eDIANA
Embedded Systems for Energy Efficient Buildings

Grant agreement no.: *100012*

D2.2-A Ontology for Device Awareness

Author(s):	**N. Arana**	**ESI**
	A. Noguero	**ESI**
	T. Padilla	**I&IMS**
	MJ. Mtz. de Lizarduy	**I&IMS**

Issue Date	30 November 2009 (m10)
Deliverable Number	D2.2-A
WP Number	WP2
Status	Delivered

Dissemination level	
X	**PU** = Public
	PP = Restricted to other programme participants (including the JU)
	RE = Restricted to a group specified by the consortium (including the JU)
	CO = Confidential, only for members of the consortium (including the JU)

Document history			
V	***Date***	***Author***	***Description***
1	*September 2009*	*I&IMS*	*ToC definition and Architecture elements analysis*
2	*October 2009*	*ESI*	*ToC modifications, objectives, applied methodology, ontology*
3	*November 2009*	*ESI, I&IMS*	*Final version*

Summary

The Ontology for Device Awareness is a public document delivered in the context of WP2, task 2.2 with regard to device awareness inside eDIANA architecture.

This document is about the ontology, a semantic technology, for device discovery and interoperability in the context of eDIANA Platform. This document goes with the prototype Ontology for Device Awareness that provides the semantic concepts of the interaction between the different components and devices to composite the internal situation of the system and its relevant external environment.

Contents

Abbreviations

CCA	Cell Control and Actuation
CDC	Cell Device Concentrator
CGS	Cell Generation and Storage
CMM	Cell Monitoring and Metering
CUI	Cell User Interface
c2MCCi	Cell Concentrator to Macro Cell Concentrator Interface
eDIANA	Embedded Systems for Energy Efficient Buildings
EDP	eDIANA Platform
iEi	Intelligent Embedded Interface
MCC	Macro Cell Concentrator
PwGRIDi	Power Grid Interface
WWWi	Internet Interface

Table of Figures

1. Introduction

Once the basic architecture of eDIANA ecosystem has been defined, the next step is to start with the modelization of the information that will flow through the different modules that will be involved into that architecture.

Eventually, all the devices and sensors participating in the cell level as well as any software application or module that could be part of the transversal middleware platform will provide heterogeneous data in several heterogeneous protocols. Part of that information is referred to the device awareness that Cell and MacroCell level have to manage to hold a complete and dynamic scenario of components, services and data flow.

Three semantic views or layers in the architecture are defined: semantic information, services and devices. The idea is to define semantically (using ontologies), all the actors (services and devices) involved in the eDIANA architecture.

There will be a semantic information layer which all involved devices (sensors, actuators, smart meters, appliances …) can access to, through their services. This access has to be defined and featured. In that semantic information layer there will be a kind of database or repository with the ontologies to which devices can connect and register themselves, more specifically services of the devices. The new services and devices registered information will be transferred to the rest actors.

From the analysis and comparison of technologies, languages, libraries and tools relating to device discovery and interoperability, an ontology prototype of the device awareness will be defined.

This document contains the description of the obtained results and the analysis process of the different aspects of the involved devices and modules. The identification of the methodologies used to obtain the ontology prototype, is also included.

The final objective of this document is to describe and explain the ontology for device awareness in eDIANA Platform.

This ontology will be the main start point to build the transversal middleware which would provide an homogeneous and common platform for all diverse devices, sensors and applications involved into the eDIANA solution.

2. Objectives of the ontology

Ontology is a description, a formal description of the concepts and relationships that can exist for an agent or a community of agents, usually inside a specific domain. It defines entities, properties, interactions, actors and basic concepts that compose the common vocabulary for all members of the domain in which it is defined.

Ontologies are defined by classes, properties that describe various features and attributes of classes and restrictions on these properties. Classes describe concepts whereas subclasses represent concepts that are more specific. The development of ontology usually includes:

- To define the classes of the ontology
- To arrange the classes hierarchically
- To define the properties and their possible values. Properties can be referred to an only class or subclass or can be use to relate different classes.

However, at least as much important as what it is, is what it is for. Some frequent uses of any ontology are:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge

One of the results of defining ontologies in a specific domain is the definition of a framework where most of the disciplines, technologies that will be involved in the domain can adopt the universe defined in the ontology. It can be the start point to several technologies, applications and software that use the ontology as knowledge base to other objectives.

The main objective of this ontology is to define the universe of concepts or classes and their relations in the domain of eDIANA Platform Architecture, related to device awareness. The eDIANA Platform Architecture provides a wide and heterogeneous list of devices in hierarchical levels: MacroCell and Cell. To allow the scalability of the architecture, the replacement, suppression or inhibition of architecture elements is essential the knowledge of the context in which each element is, their relevant characteristics to other elements as well as how they interact with other elements.

3. Analysis

At this moment of the project, a preliminary architecture for EDP, at cell level, could be defined from which to start the ontological analysis. To do this definition, we will take into account three main aspects of the system:

3.1 Basic functionalities and elements

The eDiana Platform is understood as a loosely coupled application to be deployed in a distributed fashion. Its functionalities will be compiled and grouped using the classical three-tier software architecture: user interface, business logic layer and data access layer.

Several basic functionalities may be identified for each tier.

User Interface

Regarding to each EDP deployment or scenario, a simple HMI could display data in a summarized way or, while scaling up, more sophisticated and capable HMI could be used.

- Display the current active loads: One of the keys on the EDP is the electrical load management in accordance to some energy efficiency algorithms; in this context, it seems reasonable to have the possibility to check and verify the status of the loads involved in the scenario.
- Display the comfort conditions: EDP is designed to be deployed both on residential buildings and non-residential buildings, in both cases awareness of environmental conditions and the settings imposed by the user may become key issues.
- Display the overall consumption/generation: Not only the active loads but the overall consumption/generation of the managed scenario is data to be made available for the users. This means that the user should know its consumption vs generation ratio.
- Display Energy Market Conditions: eDiana Platform as being and interoperative platform with utility companies or other energy pricing authorities should display each time the active pricing profile
- Modify the comfort standards: The platform will provide mechanisms to modify the comfort set values for a scenario or part of it.

- Modify set values for load management: The platform infrastructure will be able to give the capability for modifying the set values for load management of a scenario or any part of it.

Business Logic Layer

This layer includes the functionalities regarding the control strategies, energy constraints imposed by the utility companies or users willing and the possible interfaces with higher entities.

- Energy Efficient Comfort management algorithms: The so called BLL will include EE comfort management algorithms, which will perform an integrated lighting, loads and HVAC management.
- Demand side management algorithms: The EDP as an energy management platform will react to energy constraints and real time price signals imposed by the utility companies or by the grid switching configuration. Once these constraints are in place, load management algorithms will decide when and how to switch-off or pause or reactivate the loads.
- Energy management policy: The EDP as part of an infrastructure that may include distributed renewable energy generation and a smart grid environment, will decide, on the basis of user policy, utilities and grid constraints, and real time prices how to coordinate the parallel utilization of loads and of local generation (both thermal and electrical)
- High Level Communication with utility companies: As being in direct contact with the utilities companies and/or energy suppliers, the applications built in the platform should be able to establish a communication with them in order to negotiate or exchange all the information required to optimize the energy utilization and to trade off the best mix between energy utilization and generation..
- Energy Consumption predictive models: In order to enhance the energy efficient management, the algorithms that will manage loads, generation, if available and comfort conditions, will relay also on energy consumption prediction models. These models may be based on the mathematical formulation, typically via neural network approach, of consumption patterns or time series (similar day’s technique for example) and load profiling.
- Scheduled Occupancy and Forecast: Includes the access to scheduled use of the managed scenario and weather forecast to improve the accuracy of the management algorithms.

Data Access Layer

This point does not only include the functionalities commonly understood as DAL duties and infrastructures, but it also takes care about the low level communication processes. In fact the DAL should be understood as the layer in which the BLL relies, including both hardware and software platforms.

- Get and Set data from sensors and actuators: Some daemons are needed to manage the communication flow towards and from the sensors and actuators to some concentrator units or data brokers. The detailed list and description of sensing and actuating requirements, involved in the platform will be done in task T1.4, nevertheless some of them are immediately foreseen:
    - Indoor & Outdoor temperature sensors
    - Relative Humidity sensors
    - Sun radiation
    - Lighting sensors
    - People presence sensors
    - Air flow sensors
    - Light dimming actuators
    - Blind actuators
- Interface with electrical loads to switch them on/off: In order to react to constraints imposed by the utility companies, user preferences, grid configuration or other, it becomes necessary to know not only the consumption level of the internal electrical loads, but also have the capability to send orders to them.
- Real time gathering Real time data gathering strategies implementation would be needed. As the provision of runtime data for BLL is the main goal of the DAL the data refreshment ratio necessary for the BLL algorithms will somehow be a requirement for gathering strategies.

3.2 Identified components - devices

eDIANA Platform Architecture does not define exactly explicit devices, but it defines components that can be integrated as an only device o as a part of a device. Any device likely to be part of eDIANA Platform must be categorized as one or more components of eDIANA Platform and be compliant with their associated requirements and characteristics, providing in that way a future integration of a wide variety of devices.

Because of the goal of this work is the definition of an ontological structure for the device awareness, the identification of the components involved in the EDP architecture, as well as their attributes and behaviour, is fundamental.

The identified eDIANA Platform Components are:

- MacroCell Level Components:
  - MacroCell Level Concentrator (MMC)
  - Data gathering Component
  - Control Strategies

- Cell Level Components:
  - Cell level Monitoring and Metering (CMM)
  - Cell level Control and Actuation (CCA)
  - Cell level User Interface Channels (CUI)
  - Cell level Generation & Storage (CGS)
  - Cell level Concentrator (CDC)

