Uniform Access to Domotic Environments through Semantics

Dario Bonino, Emiliano Castellina, and Fulvio Corno

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

Abstract. This paper proposes a Domotic OSGi Gateway (DOG) able to expose different domotic networks as a single, technology neutral, home automation system. The adoption of a standard framework such as OSGi, and of sophisticated modeling techniques stemming from the Semantic Web research community, allows DOG to go beyond simple automation and to support reasoning-based intelligence inside home environments. By exploiting the DogOnt ontology for automatic device generalization, syntactic and semantic command validation, and internetwork scenario definition, DOG provides the building blocks for evolving current, isolated, home automation plants into so-called Intelligent Domotic Environments, where heterogeneous devices and domotic systems are coordinated to behave as a single, intelligent, proactive system. The paper introduces the DOG architecture by looking at functionalities provided by each of its components and by describing features that exploit ontology-modeling.

1 Introduction

Domotic systems, also known as home automation systems, have been available on the market for several years, however only in the last few years they started to spread also 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. Proliferation of technologies and communication protocols causes interoperation problems that prevent different domotic plants or components to interact with each other, unless specific gateways or adapters are used. The roots of domotic systems in simple electric automation, on the other hand, prevent satisfying the current requirements of home inhabitants, who are becoming more and more accustomed to technology and require more complex interaction possibilities.

In the literature, solutions to these issues usually propose *smart homes* [1], i.e., homes pervaded by sensors and actuators and equipped with dedicated hardware and software tools that implement intelligent behaviors. 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 current domotic systems, by adding hardware devices and software agents for supporting **interoperation** and **intelligence**. Our solution takes an *evolutionary* approach, in which commercial domotic systems are extended with a low cost device (embedded PC) allowing interoperation and supporting more sophisticated automation scenarios.

Currently available home-automation solutions rely on a hardware component called residential [2] or home gateway [3] originally conceived for providing Internet connectivity to smart appliances available in a given home. This component, in our approach, is evolved into an interoperation system, called DOG (Domotic OSGi Gateway), where connectivity and computational capabilities are exploited to bridge, integrate and coordinate different domotic networks. DOG exploits OSGi as a coordination framework for supporting dynamic module activation, hot-plugging of new components and reaction to module failures. Internal and external operations are based on the DogOnt ontology and allow supporting services that go beyond simple automation, enabling reasoning-based intelligence inside domotic environments. Internal ontology-powered operations include automatic device generalization, syntactic and semantic command validation and support the definition of inter-network scenarios. External applications can exploit DogOnt-based operations to implement sophisticated reasoning on home properties, either off-line or on-line, by interacting with DOG in real-time.

Cost and flexibility concerns take a significant part in the platform design and we propose an open-source solution capable of running on low cost hardware systems such as the ASUS eeePC 701.

The paper is organized as follows: in Section 2 the DOG platform is described, starting from high-level design issues and including the description of platform components and their interactions, while in Section 3 ontology-driven tasks in DOG are described and finally Section 4 draws conclusions.

2 DOG Architecture

DOG is a domotic gateway designed to respond to different requirements, ranging from simple bridging of network-specific protocols to complex interaction support. These requirements can be attributed to 3 priority levels: level 1 priorities include all the features needed to control different domotic systems using a single, high-level, communication protocol and a single access point, level 2 priorities define all the functionalities needed for defining inter-network automation scenarios and to allow inter-network control, e.g., to enable a Konnex switch to control an OpenWebNet light, and level 3 requirements are related to intel-

ligent behaviors, to user modeling and to adaptation. Table 1 summarizes the requirements, grouped by priority.

Table 1. Requirements for Home Gateways in Intelligent Domotic Environments.

Priority	Requirement	Description
R1 Interoperability	R1.1 Domotic network connection	Interconnection of several domotic networks.
	R1.2 Basic Interoperability	Translation / forwarding of messages across different networks.
	R1.3 High level network protocol	Technology independent, high-level network protocol for allowing neutral access to domotic networks.
	R1.4 API	Public API to allow external services to easily interact with home devices.
R2 Automation	R2.1 Modeling	Abstract models to describe the house devices, their states and functionalities, to support effective user interaction and to provide the basis for home intelligence.
	R2.2 Complex scenarios	Ability to define and operate scenarios involving different networks / components.
R3 Intelligence	R3.1 Offline Intelligence	Ability to detect misconfigurations, structural problems, security issues, etc.
	R3.2 Online Intelligence	Ability to implement runtime policies such as energy saving or fire prevention.
	R3.3 Adaptation	Learning of frequent interaction patterns to ease users' everyday activities.
	R3.4 Context based Intelligence	Proactive behavior driven by the current house state and context aimed at reaching specific goals such as safety, energy saving, robustness to failures.

