Hiding complexity and heterogeneity of the physical world in smart living environments

Abstract
Continuous technological advances lead to computerize all the electronic devices and connect them in a network, so that in the next years physical and virtual worlds will be integrated and interoperate each other at the point that browsing the reality will be similar to browsing the Web. In other words, heterogeneous networked devices, services satisfying needs of people and living environments equipped with these devices and services, will have to collaborate instead of working independently with the aim of offering to the end-users a better quality of the daily life.
The development of ubiquitous computing and communication software infrastructures has to consider the abstraction level of the implemented concepts. Therefore, abstracting concepts from direct and immediate human needs in specific smart environments, avoiding undue assumptions about the available devices or services and promoting decoupling among distinctive, physical and functional features of devices and services, is required.
This paper proposes an extensible software architecture supporting smart environments and an approach aiming at representing the physical world in a useful, comprehensible and more abstract manner, for facilitating connections with the virtual world.

1 Introduction
The way the computers are used is changing compared with the approach proposed by mainframe and client-server systems, where computers or terminals are used by persons occupying a well-defined desktop position. Nowadays, ubiquitous or pervasive systems are proliferating and one person can use any computer through the Internet, anywhere, at any time and regardless of the limitations introduced by the used technology. In general terms, a revolution in the fields of computation, communication and interaction is taking place and Weiser’s vision described in 1991 [37] is becoming a reality.
People’s thirst for technological products is growing. With the hope of simplifying their daily activities and having a better quality of the everyday life, people equip with technological products their own dresses and professional (offices, shops, laboratories, museums, classrooms, hospitals), personal (home, gardens), transit (streets, squares, airports), transport (buses, trains, cars) living environments, also referred as smart living environments or smart ambients.
The physical world is populated by common computers, but also by sensors, actuators and smaller and more personal devices, which have networking capability and can be linked by wired and wireless heterogeneous and spontaneous networks.
At the same time, academic and industrial experts feel inclined to promote technological progress. Products regarding home automation, domotic systems and smart home areas, are widely offered by the industry, while research concerning pervasive computing, ubiquitous computing, nomadic computing, ambient intelligent, context-aware computing and augmentation of the real world are being investigated in the academy.
Many challenges and future directions [10, 16, 20, 32] have been sketched. Frameworks and pieces of middleware addressing end-users needs, such as safety, saving, entertainment, comfort, and using different solutions, such as neural networks, device-independent application development processes, agents, or other approaches have been developed [21, 29, 38]. Questions regarding discovery services, automation, context awareness, physical and logical integration and interoperability, adaptiveness,
security and privacy are tackled in several projects [29]. Unfortunately, these questions are not exhaustively treated. Their treatment often requires the knowledge from a wide variety of disciplines and needs to take in consideration other already defined ad-hoc application domain solutions, system architectures, and specific experiences, for reusing them [4, 5, 7, 12]. In addition, all the produced efforts have to be reciprocally compared by academic and industrial experts.

This scenario lets glimpse a reality where the physical world (organized in living environments and populated by persons, animals and technological products) and virtual world (made up of software infrastructures that are characterized by software objects and their applications) should be integrated and should interoperate [3, 11, 25] at the point that browsing the reality will be similar to browsing the Web [6]. In the next years, this situation will imply the reciprocal collaboration of heterogeneous networked devices, services satisfying people’s needs and living environments equipped with them, in order to offer to people a better quality of the daily life. For example, let’s consider a energy saving service for automatically controlling resource consumption in a smart home ambient. This is a useful service, as it allows to resident people to save money. Many new services can be offered if collaborative work is possible. For instance, if other smart living environments have the same service, or at least an electric saving service, and if they are integrated with the electric energy saving service of an electric central, then, electric blackout and consequent economic damages and discomfort to collectivity may be avoided.