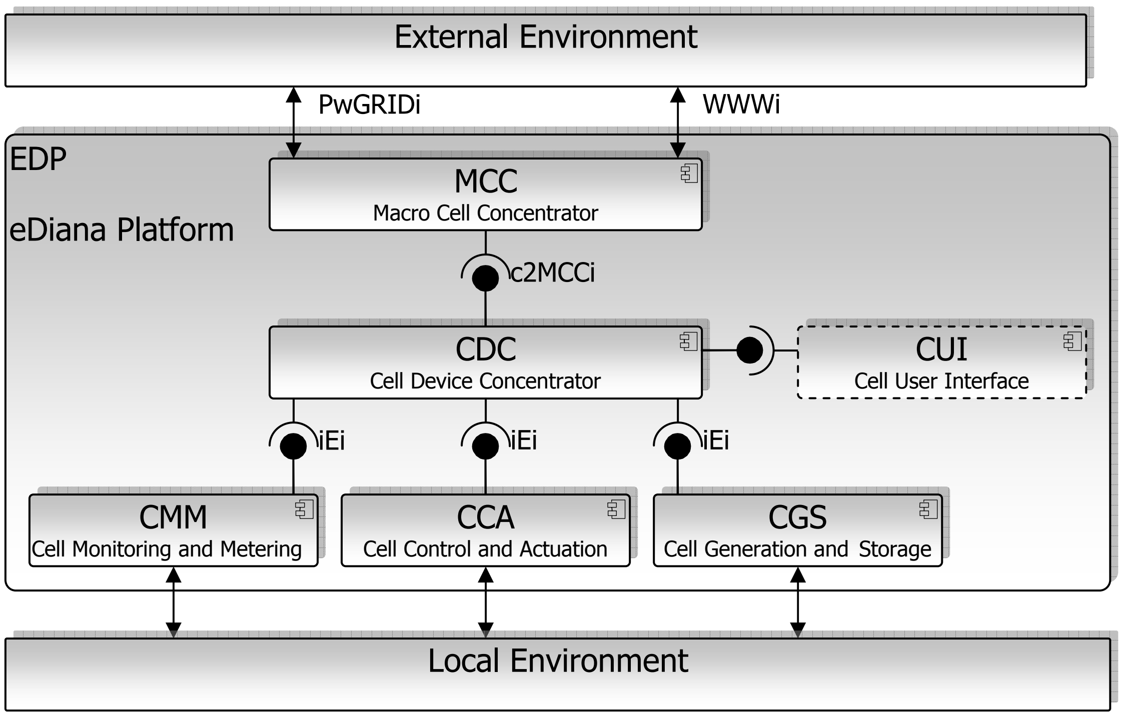
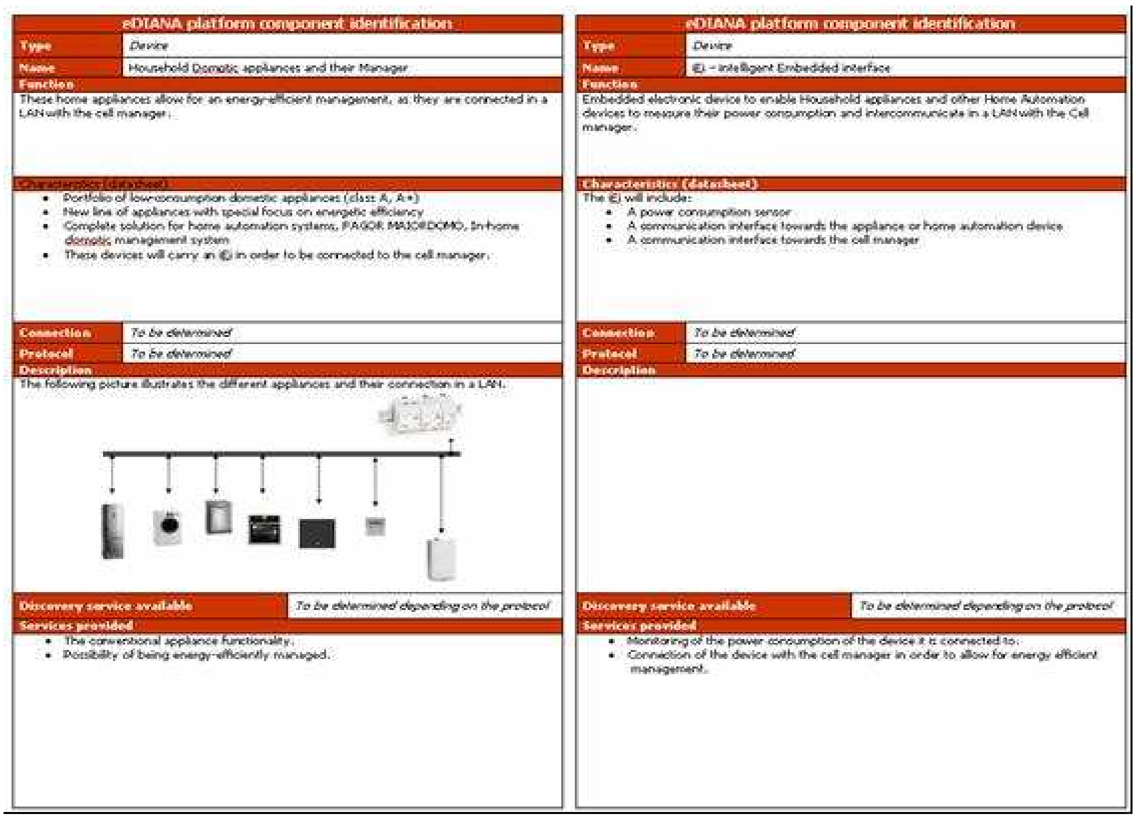
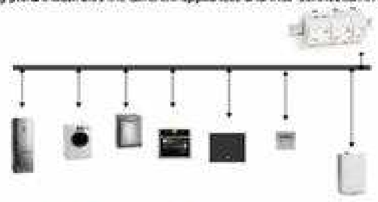


Figure 3-1 EDP reference architecture

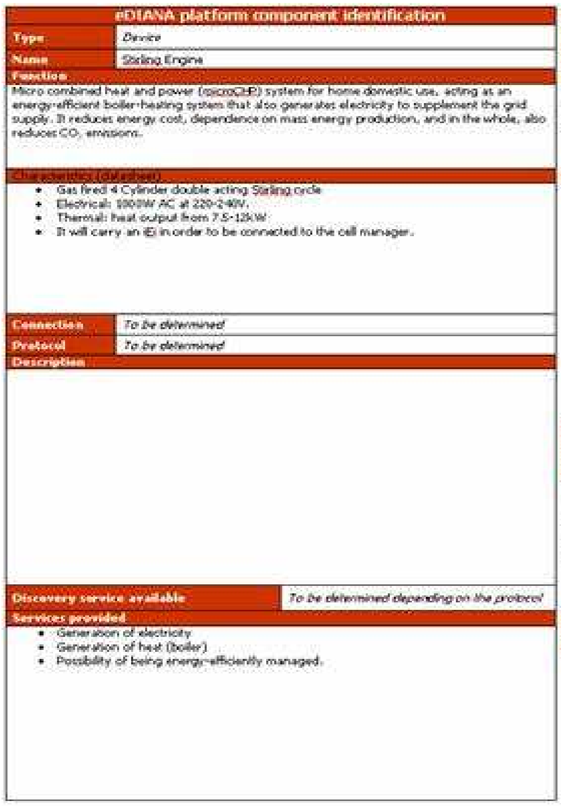
The development of the ontology has focused on the components of the Cell level, analyzing them deeply, mainly the characteristics that they must fulfil related to device awareness. The ontology classes, subclasses, properties and relations will describe how these components interact to establish a known context

Next, some examples of devices that can be integrated in eDIANA Platform.



eDIANA platform component identification	
Type	*Device*
Name	Household Domotic appliances and their Manager
Function	
These home appliances allow for an energy-efficient management, as they are connected in a LAN with the cell manager.	
Characteristics (datasheet)	
• Portfolio of low-consumption domestic appliances (class A, A+) • New line of appliances with special focus on energetic efficiency • Complete solution for home automation systems, FAGOR MAIORDOMO, In-home domotic management system • These devices will carry an iEi in order to be connected to the cell manager.	
Connection	*To be determined*
Protocol	*To be determined*
Description	
The following picture illustrates the different appliances and their connection in a LAN. 	
Discovery service available	*To be determined depending on the protocol*
Services provided	
• The conventional appliance functionality. • Possibility of being energy-efficiently managed.	

eDIANA platform component identification	
Type	*Device*
Name	iEi – intelligent Embedded interface
Function	
Embedded electronic device to enable Household appliances and other Home Automation devices to measure their power consumption and intercommunicate in a LAN with the Cell manager.	
Characteristics (datasheet)	
The iEi will include: • A power consumption sensor • A communication interface towards the appliance or home automation device • A communication interface towards the cell manager	
Connection	*To be determined*
Protocol	*To be determined*
Description	
Discovery service available	*To be determined depending on the protocol*
Services provided	
• Monitoring of the power consumption of the device it is connected to. • Connection of the device with the cell manager in order to allow for energy efficient management.	



eDIANA platform component identification

Type	*Device*
Name	Stirling Engine

Function

Micro combined heat and power (microCHP) system for home domestic use, acting as an energy-efficient boiler-heating system that also generates electricity to supplement the grid supply. It reduces energy cost, dependence on mass energy production, and in the whole, also reduces CO$_2$ emissions.

Characteristics (datasheet)

- Gas fired 4 Cylinder double acting Stirling cycle
- Electrical: 1000W AC at 220-240V.
- Thermal: heat output from 7.5-12kW
- It will carry an iEi in order to be connected to the cell manager.

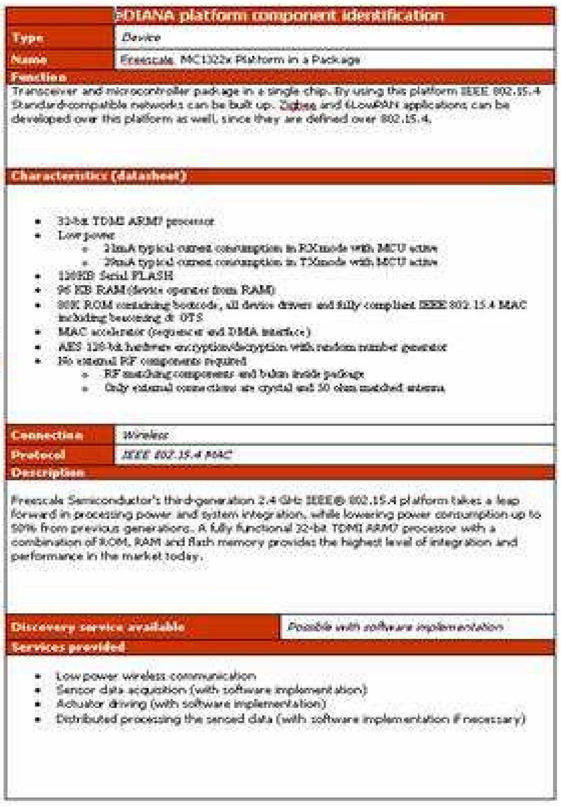
Connection	*To be determined*
Protocol	*To be determined*

Description

Discovery service available	*To be determined depending on the protocol*

Services provided

- Generation of electricity
- Generation of heat (boiler)
- Possibility of being energy-efficiently managed.



eDIANA platform component identification

Type	*Device*
Name	Freescale MC1322x Platform in a Package

Function

Transceiver and microcontroller package in a single chip. By using this platform IEEE 802.15.4 Standard-compatible networks can be built up. Zigbee and 6LowPAN applications can be developed over this platform as well, since they are defined over 802.15.4.

Characteristics (datasheet)

- 32-bit TDMI ARM7 processor
- Low power
  - 22mA typical current consumption in RX mode with MCU active
  - 29mA typical current consumption in TX mode with MCU active
- 128KB Serial FLASH
- 96 KB RAM (device operates from RAM)
- 80K ROM containing bootcode, all device drivers and fully compliant IEEE 802.15.4 MAC including beaconing & GTS
- MAC accelerator (sequencer and DMA interface)
- AES 128-bit hardware encryption/decryption with random number generator
- No external RF components required
  - RF matching components and balun inside package
  - Only external connections are crystal and 50 ohm matched antenna

Connection	*Wireless*
Protocol	*IEEE 802.15.4 MAC*

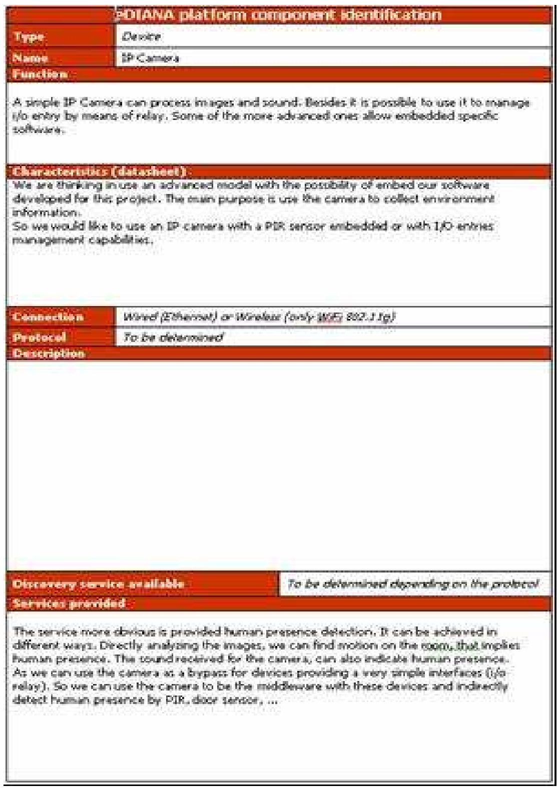
Description

Freescale Semiconductor's third-generation 2.4 GHz IEEE® 802.15.4 platform takes a leap forward in processing power and system integration, while lowering power consumption up to 50% from previous generations. A fully functional 32-bit TDMI ARM7 processor with a combination of ROM, RAM and flash memory provides the highest level of integration and performance in the market today.