Design principles include versatility, addressed through the adoption of an OSGi based architecture, advanced intelligence support, tackled by formally modeling the home environment and by defining suitable reasoning mechanisms, and accessibility to external applications, through a well defined, standard API also available through an XML-RPC [4] interface.

DOG is organized in a layered architecture with 4 **rings**, each dealing with different tasks and goals, ranging from low-level interconnection issues to high-level modeling and interfacing (Figure 1). Each ring includes several OSGi bundles, corresponding to the functional modules of the platform.

Ring 0 includes the DOG common library and the bundles necessary to control and manage interactions between the OSGi platform and the other DOG bundles. At this level, system events related to runtime configurations, errors or failures, are generated and forwarded to the entire DOG platform. Ring 1 encompasses the DOG bundles that provide an interface to the various domotic networks to which DOG can be connected. Each network technology is managed by a dedicated driver, similar to device drivers in operating systems, which abstracts network-specific protocols into a common, high-level representation that allows to uniformly drive different devices (thus satisfying requirement R1.1). Ring 2 provides the routing infrastructure for messages travelling across network

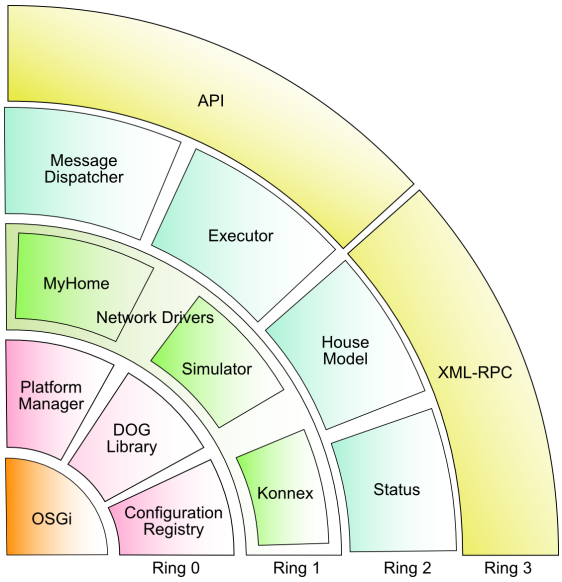

Fig. 1. DOG architecture.

drivers and directed to DOG bundles. Ring 2 also hosts the core intelligence of DOG, based on an abstract formal model of the domotic environment (**DogOnt ontology** [5]), that is implemented in the House Model bundle (R1.2, R1.3, R2.1 and, partially, R2.2). Finally, ring 3 hosts the DOG bundles offering access to external applications, either by means of an API bundle, for OSGi applications, or by an XML-RPC endpoint for applications based on other technologies (R1.4).

In the following subsections each DOG bundle is described in more detail, focusing on provided services and functionalities; particular emphasis is given to the House Model bundle that actually implements the DOG core intelligence.

2.1 Ring 0

DOG library This bundle acts as a library repository for all other DOG bundles.

Platform manager This bundle handles the correct start-up of the whole system and manages the life cycle of DOG bundles. The platform manager coordinates bundle activations, enforcing the correct start-up order, and manages bundle errors.

Configuration Registry The Configuration Registry implements the *Configurator* interface by maintaining and exporting bundle configuration parameters.

2.2 Ring 1

Network Drivers In order to interface domotic networks, DOG provides a set of Network Drivers, one per each different technology (e.g., KNX, OpenWebNet,

X10 [6], etc.). Every network driver implements a "self-configuration" phase, in which it interacts with the House Model (through the HouseModeling interface) to retrieve the list of devices to be managed, together with a description of their functionalities, in the DogOnt format. Every device description carries all the needed low-level information like the device address, according to the network dependent addressing format (simple in OpenWebNet, subdivided in group and individual addresses in KNX, etc.).
Currently three Network Drivers have already been developed: Konnex, OpenWebNet and Simulator. Konnex and OpenWebNet drivers translate DogMessages into protocol messages of the corresponding networks. The Simulator driver, instead, emulates a synthetic environment by either generating random events or by re-playing recorded event traces (this allows to simulate critical situations in a safe environment).

