Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Ontology-based Operations in DOG
6. Conclusions
Domotics

Information technology in the home

- Remote lighting and appliance control have been used for years (see X10)
- Nowadays domotics is another term for the digital home, including: the networks and devices that add comfort and convenience as well as security;
- Domotics means controlling heating, air conditioning, food preparation, TVs, stereos, lights, appliances and security system of the home
Domotics – Drawbacks (1/2)

Many vendors on the market with not compatible solutions

- Different technologies (bus, powerline, wireless)
- Different protocols (KNX, MyOpen, X10, LonWorks)
- Different device features
- Different sophistication of device firmware (from simple relay to full software-based operation)
**Domotics – Drawbacks (2/2)**

**Rooted on Simple Electric Automation**

- Only simple automation is supported
  - Simple scenarios
  - Fixed, programmed behaviors
  - Simple comfort, security and energy saving policies

- No support for more complex interactions
  - Adaptation to user preferences
  - Context detection
  - Structural verification
  - Static and dynamic reasoning on the house state
Goal

Evolving into Intelligent Domotic Environments (IDEs)

Supporting Interoperation, Integration and Intelligence by

- Adding a single (cheap) device for
  - interoperating different domotic plants
  - implementing complex behaviors

- Modeling environments in a semantic-rich, technology independent way

- Providing suitable querying and reasoning mechanism over the environment model
Domotic Systems vs Smart Home

**Smart Home**
- **Pros**
  - supports complex and intelligent behaviors
- **Cons**
  - home pervaded by sensors and actuators
  - dedicated hardware and software
  - Experimental and futuristic connotation
  - Very expensive

**Domotic Systems**
- **Pros**
  - Commercial solution
  - Modular and (relatively) easy to install and configure
  - Affordable costs
- **Cons**
  - Sparse technologies
  - Only supports simple automation
  - No support for intelligent behaviors
Starting considerations

- The sparseness of domotics solutions, the differences in languages, communication means and protocols is very similar to the “old web”

- Semantic Web technologies can help solving
  - Interoperation issues
  - Integration of different technologies

- and can support home intelligence through
  - Reasoning
  - Context Modeling
  - ...

DOG: an Ontology-Powered OSGi Domotic Gateway
Anatomy of an Intelligent Domotic Environment

DOG: an Ontology-Powered OSGi Domotic Gateway
DOG (Domotic OSGi Gateway) is a Domotic House Gateway designed for transforming commercial Domotic Systems into Intelligent Domotic Environments.

Based on OSGi architecture.

DOG provides

- Interoperation between different domotic networks through proper drivers
- Technology independent, ontology-based, house and device modeling
- Advanced, inter-network, rule-based scenario definition and operation

DogOnt is the ontology model lying at the basis of DOG
DOG Architecture

- Ring 0: the DOG common library and communication between the OSGi platform and the other rings
- Ring 1: interface to the various domotic networks
- Ring 2: routing infrastructure for messages and intelligence core of DOG (DogOnt)
- Ring 3: access to external applications
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Ontology-based Operations in DOG
6. Conclusions
Ring 0 bundles

Dog library
Library repository for all other DOG bundles. Provides the interfaces of the services implemented by DOG bundles.

Platform Manager
Handles the start-up of the whole system and manage the life cycle of DOG bundles.

Configuration Registry
Stores configuration information about each bundle.
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Ontology-based Operations in DOG
6. Conclusions
Network Drivers

- A Network Driver for **each** different domotic technology (e.g. KNX, OpenWebNet, X10, etc.)
- *Self-configuration* phase in which they retrieve the list of devices from the **House Model**.
- Network drivers translate messages back and forth between Dog bundles and network-level gateways.
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Ontology-based Operations in DOG
6. Conclusions
**Message Dispatcher**
Internal router, delivering messages (commands, state polls or notifications) to the correct destinations.

**Executor**
Semantically validates the command received from the API and forwards to the Message Dispatcher.
**Status**

Caches the states of all devices controlled by DOG.

**House Model**

Intelligence core of DOG. Based on DogOnt ontology.
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Ontology-based Operations in DOG
6. Conclusions
API
Retrieve the house configuration, to send commands to devices and to receive house events.