Discovery service available	*Possible with software implementation*

Services provided

- Low power wireless communication
- Sensor data acquisition (with software implementation)
- Actuator driving (with software implementation)
- Distributed processing the sensed data (with software implementation if necessary)



eDIANA platform component identification

Type	*Device*
Name	IP Camera

Function

A simple IP Camera can process images and sound. Besides it is possible to use it to manage i/o entry by means of relay. Some of the more advanced ones allow embedded specific software.

Characteristics (datasheet)

We are thinking in use an advanced model with the possibility of embed our software developed for this project. The main purpose is use the camera to collect environment information.
So we would like to use an IP camera with a PIR sensor embedded or with I/O entries management capabilities.

Connection	*Wired (Ethernet) or Wireless (only WiFi 802.11g)*
Protocol	*To be determined*

Description

Discovery service available	*To be determined depending on the protocol*

Services provided

The service more obvious is provided human presence detection. It can be achieved in different ways. Directly analyzing the images, we can find motion on the room, that implies human presence. The sound received for the camera, can also indicate human presence. As we can use the camera as a bypass for devices providing a very simple interfaces (i/o relay). So we can use the camera to be the middleware with these devices and indirectly detect human presence by PIR, door sensor, ...

Figure 3-2 eDIANA devices examples

These devices examples can be mapped to only one component or to several of them. For example, a smart appliance must fulfil the characteristics and behaviour of different eDIANA components, because it can be at the same time a monitoring component and an actuation component. An iEi device with a built-in PSC must be compliant with the definition of monitoring and control and actuation components.

3.3 Communication and interfaces

Although the lower layers of the communication networks stacks used in EDP are not interesting for the ontological analysis, however the identification of the protocols to be used by the different system components/devices can provide a complete vision of the communications in the whole platform.

Each device will propose its own protocol, standard or proprietary, but for all devices of Cell Monitoring and Metering (CMM), Cell Control and Actuation (CCA) and Cell Generation and Storage (CGS) that do not support the iEi protocol, a "gateway", a software running on Cell Device Concentrator (CDC) will generate missing information to be compliant with iEi protocol.

There have been identified some interfaces in eDIANA Platform, although we will only considerate those related to the device awareness in the ontology. This device awareness is focused at Cell level because is in that level where the diversity of devices is not fixed, neither their presence assured. It is at Cell level where the knowledge of the environment composition in a dynamical way is essential because of its changeability, whereas the structure of the MacroCell is more fixed as well as its interaction with Cell level.

Identified eDIANA interfaces:

- iEi: Intelligent Embedded Interface. It is the union of a PCS and an interface which connects to the CDC. This interface is in charge of the information interchange between sensors / transceivers and CDC
- C2CMCi: Concentrator to MacroCell Concentrator Interface.
- WWWi: This interface comprises the access to eDIANA Platform from internet
- PwGRIDi: this interface comprises the information interchange between eDIANA Platform with the Power Grid.

The interface that will be considered inside the device awareness ontology will be iEi, because is the interface that is in charge of the communication between Cell level elements.

3.4 Device awareness in middleware.

This section will analyze briefly several middleware that consider in their specification the context or device awareness and how they face it.

3.4.1 OSGi

The Framework forms the core of the OSGi Service Platform Specifications. It provides a general-purpose, secure, and managed Java framework that supports the deployment of extensible and downloadable applications known as "bundles" [20].

OSGi-compliant devices can download and install OSGi bundles, and remove them when they are no longer required. The Framework manages the installation and update of bundles in an OSGi environment in a dynamic and scalable fashion. To achieve this, it manages the dependencies between bundles and services in detail.

It provides the bundle developer with the resources necessary to take advantage of Java's platform independence and dynamic code-loading capability in order to easily develop services for small-memory devices that can be deployed on a large scale.

The functionality of the Framework is divided in the following layers:

- Security Layer
- Module Layer
- Life Cycle Layer
- Service Layer
- Actual Services

The part of the specification that is more interesting to the issue that this document deals is the Service Layer. In this layer, services can register, receive notifications, change their status and be searched.

Next list describe some services specification related to context and device awareness that OSGi Framework provides:

- Log Service Specification: this specification defines a general-purpose message logger inside the OSGi Service Platform. This specification establishes two services, one for logging information and another for retrieving previous logged information. The specification summarizes the methods and interfaces of the services bundle developers will use to log entries and retrieve them.

- Device Access Specification. This service specification allows the interconnection between services and devices of different vendors. It is a meeting point to add and cancel subscription to services. It supports the coordination of automatic detection and attachment of existing devices on an OSGi Platform, as well as facilitates hot-plugging and -unplugging of new devices, and downloads and installs device drivers on demand.

- Communication Admin Service Specification. It allows an Operator to set the configuration information of deployed bundles. Configuration is the process of defining the configuration data of bundles and assuring that those bundles receive that data when they are active in the OSGi Service Platform.

- Preference Service Specification. This service specification helps to have some data persistently in the system when some bundle stops and starts. The data can be related to the system or to a particular user of the service.

- Wire Admin Service Specification. The Wire Admin helps to reduce the amount of context-specific knowledge required by bundles when used in a large array of configurations. It can be said that it dynamically wires services together. Bundles participate in this wiring process by registering services that produce or consume data. The Wire Admin service wires the services that produce data to services which consume data.

- UPnP Device Service Specification. It specifies how OSGi bundles can be developed that interoperate with UPnP™ (Universal Plug and Play) devices and UPnP control points.

3.4.2 KNX

KNX is a standardized (EN 50090,ISO/IEC 14543), OSI-based network communications protocol for intelligent buildings. KNX is the successor to, and convergence of, three previous standards: the European Home Systems Protocol (EHS), BatiBUS, and the European Installation Bus (EIB). The KNX standard is administered by the Konnex Association.

KNX protocol defines several physical layers or combinations of them:

- Twisted pair cabling

- PowerLine

- RadioFrencuency

- IP enabled media like Ethernet, Bluetooth, WiFi, and IEEE 1394

However, it does not specify any hardware platform that the devices must adopt. Depending on the chosen profile, the manufacturer can develop really tiny systems with a footprint less than 5kb that can run on systems based on 8-bit processors.

One main concept of KNX protocol is "Datapoint", this term represent the process and control variables in the system. This Datapoint can be inputs, outputs, parameters, diagnostic data, etc. The Communication System and Protocol offers a reduced instruction set to read and write these Datapoint values. An application can be seen as a collection of sending and receiving Datapoints, distributed over a number of devices.

The mechanism that makes the system works is the links between Datapoints in different devices through a common identifier. One application in device A wants to send a value, e.g. a sensor data, writes the value to a sending Datapoint. The device sends a "write" message with the corresponding address and the value. A receiving Datapoint with the same address will receive this value and inform its local application.

There are types of binding Datapoints:

- Free Binding: the numerical value of the address has no semantic information to the application and there is no a priori prescription on which Datapoints may be linked to one another.
- Structured Binding: the numerical value of the address has no semantic information to the application. The address value is free but the Datapoints involved in the binding follow a precise pattern, usually corresponding to a Functional Block or Channel.
- Tagged Binding: part of the numerical value of the address is the semantic (Datapoint) identifier. The logical tag or zonning part of the address selects the intended communication partner(s) at the level of the device.

The specification provides several types of Datapoints:

- Datapoint types for common use
- Datapoint types for HVAC
- Datapoint types for Load Manager
- Datapoint types for Lighting
- Datapoint types for System
- Specialized Datapoints: Parameters

Previous to the configuration of the devices, some identification mechanisms are provided:

- Devices may be identified and subsequently accessed throughout the network either by their individual address, or by their unique serial number, depending on the configuration mode.
- Installation's extendibility and maintenance is considerably eased through product identification (i.e. a manufacturer specific reference) and functional identification (manufacturer independent) information retrievable from devices.

Mechanisms are defined around the unique serial number feature to

- get individual address of a device with a given serial number (so providing further access)
- set individual address of a device with a given serial number
- retrieve serial number of a device at a given individual address

To address many diverse needs, the KNX specification includes a *toolkit* of management features enabling the choice between several configuration modes, each adapted to different markets, local habits, level of training needed or application environment.

The specification ensures some degree of freedom for the manufacturer, whilst guaranteeing consistency and Interworking in one mode, even in a multivendor context. All configuration modes include provisions in order to extend or modify the installation using the same mode.

- System Mode: The devices that implement this mode offer the most versatile and multi-usage configuration process, while permitting a compact implementation: the complexities of binding and application configuration are shifted to a powerful configuration master. Traditionally this role is taken on by a set of PC based tools from the ETS family, supplied by the KNX Association. A database representing all possible functionalities of the devices (or products) it supports by this tool, and it is maintained by the manufacturers. Some features of this tool are:
    - Binding: setting the group addresses in order to enable group object communication between the Functional Blocks. Group objects may be set into relationship if they share the same Datapoint type. The possibly complex address and indirection tables are constructed by the configuration master (like ETS), and downloaded into the device.

  - Parameterisation: setting the parameters of the devices according to the documentation of the manufacturer.
  - Download of application program is also possible for multi-purpose devices exhibiting this special feature.

- Controller Mode: This mode is defined to support installation of a limited number of devices on one logical segment of a physical medium. An installation using the controller mode will comprise one special device called controller that is in charge of supporting the configuration process. The controller supports one or more applications (e.g. lighting). At configuration time, the role of the controller is to establish the links between the so-called "channels", which represent the set of group objects for the given functionality. The links and parameters are identified by an action of the installer, which may be different from one controller manufacturer to another. However, the set of channels supported by the KNX specification is uniquely specified, and therefore any device from any manufacturer will be taken into account by any controller from another manufacturer. The controller does not need to have any knowledge of the functionalities supported by the devices. This functionality can be read out of the devices by the controller and is "hard coded" in the called "Device Descriptor #2" implemented by each device. Following the instructions of the installer, the controller assigns individual addresses to the devices, calculates the links at Datapoint level using known rules which are part of the specification, and sets the parameters which are available for the considered devices.

- Push-Button Mode: This mode is defined to support installation of a limited number of devices on one logical segment of a physical medium. There is no need for a specialised device for configuration, therefore each device implementing the push button mode shall include the means of configuration related to its application. The devices implement fixed parametrical functionalities as described in the corresponding application specifications. Each device is provided with the ability to be dynamically configured by setting the needed individual and group addresses and parameters. Exchange of parameters is also possible, but will be mainly local on the devices. At configuration time, the installer successively designates the devices the functions of which will be linked (therefore the name push button), the way he does it is manufacturer dependent. The exchange of configuration data occurring between devices, typically sensors and actuators, is made trough a single application layer service. Each sending Datapoint device acquires itself its unique group address. This address will be given to the receiving Datapoints using the pushbutton procedure.

- Logical Tag Extended Mode: Device configuration is made using tags set locally by physical means. For the time being, Logical Tag extended is limited to HVAC applications, which need longer set of structured data. These data are exchanged via interface objects using the extended frame of the KNX protocol. Exploiting the extended address space, the tags represent powerful zoning information, which is essential in modular structured applications (e.g. heating of big buildings).

3.4.3 ZigBee (Home Automation Public Application Profile)

The Home Automation Public Application Profile defines device descriptions and standard practices for applications needed in a residential or light commercial environment. Installation scenarios range from a single room to an entire home up to 20,000 square feet (approximately 1850m2). The key application domains included in this initial version are lighting, HVAC, window shades and security. Other applications will be added in future versions [21].

Device descriptions specified in this profile are organized according the end application areas they address. A product that conforms to this specification shall implement at least one of these device descriptions and shall also include the device descriptions corresponding to all applications implemented on the product where a standard device description is specified in this profile.

A product that conforms to this specification shall implement at least one of these device descriptions and shall also include the device descriptions corresponding to all applications implemented on the product where a standard device description is specified in this profile.

