DogOnt - Ontology Modeling for Intelligent Domotic Environments

Dario Bonino and Fulvio Corno

Politecnico di Torino, Torino, Italy {dario.bonino, fulvio.corno}@polito.it

Abstract. Home automation has recently gained a new momentum thanks to the ever-increasing commercial availability of domotic components. In this context, researchers are working to provide interoperation mechanisms and to add intelligence on top of them. For supporting intelligent behaviors, house modeling is an essential requirement to understand current and future house states and to possibly drive more complex actions. In this paper we propose a new house modeling ontology designed to fit real world domotic system capabilities and to support interoperation between currently available and future solutions. Taking advantage of technologies developed in the context of the Semantic Web, the DogOnt ontology supports device/network independent description of houses, including both "controllable" and architectural elements. States and functionalities are automatically associated to the modeled elements through proper inheritance mechanisms and by means of properly defined SWRL auto-completion rules which ease the modeling process, while automatic device recognition is achieved through classification reasoning.

1 Introduction

Domotic systems, also known as "home automation" systems, have been around on the market for several years, however only few years ago they started to spread over residential buildings, thanks to the increasing availability of low cost devices and driven by new emerging needs on house comfort, energy saving, security, communication and multimedia services.

Current domotic solutions suffer from two main drawbacks: they are produced and distributed by various electric component manufacturers, each having different functional goals and marketing policies; and they are mainly designed as an evolution of traditional electric components (such as switches and relays), thus being unable to natively provide intelligence beyond simple automation scenarios. The first drawback causes an evident interoperation problem that prevents different domotic plants or components to interact with each other, unless specific gateways or adapters are used. While this was acceptable in the first evolution phase, where installations were few and isolated, now it becomes a very strong issue as many large buildings such as hospitals, hotels and universities are mixing different domotic components, possibly realized with different technologies, and need to coordinate them as a single system. On the other hand, the roots of

domotic systems in simple electric automation prevent satisfying the current requirements of home inhabitants, who are becoming more and more accustomed to technology, requiring more complex interaction possibilities.

In the literature, solutions to these issues are usually proposed by defining *smart homes*, i.e., homes pervaded by sensors and actuators and equipped with dedicated hardware and software tools that implement intelligent behaviors. Smart homes have been actively researched since the late 90's, pursuing a revolutionary approach to the home concept, from the design phase to the final deployment. Involved costs are very high and prevented, until now, a real diffusion of such systems, that still retain an experimental and futuristic connotation.

The approach proposed in this paper lies somewhat outside the smart home concept, and is based on extending domotic systems, by adding devices and agents for supporting interoperation and intelligence. Our solution takes an evolutionary approach, where commercial domotic systems are extended with a low cost device (embedded PC) allowing interoperation and supporting more sophisticated automation scenarios. In this case, the domotic system in the home evolves into a more powerful integrated system, that we call Intelligent Domotic Environment (IDE), that is able to learn user habits, to provide automatic and proactive security, to implement comfort and energy saving policies and can be immediately exploited, as technologies are low cost and commercially available. IDEs promise to achieve intelligent behaviors comparable to smart homes, at a fraction of the cost, by reusing and exploiting available technology, and providing solutions that may be deployed even today.

A key step towards the definition of IDEs is abstract and formal modeling of domotic device capabilities and functionalities, independently from technology specific aspects. For example a lamp is an object that can be electrically lit and that emits light, independently from the technology with which it is built, provided it is controllable in some way by the domotic system. Abstraction allows to bridge different technologies by associating real devices with their abstract counterparts and by translating low level information into a common, shared language.

This paper introduces DogOnt, a novel modeling language for IDEs, based on Semantic Web technologies. By adopting well known representations such as ontologies and by providing suitable reasoning facilities, DogOnt is able to face interoperation issues allowing to describe:

- where a domotic device is located;
- the set of capabilities of a domotic device;
- the technology-specific features needed to interface the device;
- the possible configurations that the device can assume;
- how the home environment is composed;
- what kind of architectural elements and furniture are placed inside the home.