As a consequence, ubiquitous computing and communication software infrastructures supporting a degree of system integration, ability to interoperate, aptitude to be extended and major abstraction, are necessary. This paper aims at addressing these aspects. With this in mind, an extensible and ubiquitous software architecture is used [8]. It is made up of a network infrastructure, a system infrastructure and a software infrastructure named Domus intelligent Keeper (DiK). The purpose of this architecture is twofold: facilitating the integration and interoperability of heterogeneous networked electronic devices based on different technologies, communication protocols and produced by different manufacturers; and hiding, to software developers and common users, the complexity and heterogeneity of both those devices and the physical world.

The paper is focused on an approach for offering an abstract representation of the physical and virtual worlds for hiding the complexity and heterogeneity of the physical world in ubiquitous computing infrastructures, and setting the basis for making the previously sketched scenario a tangible reality.

The paper is structured as follows: Section 2 presents an abstract vision of the physical and virtual worlds and shows their connections; Section 3 depicts the proposed extensible and ubiquitous software architecture; Section 4 describes the cited approach and introduces the concept of entity description graph; Section 5 discusses how the information abstracted from the physical world are used in the DiK software infrastructure; Section 6 presents related works; the final section summarizes the main conclusions and sketches future research directions

2 Characterization of the physical and virtual worlds

This section presents an abstract representation of the physical and virtual worlds and highlights their connections.

Fig.1 shows that the physical world can be characterized by a set of:

- networked entities, existent in the observed living environments. They are heterogeneous electronic networked devices, mobile or not, coming from different makers. The use of the networking techniques ensures that devices are interconnected and accessible from local- and/or wide-area networks;
- physical concepts, describing the internal or external state of a networked entity and, consequently, determining the state of the observed living environments. Examples of valid identifier terms for these concepts are LIGHT, SOUND, AIR, WATER, TEMPERATURE, FOCAL LENGTH, ELECTRICITY, DIAPHRAGM APERTURE and so on. It is also important to observe that a given physical concept can be of different types. For example, the type of the LIGHT physical concept can
be described with the terms ARTIFICIAL or NATURAL, while the type of the WATER physical concept can be described with the terms HOT or COLD.

Figure 1: Characterization and connections of the physical and virtual worlds.

The virtual world can be characterized by a set of:

- **software objects**, made up of software interfaces, modelling networked entities and their software implementations;
- **software services**, sensitive to measures of the degree of presence of a given physical concept by networked sensors.

Fig.1 shows also the connections between the two worlds. They are:

- logical connections between a networked entity and one or more physical concepts, represented by vertical arrows with white hatches. The orientation of these arrows is directed from the affecting entities to the affected concepts. For example, a networked switch is an entity able to open or close a flow of electric current. Therefore, this entity has a logical connection with the physical concept ELECTRICITY and affects its presence grade. Similarly, a networked camera that offers iris functionality is an entity able to open or close the passage of a flow of light in the camera and has a logical connection with the physical concept DIAPHRAGM APERTURE;
- one logical connection, represented by the superior horizontal arrow with white hatch and directed from the networked entities to the related software objects. The inverse orientation is invalid because characteristics of networked entities can affect the related software objects and not vice versa;
- a transmission of information, regarding the state of the observed living environments and the networked devices, represented by a white dotted arrow, directed from the physical world to the virtual world;
- a transmission of commands represented by white arrows, oriented from software services to a single or a set of software objects;
- a consequent production of effects on the physical world or the existent networked entities, represented by a nested arrow.