The specified devices until now are:

- Generic:
    - On/Off Switch
    - Level Control Switch
    - On/Off Output
    - Level Controllable Output
    - Scene Selector
    - Configuration Tool
    - Remote Control
    - Combined Interface
    - Range Extender
    - Mains Power Outlet
- Lighting:

  - On/Off Light
  - Dimmable Light
  - Color Dimmable Light
  - On/Off Light Switch
  - Dimmer Switch
  - Color Dimmer Switch
  - Light Sensor
  - Occupancy Sensor
- Closures:
  - Shade
  - Shade Controller
- HVAC:
  - Heating/Cooling Unit
  - Thermostat
  - Temperature Sensor
  - Pump
  - Pump Controller
  - Pressure Sensor
  - Flow Sensor
- Intruder Alarm Systems:
  - IAS Control and Indicating Equipment
  - IAS Ancillary Control Equipment
  - IAS Zone
  - IAS Warning Device

The ZigBee Cluster Library provides a mechanism for clusters to report changes to the value of various attributes. It also provides commands to configure the reporting parameters. The attributes that a particular cluster is capable of reporting are listed n the ZCL specification for each cluster.

The functionality made available by all supported clusters shall be that given in their ZCL specifications except where a device description in this profile includes further specification, clarification or restriction as needed for that particular device.

The Clusters used in HA Profile are:

- General
  - Basic
  - Power Configuration
  - Device Temperature Configuration
  - Identify
  - Groups
  - Scenes
  - On/Off
  - On/Off Switch Configuration
  - Level control
  - Alarms
- Measurement & Sensing
  - Iluminance Measurement
  - I luminance Level Sensing
  - MTemperature Measurement
  - Pressure Measurement
  - Flow Measurement
  - Relative Humidity Measurement
  - Occupancy sensing
- Lighting
  - Color Control
- HVAC
  - Pump Configuration and Control
  - Thermostat
  - Fan Control
  - Thermostat User Interface Configuration

- Closures
    - Shade Configuration
- Security and Safety
    - IAS ACE
    - IAS Zone
    - IAS WD

Many, if not all of the devices described in this profile will require some form of commissioning, even if the user or installer does not see it. This is because, for example, an actuating device needs to be bound to some sort of target in order to do useful work, and even if the required initializations are done at the factory before the device is sold, the required operations are virtually the same as is the outcome.

HA devices must form their own network or join an existing network. It is intended that an HA network use simple methods to form a network and to commission devices into it. The primary means of commissioning a network will use E-mode methods (button presses or similar user actions) to get nodes to join a network.

This specification has no mandates to the start-up sequence of devices or the network, however, there are some recommended practices:

- A device should be able to indicate to the user that it has decided to become the coordinator of a network.
- A device should be able to indicate to the user, that it has successfully joined a network.
- A device should be able to indicate to the user, that it is in the process of searching for or joining a network.

All HA devices must support E-mode and may optionally support S-mode. A-mode is not to be supported. E-mode commissioning may be a simple button press or may involve a separate low-cost commissioning tool (like a remote control) that is typically purchased with the vendor.s HA product.

All ZigBee HA devices are required to provide documentation with their product that explains how to perform device commissioning in using a common language set:

- Join Network: Go find and join the first available HA network.
- Form Network: For devices that can start a network.

- Allow Others to Join Network: For routers and coordinators only. Allows you can add more nodes to an existing network. This must have a mandatory timeout of 60 seconds.
- Restore to Factory Fresh Settings: Restore the device settings to fresh state (also performs leave).
- Pair Devices (End Device Bind Request): Bind to any device you can find matching clusters on. This will toggle the bind each time you do it. The ZigBee coordinator does the pairing.
- Enable Identify Mode: Sets the device in Identify mode for 60 seconds. This is used for adding devices to a group or create a scene.
- Group Nodes: Used to add devices to a group. This action sends the .Add group if Identifying. command. This adds all devices that are in .identify mode. to the group. The group ID is picked by the implementer.
- Create Scene: This action creates a scene using devices present in a group.

A device that implements the Identify client cluster must implement means for a user interaction to perform E-mode group commissioning initiated from that device. By user interaction on that device, the device shall be possible to:

- Set matching devices in identify mode.
- Perform group binding on selected matching devices.

An example of such an implementation could be:

The user interacts with a device implementing the Identify client cluster to make it enter E-mode group commissioning. When this mode is entered, the device performs the following:

- Find devices on the network that match services of a cluster on the device.
- Put the matched devices in identify mode one at a time.
- When a user interaction is performed, the device currently identifying is made subject to a group binding.

3.4.4 CORBA

CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate. The Object Management Group (OMG) is responsible for defining CORBA. The OMG comprises over 700 companies and organizations, including almost all the major vendors and developers of distributed object technology, including platform, database, and application vendors as well as software tool and corporate developers [22].

CORBA defines an architecture for distributed objects. The basic CORBA paradigm is that of a request for services of a distributed object. Everything else defined by the OMG is in terms of this basic paradigm.

The services that an object provides are given by its *interface*. Interfaces are defined in OMG's Interface Definition Language (IDL). Distributed objects are identified by object references, which are typed by IDL interfaces.

Three elements make possible the CORBA Architecture [23]:

- An object manager called ORB (Object Request Broker)
- Mechanisms to specify the interfaces: IDL (Interface Definition Language) and DII (Dynamic Invocation Interface)
- Binary protocols that will allow the communication between the ORBs: GIOP (General Inter-ORB Protocol) and IIOP (Internet Inter-ORB Protocol)

The ORB is the distributed service that implements the request to the remote object. It locates the remote object on the network, communicates the request to the object, waits for the results and when available communicates those results back to the client.

The ORB implements location transparency. Exactly the same request mechanism is used by the client and the CORBA object regardless of where the object is located. It might be in the same process with the client, down the hall or across the planet. The client cannot tell the difference.

CORBA 2.0 added interoperability as a goal in the specification. In particular, CORBA 2.0 defines a network protocol, called *IIOP* (Internet Inter-ORB Protocol), that allows clients using a CORBA product from any vendor to communicate with objects using a CORBA product from any other vendor. IIOP works across the Internet, or more precisely, across any TCP/IP implementation.

All CORBA based applications need to access to the IDL system type when they are running. This is necessary because the application need to know the types of value that are going to be used as request parameters. In the same way, the application must know all supported interface types that the objects use.

Some applications only need a static knowledge of the IDL system type, the IDL specification is compiled into the application programme language. However, other applications are unable to adopt this static definition of the interface. CORBA provides another method to define interfaces to these applications. The interfaces can be added to an interface repository service. This repository defines the operations to recover stored information at run-time. A client, using this interface registry can localize an unknown object in compilation time, ask for its interface and build a request to the ORB.

Another important part of the CORBA standard is the definition of a set of distributed services to support the integration and interoperation of distributed objects. The services, known as *CORBA Services* or COS, are defined on top of the ORB. That is, they are defined as standard CORBA objects with IDL interfaces, sometimes referred to as "Object Services." [22]:

- Object life cycle: Defines how CORBA objects are created, removed, moved, and copied
- Naming: Defines how CORBA objects can have friendly symbolic names
- Events Decouples: the communication between distributed objects
- Relationships: Provides arbitrary typed n-ary relationships between CORBA objects
- Externalization: Coordinates the transformation of CORBA objects to and from external media
- Transactions: Coordinates atomic access to CORBA objects
- Concurrency Control: Provides a locking service for CORBA objects in order to ensure serializable access
- Property: Supports the association of name-value pairs with CORBA objects
- Trader: Supports the finding of CORBA objects based on properties describing the service offered by the object
- Query: Supports queries on objects

3.5 Ontology languages

3.5.1 KEE

KEE is frame-based expert system, developed and marketed by Intellicorp in 1980's. Supports dynamic inheritance, multiple inheritance, polymorphism. Classes, metaclasses and objects are all treated alike. A class is an instance of a meta-class. Methods are written in LISP. Actions may be triggered when fields are accessed or modified. Extensive GUI integrates with objects. Can easily make object updates to be reflected on display or display selections to update fields. This can in turn trigger other methods or inference rules which may then update other parts of the display (Knowledge Engineering Environment (KEE), [2]).

3.5.2 KL-ONE

The first formalization of a semantic network based on logic was the frame-based language KL-ONE. It is the first description logic because, in addition to supporting frame representation, it was formalized logically with description-forming structures [3]. The primary unit of information corresponding to frame in KL-ONE is called a "concept", which denotes a set of objects. A Concept has a set of (syntactic) components, each denoting a property that must be true of each member of the set denoted by the "Concept". In particular, one type of component is a Roleset which is analogous to a "slot" in a "frame-like" language [4].

In the KL-ONE terminology a super class is said to subsume its subclasses by using subsume-relations. All concepts, except the top concept "Thing", must have at least one super class. Multiple inheritance is allowed.

In KL-ONE, descriptions are separated into two basic classes of concepts: primitive and defined (non-primitive). Primitives are domain concepts that are not fully defined. In this case, the properties of a concept are not sufficient to classify it. They may also be viewed as incomplete definitions. Defined concepts are complete definitions. In this case, the properties of a concept are necessary and sufficient to classify the concept.

Schmolze et al., explains primitive and non-primitive concepts by using sample concepts such as PERSON and PARENT [4]. PERSON is a primitive concept denoting the set of all persons. Primitive Concepts are interpreted as having essentially incomplete definitions, and thus, all concepts denoting "natural kinds" (e.g., people, elephants, chairs) are primitive.

- PERSON is a primitive Concept, is subsumed by
    - MAMMAL, and has a Roleset Birthdate with:
        - A number restriction of exactly one.
        - A value description of DATE.

Non-primitive Concepts are interpreted as being completely defined. An example is PARENT.

- PARENT is a non-primitive Concept, is subsumed by

- PERSON, and has a Roleset Child with:
  - A number restriction of one or more.
  - A value description of PERSON.

3.5.3 CLASSIC

CLASSIC is a knowledge representation (KR) system that follows the paradigm originally set out in the KL-ONE system: it concentrates on the definition of structured concepts, their organization into taxonomies, the creation and manipulation of individual instances of such concepts, and the key inferences of subsumption and classification [5].

The frames / concepts in CLASSIC are interpreted as descriptions rather than assertions. So for wine concept, if wine is defined as a drink with a number of other properties, then being a drink is a necessary part of being a wine, and no wine can violate this requirement. There are three kinds of formal objects in CLASSIC [5]:

- "Concepts", which are descriptions with potentially complex structure, formed by composing a limited set of description-forming operators (e.g. WHITE-FULL-BODIED-WINE).
- "Roles", which are simple formal terms for properties (e.g. grape might represent the grape(s) a wine is made from).
- "Individuals", which are simple formal constructs intended to directly represent objects in domain of interest.

3.5.4 KM

KM is a frame-based knowledge representation language with clear first-order logic semantics. KM can be used for creating knowledge bases including descriptions of things in the world as objects, events, properties and relationships; and descriptions of the rules that allow us to reason about things in the world. It contains sophisticated machinery for reasoning, including selection by description, unification, classification, and reasoning about actions using a situations mechanism. KM interpreter is implemented based on Lisp language and interpreter. The language is developed and implemented by Knowledge Systems Research Group in Department of Computer Sciences, The University of Texas at Austin. [6].

As an example, we can describe a frame in KM, showing the class "Buy", with its slots and values (some of which are themselves frames) as below:

- All "buy" events have
  - A "buyer" and a "seller" (both of type agent)

- o An object which is bought
- o Some money equal to the cost of the object
- o Two "give" subevents, in which:
  - ▪ The buyer gives the money to the seller
  - ▪ The seller gives the object to the buyer.