This information can then be leveraged by inference-based intelligent systems to provide advanced functionality required in Intelligent Domotic Environments. DogOnt is composed of two elements: the *DogOnt ontology*, expressed in OWL, which allows to formalize all the aspects of a IDE, and the *DogOnt rules*, which

ease the modeling process by automatically generating proper states and functionalities for domotic devices, and by automatically associating them to the corresponding device instances through semantic relationships. DogOnt is currently adopted to provide house modeling and reasoning capabilities to a domotic gateway called DOG (Domotic OSGi Gateway), which is under development in the authors' research group and that will be distributed as open source toolkit for building IDEs running on low cost PCs. In this context, a third component of DogOnt, namely *DogOnt queries*, not presented in this paper, supports runtime control of the IDE.

The paper is organized as follows: Section 2 introduces relevant related works, while Section 3 describes the DogOnt ontology, starting from the initial assumptions and including the most interesting modeling aspects. Section 4 shows DogOnt rules, i.e., how reasoning mechanisms can be used to ease device modeling and to decouple modeled environments from model evolutions. Section 5 finally provides final remarks and proposes future works.

2 Related Works

Modeling domotic environments through ontologies or taxonomies is an interesting field, but the amount of available literature is very limited. The main contributions are the EHS taxonomy[1] and DomoML [1]. Besides, interesting and complementary works have been done on pervasive and ubiquitous computing modeling [2] and for context representation in ambient intelligence environments [3,4,5].

The EHS taxonomy is a home appliance classification system designed by the EHS (European Home System) consortium (now evolved in the Konnex alliance[2]) that mainly describes so-called white and brown goods located in a domestic environment. It is deployed along four main classes: Meter Reading, which groups all measurement tools, House Keeping, which groups all household appliances and systems, Audio and Video, which encompasses multimedia appliances, and Telecommunication, grouping all tools able to establish a communication. This simple taxonomy has several drawbacks that prevent effective house modeling: first, it takes a somewhat incoherent modeling approach, as overlapping classes are represented as different branches in the taxonomy (thus implying un-existent disjointness unless classification under multiple branches is allowed). Second, it doesn't support non-taxonomic relationships between objects and does not address function and state modeling. Third, it does not deal with representation of appliance functions, capabilities and type of permitted operations, only allowing simple, static description of environments, without any formal notion on operating capabilities of modeled entities.

DomoML [1,6] provides a full, modular ontology for representing household environments. It describes operational and functional aspects together with some preliminary architectural and positioning information and is based on three core

[1] The European Home System, `http://www.ehsa.com`

[2] `http://www.konnex.org`

ontologies: DomoML-env, DomoML-fun and DomoML-core. DomoML-env provides primitives for the description of all "fixed" elements inside the house such as walls, furniture elements, doors, etc., and also supports the definition of the house layout by means of neighborhood and composition relationships. DomoML-fun provides means for describing the functionalities of each house device, in a technology independent manner. DomoML-core provides support for the correlation of elements described by DomoML-env and DomoML-fun constructs, including the definition of proper physical quantities. DomoML shows some shortcomings when applied to real-world domotic systems. As first, it mixes too different levels of detail in modeling. This implies, on one side, over-specification, e.g., to define that a lamp can be lit, a modeler has to describe the lamp, the attached switch button, down to the single lever. On the other side, it does not address state modeling and doesn't provide facilities to query or auto-complete models, thus requiring a cumbersome modeling effort whenever a new house must be described.

In the context of pervasive computing, the SOUPA ontology [2] provides a modular modeling structure that encompasses vocabularies for representing intelligent agents, time, space, events, user profiles, actions and policies for security and privacy. SOUPA is organized into a core set of vocabularies, and a set of optional extensions. Core vocabularies describe concepts associated with person, agent, belief-desire-intention, action policy, time, space and event. These concepts are expressed as 9 distinct ontologies aligned to well known vocabularies such as FOAF [7], DAML Time [8], OpenCyc [9] and RCC [10]. SOUPA cannot be directly applied to support interoperability and intelligence for domotic systems as many domain-specific concepts are lacking (e.g., no primitives are provided for modeling devices, functionalities, etc.), however it can be useful in a multilayered approach, where DogOnt provides domain-specific, operative knowledge and SOUPA allows modeling high level, pervasive concepts, easing the implementation of intelligent behaviors on top of it.