### 3 Software architecture

This section offers a brief overview of the extensible ubiquitous software architecture presented in [8] and shown in Fig.2. It is made up of various layers that are grouped in six levels, going from the highest, A, to the lowest, F. The typical arrangement in layers is used for obtaining a higher abstraction with each additional layer and limiting the impact of a change of a layer on the other ones.
A brief description of the different layers follows. Level F is made up of networks of interoperating heterogeneous networked electronic devices coming from different manufacturers. Level E includes the components necessary for connecting the devices of Level F drivers. They consist of the needed drivers, a hardware layer and a layer of network IP cards, audio cards, RS-232 ports, and so on. Level D includes the operating system and the Java Virtual Machine (VM) and forms the system infrastructure. Levels A, B, and C form the DiK infrastructure. In particular, level C includes the OSGi (Open Service Gateway initiative) framework [28] that represents a common environment hosting bundles. It manages the life cycle of the bundles and solves their interdependence, searches classes and resources the bundles make available, keeps a registry of services and manages the events informing the listeners when the state of a bundle is changed, a service is stored or an error occurs. Besides the OSGi framework, Level C can include other alternative solutions, such as the Java Remote Method Invocation (RMI) [23] or the Jini [22] ones. In fact, it has been designed as a dynamic container with changeable content according to the technological progress. Level B contains the Devices Virtualization layer that enables to free the developers from the burden of implementing low-level interaction with the physical world of networked devices. In addition, it promotes the decoupling of functional aspects from descriptive ones. Finally, Level A groups Logical, Services and User Interface (UI) layers, oriented to minimize the work of the end-users. In particular, the Services layer includes an intelligent service group based on a rule engine named Jess [31], which allows the execution of rules describing relations between events and actions. The rules may be created by smart environments users, or be automatically generated by a learning system that was developed on the basis of the WEKA (Waikato Environment for Knowledge Analysis) tool [36].

The cited technologies have already been used in other projects in the ubiquitous computing context. The main difference between the previous usage and the presented architectural design consists of the existence of the B layer that decouples the A layers from layers below it, so that the formation of the isolated islands of ICT facilities is contrasted. In this way, several technologies can be combined for
providing a common execution environment, where current and future heterogeneous devices and services, created under different design constraints and considerations, can be integrated. Furthermore, the proposed software architecture is open to different makers and adequately supports the daily common people decisions. They can buy and insert different new devices in their living environments and make them operative. Finally, heterogeneous networked devices are accessible in a more transparent way, simplifying the interaction of the end-users.

4. Entity description graph

The approach described in this section is oriented to represent the physical world in a useful, comprehensible and more abstract manner for facilitating the connections with the virtual world. It is based on a structure, named entity description graph, shown in Fig. 3. This structure is also used for helping the extraction of information regarding the physical world and facilitating the comparison among the different entities.

A networked entity in the observed physical world is a physical device on which it is possible to trigger actions that produce effects either internally or externally to the considered networked entity. Each entity can be defined as composite, if it is made up of a set of different other entities, or as elementary. For example, a camera is a device of composite type, composed of different elementary actuation devices, each of which relative to a different functionality, such as iris, zoom, pan, tilt.

Moreover, each networked entity can be described in an abstract manner through structured and decoupled information tuples grouped in the entity description graph. Every instance of this graph represents a different networked entity existent in the observed physical world. Each node of the entity description graph is made up of information tuples. A tuple is defined complex if it is composed of a set of other tuples, simple if it is made up of one or a set of attributes and optional if its presence is not always required. In the last case, the tuple is included in square brackets. In addition, in order to indicate that the component of a tuple (i.e., an attribute or a composing tuple) does not exist for the considered network entity, the ‘#’ symbol is used, while, for highlighting that the value of the attribute of a tuple is unknown for the considered network entity the ‘//’ symbol is adopted. An attribute is unknown when it is unavailable or undeterminable. Finally, for distinguishing two different instances of a tuple, a pedex to the symbolic representation of the tuple is added.

The graph in Fig. 3 points out that a networked entity can be described by three complex tuples:

\[ E = \langle GD, PhI, FI \rangle \] (1)

where: GD is the General Description of the entity; PhI is its Physical Identity; and FI is its Functional Identity. These tuples are collected and used in the developed DiK software infrastructure. The defined tuples are required for identifying generic characteristic features and locations of the networked entities, the physical concepts they affect, and the type of functional mechanism they exhibits.
A brief description of each item of tuple \( E \) is given in the following subsections.

### 4.1 General Description

The General Description, \( GD \), of a networked entity is a set of information defined by two simple tuples:

\[
GD = < GI , GL >
\]

(2)

where \( GI \) is the General Identity, represented by a set of attributes, such as name, manufacturer, serial, version, model and communication protocol, defining characteristics of a networked entity. \( GL \) is the entity General Location in the observed living environment.