XmlRPC bundle
It provides an XML-RPC endpoint for services offered by API bundle. It enables non-OSGi applications to control DOG.
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Ontology-based Operations in DOG
6. Conclusions
DogOnt is an ontology model designed for supporting Interoperation, Integration and Intelligence in domotic environments

- Building Thing
- Building Environment
- State
- Functionality
- Domotic Network Component
Environment Modeling (1/2)

BuildingThing

- Models all the elements of a Building Environment divided into:
  - Controllable
  - UnControllable
- The UnControllable sub-tree allows to model:
  - Furniture elements
  - Walls, floors, ceilings and other architectural elements (Architectural sub-tree)
Environment Modeling (2/2)

BuildingEnvironment

- Models rooms and architectural spaces composing a house
  - Rooms
  - External spaces such as garages, garden, etc.
Device Modeling

- Devices are modeled independently from specific technologies
- 3 Modeling axes:
  - **Typology** - describes the type of device, separating appliances and devices belonging to house plants
  - **Functionality** - describes the tasks that a device can accomplish, by defining the available commands
  - **State** - describes the conditions in which a device can be (e.g. a Lamp can be ON or OFF)
- Technology specific aspects are modeled through separate classes
  - **NetworkComponent** - the root concept for modeling every network specific information, its sub-classes reflect the different networks supported by DOG.
Typology

Controllable devices taxonomy

- **Appliances**
  - Brown Goods (TV, HiFi,...)
  - White Goods (Fridge, Dishwasher,...)

- **HousePlants**
  - Electric
  - HVAC (Heating Ventilation & Air Conditioning)
  - Security
Functionalities (1/3)

- Control Functionalities
  - Model the ability of a device to be controlled
  - Define the possible commands and their range (needed for continuous functionalities)
  - Almost every Controllable has a control functionality

- Notification Functionalities
  - Model the ability of a device to issue a notification about state/configuration changes
  - Define the possible notifications
  - Typical of Sensors and Buttons/Switches

- Query Functionalities
  - Model the ability of a device to be queried about its state/configuration
  - It’s defined for all Controllables
Every Functionality class is subdivided into

- **Continuous Functionalities**
  - Model the ability to change device properties in a continuous manner (e.g. dimming the light emitted by a lamp)

- **Discrete Functionalities**
  - Model the ability to abruptly change device properties (e.g. switching a lamp On)
DOG: an Ontology-Powered OSGi Domotic Gateway
States (1/2)

States are classified according to the kind of values they can assume:

- **Continuous states**
  - Model continuously changing qualities (e.g. the current dimming level of a lamp)
  - The current state value is stored in the `continuousValue` property.

- **Discrete states**
  - Model discretely changing qualities (e.g. the lamp being On or Off)
  - The current state value is stored in the `discreteValue` property.
  - Possible states are listed in the `possibleStates` property.
States (2/2)

State

Continuous State

LightIntensity State

LightIntensityState Instance

sample_dimmer_lamp

valueContinuous

100%

hasState
DimmerLamp modeling example

sample_dimmer_lamp

- hasControl: switch_sample_dimmer_lamp
- contains: sample_living_room

- hasState: LightIntensityStateInstance

- hasFunctionality: QueryFunctionalityInstance (getInstance)
  - commands: Get

- hasFunctionality: LightRegulationFunctionalityInstance
  - MaxValue: 100%
  - MinValue: 0%
  - commands: StepUp, StepDown, Set

- hasFunctionality: OnOffFunctionalityInstance
  - commands: On, Off

inherited from Controllable
inherited from Lamp
Outline

1 Introduction

2 Objectives

3 DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles

4 DogOnt

5 Ontology-based Operations in DOG

6 Conclusions
The House Model is the core of the DOG intelligence.

It is based on a formal model defined by DogOnt ontology.

DogOnt is designed for supporting Interoperation, Integration and Intelligence in domotic environments.