3 DogOnt ontology

The DogOnt ontology is designed with a particular focus on interoperation between domotic systems. Base assumptions are directly driven by real-world case studies [11], mainly focusing on device, state and functionality modeling. DogOnt (whose features are reported in Table 1) is deployed along 5 main hierarchy trees (Figure 1):

- *Building Thing*: modeling available things (either controllable or not);
- *Building Environment*: modeling where things are located;
- *State*: modeling the stable configurations that controllable things can assume;
- *Functionality*: modeling what controllable things can do;
- *Domotic Network Component*: modeling features peculiar of each domotic plant (or network).

Table 1. DogOnt ontology statistics.

Feature	**Value**
Expressivity	$ALCHOIN(D)$
Named Classes	167
Number of Siblings per Node (mean)	5
Restrictions	126
Universal restrictions	21
Cardinality restrictions	54
hasValue restrictions	41
Object properties	18
Datatype properties	26

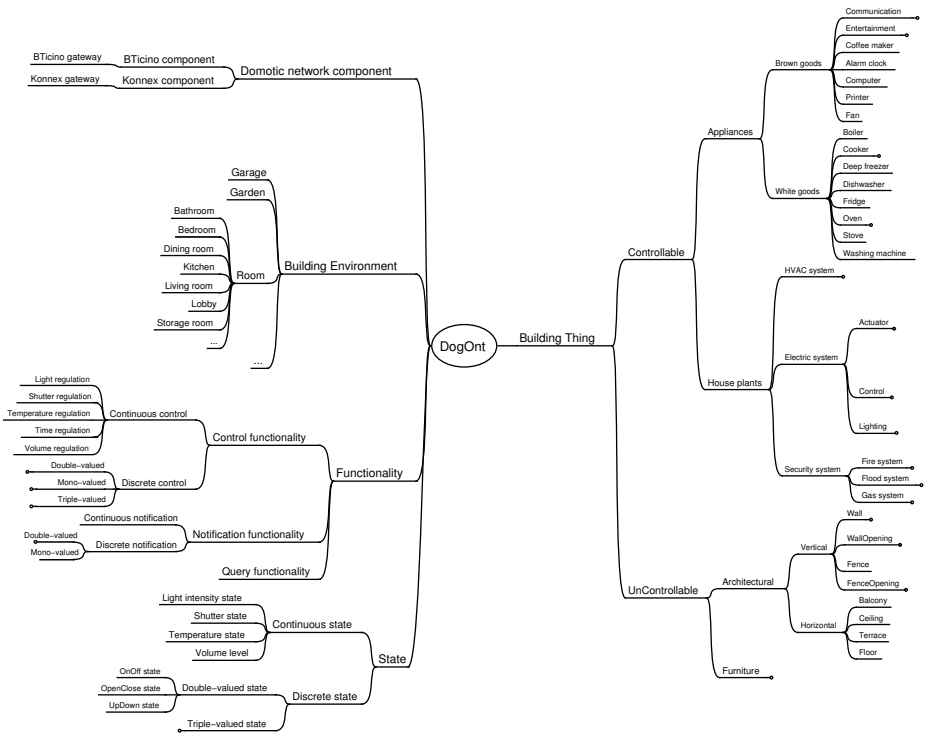

Fig. 1. An overview of the DogOnt ontology.

3.1 Environment modeling

Environment modeling is achieved by means of the DogOnt concepts inheriting from *Building Environment* and from *Building Thing* (Figure 1), both already defined in DomoML [1] but differently formalized in DogOnt to overcome the limitations in IDE modeling described in section 2. Modeling detail is limited to the minimal set of primitives needed to locate domotic components, furniture elements and appliances inside a single flat or living unit. Entire buildings can be represented by extending this section of the ontology through subclassing of *Building Environment* and through the definition of proper relationships (e.g. by introducing principles from spatial modeling and reasoning [12,13]).

