for situational knowledge. In particular, annotating sensor data with spatial, temporal, and thematic semantic meta data. The SEIPF publishing framework exposes information in a pure RDF along with the option to view ontologies to understand the whole structure of information instead of annotating information. The framework also provides a mechanism to authenticate and authorize third party applications, services or automated agents, and provides the consumer information in a restricted manner.

12.7 Synopsis

This chapter presents a Semantic Energy Information Publishing Framework (SEIPF) that can be installed on a residential gateway (Dog2.0) to publish power consumption information of different appliances in a house environment. A new modular E.P ontology to model the power consumption of different appliances in the house is proposed. Modularity allows the E.P ontology to be plugged in the DogOnt ontology that models the domotic plant in a house. The SEIPF exposes the power consumed by different appliances and their properties in a pure RDF format. The goal of our approach is to make power consumption information machine understandable to support distributed applications such as intelligent negotiation. This will enable today third-party applications, services or agents (given authorization) to access the power consumption of a house or a building and help build standalone or data sharing applications to evolve a energy aware and energy efficient society.

We plan to extend the SEIPF by taking into account the time behavior, enabling it to publish energy figures over time intervals, thus reducing requirements over the polling intervals of client applications. In the future, the SEIPF could also help us

to build systems where energy consumption can be co-ordinated between different consumers. The semantic nature of the exposed data will help building applications where automated intelligent negotiation and consumption coordination can take place, evolving in to an intelligent energy grids.

Chapter 13

Conclusion

This thesis focused on the role of semantic web technologies in smart environments. In particular, the potential role in defining mechanisms for user intelligible goals and semantic data exchange in smart environments was discussed. This chapter concludes the thesis and provides future directions.

In order to model user intelligible goals in smart environments, a *Domotic Effects* framework was presented. Part II presented different aspects of *Domotic Effects* framework. The framework is designed as an ontology driven software that consists of evaluation, enforcement and optimization components. In literature, the *DE* framework can be considered among the initial approaches to model user intelligible goals. And then, using those goals to control and monitor smart environments. This thesis includes complete detail from conception and development to experimentation. The *DE* framework is currently an active research topic and in the future, the research will move in the direction of addressing following issues:

1. The thesis discusses the *DE* framework restricted to the Boolean application domain. Though the modeling of *DE* framework (Chapter 6) provides flexibility to define AmI layer for real-valued application domains, the evaluation and enforcement approaches for real-valued application domains need to be explored in the future.

2. Different requirements of the *DE* framework, i.e., modeling, evaluation, enforcement and optimization, were designed and tested individually. However, no qualitative comparison of features between *DE* framework and other approaches (in literature) is made in the thesis. In the future, such comparison will be necessary to understand the advantages and pitfalls of *Domotic Effects* based approach.

3. In order to understand the adaptability of the *DE* framework for users, a user study is needed in the future.

4. In order to enable users to easily define domotic effects on their environments, an intuitive GUI is needed in the future.

5. As defined in Chapter 5 and Chapter 8, at any instant the environment possesses a *global state* $g \in \mathcal{G}$ and the satisfiability of the user's request $\mathcal{R}$ amounts to function $F_{\mathcal{R}}(g)$. $F_{\mathcal{R}}(g)$ would give a new a *global state* $g' \in \mathcal{G}$. The problem of finding a *global evolution* $\mathcal{E}$ from g to g' is still open, i.e.,

$$g \xrightarrow{\mathcal{E}} g'$$

, where a *global evolution* $\mathcal{E}$ represents a sequence of commands and notification to (and from) individual devices.

6. Chapter 9 discussed the optimization of power consumption for a user's request $\mathcal{R}$. However, only active power was considered while designing the heuristic. In the future for more efficient energy optimization, measurements like reactive power consumption of devices and energy needed for the evolution $\mathcal{E}$ should also be considered. Moreover, heuristics may be designed to address other features like device wearing out.

As mentioned in Chapter 1, the issue of semantic data exchange was discussed within the context of Energy Management Systems. Part III discussed two approaches in order to publish power consumed by different devices in a smart environment. Though SEIPF was designed specifically for energy management systems, the LO(D)D framework is generic in nature. The work on *LO(D)D* framework is being done in the SMILE-O project and currently it is an active research topic. In the future, the research will move in the direction of addressing following issues:

1. Chapter 11 presented a prototype implementation of the LO(D)D architecture. In order to integrate LO(D)D within SMILE-O project, the implementation will be improved in the future. Furthermore, rigorous experimentation need to be performed so that the results can be solidified.

2. Currently, LO(D)D architecture provides the publisher component for stationary producer applications. However, in the future, mobile application will also be considered.

3. In the future, LO(D)D will be used to build and study federation of heterogeneous data (coming from sensors and mobile devices). In order to do so, the scalability of the LO(D)D architecture also needs to be investigated.

4. In the future, quantitative and qualitative comparison will be performed against rival approaches. It will help understand the advantages and pitfalls of *LO(D)D*.