DogOnt supports several critical features of DOG.
House Model and DogOnt (2/2)

- A central **point of configuration** for devices
- Specific **uniform set of devices, states and functionalities**
- Enables **syntactic and semantic check of commands**
- Top-down **inter-plant** scenarios which involve devices
- Provides interoperation between plants (e.g. allowing a BTicino button to control a KNX light)
Start-up

Platform Manager

Network Driver (Konnex)

Dispatcher

House Model

- start()
- start()
- start()

getAllDevices(Konnex)

Konnex devices

* repeat until all devices are mapped

Create Mappings

Generalize(?x)

Generalize(?x)

Father(?x)

Father(?x)
Command Validation

1. Start Validation
2. Syntax OK?
   - SYNTACTIC VALIDATION
   - get Device Description
   - commands AND values match?
   - SEMANTIC VALIDATION
   - Discard message
3. Check Syntax
   - get Device Description
   - return Description
4. Check Semantics
5. Forward message
6. Dog Message
7. Executor
8. House Model
9. Dispatcher
Inter-network scenarios

Diagram showing the flow of control between different components:

- External App.
- API
- House Model
- Executor
- Dispatcher

Actions:
- allLampsOFF
- ?lamp
- all Lamps
- Set(?x, OFF)
- Validate
- Set(?x, OFF)
- To Network Drivers
  - Set(?x, OFF)
  - Set(?x1, OFF)
  - Set(?x2, OFF)
Advanced Intelligence in DOG

- Transitive closure and Classification Reasoning to **decouple evolution of the model and domotic systems**

- Structural verification of domotic environments through SWRL constraints

- Dynamic detection of safety critical situations (smoke propagation, safe exit path) using rule-based reasoning

- On-going work on automatic generation of interoperation rules from DogOnt
Experimental set-up

Technologies
- Eclipse Equinox OSGi framework
- Jena and Pellet
- MyOpen and KNX

Components
- DOG runs on an ASUS eeePC701
  - 900MHz Celeron processor
  - 512MByte RAM
  - 4GByte SSD
- KNX demo case built by the authors
- MyOpen demo case offered by BTicino
Reference Environment

Domotic Devices

- 27 Push Buttons
- 7 Lamps
- 23 Plugs
- 7 Door Actuators
- 7 Door Sensors
- 6 Window Actuators
- 6 Window Sensors
- 6 Shutter Actuators
- 5 Infrared Sensors
- 6 Smoke Sensors
Ontology-based Operations in DOG

Operations supported by DogOnt

- **Installation** (~40s)
  - Model Reasoning (transitive closure + classification)
- **Start-up** (< 3s)
  - SPARQL queries for associating devices to drivers
- **Validation** (<100ms)
  - SPARQL queries for gathering allowed commands and their ranges
  - Comparison between requested and allowed operations
- **Inter-network scenarios**
  - SPARQL queries for gathering specific device types (e.g. Lamps)
  - Generation of commands on the basis of device types (e.g. all Lamps ON)
SPARQL queries

Controllable query excerpt

```sparql
SELECT DISTINCT ?x ?y WHERE {
...
UNION
{?x rdfs:subClassOf dogont:Controllable . ?x rdfs:subClassOf ?s.
?s rdfs:subClassOf [rdf:type owl:Restriction;
owl:onProperty dogont:hasFunctionality;
owl:someValuesFrom ?y] . ?y rdfs:subClassOf
dogont:Functionality;}
UNION
{?x rdfs:subClassOf dogont:Controllable . ?x rdfs:subClassOf ?s.
?s rdfs:subClassOf [rdf:type owl:Restriction;
owl:onProperty dogont:hasFunctionality;
owl:allValuesFrom ?u] . ?u owl:unionOf [
list:member[rdf:type ?v; rdfs:subClassOf ?y;]]
. ?y rdfs:subClassOf dogont:Functionality;}
...
FILTER(?x != owl:Nothing) . FILTER(?x != owl:Thing)
}ORDER BY ?x ?y
```
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Ontology-based Operations in DOG
6. Conclusions
Conclusions

- We developed DOG: an ontology-powered OSGi Domotic Gateway
- Dog currently uses DogOnt ontology, that allows to control several, different, domotic plants, at the same time
- Dog will transform your Domotic plants into Intelligent Domotic Environments.

http://domoticdog.sourceforge.net
References

http://domoticdog.sourceforge.net

BONINO D; CASTELLINA E; CORNO F.; GALE A; GARBO A; PURDY K; SHI F, A blueprint for integrated eye-controlled environments, UNIVERSAL ACCESS IN THE INFORMATION SOCIETY, 2009, Vol. 8/4, ISSN: 1615-5289, DOI: 10.1007/s10209-009-0145-4


License

This document is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License.

http://creativecommons.org/licenses/by-nc-sa/3.0/