The *BuildingEnvironment* tree supports a coarse representation of domestic environments, as whole architectural units, including: several types of *Room*, the *Garage* and the *Garden*. The *BuildingThing* tree, instead, represents all the elements that can be located or that can take part in the definition of a *BuildingEnvironment*. DogOnt defines a clear separation between objects that can be controlled by a domotic system (*Controllable* class) and all the other objects that can be found in a home (*UnControllable* class); they are explicitly modeled as disjoint classes.

Controllable objects can be appliances or can belong to house plants such as the HVAC[3] plant. Appliances are modeled through the homonymous class and are further subdivided in *White Goods* and *Brown Goods*, according to the EHS taxonomy. *House plants* include *HVAC systems*, *electric systems* and *security systems*. They differ from appliances as they are usually installed in fixed positions, and they encompass several components that must be coordinated to reach a specific goal (e.g., delivering electrical power).

Uncontrollable objects are all the home components that cannot be directly controlled by a domotic system. They are mainly subdivided in *Furniture* and *Architectural* elements. *Furniture* models all the elements usually adopted as furniture like chairs, cupboards, desks, etc. Instead, *Architectural* objects model all the elements that define a living environment such as *Wall*s, *Floor*s, etc. They are mainly grouped in *Vertical* and *Horizontal* elements, which are further subdivided in subclasses. Architectural modeling is somewhat limited to simple partitions (walls, floors) and openings (windows, doors) and may be extended for implementing advanced modeling, e.g., to support architectural design.

3.2 Device modeling

All the objects referred to as "device," in this paper, are objects belonging to the *Controllable* sub-tree. A controllable object differs from an uncontrollable one as it must satisfy several restrictions on *Functionalities* and *State*s. It must possess at least the functionality of being queried about its operative condition (*QueryFunctionality*) and it must possess a state, intended as the ability of reaching a stable configuration identifiable in some way: a lamp is able to be

[3] Heating, Ventilating, and Air Conditioning

steadily on or off, a flashing light can be on, off or flashing, a shutter can be moving up, down or being steady, etc.

Example (part 1): We consider a dimmer lamp connected to a KNX network, and located in the living room, as sample device. On the basis of formalization defined until now, our dimmer lamp is an instance of the class *Dimmer Lamp*, which in turn is a *Lamp*, a *Lighting*, an *Electric System*, a *HousePlant* and a *Controllable* object. The corresponding OWL formalization fragment is reported in Figure 2. As can be easily noticed, while the position inside the house is explicitly modeled, as well as the fact that our dimmer lamp is actually connected to a KNX network, and can be controlled by a switch. The inheritance of characteristics from ancestors such as *Lighting* or *Controllable* is left to a simple reasoning step, where the transitive closure of the model is computed and properties are propagated along the ontology *isA* relations.

```
<DimmerLamp rdf:ID="sample_dimmer_lamp">
  <isIn rdf:resource="#sample_living_room"/>
  <rdf:type rdf:resource="#KNXNetworkComponent"/>
  <individualAddress>101</individualAddress>
  <groupAddress>12</groupAddress>
  <hasControl rdf:resource="#switch_sample_dimmer_lamp"/>
</DimmerLamp>
```

Fig. 2. Example (part 1) - representation of a sample DimmerLamp instance.

3.3 Functionality modeling

Each device class, in DogOnt, is associated to a set of different functionalities, by means of the *hasFunctionality* relationship. Several approaches can be chosen for functionality modeling: a compositional approach, as in DomoML, where the functionalities of a given object derive from the composition of functionalities provided by its components, or a descriptive model, where functionalities are described apart and then associated to the single devices. DogOnt takes this last approach and models functionalities by objectives and by variation modality. This allows to use only one instance per functionality, as device capabilities are modeled independently from device classes.