Bibliography

[1] T.B. Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[2] G. Antoniou and F. Van Harmelen. *A semantic web primer*. the MIT Press, 2004.

[3] I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler. Semantic web architecture: Stack or two towers? *Principles and Practice of Semantic Web Reasoning*, pages 37–41, 2005.

[4] N. Shadbolt, W. Hall, and T. Berners-Lee. The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96–101, 2006.

[5] G. Klyne, J.J. Carroll, and B. McBride. Resource description framework (rdf): Concepts and abstract syntax. *W3C recommendation*, 10, 2004.

[6] World Wide Web Consortium et al. Rdf vocabulary description language 1.0: Rdf schema. *W3C recommendation*, pages 02–10, 2004.

[7] D.L. McGuinness, F. Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10:2004–03, 2004.

[8] E. Prud'Hommeaux and A. Seaborne. SPARQL query language for RDF. Technical Report January, W3C, 2008.

[9] Tim Berners-Lee. Keynote on web architecture, 2006.

[10] C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[11] C. Diane and D. Sajal. *Smart environments: Technology, protocols and applications*. Wiley-Interscience, 2004.

[12] M. Weiser. The computer for the 21st century. *Scientific American*, 272(3):78–89, 1995.

[13] X. Wang, J.S. Dong, C.Y. Chin, S.R. Hettiarachchi, and D. Zhang. Semantic space: an infrastructure for smart spaces. *Pervasive Computing, IEEE*, 3(3):32–39, 2004.

[14] S. Bader and M. Dyrba. Goalaviour-based control of heterogeneous and distributed smart environments. In *Intelligent Environments (IE), 2011 7th International Conference on*, pages 142 –148, july 2011.

[15] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu. Sensor-based activity recognition. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(6):790 –808, nov. 2012.

[16] Brandon Davito, Humayun Tai, , and Robert Uhlaner. The smart grid and the promise of demand-side-management. Technical report, McKinsey & Company, 2011.

[17] P. Palensky and D. Dietrich. Demand side management: Demand response, intelligent energy systems, and smart loads. *Industrial Informatics, IEEE Transactions on*, 7(3):381–388, 2011.

[18] H. Sui, Y. Sun, and W.J. Lee. A demand side management model based on advanced metering infrastructure. In *Electric Utility Deregulation and Restructuring and Power Technologies (DRPT), 2011 4th International Conference on*, pages 1586–1589. IEEE, 2011.

[19] A. Faruqui, R. Hledik, and J. Tsoukalis. The power of dynamic pricing. *The Electricity Journal*, 22(3):42–56, 2009.

[20] M.A.A. Pedrasa, T.D. Spooner, and I.F. MacGill. Coordinated scheduling of residential distributed energy resources to optimize smart home energy services. *Smart Grid, IEEE Transactions on*, 1(2):134–143, 2010.

[21] Yann Riche, Jonathan Dodge, and Ronald A. Metoyer. Studying always-on electricity feedback in the home. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1995–1998, New York, NY, USA, 2010. ACM.

[22] Simon Roberts, Helen Humphries, and Verity Hyldon. Consumer preferences for improving energy consumption feedback. Technical report, Centre for Sustainable Energy, 2004.

[23] Markus Weiss, Friedemann Mattern, Tobias Graml, Thorsten Staake, and Elgar Fleisch. Handy feedback: connecting smart meters with mobile phones. In *Proceedings of the 8th International Conference on Mobile and Ubiquitous Multimedia*, MUM '09, pages 15:1–15:4, New York, NY, USA, 2009. ACM.

[24] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the eleventh ACM Symposium on Operating systems principles*, SOSP '87, pages 123–138. ACM, 1987.

[25] Ivan Herman. Semantic web activity statement. Technical report, W3C, 2001.

[26] W3C. RDF primer. Technical report, W3C, 2004.

[27] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (xml). *World Wide Web Journal*, 2(4):27–66, 1997.

[28] T. Gruber. What is an ontology? *Knowledge Acquisition*, 5(2):199–220, 1993.

[29] W3C. OWL : Ontology Web Language. Technical report, W3C, 2004.

[30] O. Hartig, C. Bizer, and J.C. Freytag. Executing sparql queries over the web of linked data. In *Proceedings of the 8th International Semantic Web Conference*, pages 293–309. Springer-Verlag, 2009.

[31] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.

[32] B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, pages 524–538. Springer-Verlag, 2008.

[33] Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, Volume One. Technical Report W3C Recommendation 15 December 2004, W3C, 2004.

[34] T. Heath and C. Bizer. Linked data: Evolving the web into a global data space. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136, 2011.

[35] Dario Bonino and Fulvio Corno. Dogont - ontology modeling for intelligent domotic environments. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 790–803. Springer Berlin Heidelberg, 2008.

[36] Dario Bonino, Emiliano Castellina, and Fulvio Corno. The DOG Gateway: Enabling Ontology-based Intelligent Domotic Environments. *IEEE Transactions on Consumer Electronics*, 54:4:1656–1664, November 2008.