For example, an Axis networked camera, installed in a garden, is a composite networked entity that can be characterized by the General Description \( GD=<GI, GL> \), where \( GI=<\text{network camera, Axis, #, #, 2130 PTZ, TCP/IP}> \) and \( GL=<\text{garden}> \) is its location as a topographical information.

### 4.2 Physical Identity

The Physical Identity, \( PhI \), of a networked entity is a set of information defined by two simple tuples:

\[
PhI = < PhA , B >
\]

(3)

where \( PhA \) represents the Physical Aspect of the entity and \( B \) its Behaviour.

The Physical Aspect is described by three attributes:

\[
PhA = < PI , { PT }, C >
\]

(4)

where: \( PI \) is the Identifier of a physical concept, \( PT \) is an optional attribute representing the Type of the physical concept; and \( C \) is the Consumption required from the networked entity for affecting the value of the considered physical concept.

For example, a networked rolling shutter can affect the brightness degree of a given room, and, therefore, has a physical aspect described by the tuple \( PhA=<\text{LIGHT, NATURAL, 60Watt}> \); while, the iris functionality of a networked camera has a physical aspect described by the tuple \( PhA=<\text{DIAPHRAGM APERTURE, AUTO, //}> \). As additional example, a brightness sensor has a physical aspect described by the tuple \( PhA=<\text{LIGHT, #, //}> \), where the middle attribute is not defined, as the physical aspect of a sensor device does not require specification of the type of the pertinent physical concept.

The Behaviour is described by four attributes:

\[
B = < A , BAct , BEff , BWT >
\]

(5)

where: \( A \) represents the Logical Connection type of a networked entity to a physical concept; \( BAct \) represents a feasible Action on an elementary device for affecting a physical concept; \( BEff \) represents the produced Effect type on a given physical concept; and \( BWT \) concerns the action’s Working Time.

One or more internal and/or external logical connections to pertinent physical concepts exist for each elementary or composed networked entity in real world. This logical connection describes the internal and/or external state of the considered networked device. Pertinent physical concepts can be derived from the analysis of datasheets of the considered networked device. A logical connection from a networked device to a physical concept exists when the considered networked entity can affect the measure of the considered physical concept through its functionalities. In order to represent internal logical connections, the “<” symbol is used, while for external one the “>” symbol is employed.

In addition, it is assumed that feasible actions on physical concepts can be only of two types: increase represented with the “+” symbol and decrease represented with the “-” symbol. The actions of turned on and turned off are included in those of increase and decrease, respectively. In particular, turned on regards an increase of a value different from zero, and turned off concerns a decrease of value equal to zero, if
zero is the inferior extreme of the range of possible increments. In this context, a quantitative scale of values for a physical concept is used, even if a qualitative scale could also be used.

When an action is triggered, an effect is produced. It can be of different types: null, represented by the “=” symbol; in agreement with the pertinent action, represented by “+”; in disagreement with the pertinent action, represented by “-”. For example, the action of increase on a rolling shutter is not in agreement with the produced effect, as this action involves a decreasing of the natural light.

In addition, the maintenance of an effect can require a consumption of continuous or temporary power. This concept is depicted in the action’s working time value that can be equal to either CONTINUOUS or TEMPORARY. For example, the maintenance of the produced effect by a rolling shutter on the natural light, or by the iris functionality of a network camera on the diaphragm aperture, requires a TEMPORARY action’s working time, while the produced effect by a lamp on the artificial light requires a CONTINUOUS action’s working time.

As a final example, the behaviour of the increase of a networked rolling shutter is represented by \( B_1 = \langle >, +, -, TEMPORARY > \); the one of the iris functionality of a networked camera can correspond to the tuple \( B_2 = \langle <, +, +, TEMPORARY > \); and the one of the decrease of a networked lamp can be represented by \( B_3 = \langle >, +, +, CONTINUOUS > \).