Each functionality defines the commands to modify a given device property (e.g., light intensity) and the values they can assume. Functionalities are divided in different classes on the basis of their goals: *Control Functionalities* model the ability to control a device or a part of it, e.g., to open up a shutter. *Notification Functionalities* represent the ability of a device to autonomously advertise its internal state and in particular the ability of detecting and signaling state changes. *Query Functionalities* encompass the capabilities of a device to be queried, or polled, about its condition, e.g., failure, internal state values, etc.

Functionalities are also modeled according to the way they modify the internal state of a given device; two main variation families are provided: *Continuous*

Functionalities that allow to change device characteristics (e.g., the light intensity) in a continuous manner, between a minimum and a maximum value, and *Discrete Functionalities* that only allow abrupt changes of device properties, e.g., to switch a light on. Most domotic devices such as switches, plugs and lights can be controlled by means of only 2 or 3 different commands (e.g., on-off, open-close, up-down-rest, etc.) while functionalities controlled by more than 3 commands are rare. Reflecting this situation, the DogOnt ontology explicitly models *Discrete Functionalities* as mono-, double- and triple-valued functionalities. Clearly, these 3 subclasses do not define the complete universe of possible discrete functionalities, therefore multi-valued capabilities can be modeled by directly instantiating the *Discrete Functionality* class.

Example (part 2): Let us re-consider the sample dimmer lamp definition started in section 3.2; the capabilities of the lamp can be modeled by means of proper functionalities. Our dimmer lamp, being an instance of the class *Dimmer Lamp*, is connected to the unique *LightRegulationFunctionalityInstance* (continuous) defined in DogOnt, which models the ability to dim the emitted light. As *DimmerLamp* is a subclass of *Lamp*, it inherits the related *OnOffFunctionalityInstance* (discrete) modeling the capability to switch on and off the lamp. Finally, a *Lamp* is a specific subtype of *Controllable* to which is associated the *QueryFunctionalityInstance* representing the capability of the lamp to be queried about its characteristics (e.g., the dimming level). Figure 3 shows the corresponding OWL excerpt.

```
<owl:Class rdf:ID="DimmerLamp">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  Lamp that varies the level of illumination</rdfs:comment>
<owl:equivalentClass>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:hasValue>
          <LightRegulationFunctionality
           rdf:ID="LightRegulationFunctionalityInstance"/>
        </owl:hasValue>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasFunctionality"/>
        </owl:onProperty>
      </owl:Restriction>
      <owl:Class rdf:about="#Lamp"/>
    </owl:intersectionOf>
  </owl:Class>
</owl:equivalentClass>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
 DimmerLamp</rdfs:label>
</owl:Class>
```

Fig. 3. The definition of DimmerLamp with associated functionalities.

3.4 State modeling

States are modeled following the same descriptive approach adopted for functionalities; they must be instantiated for each home device instance since different

devices belonging to the same conceptual class, e.g., *Lamp*, can be in different conditions, e.g., on or off.

States are classified according to the kind of values they can assume: continuously changing qualities are modeled as *Continuous States* with an associated *continuousValue* datatype property of type `xsd:float`. Instead, qualities that can only assume discrete values (e.g., On/Off, Up/Down, etc.) are classified as *Discrete States*. Discrete states are subdivided in double- and triple-valued states, while states having more than 3 stable configurations are modeled by directly instantiating the *Discrete State* concept (see Figure 1 for the complete hierarchy).