[37] O.S.G. Alliance. *Osgi service platform, release 3*. IOS Press, Inc., 2003.

[38] S. Davidoff, M.K. Lee, C. Yiu, J. Zimmerman, and A.K. Dey. Principles of smart home control. In *Proceedings of the 8th international conference on Ubiquitous Computing*, pages 19–34. Springer-Verlag, 2006.

[39] M. Garcia-Herranz, P. Haya, and X. Alaman. Towards a ubiquitous end-user programming system for smart spaces. *Journal of Universal Computer Science*, 16(12):1633–1649, 2010.

[40] M. Garcia-Herranz, P.A. Haya, A. Esquivel, G. Montoro, and X. Alaman. Easing the smart home: Semi-automatic adaptation in perceptive environments. *Journal of Universal Computer Science*, 14(9):1529–1544, 2008.

[41] F. Kawsar, T. Nakajima, and K. Fujinami. Deploy spontaneously: supporting end-users in building and enhancing a smart home. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 282–291. ACM, 2008.

[42] A. Katasonov. Enabling non-programmers to develop smart environment applications. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 1059–1064. IEEE, 2010.

[43] P. Rashidi and D.J. Cook. Keeping the resident in the loop: Adapting the smart home to the user. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(5):949 –959, sept 2009.

[44] E. Salomons, W. Teeuw, H. van Leeuwen, and P. Havinga. Persona-based adaptation in a smart green home. In *Intelligent Environments (IE), 2012 8th International Conference on*, pages 355–358. IEEE, 2012.

[45] S.T. Cheng, C.H. Wang, and C.C. Chen. An adaptive scenario based reasoning system cross smart houses. In *Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on*, pages 549–554. IEEE, 2009.

[46] A.K. Dey, G.D. Abowd, and D. Salber. A context-based infrastructure for smart environments. In *Managing Interaction in Smart Environments, MANSE'09. 1st International Workshop on*, pages 114–128. Springer, 1999.

[47] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and JC Burgelman. Ambient intelligence: From vision to reality. Technical report, IST Advisory Group, 2003.

[48] M. van Doorn, A. de Vries, and E. Aarts. End-user software engineering of smart retail environments: the intelligent shop window. In *Ambient Intelligence: European Conference, AmI 2008, Nuremberg, Germany, November 19-22, 2008. Proceedings*, volume 5355, page 157. Springer-Verlag New York Inc, 2008.

[49] T. Heider and T. Kirste. Supporting goal-based interaction with dynamic intelligent environments. In *ECAI*, pages 596–602. Fraunhofer Publica (Germany), 2002.

[50] J.L. Encarnaçao and T. Kirste. Ambient intelligence: Towards smart appliance ensembles. In Matthias Hemmje, Claudia NiederÃ©e, and Thomas Risse, editors, *From Integrated Publication and Information Systems to Information and Knowledge Environments*, volume 3379 of *Lecture Notes in Computer Science*, pages 261–270. Springer Berlin Heidelberg, 2005.

[51] Michael Hellenschmidt. Distributed implementation of a self-organizing decentralized multimedia appliance middleware. In Nigel Davies, Thomas Kirste, and Heidrun Schumann, editors, *Mobile Computing and Ambient Intelligence: The Challenge of Multimedia*, number 05181 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. IBFI, Germany.

[52] H. Dibowski, J. Ploennigs, and K. Kabitzsch. Automated design of building automation systems. *Industrial Electronics, IEEE Transactions on*, 57(11):3606–3613, 2010.

[53] R. Velik and H. Boley. Neurosymbolic alerting rules. *Industrial Electronics, IEEE Transactions on*, 57(11):3661–3668, 2010.

[54] R. Velik and G. Zucker. Autonomous perception and decision making in building automation. *Industrial Electronics, IEEE Transactions on*, 57(11):3645–3652, 2010.

[55] R. Velik, G. Zucker, and D. Dietrich. Towards automation 2.0: a neurocognitive model for environment recognition, decision-making, and action execution. *EURASIP Journal on Embedded Systems*, 2011:4, 2011.

[56] F. Amigoni, N. Gatti, C. Pinciroli, and M. Roveri. What planner for ambient intelligence applications? *Systems, Man and Cybernetics, Part A: Systems*

and Humans, IEEE Transactions on, 35(1):7 – 21, 2005.

[57] E. Kaldeli, E.U. Warriach, J. Bresser, A. Lazovik, and M. Aiello. Interoperation, composition and simulation of services at home. In *Service-Oriented Computing: 8th International Conference, ICSOC 2010, San Francisco, CA, USA, December 7-10, 2010. Proceedings*, volume 6470, page 167. Springer-Verlag New York Inc, 2010.

[58] Dario Bonino, Fulvio Corno, and Luigi De Russis. A user-friendly interface for rules composition in intelligent environments. In Paulo Novais, Davy Preuveneers, and Juan Corchado, editors, *Ambient Intelligence - Software and Applications*, volume 92 of *Advances in Intelligent and Soft Computing*, pages 213–217. Springer Berlin / Heidelberg, 2011.