This definition is represented in KM language as below:

```
(Buy  has  (superclasses  (Event)))  ;  Properties  of  the  class  ('owns'
properties)

(every  Buy  has                    ;  Properties  of  its  members  ('template'
properties)
  (buyer ((a Agent)))
  (object ((a Thing)))
  (seller ((a Agent)))
  (money ((the cost of (the object of Self))))
  (subevent1 ((a Give with
               (agent ((the buyer of Self)))
               (object ((the money of Self)))
               (rcpt ((the seller of Self))))))
  (subevent2 ((a Give with
               (agent ((the seller of Self)))
               (object ((the object of Self)))
               (rcpt ((the buyer of Self)))))))
```

3.5.5 F-logic

Michael Kifer **¡Error! No se encuentra el origen de la referencia.** proposed a language called F-logic (Frame Logic) that accounts in a clean and declarative fashion for most of the structural aspects of object oriented and frame based languages. These features include object identity, complex objects, inheritance, polymorphic types, query methods, encapsulation and others. He claims that one of the main problems with the object oriented approach is the lack of logical semantics that traditionally has been playing an important role in database programming languages while deductive databases rely on a flat data model and do not support data abstraction. F-logic combines the two approaches **¡Error! No se encuentra el origen de la referencia.**.

Major strengths of F-logic are extensibility and his capacities to directly represent fundamental concepts that come from object oriented programming and frame based languages. F-logic makes a number of central aspects of object oriented programming to become compatible with logic paradigm. F-Logic main weakness is related to mathematic and logic concepts needed to program in this language. F-logic does not possess cardinality restrictions.

3.5.6 CycL

CycL is an ontology language used by Cyc project, an artificial intelligence project that attempts to assemble a comprehensive ontology and knowledge base of everyday common sense knowledge, with the goal of enabling AI applications to perform human-like reasoning. The original version of CycL was a frame language, but the modern version is a declarative language based on classical first-order logic, with extensions for modal operators and higher order quantification. CycL has some basic ideas:

- Naming the constants used to refer to information for represented concepts.
- Grouping the constants together in a generalization/specialization hierarchy (usually called categorization).
- Stating general rules that support inference about the concepts.
- The truth or falsity of a CycL sentence is context-relative; these contexts are represented in CycL as Microtheories.

It has six expression types: Constants, Formulas and Truth-function, Function-denotional, Variables and Quantifiers. CycL's major strengths are expressiveness, precision, meaning and use-neutral representation and it is major weakness is the focus within the Cyc project has been on the engineering of large common sense knowledge bases, and not on the advancement of deductive reasoning technology.

3.5.7 LOOM

Loom is a language and environment for constructing intelligent applications. The heart of Loom is a knowledge representation system that is used to provide deductive support for the declarative portion of the Loom language. Declarative knowledge in Loom consists of definitions, rules, facts, and default rules. A deductive engine called a classifier utilizes forward-chaining, semantic unification and object-oriented truth maintenance technologies in order to compile the declarative knowledge into a network designed to efficiently support on-line deductive query processing [8].

3.5.8 KIF

KIF (Knowledge Interchange Format) is a computer-oriented language for the interchange of knowledge among disparate programs. It has declarative semantics (i.e. the meaning of expressions in the representation can be understood without appeal to an interpreter for manipulating those expressions); it is logically comprehensive (i.e. it provides for the expression of arbitrary sentences in the first-order predicate calculus); it provides for the representation of knowledge about knowledge [9].

3.5.9 Ontolingua

Ontolingua was created in 1992 at Stanford University and it's a language based in KIF (Knowledge Interchange Format). Combines frames paradigm and first order predicates. Beyond all the languages used to represent ontologies, Ontolingua language is the one with the biggest expressiveness. It can represent concepts, concepts taxonomies, n-ary relationships, axioms, instances and procedures. Ontolingua doesn't permit reasoning [10].

3.5.10 SHOE

SHOE was developed in 1996 at University of Maryland and stands for Simple HTML Ontology Extensions. SHOE is a small extension to HTML which allows web page authors to annotate their web documents with machine-readable knowledge. SHOE makes real intelligent agent software on the web possible. SHOE is a language in which categories, relationships, attributes, inferences, etc. can be defined by ontologies, but SHOE itself does not define them. This is the job of ontology designers for specific tasks or domains. SHOE was designed with the needs of the web in mind. It has limited semantics to make it possible to handle large amounts of data. However, simple database semantics are not enough for web data. SHOE provides true knowledge-base semantics. It has a variety of mechanisms that try to deal with the fact that the data out there is distributed and under no one's total control [10].

3.5.11 RDF / RDFS

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web [11].

The underlying structure of any expression in RDF is a collection of triples, each consisting of a subject, a predicate (also called a property) and an object [12].

RDF Schema (or RDFS) is an extensible knowledge representation language, providing basic elements for the description of ontologies, otherwise called RDF vocabularies, intended to structure RDF resources. The first version was published by W3C in April 1998, and the final W3C recommendation was released in February 2004.

In addition to inheriting basic features from Frame Systems, RDFS provides ontology constructs that make relations less dependent on concepts: users can define relations as an instance of rdf:Property, describe inheritance relations between relations using rdfs:subPropertyOf, and then associate defined relations with classes using rdfs:domain or rdfs:range [13].

3.5.12 DAML+OIL

The DARPA Agent Markup Language (DAML) is a agent markup language developed by the Defense Advanced Research Projects Agency (DARPA) for the semantic web. The DAML language was developed as an extension to XML and the RDF.

RDF Schema (RDFS) in particular is recognizable as an ontology/knowledge representation language: it talks about classes and properties (binary relations), range and domain constraints (on properties), and subclass and subproperty (subsumption) relations. RDFS is, however, a very primitive language (the above is an almost complete description of its functionality), and more expressive power would clearly be necessary/desirable in order to describe resources in sufficient detail. Moreover, such descriptions should be amenable to automated reasoning if they are to be used effectively by automated processes, e.g., to determine the semantic relationship between syntactically different terms. The recognition of these requirements has led to the development of DAML+OIL, an expressive Web ontology language. DAML+OIL is the result of a merger between DAML-ONT, a language developed as part of the US DARPA Agent Markup Language (DAML) programme and OIL (the Ontology Inference Layer) [14].

3.5.13 OWL

OWL (Web Ontology Language), which has been developed by W3C Web Ontology Working Group, is a formal knowledge representation language for defining ontologies. OWL is extensively used in the life sciences community, where it has rapidly become a de facto standard for ontology development and data interchange. OWL has evolved primarily from Description Logics and frames.

In fact OWL recommendation includes three languages, namely OWL Lite, OWL DL, and OWL Full, in the order of expressive power. Briefly, as stated by Antoniou [15]:

- OWL Lite adds the possibility to express definitions and axioms, together with a limited use of properties to define classes;
- OWL DL supports those users who want the maximum expressiveness while retaining good computational properties;
- OWL Full is meant for users who want maximum expressiveness with no computational guarantees.

3.5.14 OWL Lite

OWL Lite is a variant of description logic languages. It can be seen as a variant of SHIF(D) logic

Language. The SHIF(D) language allows complex class descriptions, including conjunction, disjunction, negation, existential and universal value restrictions for roles, role hierarchies, transitive roles, inverse roles, a restricted form of cardinality restrictions (cardinality 0 or 1) and (limited) support for concrete domains [16].

If tractable inference and simple syntax is important than OWL Lite is the choice for defining an ontology [17]. OWL Lite is a syntactic subset of OWL DL. Some of the constructors and axioms allowed in OWL DL is not allowed, but it has been shown that by using the allowed language constructs it is possible to maintain most of the expressiveness of OWL DL [18].

3.5.15 OWL DL

OWL DL is the most well-known and researched variant of OWL languages. The only descriptions in OWL DL that cannot be expressed in OWL Lite are those containing individuals and cardinalities higher than one [16]. The extension from OWL Lite to OWL DL lifts the restriction on cardinality constraints to have only 0/1 values. Technically, it restricts the form of number restrictions to be unqualified, and adds a simple form of Datatypes. From the perspective of language constructs, OWL DL allows additional constructs on top of OWL Lite. Owl has closely the same expressiveness as SHOIN(D) description language.

3.5.16 OWL Full

OWL Full is meant for the ones who want maximum expressiveness, syntactic freedom, and full compatibility with RDF Schema, with no computational guarantees. OWL Full provides a more complete integration with RDF, but its formal properties are less well understood, and key inference problems would certainly be much harder to compute. As being the most expressive species of OWL, it layers on top of both RDFS and OWL DL. OWL Full is fully upward-compatible with RDF, both syntactically and semantically: any legal RDF document is also a legal OWL Full document, and any valid RDF/RDF Schema conclusion is also a valid OWL Full conclusion.

3.5.17 OWL 2 – The Next Step

Although OWL 1 has been widely accepted by the industry, it had its shortcomings, coming from its design. To tackle these problems a working group was formed in W3C to settle down the revisions from the theoreticians, implementers and users. The changes were first filed with the name OWL 1.1 but the changes were substantial enough to call the language with a different name, namely OWL 2, which addresses the shortcomings of OWL 1 in the following areas [19]:

- Expressivity Limitations
- Syntax Issues
- Metamodeling
- Annotations
- OWL Semantics

W3C Working Group continues to work on OWL 2, but there are some draft specifications. Some tools like Protégé and Pellet supports OWL 2. The latest (as of November 2008) of OWL 2 specifications is filed under http://www.w3.org/TR/2008/WD-owl2-syntax-20081008/

4. Applied methodology.

In this chapter, it could be described which methodologies or tools will be used to define the ontology prototype for device awareness.

It is not an only and correct way to develop ontologies, but almost in all the utilized methodologies there are some basic rules that are fulfilled [19]:

- There is not one correct way to model a domain, there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.
- Ontology development is necessarily an iterative process.
- Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain.

Following these rules, and applying them to the objective of modelling with this ontology a specific part of the reality of eDIANA Platform, in particular de device awareness between Cell level components and devices, the next steps have been completed in some iterations. Nevertheless the explicitness that an ontology can achieve is almost infinite,

4.1 Determine the domain and scope of the ontology

The first step developing ontologies is to determine the domain and scope of the ontology, to answer next questions can help to complete this task:

- What is the domain that the ontology will cover?

The domain of the ontology is clearly the eDIANA Platform, more specifically, the Cell level, covering the knowledge of device awareness of this level components and devices.

- For what we are going to use the ontology?

The main use of the ontology will be to provide a unified description of the domain to all the members of the project, as well as to have a knowledge repository where later works can be reflected.

- For what types of questions the information in the ontology should provide answers?

The questions would be referred to topics like these:

- o Environment understanding
- o Types of components and devices
- o Profiles of components and devices
- o Identification of components and devices
- o Ad-hoc detection
- o Components and devices services

4.2 Enumerate important terms in the ontology

This step is a kind of brainstorming about all concepts involved in the domain, without paying much attention to the relation between concepts, terms, etc. Once establish a term, it is analyzed to discover more new issues that complete the original term.

Next steps will be in charge of analyze those terms, concepts, issues and hierarchize them.

4.3 Define the classes and the class hierarchy

It has been selected a top-down development process for the first iteration, starting from the eDIANA Platform and the most high level: MacroCell, to the next level: Cell.

In this step, all the terms provided by previous step have been categorized into classes, and these classes organized in the resultant hierarchy. During this step the analysis of the proposed terms causes the addition of more related concepts.

Following iterations have combined top-down and down-top processes in order to find missing concepts.

4.4 Define the properties of classes—slots

In this step, the analysis has gone deep into the classification of the terms, finding out which of them correspond better to properties of classes, than to relational classes or entities themselves.

This step has been made at the same time as the definition of the classes and their hierarchy. Sometimes, the discovering of a new property caused the definition of classes or subclasses, this is the reason because several iterations have been made through each class and properties.

4.5 Define the facets of the properties

This step has consisted of defining more precisely the characteristics, features or facets of the properties of the classes, including relational properties and feature properties.