Discrete states are characterized by the *valueDiscrete* datatype property that describes the current state and by a variable set of *possibleStates* (datatype property) that models all the possible values that *valueDiscrete* can assume for the current state type. Figure 4 reports the definition of the *OnOffState* typically associated to all *Lamp* instances, the *valueDiscrete* property is defined in the *Discrete State* class and inherited by all subclasses.

```
<owl:Class rdf:about="#OnOffState">
 <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  State: on - off</rdfs:comment>
 <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  OnOffState</rdfs:label>
 <rdfs:subClassOf rdf:resource="#DoubleValuedState"/>
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty>
       <owl:DatatypeProperty rdf:about="#possibleStates"/>
     </owl:onProperty>
     <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      On</owl:hasValue>
   </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Off</owl:hasValue>
     <owl:onProperty>
       <owl:DatatypeProperty rdf:about="#possibleStates"/>
     </owl:onProperty>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Fig. 4. The definition of the OnOffState.

Example (part 3): Having defined state modeling, the sample dimmer lamp instance can now be completely defined (Figure 5). As can easily be noticed, many properties of this simple object are not explictly modeled, e.g., its associated functionalities and the relative commands, but need to be deduced by performing a simple reasoning step that computes the transitive closure of the model, turning implicit knowledge into explicit information (Figure 6).

3.5 Network modeling

Interoperation between domotic systems requires the definition of a technology independent formalization that allows to operate seamlessly with different de-

```
<DimmerLamp rdf:ID="sample_dimmer_lamp">
 <isIn rdf:resource="#sample_living_room"/>
  <rdf:type rdf:resource="#KNXNetworkComponent"/>
  <individualAddress>101</individualAddress>
  <groupAddress>12</groupAddress>
  <hasControl rdf:resource="#switch_sample_dimmer_lamp"/>
  <hasState>
     <LightIntensityState
       rdf:ID="DimmerLamp_Livingroom1_LightIntensityState"/>
   </hasState>
</DimmerLamp>
```

Fig. 5. The OWL definition of the sample DimmerLamp instance.

vices, produced by different manufacturers and operating in plants with different technologies. A minimum set of plant-dependent knowledge must, however, be available for enabling interoperation systems (gateways) to interact with physical devices. DogOnt models such information by means of an ontology branch stemming from the *Domotic Network Component* concept.

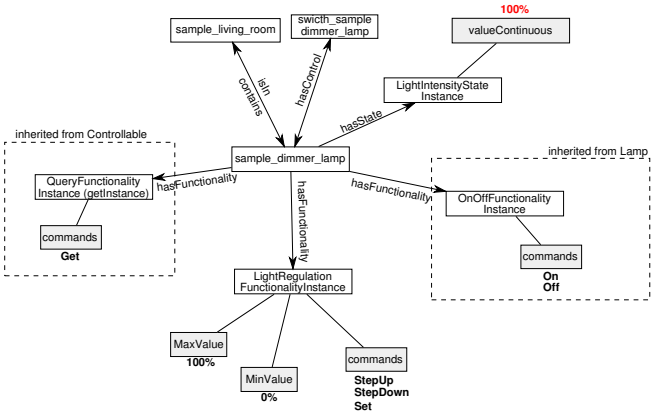

Fig. 6. The complete definition of the sample DimmerLamp instance, with inherited properties.

Every controllable object belonging to a domotic plant is modeled as an instance of a specific controllable concept (e.g., a *Lamp*) and, at the same time, as an instance of a proper *Network Component*. Currently 2 network components are already modeled: the *KNXNetworkComponent*, representing KNX-compliant devices and the *BTicinoNetworkComponent*, representing BTicino MyHome devices.

No subclasses are required except for network-level gateways that need more fine grained descriptions to model features such as IP addresses, polling intervals, etc. (see Table 2 for a complete reference). These features are needed by integration systems to interface domotic networks, thus enabling interoperation.