4.3 Functional Identity

Functional Identity of a networked entity is the following simple tuple:

\[
FI = \langle DI , DT >
\]  

(6)

where \( DI \) represents Device functional Identifier and \( DT \) is the Device functional identifier Type. According to the functional point of view, networked entities can be grouped in two families of elementary devices:

- sensors, capturing information from the networked devices and/or the environment, and, then, producing events;
- actuators, triggering actions on the networked devices in the considered environment and, then, consuming events.

They can also be defined as a set of these elementary devices. The functional point of view allows defining networked devices in an abstract manner, which is independent from their type described as simple \( GI \) tuple, and nature, described as complex \( PhI \) tuple. Therefore, two devices, having a distinct type and nature, may share the same actuation mechanism. For example, a networked lamp is a device different from an alarm. The lamp is logically associated to the electric light concept and may change the state of the environment area where it is installed by providing or not providing light on the basis of the switch on/off actuation mechanism. In an analogous manner, the alarm is logically associated to the sound concept and may change the state of the environment area hosting it by providing or not providing noise in accordance with its open/close actuation mechanism. In spite of their different type and nature, the lamp and alarm are devices share an actuation mechanism with the same working procedure.

In addition, sensors and actuators can be still specialized in other objects. For example, the networked rolling shutter has a mechanism of actuation that is different from that of the networked lamp and alarm. In fact, it cannot be defined on the basis of two values but by considering a set of valid values. For instance, it may have five possible valid values, such as absent, low, medium, high, highest, modelling five different positions and brightness degrees. Besides the device functional identifier, a device functional identifier type exists, too.

To summarize, the functional identity of the considered rolling shutter can be represented by the tuples: \( FI_1 = \langle ACTUATOR, SET_VALUES > \), while that of the lamp can correspond to \( FI_2 = \langle ACTUATOR, BINARY > \). Terms SET_VALUES and BINARY indicate an actuator with a valid set of options and one with only two valid options, respectively.
5 Utility of functional and descriptive tuples

The structured and decoupled tuples in (2), in (3) and in (6) are the result of the separation of the functional and descriptive aspects of the networked entities installed in the physical world. Information captured by instances of these tuples is mapped to the Devices Virtualization, Logical and Services layers of the DiK software infrastructure. The next subsections provide a description of the three layers in Levels A and B of Fig.2.

5.1 Device Virtualization layer

The Devices Virtualization Layer regards the functional characterization of the devices installed in the living environments. A software object is defined for each instance of the (6) tuple, regarding networked entities in the physical world. In particular, the software object is composed of the implementation, maintained at level B, and the interface, maintained at level C of Fig.2. A simplified view of the interface is shown in Fig. 4.

Fig. 4 highlights that:

• A first level of specialization of the generic devices of type Sensor and Actuator exists. For example, the networked lamp is a device of Actuator type, which can be characterized by a BinaryActuator interface assuming only two possible valid values. While the rolling shutter is a device of Actuator type, which can be characterized by a SetValueActuator interface assuming different discrete defined values. Furthermore, a device having values inside a given continuous range can be characterized by a RangeValuesActuator interface. Besides those discussed, further specialization levels can be identified.

• Interface Device is characterized by methods adding and removing the EventListener objects. These methods are used from clients for registering and un-registering an event listener to the Device. Thus, for taking decisions, clients can be notified of changes in the state of the devices, in a push way. Listener and event interface hierarchies are also defined.

• Complex devices, such as cameras, can be considered as a specialization of the MultiDevices object. Fig. 5 shows the composite pattern used for developing this kind of object. This solution offers the possibility to model any complex device in a simple and extensible way. In fact, Fig. 5 exhibits an
interface declaring methods for adding/removing single devices to/from the set of devices composing the considered complex device.

The interface hierarchy shown in Fig. 4 is not complete. It shows that it permits the realization of reusable software components and that the Devices Virtualization layer is still valid even when the hierarchy is extended for including new devices, independently from their complexity.

![Interface Diagram](image)

**Figure 5:** Interface MultiDevices.

### 5.2 Logical layer

The Logical layer offers a representation of the physical world in a useful, comprehensible and as more abstract as possible. It collects information required for identifying generic characteristic features and locations of the networked entities, and the physical concepts they affect and the manner this happens.