[59] Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, and Koen De Bosschere. Towards an extensible context ontology for Ambient Intelligence. In *Second European Symposium on Ambient Intelligence*, volume 3295 of *LNCS*, pages 148 – 159, Eindhoven, The Netherlands, Nov 8 – 11 2004. Springer.

[60] H. Chen, F. Perich, T. Finin, and A. Joshi. Soupa: standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 258 – 267, aug. 2004.

[61] F. Ramparany, R. Poortinga, M. Stikic, J. Schmalenstroer, and T. Prante. An open context information management infrastructure the IST-amigo project. In *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on*, pages 398–403. IET, 2008.

[62] Francesco Furfari, Lorenzo Sommaruga, Claudia Soria, and Roberto Fresco. DomoML: the definition of a standard markup for interoperability of human home interactions. In *EUSAI '04: Proceedings of the 2nd European Union symposium on Ambient intelligence*, pages 41–44, New York, NY, USA, 2004. ACM.

[63] Juan Ye, Graeme Stevenson, and Simon Dobson. A top-level ontology for smart environments. *Pervasive and Mobile Computing*, 7(3):359 – 378, 2011. Knowledge-Driven Activity Recognition in Intelligent Environments.

[64] Ioanna Roussaki, Ioannis Papaioannou, Dimitrios Tsesmetzis, Julia Kantorovitch, Jarmo Kalaoja, and Remco Poortinga. Ontology based service modelling for composability in smart home environments. In Max Muhlhauser, Alois Ferscha, and Erwin Aitenbichler, editors, *Constructing Ambient Intelligence*, volume 11 of *Communications in Computer and Information Science*, pages 411–420. Springer Berlin Heidelberg, 2008.

[65] D. Bonino, E. Castellina, and F. Corno. The dog gateway: enabling ontology-based intelligent domotic environments. *Consumer Electronics, IEEE Transactions on*, 54(4):1656 –1664, november 2008.

[66] Melvin A. Breuer and Arthur D. Friedman. *Diagnosis and Reliable Design of Digital Systems*. Computer Science Press, 1976.

[67] D. Chen, J. Yang, and H.D. Wactlar. Towards automatic analysis of social interaction patterns in a nursing home environment from video. In *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 283–290. ACM, 2004.

[68] D. Le Berre and A. Parrain. The Sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[69] K. Erol, J. Hendler, and D.S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[70] Diane J Cook, Juan C Augusto, and Vikramaditya R Jakkula. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277–298, 2009.

[71] D.J. Cook, M. Youngblood, E.O. Heierman III, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja. MavHome: An agent-based smart home. In *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 521–524. IEEE, 2003.

[72] C. Ge, Y. Li, X. Zhi, and W. Tong. The intelligent stb-implementation of next generation of residential gateway in digital home. In *Pervasive Computing and Applications, 2007. ICPCA 2007. 2nd International Conference on*, pages 256–261. IEEE, 2007.

[73] Department of Energy, USA. 2008 buildings energy data book. Technical report, Buildings Technologies Program Energy Efficiency and Renewable Energy, 2009.

[74] S. Lukovic, V. Congradac, and F. Kulic. A system level model of possible integration of building management system in smartgrid. In *Complexity in Engineering, 2010. COMPENG '10.*, pages 58 –60, feb. 2010.

[75] I. Koutsopoulos and L. Tassiulas. Challenges in demand load control for the smart grid. *Network, IEEE*, 25(5):16–21, 2011.

[76] P. McDaniel and S. McLaughlin. Security and privacy challenges in the smart grid. *Security & Privacy, IEEE*, 7(3):75–77, 2009.

[77] Nelson Fumo, Pedro Mago, and Rogelio Luck. Methodology to estimate building energy consumption using energyplus benchmark models. *Energy and Buildings*, 42(12):2331 – 2337, 2010.

[78] L. Pérez-Lombard, J. Ortiz, and C. Pout. A review on buildings energy consumption information. *Energy and Buildings*, 40(3):394–398, 2008.

[79] T. Hubert and S. Grijalva. Realizing smart grid benefits requires energy optimization algorithms at residential level. In *Innovative Smart Grid Technologies (ISGT), 2011 IEEE PES*, pages 1 –8, 2011.

[80] O.A. Sianaki, O. Hussain, T. Dillon, and A.R. Tabesh. Intelligent decision support system for including consumers preferences in residential energy consumption in smart grid. In *Proceedings of the 2010 Second International Conference on Computational Intelligence, Modeling and Simulation*, pages 154–159, 2010.

[81] H. Zhang, X. Xia, and J. Zhang. A residential energy and power conservation system utilizing an optimization model. In *AFRICON, 2009. AFRICON '09.*, pages 1 –6, sept. 2009.

[82] T.T. Kim and H.V. Poor. Scheduling power consumption with price uncertainty. *Smart Grid, IEEE Transactions on*, 2(3):519 –527, sept. 2011.