Table 2. Specific features of network-level gateways.

Gateway-specific property	BTicino	KNX
connection timeout	x	x
connection trials before failure	x	x
IP address	x	x
port	x	x
sleeptime	x	x
multicast address	-	x
UDP port	-	x
polling interval	-	x

4 DogOnt - rules

DogOnt provides different reasoning mechanisms responding to different goals: to ease model instantiation, to verify the formal correctness of model instantiations (consistency checking) and to support automatic recognition of device instances from their features (i.e., to support scalability and model evolution):

Model instantiation is a relatively complex task that requires generating device, and state, instances, and to properly link them by means of relationships. Due to the many modeling aspects considered in DogOnt, this process can be quite difficult and error prone. Therefore, a suitable set of autocompletion rules has been defined, which allows, with a single deduction step, to automatically create states and relationships associated to specific type of devices.

Consistency checking allows to verify the formal correctness of generated model instantiations, ensuring correct operation of systems built on top of them.

Classification reasoning is used to automatically recognize device classes, starting from device functional descriptions. This allows to decouple the DogOnt model evolution from model instantiation, thus enabling systems to operate with unknown device classes, on one side, and to automatically re-classify existing devices with respect to new classes defined in forecoming DogOnt model versions, on the other side.

4.1 Rule-based model instantiation

DogOnt represents each device as an object having a given set of functionalities and states (Section 3). Functionalities are automatically added to every device instance by means of suitable restrictions defined at the class level. They are shared by all devices of the same class. On the contrary, states are peculiar of each device.

Manually creating and associating states and devices is absolutely tedious for designers as it is repetitive and can lead to modeling errors and/or inconsistencies (this also happens in the approach taken by DomoML). Thanks to

the repetitive nature of the operation, the process can be easily automated. Rule-based reasoning can, in fact, generate the needed instances and links in an automatic and verified manner.

Several rule languages can be applied to the DogOnt ontology; among them, SWRL [14] appears the most suitable solution as it allows to directly embed rules in the DogOnt ontology (SWRL constructs can, in fact, be expressed in OWL). SWRL is a powerful rule language based on Horn-like rules [15], that guarantees decidability in finite time. The most interesting feature of SWRL is the ability to provide/define so-called built-in operators, that can implement non-logic operations such as mathematic calculations, string functions, etc. Built-ins [14] can either be customly built or can belong to standard sets defined in SWRL-B[4] and SWRL-X[5] libraries. SWRL-X, in particular, provides class/instance generation built-ins that, combined with string elaboration built-ins and proper modeling, allow to autocomplete states for device instances. Figure 7 shows a sample SWRL rule for attaching state instances to *SimpleLamp* instances. It must be noticed that to generate human-readable labels and identifiers for SWRL-created states, instances must have a human-readable `rdfs:label`.