2.3 Ring 2

Message Dispatcher This bundle acts as an internal router, delivering messages to the correct destinations, be they the network drivers (commands and state polls) or other DOG bundles (notifications). Routing is driven by a routing table where network drivers are associated to high-level device identifiers, enabling DOG to deliver commands to the right domotic network.
Executor The Executor validates commands received from the API bundle, either directly or through the XML-RPC protocol. Commands are syntactically validated, by checking the relation between the DogMessage type declaration and the DogMessage content, and semantically validated, by checking them against the set of commands modeled by the HouseModel ontology. If all checks are passed, the Executor forwards messages to the Message Dispatcher for the final delivery. Otherwise messages are dropped, avoiding to generate a platform inconsistent state.
Status The Status bundle caches the states of all devices controlled by DOG by listening for notifications coming from network drivers. This state cache is extensively used in DOG to reduce network traffic on domotic busses, and to filter out un-necessary commands, e.g., commands whose effects leave the state of the destination devices unchanged.
House Model The House Model is the core of intelligence of the DOG platform. It is based on a formal model of the house defined by instantiating the **DogOnt ontology** [5]. DogOnt is a OWL meta-model for the domotics domain describing where a domotic device is located, the set of its capabilities, the technology-specific features needed to interface it, and the possible configurations it can assume. Additionally, it models how the home environment is composed and what kind of architectural elements and furniture are placed inside the home.

It is organized along 5 main hierarchy trees (Figure 2), including: *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; and *Domotic Network Component*, modeling peculiar

features of each domotic plant (or network). The *BuildingThing* tree subsumes

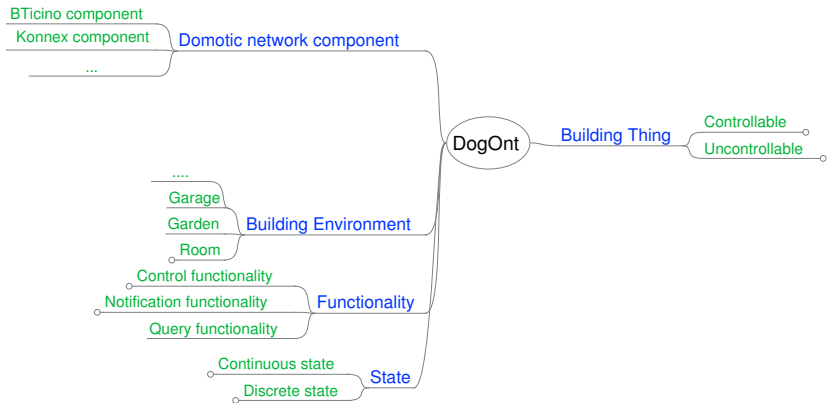

Fig. 2. An overview of the DogOnt ontoloy

the *Controllable* concept and its descendants, which are used to model devices belonging to domotic systems or that can be controlled by them.

Devices are described in terms of capabilities (*Functionality* concept) and possible configurations (*State* concept). Functionalities are mainly divided in *Continuous* and *Discrete*, the former describing capabilities that can be variated continuously and the latter referring to the ability to change device configurations in a discontinuous manner, e.g., to switch on a light. In addition they are also categorized depending on their goal, i.e. if they allow to control a device (*Control Functionality*), to query a device condition (*Query Functionality*) or to notify a condition change (*Notification Functionality*). Each functionality instance defines the set of associated commands and, for continuous functionalities, the range of allowed values, thus enabling runtime validation of commands issued to devices. Devices also possess a state instance deriving from a *State* subclass, which describes the stable configurations that a device can assume. Each *State* class defines the set of allowed *state values*; states, like functionalities, are divided in *Continuous* and *Discrete*.

DOG uses the DogOnt ontology for implementing several functionalities (Section 3) encompassing **command validation** at run-time, using information encoded in functionalities, **stateful operation**, using the state instances associated to each device, **device abstraction** leveraging the hierarchy of classes in the controllable subtree. The last operation, in particular, allows to deal with unknown devices treating them as a more generic type, e.g., a dimmer lamp can be controlled as a simple on/off lamp. Ontology instances used to model environments controlled by DOG are created off-line by means of proper editing tools, some of which are currently being designed by the authors, and may leverage auto-discovery facilities provided by the domotic systems interfaced by DOG.