[83] A.-H. Mohsenian-Rad and A. Leon-Garcia. Optimal residential load control with price prediction in real-time electricity pricing environments. *Smart Grid, IEEE Transactions on*, 1(2):120 –133, sept. 2010.

[84] A. Mohsenian-Rad, V.W.S. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia. Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid. *Smart Grid, IEEE Transactions on*, 1(3):320 –331, dec. 2010.

[85] Energy consumption by sector, 2005.

[86] P. Lalanda and J. Bourcier. Towards autonomic residential gateways. In *IEEE International Conference on Pervasive Services (ICPS 2006)*, pages 329–332. IEEE, 2006.

[87] C. Escoffier, J. Bourcier, P. Lalanda, and J. Yu. Towards a home application server. In *IEEE Consumer Communications and Networking Conference, 2008*, pages 321–325, 2008.

[88] S.L. Chung and W.Y. Chen. MyHome: A Residential Server for Smart Homes. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 664–670. Springer, 2010.

[89] J. Bourcier, A. Chazalet, M. Desertot, C. Escoffier, and C. Marin. A dynamic-SOA home control gateway. In *IEEE International Conference on Services Computing, 2006. SCC'06*, pages 463–470, 2006.

[90] IEA. World energy outlook 2009 fact sheet: Why is our current energy pathway unsustainable? Technical report, International Energy Agency, 2009.

[91] T. Gu, X.H. Wang, H.K. Pung, and D.Q. Zhang. An ontology-based context model in intelligent environments. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, pages 270–275, 2004.

[92] O. Lassila and R.R. Swick. Resource Description Framework (RDF) model and syntax, 1999.

[93] A. Passant and P.N. Mendes. sparqlPuSH: Proactive notification of data updates in RDF stores using PubSubHubbub. In *Scripting for the Semantic Web Workshop (SFSW2010) at ESWC2010*, 2010.

[94] P. Barnaghi, M. Presser, and K. Moessner. Publishing Linked Sensor Data. In *Proceedings of the 3rd International Workshop on Semantic Sensor Networks*, 2010.

[95] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, pages 635–644. ACM, 2011.

[96] D.F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th international conference on World wide web*, pages 1061–1062. ACM, 2009.

[97] S. Groppe, J. Groppe, D. Kukulenz, and V. Linnemann. A SPARQL engine for streaming RDF data. In *Signal-Image Technologies and Internet-Based System, 2007. Third International IEEE Conference on*, pages 167–174. IEEE, 2007.

[98] D. Le-Phuoc, M. Dao-Tran, J.X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *Proceedings of the 10th international conference on The semantic web-Volume Part I*, pages 370–388. Springer-Verlag, 2011.

[99] Sven Meyer and Andry Rakotonirainy. A survey of research on context-aware homes. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 159–168, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[100] George Demiris, Marilyn J Rantz, Myra A Aud, Karen D Marek, Harry W Tyrer, Marjorie Skubic, and Ali A Hussam. Older adults' attitudes towards and perceptions of 'smart home' technologies: a pilot study. *Informatics for Health and Social Care*, 29(2):87–94, 2004.

[101] F. Razzak, D. Bonino, and F. Corno. Mobile interaction with smart environments through linked data. In *IEEE International Conference on Systems, Man, and Cybernetics, October 10-13, 2010*, pages 2922–2929, 2010.

[102] Chi-Chun Pan, Prasenjit Mitra, and Peng Liu. Semantic access control for information interoperation. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 237–246, New York, NY, USA, 2006. ACM.

[103] Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In *The Semantic Web Conference - ISWC 2006*, pages 473–486, 2006.

[104] Cecilia M. Ionita and Sylvia L. Osborn. Specifying an access control model for ontologies for the semantic web. In *Secure Data Management*, pages 73–85,

2005.

[105] D.Recordon and D.Reed. OpenID 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, pages 11–16, 2006.

[106] Dario Bonino and Fulvio Corno. DogSim: A state chart simulator for Domotic Environments. In *8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010*, pages 208–213. IEEE, March 2010.

[107] John E. Seem. Using intelligent data analysis to detect abnormal energy consumption in buildings. *Energy and Buildings*, 39(1):52 – 58, 2007.

[108] Google. Google power meter, 2009.

[109] M. Weiss, F. Mattern, T. Graml, T. Staake, and E. Fleisch. Handy feedback: Connecting smart meters with mobile phones. In *Proceedings of the 8th International Conference on Mobile and Ubiquitous Multimedia*, pages 1–4. ACM, 2009.

[110] Amit Sheth, Cory Henson, and Satya S. Sahoo. Semantic sensor web. *IEEE Internet Computing*, 12:78–83, 2008.

Appendices

Appendix A

Use cases

The appendix provides the functional representation of six use cases, i.e., Secure Home, Bath Room Illumination, Home Illumination, Afternoon Lunch Cooking, Air Flow and Morning wakeup.