For each instance of the (2) and (3) tuples, regarding networked entities in the physical world, a software object exists. The implementation of the software objects is maintained in the Logical layer of Level A in Fig. 2; while its interface is maintained in the same layer for (3) tuples, and in level C for the (2) tuples. In particular, Fig. 4 and Fig 6 show that the Device and DevicePhysicalInfo interfaces are characterized by getting and setting methods for accessing and/or manipulating information regarding the (2) and (3) tuples, with reference to a given networked entity in the real physical world.

![Interface Diagram](image)

**Figure 6:** Interface for accessing the device physical identity.

### 5.3 Services layer

The Service layer is made up of a set of services, operating in living environments where networked devices are constantly moved, upgraded, connected/disconnected, and turned on/off. Moreover, generally, each service must rely on many other services to perform its tasks, even if the exact identities of these services will inevitably change over time [1]. In this scenario, the challenge is to build a federation of services made of components that do not know with what they connect until they have run-time information [33] and the networked devices affected by services.
In the proposed approach, this can be performed through a dynamic binding process, and instances of information in the (2) and (3) tuples are the instruments through which a service can realize this process. In fact, a service can dynamically bind a perception coming from the physical world to a suitable method of a software object in the virtual world. As a consequence, actions are triggered on the networked entities of the physical world with the production of their external or internal effects. These effects affect the physical concepts value.

A way to adjust these values can be using a regulation service. Let’s consider the idea of decrease the physical concepts LIGHT and TEMPERATURE of a room and an oven. In the case of the room, the regulation service invokes a localization service that checks all the networked entities with tuple $GL$ matches the room, and returns a list of the found entities. Then, the regulation service checks from this list all the networked entities with attribute $PI$ equal to either LIGHT or TEMPERATURE, and obtains a new list of found entities, which is a subset of the previous one. Yet, the regulation service can invoke, for example, a saving strategy that can decide to decrease the light of the devices having consumption higher than a given value, or of all the devices with the attribute $PT$ equal to ARTIFICIAL. During the strategy application, a process of dynamic binding is performed. In the case of the oven, the regulation service can directly execute the dynamic binding to methods for decreasing light and temperature of the networked oven.

Describing behaviours of networked entities in terms of physical concepts (which physical concepts a networked entity affects, how and with which type of functional mechanism), allows designing services that are independent from the end-users, devices and living environments. In this way, services are not focused on user tasks, actions, device functionalities or user’s interactions with the interfaces on a specific living environment. This is a key idea for setting the basis in order to realize networked devices groups and/or pervasive and ubiquitous systems working in collaborative way.

6 Related work

The approach proposed in this paper and used in the implemented software architecture shown in Fig.2 aims at hiding complexity and heterogeneity of the physical world in ubiquitous computing infrastructures for helping the creation of a common execution environment for contrasting formation of isolated island of ubiquitous facilities in favour of collaborative work.

Idea of realizing a common execution environment is not a novelty. Many scientific disciplines are interested in using a global distributed infrastructure supporting their activities and facilitating information exchange among different systems or components. Grid technology [13] and efforts like Globe [35], Globus [14] address this subject offering an environment where two or more systems or components can exchange information. Unfortunately, they are conceived for distributed computing originally developed for small and relatively static computer networks [17]. Therefore, they are not well suited for ubiquitous computing, occurring in large and highly dynamic distributed systems, which are always working and available. One.world [15, 17] is a system architecture addressing the considered subject. It is largely implemented in Java, its data management is based on tuples and its control flow is expressed through asynchronous events that are simply tuples and are processed by event handlers. Its goal is to provide an integrated and comprehensive framework for building pervasive applications. Its end-users are application developers and administrators. Substantially, Levels C and B of the architecture proposed in this paper have the same goal. Difference with system architecture in Fig.2 is on its arrangement and used technologies. In particular, Level C use the OSGi framework that represents a service and discovery-focused standard whose adoption is possible by a broad category of vendors of heterogeneous networked devices.