2.4 Ring 3

API Services provided by DOG are exposed to external OSGi-based applications by means of the API bundle that allows to retrieve the house configuration, to send commands to devices managed by DOG and to receive house events.
XmlRPC The XmlRPC bundle simply provides an XML-RPC endpoint for services offered by the API bundle, thus enabling non-OSGi applications to exploit DOG services.

3 Ontology-based interoperation in DOG

Differently from currently available solutions for domotic interoperability, DOG is **strongly based on semantics** and exploits the DogOnt ontology to tackle many issues, related to internal working and to interaction with external applications. Following sections describe in more detail how DogOnt is exploited in DOG's internal operations.

3.1 Start-up

In the start-up phase, information contained in the DogOnt ontology instantiation that models the controlled environment, and exposed through the House Model bundle, is queried to configure network drivers and to deal with unknown device types. When a DOG instance is run, DOG bundles are started, with a bootstrap order defined by the Platform Manager bundle. The House Model is one of the first available services and is used by network drivers to get the list of their managed devices. The first interaction step involves querying a DogOnt instantiation, using constructs defined in the W3C standard query language for RDF (SPARQL [7]), for **extracting device descriptions** (Figure 3), filtered by technology (e.g., searching specific *DomoticNetworkComponent* instances). Each device description contains the device name as well as all the low level details needed by drivers to communicate with the corresponding real device.

```
SELECT ?x WHERE
{ ?x a d:OpenWebNetComponent}                    (a)

SELECT ?x WHERE
{ ?x a d:KonnexComponent}                        (b)
```

Fig. 3. SPARQL queries for retrieving all BTicino OpenWebNet (a) and all KNX (b) devices in DogOnt.

Once the complete device list is received, each driver builds a mapping table for translating high-level commands and states defined in DogOnt into suitable sequences of protocol-specific messages. In this phase, drivers can possibly find unsupported devices, i.e., devices that they cannot control as no mapping

between high and low level messages is defined, yet. In this case, a further interaction with the House Model requests a **generalization step** for instances of unknown devices. For each unknown device, the House Model retrieves the super-classes and provides their descriptions back to the network drivers. In this way specific devices (e.g., a dimmer lamp) can be treated as more generic and simpler ones (e.g., a lamp), for which network drivers have the proper mapping information. This automatic reconfiguration capability is the key to deal with unsupported devices and sustains DOG scalability: even if devices (and their formalization) evolve more rapidly than drivers, they can still be controlled by DOG, although in a restricted manner.

3.2 Runtime command validation

At runtime, the DogOnt instantiation exposed by the House Model is exploited to semantically **validate** received requests and internally generated commands. For each DogMessage requiring the execution of a command, i.e., requiring an action on a given domotic component, the command value is validated against the set of possible values defined in DogOnt for that component type. Validation proceeds as follows: when a DogMessage containing a command needs validation, the House Model queries DogOnt for allowed commands (Figure 4) and, if necessary, retrieves the allowed range associated to each of its parameters. If the DogMessage command complies with constraints extracted from the ontology model, the command is considered valid and propagation to the DOG Message Dispatcher is allowed, otherwise the command is rejected and the message dropped without any further consequences (except logging).

```
 SELECT ?h WHERE
{ d:DimmerLamp rdfs:subClassOf ?q.
?q rdfs:subClassOf ?y.
?y rdf:type owl:Restriction.
?y owl:onProperty d:hasFunctionality.
?y owl:hasValue ?z.
?z a ?p. ?p rdfs:subClassOf ?l.
?l rdf:type owl:Restriction.
?l owl:onProperty d:commands.
?l owl:hasValue ?h}
```

Fig. 4. The SPARQL query needed to retrieve the commands that can be issued to a specific device, e.g. a DimmerLamp.

3.3 Inter-network scenarios

Together with validation and automatic generalization of devices, the semantic House Model assumes a crucial role in the definition of scenarios and commands involving more than one domotic network. This is a typical case for **home scenarios**, i.e., for commands that coordinate different devices to reach a given

high-level goal, for example to set-up a comfortable environment for watching the TV.

If a scenario involves devices belonging to different domotic plants, the abstraction introduced by DogOnt allows to define operations in a technology neutral way and to properly generate the corresponding DogMessages that are then converted into network specific calls.