Table A.1. Secure Home use case

$SecureHome\_All = \mathbf{OR}(SecureHome\_Scenario\_1, SecureHome\_Scenario\_2);$
$SecureHome\_Scenario\_1 = \mathbf{AND}(LivingRoom\_l2\_WS\_CloseDown, BathRoom\_WS\_CloseDown, Kitchen\_WS\_CloseDown,$
$BedRoom\_l1\_WS\_North\_CloseDown, BedRoom\_WS\_West\_CloseDown, DoorActuator\_d4\_lobby\_ext\_Close,$
$LivingRoom\_L1\_WS\_CloseDown);$
$LivingRoom\_l2\_WS\_CloseDown = \mathbf{AND}(WindowActuator\_w6\_living\_Close, ShutterActuator\_sh2\_living\_Down);$
$LivingRoom\_L1\_WS\_CloseDown = \mathbf{AND}(WindowActuator\_w5\_living\_Close, ShutterActuator\_sh1\_living\_Down);$
$BathRoom\_WS\_CloseDown = \mathbf{AND}(WindowActuator\_w3\_bath\_Close, ShutterActuator\_bath\_Down);$
$BedRoom\_l1\_WS\_North\_CloseDown = \mathbf{AND}(WindowActuator\_w1\_living\_Close, ShutterActuator\_sh1\_Down);$
$BedRoom\_WS\_West\_CloseDown = \mathbf{AND}(WindowActuator\_w2\_Close, ShutterActuator\_sh2\_Down);$
$SecureHome\_Scenario\_2 = \mathbf{AND}(Secure\_LivingRoom, Secure\_BedRoom, Secure\_BathRoom, Secure\_Lobby, Secure\_Kitchen);$
$Secure\_LivingRoom = \mathbf{AND}(DoorActuator\_d7\_kitchen\_Close, DoorActuator\_d6\_living\_Close,$
$Tv\_LivingRoom\_Off, LivingRoom\_L1\_WS\_CloseDown, LivingRoom\_l2\_WS\_CloseDown);$
$Secure\_BedRoom = \mathbf{AND}(DoorActuator\_d1\_bed\_Close, BedRoom\_l1\_WS\_North\_CloseDown, BedRoom\_WS\_West\_CloseDown);$
$Secure\_BathRoom = \mathbf{AND}(BathRoom\_Ws\_CloseDown, DoorActuator\_d2\_bath\_Close);$
$Secure\_Lobby = \mathbf{AND}(DoorActuator\_d6\_living\_Close, DoorActuator\_d5\_kitchen\_Close,$
$DoorActuator\_d4\_lobby\_ext\_Close, DoorActuator\_d3\_lobby\_stor\_Close, DoorActuator\_d1\_bed\_Close,$
$DoorActuator\_d2\_bath\_Close);$
$Secure\_Kitchen = \mathbf{AND}(DoorActuator\_d5\_kitchen\_Close, DoorActuator\_d7\_kitchen\_Close, Kitchen\_WS\_CloseDown);$
$Kitchen\_WS\_CloseDown = \mathbf{AND}(WindowActuator\_w4\_kitchen\_Close, ShutterActuator\_kitchen\_Down);$

Table A.2. Bathroom Illumination functional form

$Illumination = \mathbf{Or}(ArtificialIllumination, NaturalIllumination)$
$ArtificialIllumination = \mathbf{Alternate}(CeilingLampIllumination, MirrorLampIllumination)$
$MirrorLampIllumination = \mathbf{And}(LeftMirrorLampIllumination, RightMirrorLampIllumination)$
$RightMirrorLampIllumination = \mathbf{SE}(l9, OnState\_lamp9)$
$LeftMirrorLampIllumination = \mathbf{SE}(l8, OnState\_lamp8)$
$CeilingLampIllumination = \mathbf{SE}(l2, OnState\_lamp2)$
$NaturalIllumination = \mathbf{SE}(ShutterBath, UpStateValue\_ShutterBath)$