Besides OSGi, other standard like Jini, HAVI (Home Audio/Video interoperability) [19], UPnP (Universal Plug and Play) [34], and so on, exist. They are complementary, rather than competitive, even if they are sometimes partially overlapped in some provided facilities.

Standardizing communication at the lower level of the architecture in Fig.2 is only the starting point toward contrasting formation of isolated island of ubiquitous facilities in favour of collaborative work.
Research effort toward the coordination mechanism at a higher level is also required, for allowing networked devices and smart environments to appropriately use the exchanged information without an a priori knowledge regarding with whom they are connected and how to communicate with it [10]. Concerning this, academic and industrial research groups have investigated various approaches. The CoolTown project [26] uses more abstract and much less detailed service specifications in web-based systems. Its approach gives to physical entities (people, places, and things) a presence on the web that can be associated with their physical location. The one.world project is based on three main abstractions: task representing computations, tuple representing persistent data and environments providing structure and control [15]. The work at Stanford [24] is characterized by a common tuple space protocol as coordination infrastructure for interactive workspaces. The Speakeasy project at Xerox PARC [27] uses interfaces extending the behaviour of entities in the environment, while the end user provides the semantic knowledge for deciding when and whether to use a particular entity. These systems use specialized representation mechanisms of contextual connecting information with the physical world, while the use of ontology is investigated in other framework like COBRA [9], SOCAM [18], GAIA [30].

In the architecture in Fig.2, a specialized representation mechanism is used. It is conceived as a flexible instrument for comparing heterogeneous networked entities and perceiving their similarity degree. Moreover, it supports the building of federations of networked devices and software components implementing ad hoc services. A federation can be based on one or more functional, typological, topological, physical, behavioural aspects, or even on aspects with a finer grain in accordance with the depth of the entity description graph. For considering other future aspects, it can be simply extended the branches of the entity description graph, limiting the impact of new introduced aspects on the work of the existent entities.

7. Conclusions and future works

Individual networked devices and smart living environments, are characterized by their own application domains, requirements and goals. These features represent drawbacks that lead to the formation of isolated islands of ICT (Information and Communication Technology) facilities and lack of collaboration in and among smart environments.

To abstract concepts from direct and immediate human needs in specific smart environments, avoid undue assumptions about the available devices or services [2] and promote decoupling among distinctive, physical and functional features of devices and services, is crucial in order to contrast the described drawbacks. With this in mind, this paper proposes a software architecture and an approach, oriented to represent the physical world in a useful, comprehensible and more abstract manner for facilitating connections with the virtual world. An entity description graph is used for supporting the information extraction from the physical world and facilitating the comparison among different entities. The aim is to build a federation of services and the networked devices they affect. The services are made of components that do not have knowledge regarding with which components they are connected before run-time information are received [33].

Every instance of this graph represents a different networked entity existing in the observed physical world. Each node of the entity description graph is made up of information tuples, aiming at describing the generic characteristic features and locations of the networked entities, the physical concepts they affect, and the type of functional mechanism.

Authors’ research activity is carried on within a wider Demonstrator project involving many researchers and industrial partners, and aiming at analysing, defining and realizing hardware and software platforms for permitting the provision of networked services in safety, security and privacy conditions and the implementation of advanced technologies. In particular, the activities involving the authors aim at developing a platform in the field of home automation that is endowed with different levels of intelligence as awareness, reactivity and adaptiveness, and allowing the activation of a service through different channels of interaction like mobile, web and the classical (AWT/Swing) user interface client. In this way,
a suitable application with reference to the user interface typology accessing the target service can be implicitly chosen. Many directions could be still investigated. Author will focus on two main directions. The first regards the further extension of the DiK intelligence component for automatically facilitating the evolution of the software infrastructure, on the basis of people’s continuous changing habits and modifications of the networked devices adopted in the living environments. Another future research direction regards the experimentation in real context of the proposed approach and the evaluation of the performance of the proposed extensible and ubiquitous architecture.

References


