Fast self-configuration in service-oriented Smart Environments for real-time applications

Borja Bordel[a,*], Ramón Alcarria[b] , Diego Sánchez de Rivera[a], Diego Martín[a] and Tomás Robles[a]
[a]*Department of Telematics Systems Engineering. Universidad Politécnica de Madrid. Avenida Complutense nº 30. 28040 - Madrid (España); E-Mails: bbordel@dit.upm.es; diego.sanchezderiveracordoba@gmail.com; diego.martin.de.andres@upm.es; tomas.robles@upm.es*
[b]*Department of Topographic Engineering and Cartography. Universidad Politécnica de Madrid. Campus Sur, 28031 Madrid, Spain; E-Mail: ramon.alcarria@upm.es*

Abstract. Smart Environments (SE) aim to satisfy the experience of individuals by the provision of systems, services and devices. Although many approaches may be followed in order to build a SE, a service-based design is the most promising one nowadays. In these architectures, both physical devices and even humans may provide low-level services based on their capabilities. These low-level services are composed and connected in order to create high-level services, employed by managers to describe the behavior of the SE. This composition process traditionally requires the human intervention, so changes in services are not automatic. However SE should operate at real-time, thus the collection of available services must be always updated. In the state of the art this challenge has been only partially covered, and proposed solutions are still heavy and manual. Therefore, in this work we propose a new fast semi-automatic self-configuration (and automatic self-adaptation) technology for service-oriented SE. The proposed technology is based on the definition of templates at different levels which are related by means of semantic information. Moreover, simple binary mathematical operations are included in order to reduce the convergence time and improve the scalability. A first reduced extension for hybrid approaches (including both humans and physical devices) is also provided. Finally, an experimental validation is conducted in order to evaluate the convergence time of the proposed solution in different scenarios, and demonstrate it is scalable towards future deployments involving thousands of devices and (possibly) humans.

Keywords: Smart environments; Self-configuration; Services composition; Quality of service; Real-time applications

* Corresponding author. E-mail: bbordel@dit.upm.es

1. Introduction

Cook and Das [1] define a Smart Environment (SE) as one that is able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment.

Traditionally, SE can be understood as monolithic systems where the different existing layers are strongly integrated. Thus, in these architectures, the operation of SE is described as ad hoc feedback loops of (i) perceiving the state of the environment, (ii) reasoning about the sensed state together with some goals defined by users and (iii) acting over the environment to change its state. Thus, perception of the environment is a bottom-up process [2]. With this view, every component in the system depends on each other (e.g. the users' goals should be measurable employing the underlying hardware devices), and various experts must be involved not only in the system deployment but also during the operation [3].

With the objective of getting independent the different elements in a SE and, overall, of enabling domain experts with no programming or technological skills (such as health care professionals) to operate SE without technical assistance, newer service-oriented architectures are being applied to SE. Service-based designs are currently very promising [4][5], as they define different levels of abstraction which are independent and might be employed in very different contexts (services are encapsulated and can be easily transferred to new systems and applications). Thus, economies of scale are enhanced, and the deployment of SE is promoted.

The introduction of service-based approaches causes the traditional SE operation to change [6]. Particularly, a new first discovery process must be considered, where low-level services (those which are directly supported by the hardware capabilities) are listed and transformed into medium-level and high-level services. In that way, the SE operation should be described as, (i) the physical layer publishes the available services in a central server by means of a transformation process, (ii) then, the user's applications look in the central server for the required services to improve the inhabitants experience following the rules indicated by system managers and (iii) using the selected services the applications monitor the state of the environment, reason about that state and act to change the situation. This new approach maintains the bottom-up perception of Smart Environments, but turns independent all the different layers without including heavy software elements such as specialized operating systems [2].

Another advantage of applying a service-based approach is the possibility of including humans as service providers [7]. In this context, a "service provided by humans" is any work performed by people at the request of the SE, with the aim of improving the overall experience of the inhabitants. For example, in a certain SE the system may ask a person (by means of a mobile application, or any other mechanism) to open a window if the ambient temperature goes above the specified range. Then, it is said the service "open a window" is provided by humans. Traditionally, studies on the relation between humans and SE are focused on behavior monitoring [8][9] or activities recognition [10][11]. In that way, humans are usually considered as "consumers" (i.e. applicants for system actions to improve their experience). However, projects such as the MavHome [12] (Managing an Adaptive Versatile Home) have introduced humans in SE in various roles (a separated problem known as "humans-in-the-loop" [13]). Figure 1 shows the resulting scheme for service-based Smart Environments. As can be seen, humans appear as services consumers (in the top part of the architecture) and as services providers (in the physical layer, together with hardware components).