Table A.3. Home Illumination use case

$$Home\_Illumination = \mathbf{OR}(Natural\_Illumination\_Home, Artificial\_Illumination\_Home);$$
$$Natural\_Illumination\_Home = \mathbf{AND}(BedRoom\_Natural\_Illumination, DoorActuator\_d7\_kitchen\_Open, DoorActuator\_d6\_living\_Open, BathRoom\_WS\_CloseUp, DoorActuator\_d5\_kitchen\_Open, DoorActuator\_d1\_bed\_Open, LivingRoom\_Natural\_Illumination, Kitchen\_WS\_CloseUp);$$
$$BedRoom\_Natural\_Illumination = \mathbf{AND}(BedRoom\_WS\_West\_Closeup, BedRoom\_l1\_WS\_North\_CloseUp);$$
$$BedRoom\_l1\_WS\_North\_CloseUp = \mathbf{AND}(WindowActuator\_w1\_living\_Close, ShutterActuator\_sh1\_Up);$$
$$BedRoom\_WS\_West\_CloseUp = \mathbf{AND}(WindowActuator\_w2\_Close, ShutterActuator\_sh2\_Up);$$
$$LivingRoom\_Natural\_Illumination = \mathbf{OR}(LivingRoom\_l1\_WS\_CloseUp, LivingRoom\_l2\_WS\_CloseUp);$$
$$LivingRoom\_l1\_WS\_CloseUp = \mathbf{AND}(WindowActuator\_w5\_living\_Close, ShutterActuator\_sh1\_living\_Up);$$
$$LivingRoom\_l2\_WS\_CloseUp = \mathbf{AND}(WindowActuator\_w6\_living\_Close, ShutterActuator\_sh2\_living\_Up);$$
$$Artificial\_Illumination\_Home = \mathbf{AND}(Lobby\_Illumination, Lamp6\_Kitchen\_On, artificiallyIlluminatedBath, Lamp1\_BedRoom\_On, Lamp7\_LivingRoom\_On);$$
$$Lobby\_Illumination = OR(Lobby\_Illumination\_All, Lobby\_Illumination\_Alternate);$$
$$Lobby\_Illumination\_Alternate = \mathbf{ALTERNATE}(Lamp4\_Lobby\_On, Lamp5\_Lobby\_On);$$
$$Lobby\_Illumination\_All = \mathbf{AND}(Lamp4\_Lobby\_On, Lamp5\_Lobby\_On);$$
$$artificialIllumination = \mathbf{ALTERNATE}(celingLamp\_On, MirrorLampsOn);$$
$$MirrorLampsOn = \mathbf{AND}(Lamp9\_On, Lamp8\_On);$$

Table A.4. Afternoon Lunch Cooking use case

$Afternoon\_Lunch = \mathbf{AND}(Oven\_Kitchen\_On, Tv\_Kitchen\_On, Kitchen\_CookingDay\_Scenario\_Alt)$;
$Kitchen\_CookingDay\_Scenario\_Alt = \mathbf{ALTERNATE}(Kitchen\_CookingDay\_Scenario\_1, Kitchen\_CookingDay\_Scenario\_2)$;
$Kitchen\_CookingDay\_Scenario\_1 = \mathbf{AND}(ExhaustFan\_On, DoorActuator5\_Close, DoorActuator7\_Close, Lamp6\_Off, Kitchen\_WS\_Day\_Scenario)$;
$Kitchen\_WS\_Day\_Scenario = \mathbf{ALTERNATE}(Kitchen\_WS\_CloseUp, Kitchen\_WS\_OpenDown)$;
$Kitchen\_WS\_CloseUp = \mathbf{AND}(WindowActuator\_Kitchen\_Close, Shutter\_Kitchen\_Up)$;
$Kitchen\_WS\_OpenDown = \mathbf{AND}(WindowActuator\_Kitchen\_Open, Shutter\_Kitchen\_Down)$;
$Kitchen\_CookingDay\_Scenario\_2 = \mathbf{AND}(ExhaustFan\_On, DoorActuator5\_Close, DoorActuator7\_Close, Lamp6\_On, Kitchen\_WS\_CloseDown)$;
$Kitchen\_WS\_CloseUp = \mathbf{AND}(WindowActuator\_Kitchen\_Close, Shutter\_Kitchen\_Down)$;

Table A.5. Air Passage use case

$AirPassage\_All = \mathbf{AND}(AirPassage\_LRBR\_Scenario\_1, AirPassage\_LRBR\_Scenario\_2, AirPassage\_LRKT\_Scenario\_1, Door\_Kitchen\_d5\_Open)$;

$AirPassage\_LRBR\_Scenario\_1 = \mathbf{AND}(LivingRoom\_Windows\_Open\_Any, DoorActuator\_d6\_living\_Open, DoorActuator\_d1\_Bed\_Open, BedRoom\_l1\_WS\_North\_OpenUp)$;

$LivingRoom\_Windows\_Open\_Any = \mathbf{OR}(LivingRoom\_Windows\_Open\_Alternate, LivingRoom\_Windows\_Open)$;

$LivingRoom\_Windows\_Open\_Alternate = \mathbf{ALTERNATE}(LivingRoom\_Windows\_North\_Open, LivingRoom\_Windows\_South\_Open)$;

$LivingRoom\_Windows\_South\_Open = \mathbf{NOT}(LivingRoom\_Windows\_North\_Open)$;

$LivingRoom\_Windows\_North\_Open = \mathbf{AND}(LivingRoom\_l1\_WS\_OpenUp, LivingRoom\_l2\_WS\_CloseDown)$;

$LivingRoom\_l1\_WS\_OpenUp = \mathbf{AND}(WindowActuator\_w5\_living\_Open, ShutterActuator\_sh1\_living\_Up)$;

$LivingRoom\_l2\_WS\_CloseDown = \mathbf{AND}(WindowActuator\_w6\_living\_Close, ShutterActuator\_sh2\_living\_Down)$;

$LivingRoom\_Windows\_Open = \mathbf{AND}(LivingRoom\_l2\_WS\_OpenUp, LivingRoom\_l1\_WS\_OpenUp)$;

$LivingRoom\_l2\_WS\_OpenUp = \mathbf{AND}(WindowActuator\_w6\_living\_Open, ShutterActuator\_sh2\_living\_Up)$;