The most common values of the facets are:

- String
- Number
- Boolean
- Enumerated
- Ranges

5. Semantic structure of the ontology.

After the analysis, this chapter could contain the obtained results. The detailed description of the semantic structure or architecture based on three layers o views: information, services and devices.

The following section will cover these three semantic layers, defining the semantics of each concept in the ontology. Additionally, each section is provided with a tree diagram showing the concept taxonomy of the eDIANA context awareness ontology. Relations between classes are described in detail in the description of each ontology node.

5.1 Information layer.

The information layer of the eDIANA ontology for device awareness contains different categories of information that will be referenced by the elements defined in the other two layers, namely services layer and devices layer. The elements contained in this layer will be described in the following sections.

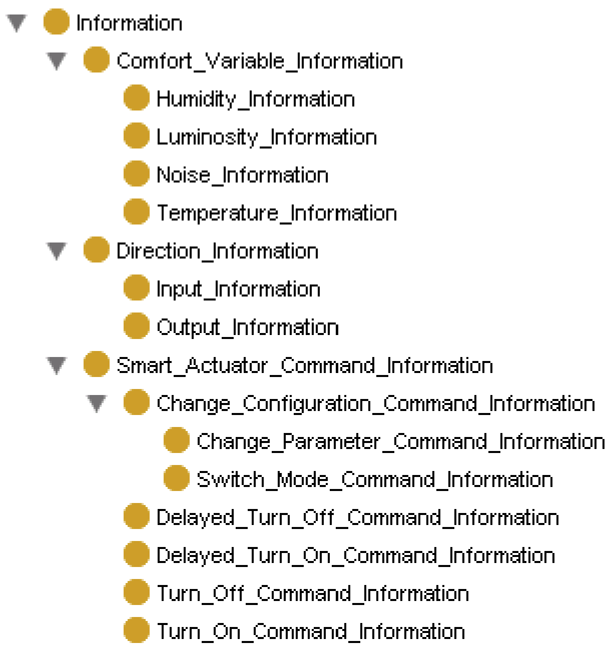

Information
Comfort_Variable_Information
Humidity_Information
Luminosity_Information
Noise_Information
Temperature_Information
Direction_Information
Input_Information
Output_Information
Smart_Actuator_Command_Information
Change_Configuration_Command_Information
Change_Parameter_Command_Information
Switch_Mode_Command_Information
Delayed_Turn_Off_Command_Information
Delayed_Turn_On_Command_Information
Turn_Off_Command_Information
Turn_On_Command_Information

Figure 5-1 Information layer

5.1.1 Information

The Information class is the top level class of the information layer.

Generalizations

- Thing

Object Properties

None

Data Properties

None

Restrictions

None

5.1.2 Direction_Information

The Direction_Information class is the top level class for the classes that describe the semantic direction of the data in a user interface device.

Generalizations

- Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.3 Input_Information

The Input_Information class provides the semantics of a user interface device intended for providing information to the CDC.

Generalizations

- Direction_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.4 Output_Information

The Output_Information class provides the semantics of a user interface device intended for rendering information obtained from the CDC.

Generalizations

- Direction_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.5 Comfort_Variable_Information

The Comfort_Variable_Information class is the top level class for the classes that describe the semantics of the different comfort variables addressed in the eDIANA user policies. These classes will be referenced by actuators, sensors and appliances.

Generalizations

- Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.6 Humidity_Information

The Humidity_Information class refers to a simple device (actuator, sensor or appliance) directly or indirectly related with the humidity comfort property (e.g. humidity sensors, dehumidifiers, etc.). This information will make the CDC aware of the fact that this component provides information usable by the control algorithms related to humidity.

Generalizations

- Comfort_Variable_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.7 Luminosity_Information

The Luminosity_Information class refers to a simple device (actuator, sensor or appliance) directly or indirectly related with the luminosity comfort property (e.g. lighting sensors, lamps, blinds, etc.). This information will make the CDC aware of the fact that this component provides information usable by the control algorithms related to luminosity.

Generalizations

- Comfort_Variable_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.8 Noise_Information

The Noise_Information class refers to a simple device (actuator, sensor or appliance) directly or indirectly related with the noise comfort property (e.g. sound sensors, noisy appliances such as washing machines, etc.). This information will make the CDC aware of the fact that this component provides information usable by the control algorithms related to noise.

Generalizations

- Comfort_Variable_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.9 Temperature_Information

The Temperature_Information class refers to a simple device (actuator, sensor or appliance) directly or indirectly related with the temperature comfort property (e.g. thermometers, heating systems, air conditioning systems, etc.). This information will make the CDC aware of the fact that this component provides information usable by the control algorithms related to temperature.

Generalizations

- Comfort_Variable_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.10 Smart_Actuator_Command_Information

The Smart_Actuator_Command_Information class is the top level class for the classes that define the semantics of the different command types that a smart actuator could process.

Generalizations

- Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.11 Change_Configuration_Command_Information

The Change_Configuration_Command_Information class categorizes the commands related with configuration changes.

Generalizations

- Smart_Actuator_Command_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.12 Change_Parameter_Command_Information

The Change_Parameter_Command_Information class is a kind of configuration change command. Specifically, this class refers to a concrete configuration change that affects a concrete parameter of the smart actuator, such as the selected program in a washing machine.

Generalizations

- Change_Configuration_Command_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.13 Switch_Mode_Command_Information

The Switch_Mode_Command_Information class is a kind of configuration change command. Specifically, this class refers to a concrete configuration change that affects operating mode of the smart actuator, such as low power mode.

Generalizations

- Change_Configuration_Command_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.14 Delayed_Turn_Off_Command_Information

The Delayed_Turn_Off_Command_Information class refers to a command capable of setting an actuator to turn off a device at a given moment in time.

Generalizations

- Smart_Actuator_Command_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.15 Delayed_Turn_On_Command_Information

The Delayed_Turn_On_Command_Information class refers to a command capable of setting an actuator to turn on a device at a given moment in time.

Generalizations

- Smart_Actuator_Command_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.16 Turn_Off_Command_Information

The Turn_Off_Command_Information class refers to a command intended to turn a device of the platform off.

Generalizations

- Smart_Actuator_Command_Information

Object Properties

None

Data Properties

None

Restrictions

None

5.1.17 Turn_On_Command_Information

The Turn_On_Command_Information class refers to a command intended to turn a device of the platform on.

Generalizations

- Smart_Actuator_Command_Information

Object Properties

None

Data Properties

None

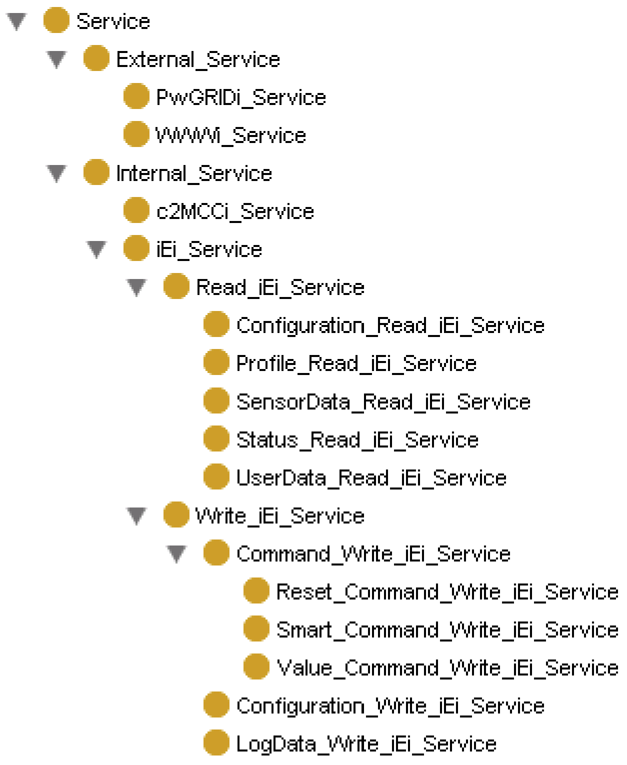
Restrictions

None

5.2 Service layer

The Service structure layer on the eDIANA Context Awareness ontology focuses on the definition of the different interfaces present in eDIANA. This definition has been made at a very high level, since the concrete definition of the interfaces hasn't been created yet; however, the structure proposed in this deliverable for the Service structure layer can be easily extended during posterior phases of the project maintaining consistency.

The classes defined in the Service structure layer will be further described in the following sections.



```
▼ Service
    ▼ External_Service
        PwGRIDi_Service
        WWWi_Service
    ▼ Internal_Service
        c2MCCi_Service
        ▼ iEi_Service
            ▼ Read_iEi_Service
                Configuration_Read_iEi_Service
                Profile_Read_iEi_Service
                SensorData_Read_iEi_Service
                Status_Read_iEi_Service
                UserData_Read_iEi_Service
            ▼ Write_iEi_Service
                ▼ Command_Write_iEi_Service
                    Reset_Command_Write_iEi_Service
                    Smart_Command_Write_iEi_Service
                    Value_Command_Write_iEi_Service
                Configuration_Write_iEi_Service
                LogData_Write_iEi_Service
```

Figure 5-2 Service layer

5.2.1 Service

The Service class is the top level class of the services layer.

Generalizations

- Thing

Object Properties

None

Data Properties

None

Restrictions

None

5.2.2 External_Service

The External_Service class is the top level class of the services defined between the eDIANA platform and external.

Generalizations

- Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.3 PwGRIDi_Service

The PwGRIDi_Service class provides the semantics of the interface between the eDIANA platform and the power grid of an energy provider.

Generalizations

- External_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.4 WWWi_Service

The WWWi_Service class provides the semantics of the interface between the eDIANA platform and the Internet. More concretely, this class refers to the service between the platform and an ISP.

Generalizations

- External_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.5 Internal_Service

The Internal_Service class is the top level class of the services defined between components inside the eDIANA platform.

Generalizations

- Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.6 Internal_Service

The Internal_Service class is the top level class of the services defined between components inside the eDIANA platform.

Generalizations

- Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.7 c2MCCi_Service

The c2MCCi_Service class provides the semantics of the interface between a Macro Cell Concentrator and a Cell Concentrator or between two Macro Cell Concentrators.

Generalizations

- Internal_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.8 iEi_Service

The iEi_Service class provides the semantics of the interface between a Cell Concentrator and any device connected to the Cell. This is only a top level class further refined to define specific iEi services applicable to the different eDIANA device types.

Generalizations

- Internal_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.9 Read_iEi_Service

The Read_iEi_Service class refers to all the iEi services intended to read a data value from the remote device (e.g. data from a sensor, configuration parameters, etc.).

Generalizations

- iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.10 Configuration_Read_iEi_Service

The Configuration_Read_iEi_Service class refers to services intended to get the current configuration of a device.

Generalizations

- Read_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.11 UserData_Read_iEi_Service

The UserData_Read_iEi_Service class refers to services provided by user interface devices to read the commands from the users.

Generalizations

- Read_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.12 Profile_Read_iEi_Service

The Profile_Read_iEi_Service class refers to the service intended to provide to the Cell Concentrator the characteristics of a device inside a Cell.

Generalizations

- Read_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.13 Status_Read_iEi_Service

The Status_Read_iEi_Service class refers to the iEi services intended to read the current status of a remote device.

Generalizations

- Read_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.14 SensorData_Read_iEi_Service

The SensorData_Read_iEi_Service class refers to the iEi services intended to retrieve a data value from a remote sensor device.

Generalizations

- Read_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.15 Write_iEi_Service

The Write_iEi_Service class refers to all the iEi services intended to write data to the remote device (e.g. actuator commands, new configuration parameters, etc.).

Generalizations

- iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.16 LogData_Write_iEi_Service

The LogData_Write_iEi_Service class refers to the services intended to write log data to the remote device (e.g. a web server, a database, etc.).

Generalizations

- Write_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.17 Configuration_Write_iEi_Service

The Configuration_Write_iEi_Service class refers to all the iEi services intended to modify the configuration parameters of a remote device.