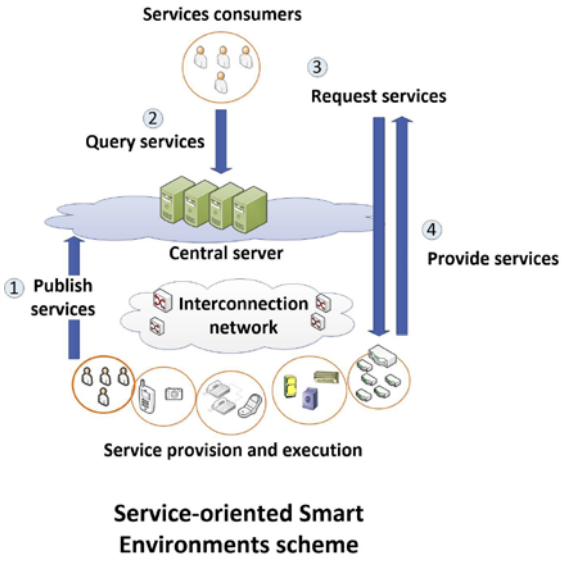

Fig. 1. Service-oriented Smart Environment scheme

Nevertheless, service-based systems introduce new challenges and problems. In traditional SE based on feedback loops, the system is designed to react to the stimuli before the state in the physical world changes

(the stability criterion). Service-oriented solutions should maintain these temporal requirements as SE are much more valuable if real-time operation is supported [14]. However, in these systems, the necessary abstraction process to transform low-level services into high-level services is often performed by people, as it is required to understand the "meaning" (i.e. the semantic) of every service to perform it [15]. Thus, the cited abstraction process cannot be executed during the operation time as it prevents the fulfillment of the real-time requirements (because it is a manual process and, then, very time consuming). Thus, the definition of services at the different levels is performed during the configuration time, and the created service collections remain immutable until an expert updates them manually. Nonetheless, changes in the physical platform occur in the same way that in the traditional SE, so commonly there are services configured by experts which are not executable during a certain time due to variations in the hardware components, or even in humans (life cycle, interferences, temperature…) [16].

In order to address these inconsistencies, two types of solutions have been proposed. First, some solutions try to find out if a certain service is available before invoking it [17]. In order to do that, information about the hardware devices is provided to high-level components, a solution which breaks the independency among abstraction layers typical of service-based solutions. Moreover, transferring exhaustive descriptions about low-level components to high-level layers (and later processing them) is a very costly process, so real-time operation in the SE is usually compromised [18].

On the other hand, some solutions include new functionalities in hardware components so that they can refuse non-executable service invocations [19]. If a certain invocation is refused, the high-level layers must look for another equivalent service and repeat the call [20]. This iterative process may be really complex and time consuming, so many times real-time operation is not supported [17].

Therefore, the objective of this paper is to address these problems, proposing a semi-automatic self-configuration (and automatic self-adaptation) technology for service-oriented Smart Environments, compatible with real-time applications, and being able to maintain the independency among the different abstraction levels. In order to do that, we define various types of templates at different levels; we represent information by means of binary vectors which may be processed using simple operations and include semantic and Quality-of-Service (QoS) information in order to maintain updated at every time the list of available services.

The authors, besides, implemented the proposed solutions and carried out an experimental validation in order to validate its performance (especially, its convergence time and scalability).

The rest of the paper is organized as follows: Section 2 introduces the state of the art in the different technologies involved in the proposed problem and solution. Section 3 presents the proposed technology. Section 4 provides an experimental validation of the proposal. Finally, Sections 5 and 6 explain some results of this experimental validation and the conclusions of our work.

2. State of the art

A complete solution for service-based SE requires implementing two different procedures: configuration mechanisms and adaptation mechanisms.

A configuration mechanism is the process through which all components in a system perform the installation procedures to get the basic operational parameters and to obtain the necessary initial information for operation. The configuration process takes place at the pre-operational phase [21]. Thus, automatic solutions are not needed, as (in theory) the pre-operational time may be as large as required. However, in practice, it is impossible to manually configure systems including a great number of devices, so (at least) semi-automatic solutions are needed in order to perform a first automatic phase which reduces the information volume until it is manageable by people. As, partially, the system may be configured by itself, these technologies are usually known as semi-automatic self-configuration solutions.