Example A very common scenario, available in almost all domotic environments is the "switch-all-lights-off" scenario, that turns off all the lights of a given domotic home. If the sample home contains more than one domotic plant, DOG allows to implement the scenario easily, by means of its House Model bundle. Thanks to the abstraction provided by the DogOnt instantiation managed by the House Model, the "switch-all-lights-off" can be simply modeled by a rule stating that all *Lamp* devices shall receive an *OFF* command, defined by the basic *OnOffFunctionalityInstance* associated to each *Lamp* (Figure 5).

```
Lamp(?x)^hasState(?x,?y)^DiscreteState(?y)^
^valueDiscrete(?y,?z)^equals(?z,"ON")->
valueDiscrete(?y,"OFF")

Lamp(?x)^hasState(?x,?y)^ContinuousState(?y)^
valueContinuous(?y,?z)^ge(?z,0)->
valueContinuous(?y,0)
```

Fig. 5. The switch-all-lights-off rule, in Turtle notation.

This rule, when triggered by a call to the API bundle, requires a reasoning step, called Transitive Closure, that allows to propagate properties (e.g., functionalities) along the ontology hierarchy, thus allowing to recognize all the instances of *Lamp* descendants as *Lamps*. For each of them, a suitable DogMessage is generated, carrying detailed information about the destination device, modeled in the ontology by subclassing the proper *DomoticNetworkComponent*. Resulting DogMessages are then propagated by the Message Dispatcher to the network drivers, which, in turn, power off the corresponding lamp devices, be they simple lamps, dimmers or very complex illumination systems.

4 Conclusions

This paper proposed the DOG platform as a means to smoothly evolve simple, isolated domotic networks into comprehensive, home-wide Intelligent Domotic Environments. Several issues have been addressed related to network interoperation, to flexible integration and to cost affordability. Based on a sound, dynamic framework such as OSGi, DOG allows to manage different networks in a light-weight manner, exploiting the DogOnt ontology to support complex interoperation, generalization and validation tasks.

DOG has been implemented in Java, as a set of 12 OSGi bundles running on the Equinox open source framework [8]. The DogOnt ontology is managed by the HouseModel using the HP Jena API [9] while the XML-RPC module exploits the Apache XML-RPC API [10]. DOG is currently released under the Apache license and can be freely downloaded at `http://domoticdog.sourceforge.net`. This web site also publishes a video demonstration of DOG, the DOG documentation files and links to download the DogOnt model. DOG runs over very cheap devices such as an ASUS eeePC 701 (which costs less than 300 Euros), and it is used by the authors to interoperate a BTicino MyHome demo-box and a Konnex demo-box crafted by using off-the-shelf components from Merten and Siemens.

Many interesting research streams stem from this first work on semantics-based domotic interoperation; the authors are currently using the DogOnt model to perform structural and functional reasoning on the environment controlled by DOG and, at the same time, they are developing semantic-based autonomic policies for DOG, in order to exploit the DogOnt formal model to achieve desired home configurations, even in case of device failures.

5 Acknowledgements

This work has been partially funded by the European Commission under the EU IST 6th framework program, project 511598 "COGAIN: Communication by Gaze Interaction". The sole responsibility of this work lies with the authors and the Commission is not responsible for any use that may be made of the information contained therein.

References

1. Scott Davidoff, Min Kyung Lee, John Zimmerman, and Anind Dey. Principle of Smart Home Control. In *Proceedings of the Conference on Ubiquitous Computing*, pages 19–34. Springer, 2006.
2. Sheng-Luen Chung and Wen-Yuan Chen. MyHome: A Residential Server for Smart Homes. *Knowledge-Based Intelligent Information and Engineering Systems*, 4693/2007:664–670, 2007.
3. Home gateway technical requirements: Residential profile. Technical report, Home Gateway Initiative, 2008.
4. Dave Winer. XML-RPC specification. Technical report, UserLand Software, 2003.
5. D. Bonino and F. Corno. DogOnt - Ontology Modeling for Intelligent Domotic Environments. In *7th International Semantic Web Conference*, Lecture Notes on Computer Science, pages 790–803. Springer-Verlag, 2008.
6. Dave Rye. The X10 PowerHouse powerline interface. Technical report, X10 PowerHouse, 2001.
7. Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF (W3C recommendation). http://www.w3.org/TR/rdf-sparql-query/, January 2008.
8. The Eclipse Equinox project. `http://www.eclipse.org/equinox/`.
9. The HP Jena API. `http://jena.sourceforge.net/`.
10. The Apache XML-RPC API. `http://ws.apache.org/xmlrpc/`.