Generalizations

- Write_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.18 Command_Write_iEi_Service

The Command_Write_iEi_Service class refers to all the iEi services intended to send commands to remote actuator devices.

Generalizations

- Write_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.19 Reset_Command_Write_iEi_Service

The Reset_Command_Write_iEi_Service class refers to an iEi service intended to reset a remote device to its initial state.

Generalizations

- Command_Write_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.20 Smart_Command_Write_iEi_Service

The Smart_Command_Write_iEi_Service class refers to all the iEi services intended to send commands to remote smart actuator devices.

Generalizations

- Command_Write_iEi_Service

Object Properties

None

Data Properties

None

Restrictions

None

5.2.21 Value_Command_Write_iEi_Service

The Value_Command_Write_iEi_Service class refers to all the iEi services intended to send value commands to remote value driven actuator devices (e.g. percentage valve opener, variable light actuator).

Generalizations

- Command_Write_iEi_Service

Object Properties

None

Data Properties

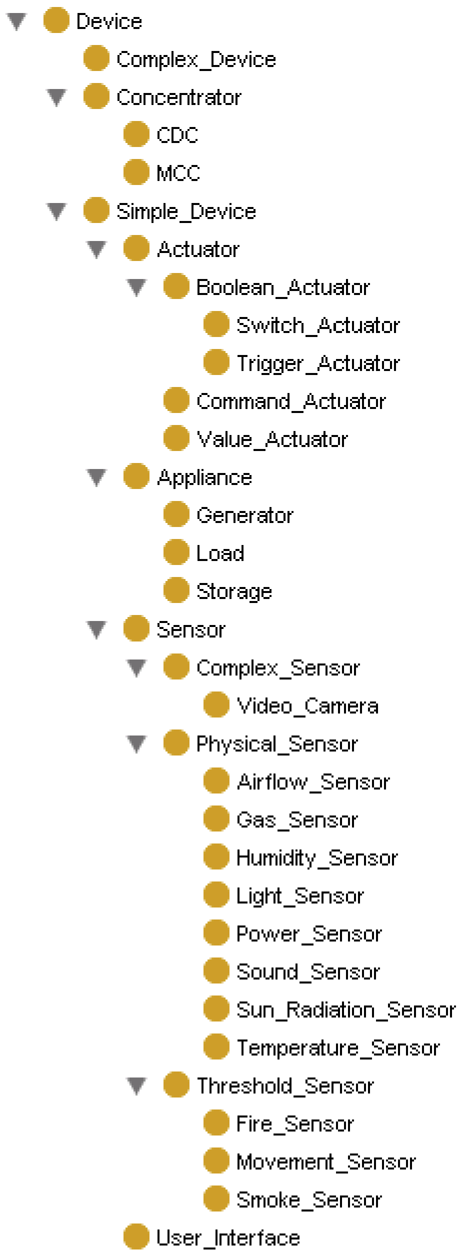
None

Restrictions

None

5.3 Device layer

The Device layer of the eDIANA ontology for device awareness contains different categories of devices that compose the eDIANA platform. This taxonomy of devices will enable device awareness services and plug-and-play services by characterizing the devices, their properties and their interfaces. The elements contained in this layer will be described in the following sections.



- Device
  - Complex_Device
  - Concentrator
    - CDC
    - MCC
  - Simple_Device
    - Actuator
      - Boolean_Actuator
        - Switch_Actuator
        - Trigger_Actuator
      - Command_Actuator
      - Value_Actuator
    - Appliance
      - Generator
      - Load
      - Storage
    - Sensor
      - Complex_Sensor
        - Video_Camera
      - Physical_Sensor
        - Airflow_Sensor
        - Gas_Sensor
        - Humidity_Sensor
        - Light_Sensor
        - Power_Sensor
        - Sound_Sensor
        - Sun_Radiation_Sensor
        - Temperature_Sensor
      - Threshold_Sensor
        - Fire_Sensor
        - Movement_Sensor
        - Smoke_Sensor
    - User_Interface

Figure 5-3 Device layer

5.3.1 Device

The Device class is the top level class of the information layer.

Generalizations

- Thing

Object Properties

- provided_services: Service (multiple).

  The *provided_services* property relates a device with the set of services that it provides.

Data Properties

- location: string (single).

  The *location* property provides the information of the location where this device is placed.

Restrictions

None

5.3.2 Simple_Device

The Simple_Device class represents any kind of simple device types (namely sensor, actuator, appliance and user interface).

Generalizations

- Device

Object Properties

None

Data Properties

None

Restrictions

None

5.3.3 Actuator

The Actuator class represents any kind of actuator devices, such as valve openers, light switches, etc.

Generalizations

- Simple_Device

Object Properties

- related_appliance: Appliance (single).

  The *related_appliance* property relates the actuator device with the appliance (e.g. motor, washing machine, lamp, etc.) this actuator affects to.

- related_comfort_properties: Comfort_Variable_Information (multiple).

  The *related_comfort_properties* property relates an actuator with any comfort variables that depend on it. This information will be used by the CDC to be able to apply different control algorithms to the different actuators, sensors and appliances taking into account their dependencies with user comfort parameters.

Data Properties

None

Restrictions

[1] An actuator may only include in its *provided_services* property services from the following list: *Profile_Read_iEi_Service, Configuration_Read_iEi_Service, Status_Read_iEi_Service, Command_Write_iEi_Service or Configuration_Write_iEi_Service*. Other services are not allowed for an actuator.

5.3.4 Boolean_Actuator

The Boolean_Actuator class represents actuators that may only take two different states: on and off.

Generalizations

- Actuator

Object Properties

None

Data Properties

None

Restrictions

[1] A boolean actuator may not include in its *provided_services* property services from the following list: *Smart_Command_Write_iEi_Service*. This restriction refines the restriction introduced in the parent class Actuator.

5.3.5 Switch_Actuator

The Switch_Actuator class represents a boolean actuator that can be remotely turned on and off.

Generalizations

- Boolean_Actuator

Object Properties

None

Data Properties

None

Restrictions

None

5.3.6 Trigger_Actuator

The Trigger_Actuator class represents a boolean actuator that after being activated requires human interaction to be reset to its initial state (e.g. fire alarm activator).

Generalizations

- Boolean_Actuator

Object Properties

None

Data Properties

None

Restrictions

None

5.3.7 Value_Actuator

The Value_Actuator class represents actuator devices that receive a value within a value range as input, such as valve openers, variable lighting lamp actuators, etc.

Generalizations

- Actuator

Object Properties

None

Data Properties

- is_digital: boolean (single).

  The *is_digital* property defines whether the actuator receives a digital value or an analog value.

- max_value: double (single).

  The *max_value* property defines the maximum value the actuator can accept.

- min_value: double (single).

  The *min_value* property defines the minimum value the actuator can accept.

Restrictions

[1] A value actuator may not include in its *provided_services* property services from the following list: *Smart_Command_Write_iEi_Service, Reset_Command_Write_iEi_Service*. This restriction refines the restriction introduced in the parent class Actuator.

5.3.8 Smart_Actuator

The Smart_Actuator class represents actuator devices capable or processing complex commands and provide complex actuating mechanisms. Examples of smart actuators can be found in intelligent appliances, such as programmable washing machines or air conditioning systems.

Generalizations

- Actuator

Object Properties

- accepted_commands: Smart_Actuator_Command_Information (multiple).

The *accepted_commands* property relates the smart actuator device with the command set it is capable of managing.

Data Properties

None

Restrictions

[1] A value actuator may not include in its *provided_services* property services from the following list: *Value_Command_Write_iEi_Service, Reset_Command_Write_iEi_Service*. This restriction refines the restriction introduced in the parent class Actuator.

5.3.9 Sensor

The Sensor class represents any kind sensing device, such as power consumption sensors, light sensors, etc. More complex sensing devices, such as video cameras are also included in this category.

Generalizations

- Simple_Device

Object Properties

- related_comfort_properties: Comfort_Variable_Information (multiple).

The *related_comfort_properties* property relates a sensor with any comfort variables that depend on it. This information will be used by the CDC to be able to apply different control algorithms to the different actuators, sensors and appliances taking into account their dependencies with user comfort parameters.

Data Properties

- is_calibrable: boolean (single).

  The *is_calibrable* property defines whether a sensor can be calibrated or not by the CDC controls.

- coverage_area: double (single).

  The *coverage_area* property provides the circular area value (in m^2) from the location of this device that is covered by the sensor.

Restrictions

[1] A sensor may only include in its *provided_services* property services from the following list: *Profile_Read_iEi_Service, Configuration_Read_iEi_Service, Status_Read_iEi_Service, SensorData_Read_iEi_Service, Reset_Command_Write_iEi_Service or Configuration_Write_iEi_Service*. Other services are not allowed for a sensor.

5.3.10 Physical_Sensor

The Physical_Sensor class represents any kind sensing device intended to measure a physical magnitude, such as power consumption, light, temperature etc.

Generalizations

- Sensor

Object Properties

None

Data Properties

- unit: String (single).

  The *unit* property defines unit which characterizes the physical value measured by the sensor.

- max_value: double (single).

The *max_value* property defines the maximum value the sensor can provide.

- min_value: double (single).

The *min_value* property defines the minimum value the sensor can provide.

Restrictions

None

5.3.11 Airflow_Sensor

The Airflow_Sensor class represents a sensing device intended to measure the flow of air.

Generalizations

- Physical_Device

Object Properties

None

Data Properties

None

Restrictions

None

5.3.12 Gas_Sensor

The Gas_Sensor class represents a sensing device intended to measure the concentration of a certain gas in the air.

Generalizations

- Physical_Device

Object Properties

None

Data Properties

- gas_type: String (single).

  The *gas_type* property defines which gas type the sensor measures.

Restrictions

None

5.3.13 Humidity_Sensor

The Humidity_Sensor class represents a sensing device intended to measure the relative humidity of the air.

Generalizations

- Physical_Device

Object Properties

None

Data Properties

None

Restrictions

[1] The *related_comfort_properties* of a humidity sensor must always be set to *Humidity_Information*.

5.3.14 Light_Sensor

The Light_Sensor class represents a sensing device intended to measure the luminosity in an area.

Generalizations

- Physical_Device

Object Properties

None

Data Properties

None

Restrictions

[1] The *related_comfort_properties* of a light sensor must always be set to *Luminosity_Information*.

5.3.15 Power_Sensor

The Power_Sensor class represents a sensing device intended to measure the power produced or consumed at a certain appliance in the eDIANA platform.

Generalizations

- Physical_Device

Object Properties

None

Data Properties

- is_generated: boolean (single).

  The *is_generated* property defines whether the data provided by the sensor refers to consumed or generated power.

Restrictions

None

5.3.16 Sound_Sensor

The Sound_Sensor class represents a sensing device intended to measure the sound intensity in an area.

Generalizations

- Physical_Device

Object Properties

None

Data Properties

None

Restrictions

[1] The *related_comfort_properties* of a sound sensor must always be set to *Noise_Information*.

5.3.17 Sun_Radiation_Sensor

The Sun_Radiation_Sensor class represents a sensing device intended to measure the intensity of the sun radiation in an area.

Generalizations

- Physical_Device

Object Properties

None

Data Properties

None

Restrictions

None

5.3.18 Temperature_Sensor

The Temperature_Sensor class represents a sensing device intended to measure the temperature of a certain area.

Generalizations

- Physical_Device

Object Properties

None

Data Properties

None

Restrictions

[1] The *related_comfort_properties* of a temperature sensor must always be set to *Temperature_Information*.

5.3.19 Threshold_Sensor

The Threshold_Sensor class represents any sensing device that triggers when a certain variable surpasses a threshold value. Examples of this kind of sensors are smoke sensors, fire sensors, etc.

Generalizations

- Sensor

Object Properties

None

Data Properties

None

Restrictions

None

5.3.20 Fire_Sensor

The Fire_Sensor class represents a sensing device that triggers when the temperature surpasses a threshold value.