```
SimpleLamp(?x)^rdfs:label(?x,?y)^
swrlb:stringConcat(?z,?y,"_OnOffState")^
swrlx:createOWLThing(?w,?z)->
OnOffState(?w)^rdfs:label(?w,?z)^hasState(?x,?w)
```

Fig. 7. The auto-completion rule for SimpleLamp states.

SWRL rules can be executed by any SWRL-compliant rule engine. In Protégé [16], for example, the SWRLTab allows to use the Jess[6] rule engine to execute SWRL statements, embedding the newly generated knowledge in the active model. In this way, during house modeling, a domotic designer can instantiate the needed devices without caring of states. Then, she can run the Jess engine with the DogOnt SWRL rules obtaining as result a complete model. Consistency problems are completely avoided for state modeling as rules are predefined and a priori validated.

4.2 Classification reasoning

Classification reasoning is a type of automated inference that allows to infer the class(es) to which an instance belongs, by checking its properties against the set of necessary and sufficient conditions that define class membership. These conditions are usually adopted in consistency checking to ensure that class instances

[4] `http://www.daml.org/rules/proposal/builtins.html`
[5] `http://swrl.stanford.edu/ontologies/built-ins/3.3/swrlx.owl`
[6] The Jess rule engine, `http://www.jessrules.com/`

respect the restrictions defined on properties for individuals belonging to a given class. For example, a *DimmerLamp* instance must have a *LightRegulationFunctionality*, must only possess one *LightIntensityState* and must be a *Lamp*. Every asserted *DimmerLamp* instance respecting these constraints is valid. This kind of reasoning can also be used to discover new class memberships, i.e., to infer instance types: in a sample scenario a given domotic system provides a device able to variate light intensity. The home modeler does not know the existence of the *DimmerLamp* class, and decides to model the device as a *Lamp* instance, with a *LightIntensityState* and a *LightRegulationFunctionality*. Classification reasoning allows to discover that the modeled device is actually a *DimmerLamp*, and can therefore be treated in the way asserted *DimmerLamp*s are.

Classification reasoning is a fundamental part of formal modeling of home environments for domotic interoperability. In fact, as manufacturers are always adding new features to their networks and new technologies are emerging too, house models frequently become un-synchronized with the actual environment configuration or capabilities. Being able to automatically discover new classes for already defined instances allows to easily extend/amend the DogOnt ontology without risking to disrupt existing models.

5 Conclusions

In this paper we introduce DogOnt, a new modeling approach for domotic environments composed of the DogOnt ontology and a set of DogOnt rules. DogOnt provides functionality and state auto-completion, and supports model evolution through classification reasoning. Modeling is done at a detail level that reflects the actual needs of interoperation between real-world domotic systems and supports the development of Intelligent Domotic Environments. Architectural modeling is also provided, although in a very limited, but extensible form. Novelty points include the descriptive modeling approach, more flexible than approaches available in the literature, and the definition of auto-completion mechanisms through reasoning, which eases the house modeling process.

Formal modeling of house environments through ontology-based technologies is a promising research stream for domotic systems and smart homes. Ontologies allow, on one side, to achieve a natural abstraction of networks and devices that can be used for supporting interoperability. On the other hand, well studied reasoning techniques can both ease the modeling process and provide support for the implementation of complex, intelligent behaviors inside domotic homes.

The authors are currently working on the design and implementation of DOG, a domotic gateway based on DogOnt and OSGi, on structural checks, and on verification of safety properties through rule-based reasoning on DogOnt.

References

1. Lorenzo Sommaruga, Antonio Perri, and Francesco Furfari. DomoML-env: an ontology for Human Home Interaction. In *Proceedings of SWAP 2005, the 2nd Italian*

Semantic Web Workshop, Trento, Italy, December 14-16, 2005, CEUR Workshop Proceedings, 2005.
2. Harry Chenand Tim Finin and Anupam Joshi. *Ontologies for Agents: Theory and Experiences*, chapter The SOUPA Ontology for Pervasive Computing, pages 233–258. Birkhäuser Basel, Los Alamitos, CA, USA, 2005.
3. 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.
4. Daqing Zhang, Tao Gu, and Xiaohang Wang. Enabling Context-aware Smart Home with Semantic Technology. *International Journal of Human-friendly Welfare Robotic Systems*, 6(4), 2005.
5. Anders Kofod-Petersen and Agnar Aamodt. Contextualised Ambient Intelligence through Case-Based Reasoning. In T. R. Roth-Berghofer, Mehmet H. Göker, and H. Altay Güvenir, editors, *Proceedings of the Eighth European Conference on Case-Based Reasoning (ECCBR 2006)*, volume 4106 of *Lecture Notes in Computer Science*, pages 211–225, Ölüdeniz, Turkey, September 2006. Springer Verlag.
6. 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.
7. Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.91. http://xmlns.com/foaf/spec/, November 2007.
8. Jerry Hobbs and James Pustejovsky. Annotating and Reasoning about Time and Events. In *2003 AAAI Spring Symposium*, 2003.
9. Stephen L. Reed and Douglas B. Lenat. Mapping Ontologies into Cyc. In *AAAI 2002 symposium*, 2002.
10. Sanjiang Li and Mingsheng Ying. Region Connection Calculus: its models and composition table. *Artificial Intelligence*, 145(1-2):121–146, 2003.
11. Paolo Pellegrino, Dario Bonino, and Fulvio Corno. Domotic House Gateway. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1915–1920, New York, NY, USA, 2006. ACM.
12. J. Renz and B Nebel. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the Region Connection Calculus. *Artificial Intelligence*, 108:69–123, 1999.
13. J. Renz and G Ligozat. Weak Composition for Qualitative Spatial and Temporal Reasoning. In *Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05)*, 2005.
14. Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.daml.org/2003/11/swrl/`, November 2003.
15. Alfred Horn. On Sentences Which are True of Direct Unions of Algebras. *The Journal of Symbolic Logic*, 16:14–21, 1951.
16. John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubezy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *Int. Journal of Human-Computer Studies*, 58(1):89–123, 2003.