$LivingRoom\_l1\_WS\_OpenUp = \mathbf{AND}(WindowActuator\_w5\_living\_Open, ShutterActuator\_sh1\_living\_Up)$;

$BedRoom\_l1\_WS\_North\_OpenUp = \mathbf{AND}(WindowActuator\_w1\_living\_Open, ShutterActuator\_sh1\_Up)$;

$AirPassage\_LRBR\_Scenario\_2 = \mathbf{AND}(DoorActuator\_d1\_bed\_Open, LivingRoom\_Windows\_Open\_Any, DoorActuator\_d6\_living\_open, BedRoom\_WS\_West\_OpenUp)$;

$BedRoom\_WS\_West\_OpenUp = \mathbf{AND}(WindowActuator\_w2\_Open, ShutterActuator\_sh2\_Up)$;

$AirPassage\_LRKT\_Scenario\_1 = \mathbf{AND}(LivingRoom\_Windows\_Open\_Any, Kitchen\_WS\_OpenUp, DoorActuator\_d7\_kitchen\_Open, ExhaustFan\_Kitchen\_On)$;

$Kitchen\_WS\_OpenUp = \mathbf{AND}(WindowActuator\_w4\_kitchen\_Open, ShutterActuator\_kiitchen\_Up)$;

Table A.6. Morning Wake Up use case

$Morning\_WakeUp = \mathbf{AND}(BathRoomIllumination, Radio\_BathRoom\_On, Tv\_Kitchen\_On,$
$BedRoom\_Natural\_Illumination, Kitchen\_Cooking\_Day\_Scenario\_1, GasHeater\_BedRoom\_On);$
$BathRoomIllumination = \mathbf{OR}(artificialIllumination, ShuterBathUp);$
$artificialIllumination = \mathbf{ALTERNATE}(celingLamp\_On, MirrorLampsOn);$
$MirrorLampsOn = \mathbf{AND}(Lamp9\_On, Lamp8\_On);$
$Kitchen\_CookingDay\_Scenario\_1 = \mathbf{AND}(ExhaustFan\_On, DoorActuator5\_Close, DoorActuator7\_Close,$
$Lamp6\_Off, Kitchen\_WS\_Day\_Scenario);$
$Kitchen\_WS\_Day\_Scenario = \mathbf{ALTERNATE}(Kitchen\_WS\_CloseUp, Kitchen\_WS\_OpenDown);$
$Kitchen\_WS\_CloseUp = \mathbf{AND}(WindowActuator\_Kitchen\_Close, Shutter\_Kitchen\_Up);$
$Kitchen\_WS\_OpenDown = \mathbf{AND}(WindowActuator\_Kitchen\_Open, Shutter\_Kitchen\_Down);$
$BedRoom\_Natural\_Illumination = \mathbf{AND}(BedRoom\_WS\_West\_CloseUp, BedRoom\_L1\_WS\_North\_CloseUp);$
$BedRoom\_WS\_West\_CloseUp = \mathbf{AND}(WindowActuator\_w2\_Close, ShutterActuator\_sh2\_Up);$
$BedRoom\_L1\_WS\_North\_CloseUp = \mathbf{AND}(WindowActuator\_w1\_Close, ShutterActuator\_sh1\_Up);$

Appendix B

Publications

B.1 International Journals

1. Fulvio Corno, Faisal Razzak (2012) **Intelligent Energy Optimization for User Intelligible Goals in Smart Home Environments** In: IEEE TRANSACTIONS ON SMART GRID. vol. 3/4, pp. 2128 - 2135.

2. Dario Bonino, Fulvio Corno, Faisal Razzak (2011) **Enabling Machine Understandable Exchange of Energy Consumption Information in Intelligent Domotic Environments** In: ENERGY AND BUILDINGS, vol. 43/6, pp. 1392-1402.

B.2 Proceedings

1. Fulvio Corno, Faisal Razzak (2012) **Publishing LO(D)D: Linked Open (Dynamic) Data for Smart Sensing and Measuring Environments** In: PROCEDIA COMPUTER SCIENCE, vol. 10C, pp. 381-388.

2. Faisal Razzak (2012) **Spamming the Internet of Things: A Possibility and its probable Solution** In: PROCEDIA COMPUTER SCIENCE, vol. 10, pp. 658-665.

3. Faisal Razzak (2012) **Semantic Web Technologies role in Smart Environments** In: OTM Workshops, LNCS 7567, pp. 54–58 Springer – (International Workshop)

4. Faisal Razzak, Dario Bonino, Fulvio Corno (2010) **Mobile Interaction with Smart Environments through Linked Data** In: IEEE International Conference on Systems, Man, and Cybernetics, Istanbul, Turkey, October 10-13. pp. 2922-2929

5. Emiliano Castellina, Faisal Razzak, Fulvio Corno (2009) **Environmental Control Application compliant with Cogain Guidelines** In: COGAIN 2009: Gaze Interaction For Those Who Want It Most, COGAIN 2009, Copenaghen, Denmark