Generalizations

- Threshold_Sensor

Object Properties

None

Data Properties

None

Restrictions

None

5.3.21 Smoke_Sensor

The Smoke_Sensor class represents any sensing device that triggers when the smoke level surpasses a threshold value.

Generalizations

- Threshold_Sensor

Object Properties

None

Data Properties

None

Restrictions

None

5.3.22 Movement_Sensor

The Movement_Sensor class represents any sensing device that triggers when the movement of an object is detected within the coverage area of the sensor.

Generalizations

- Threshold_Sensor

Object Properties

None

Data Properties

None

Restrictions

None

5.3.23 Complex_Sensor

The Complex_Sensor class represents any sensing device that provides complex data, such as a video stream.

Generalizations

- Sensor

Object Properties

None

Data Properties

None

Restrictions

None

5.3.24 Video_Camera

The Video_Camera class represents device which provides a video stream captured in a certain area.

Generalizations

- Complex_Sensor

Object Properties

None

Data Properties

None

Restrictions

None

5.3.25 Appliance

The Appliance class is represents any kind of appliance used in the eDIANA platform (e.g. televisions, washing machines, lamps, air conditioning systems...). These appliances may consume, produce or store energy.

Generalizations

- Simple_Device

Object Properties

- related_comfort_properties: Comfort_Variable_Information (multiple).

  The *related_comfort_properties* property relates an appliance with any comfort variables that depend on it (e.g. a motor controlling the blinds of a window is related to the lighting management). This information will be used by the CDC to be able to apply different control algorithms to the different actuators, sensors and appliances taking into account their dependencies with user comfort parameters.

Data Properties

- is_programmable: boolean (single).

  The *is_programmable* property defines whether an appliance can be programmed or not by the CDC to start/stop at a concrete time instant.

- priority: int (single).

  The *priority* property provides the CDC information on the importance of the appliance for a user. For example, higher priority appliances will be the last to be turned off by the CDC when a cell is exceeding its power consumption limits.

Restrictions

[1] An appliance may only include in its *provided_services* property services from the following list: *Profile_Read_iEi_Service*. Other services are not allowed for an appliance.

5.3.26 Generator

The Generator class represents an appliance used in the eDIANA platform capable of generating electric power, such as photovoltaic panels.

Generalizations

- Appliance

Object Properties

None

Data Properties

None

Restrictions

None

5.3.27 Storage

The Storage class represents an appliance used in the eDIANA platform intended to store electric power, such as batteries.

Generalizations

- Appliance

Object Properties

None

Data Properties

None

Restrictions

None

5.3.28 Load

The Load class represents an appliance used in the eDIANA platform that consumes electric power to develop its functionalities, such as a lamp or a washing machine.

Generalizations

- Appliance

Object Properties

None

Data Properties

None

Restrictions

None

5.3.29 User_Interface

The User_Interface class is represents any kind of simple device intended to be used by humans to obtain data from or to provide information to the Cell. Such devices can be web pages, consoles, control panels, etc.

Generalizations

- Simple_Device

Object Properties

- direction: Direction_Information (multiple).

  The *direction* property defines whether a user interface device provides input information, receives output information or both.

Data Properties

- is_graphical: boolean (single).

  The *is_graphical* property provides the CDC information on the graphical capabilities of the user interface device. The CDC may decide to send different data to output interfaces to graphical and non-graphical interfaces to optimize the performance of these devices.

Restrictions

[1] A user interface device may only include in its *provided_services* property services from the following list: *Profile_Read_iEi_Service, Configuration_Read_iEi_Service, Status_Read_iEi_Service, UserData_Read_iEi_Service, LogData_Write_iEi_Service or Configuration_Write_iEi_Service*. Other services are not allowed for a user interface device.

5.3.30 Complex_Device

The Complex_Device class is represents complex devices, such as smart washing machines, or variable lamps with light sensors. These devices can be described as compositions of simpler devices.

Generalizations

- Device

Object Properties

- composing_devices: Simple_Device (multiple).

  The *composing_devices* property relates a complex device with the simple devices that compose its full functionality.

Data Properties

None

Restrictions

[1] All the devices included in the *composing_devices* property must have the same location as the complex device they compose.

5.3.31 Concentrator

The Concentrator class represents the concentrator devices of the Cell and MacroCell levels of the eDIANA platform. These devices are not connected to the cells; instead concentrators will connect to each other and to external elements (such as the Internet and Power suppliers).

Generalizations

- Device

Object Properties

None

Data Properties

None

Restrictions

[1] The *provided_services* object property, inherited from Device, may only take the values c2MCCi_Service or subclasses of External_Service.

5.3.32 CDC

The CDC class represents the Cell level concentrator of the eDIANA platform.

Generalizations

- Concentrator

Object Properties

None

Data Properties

None

Restrictions

None

5.3.33 MCC

The MCC class represents the MacroCell concentrator of the eDIANA platform.

Generalizations

- Concentrator

Object Properties

None

Data Properties

None

Restrictions

None

6. Ontology prototype.

Together with this document, the prototype of the Ontology for Device Awareness is provided in the files:

- eDIANA_ContextAwareness.owl
- eDIANA_ContextAwareness.dot-input
- eDIANA_ContextAwareness.pprj
- eDIANA_ContextAwareness.repository

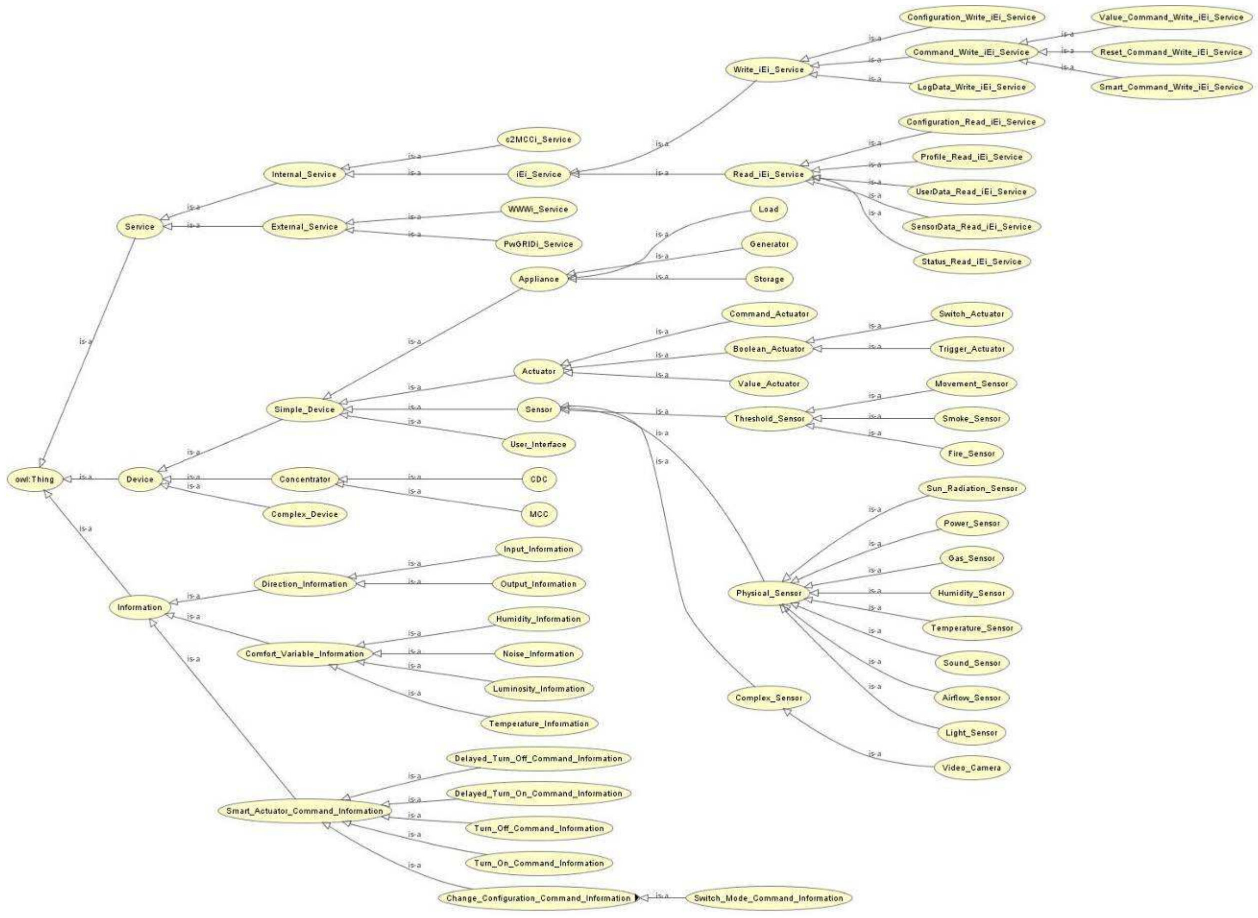


Figure 6-1 Ontology for Device Awareness

7. Conclusions

As has been said in the introduction chapter, the resultant ontology prototype should be the very start point to continue with the definition of the middleware level.

Acknowledgements

The eDIANA Consortium would like to acknowledge the financial support of the European Commission and National Public Authorities from Spain, Netherlands, Germany, Finland and Italy under the ARTEMIS Joint Technology Initiative.

References

[2] Knowledge Engineering Environment (KEE), The Encyclopedia of Computer Languages.
[3] Daconta, M. C. Obrst, L. J. Smith, K.T. 2003. The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management.
[4] Schmolze, J. G. Lipkis, T. A. Classification in the KL-ONE Knowledge Representation System.
[5] Brachman, R. J. McGuiness, D. L. Patel-Schneider, P. F. Resnick, L. A. Borgida, A. 1991. Living With Classic: When and How to Use a KL-ONE-like language.
[6] KM Project Website, http://www.cs.utexas.edu/~mfkb/km/
[7] Kifer, M. Lausen, G. Wu, J. 1995. Logical Foundations of Object-Oriented and Frame-Based Languages
[8] LOOM Project Website, http://www.isi.edu/isd/LOOM/
[9] Knowledge Interchange Format (KIF) Project Website, http://logic.stanford.edu/kif/kif.html
[10] The SEED research laboratory Website, http://seed.uma.pt/index.php?option=com_content&task=view&id=9&Itemid=51
[11] W3C Website, RDF primer, http://www.w3.org/TR/rdf-primer/
[12] W3C Website, Resource Description Framework (RDF): Concepts and Abstract Syntax, http://www.w3.org/TR/rdf-concepts/
[13] Ding, L. Kolari, P. Ding, Z. Avancha, S. Finin, T. Joshi, A. 2005. Using Ontologies in the Semantic Web: A Survey.
[14] Horrocks, I. 2002. DAML+OIL: A Description Logic for the Semantic Web.
[15] Antoniou, G. Franconi, E. Van Harmelen, F. Introduction to Semantic Web Ontology Languages.
[16] Bruijn, J. Polleres, A. Lara, R. Fensel, D. 2005. OWL.
[17] Horrocks, I. Patel-Schneider, P. F. Van Harmelen, F. From SHIQ and RDF to OWL. The Making of a Web Ontology Language.
[18] Horrocks, I. Patel-Schneider, P. F. McGuiness, D. L. Welty, C. A. OWL: A Description Logic Based Ontology Language for the Semantic Web.
[19] Cuenca, B. Horrocks, I. Motik, B. Parsia, B. Patel-Schneider, P. Sattler, U. OWL2: The Next Step for OWL.

[20] Noy, Natalya F. and McGuinness, Deborah L. Ontology Development 101: A Guide to Creating Your First Ontology. Stanford University
[21] OSGi Alliance. OSGi Service Platform Service Compendium
[22] ZigBee Home Automation Public Application Profile. Home Automation Profile Specification, ZigBee Alliance
[23] MageLang Institute. Introduction to CORBA 1998
[24] Common Object Request Broker Architecture (CORBA) *for embedded* Specification. Object Management Group, Inc.