On the other hand, an adaptation mechanism is focused on dynamically modifying the system configuration without restarting or stopping the system operation. Although various types of adaptation may be defined, in this context, adaptation represents the system reaction when an unexpected change occurs in any of its components. The basic adaptation actions include removal, addition, and replacement of hardware components [22]. As the adaptation process occurs at operation time, and (while being executed) the correct system behavior is not guaranteed, adaptation algorithms (in theory) should be fast and are always automatic (so normally they are named as self-adaptation policies).

Self-adaptation mechanisms, moreover, usually depend on the selected configuration solution. In general, if the implemented configuration mechanism is a self-configuration one, no specific self-adaptation solution is defined: it is enough (as occurs in the Internet and the OSPF protocol) to dynamically refresh the configuration process during the operation, and the system gets automatically adapted to the new circumstances. In any case, if a different solution for self-adaptation is required, it must be specifically designed to update the configuration of the system under study.

In our proposal, two different mechanisms are defined. On the one hand, a semi-automatic self-configuration solution, based on the definition (by domain experts) of service semantic templates which are instantiated automatically using the information provided by the hardware platform is proposed. Hardware information is also automatically collected using low-level templates and binary operations. Mechanisms to turn independent both levels are also included.

On the other hand, as the instantiation of semantic templates is a very complex process [23], a lighter self-adaptation algorithm is defined. In particular, during the configuration process, QoS data are also collected, which are employed to define services as available, unavailable or deleted. Using this solution, the system may be updated in a very fast way and, any case, if required, it is also possible to perform a hard refresh of the configuration process.

Moreover, as we are seeing, the proposed technologies may be extended to hybrid systems including devices and humans.

Then, in this section we analyze the state of the art on various issues: from previous self-configuration and self-adaptation solutions, to technologies related to the semantic service composition.

Previous configuration solutions for service-oriented Smart Environments are divided, basically, into two different groups.

In the **first group**, solutions based on the virtualization of the hardware platform in a central computing core can be found. The idea of these proposals is to create a virtual model for each one of the physical components in the system and, then, obtain the list of available services and configure the system implementing the configuration policies into the virtualized hardware platform and defining mapping techniques. Some of these works propose a specialized middleware layer which maintains the virtualized model [24][25]. Others create an agent-based system representation where every device is represented by an agent in an agent directory [3] (agent-based solutions present the advantage of being able to be extended to hybrid systems). This approach, however, present two main disadvantages. First composed services are manually defined by experts during the system configuration and human intervention is required to apply any update to the service collections (for example, in [24] composed service "healthcare virtual sensor" it is created by joining three types of sensor services). And, second, services in the different abstraction layers are not independent, as (in order to design service composition) experts must know both levels: the underlying hardware platform and the high-level services to be offered (as indicated in the previous example).

Solutions based on the use of semantic technologies together with the virtual hardware platform are a special case in this first group [26]. Although these technologies could be employed to make independent the different abstraction layers in the architecture, these works usually create a unique medium-level layer where semantic is employed to define ontologies with the objective of detecting services with "the same meaning" and avoid replicated services in the service catalogue. In that way, problems previously cited usually appears too in this type of proposals.

In comparison to these previous works, our solution employs light semantic technologies to reach a semi-automatic solution where abstraction levels are independent. In particular, experts must only create service semantic templates in any domain language (so programming skills are not required). These templates are hardware independent, so they may be employed in any context, system or application (which makes easy to share knowledge and reduce costs). In a second phase, these templates are automatically instantiated by a composition engine using medium-level information from the hardware devices. This additional medium-level layer decouples the high-level services from hardware platform, so no knowledge about the technological basis is required to operate a SE (which meets the needs of experts such as health care professionals).

As a result, the proposed technology includes three different independent abstraction layers. In the **second group** of previously existing configuration technologies, solutions are based on the exchange of large amounts of data (among the different components) describing in a comprehensive way the system state. Then, devices independently calculate their own system state. In general, these solutions are automatic: the system runs a configuration procedure during which the operation is stopped and, once fin-

ished, the operation is started. For example, in [27] a distributed technology for self-configuration is described (each device creates a description file which is processed by the rest of the components). As automatic solutions, these configuration policies do not require to be complemented with adaptation mechanisms (which is an important advantage). Nevertheless, generating, exchanging and processing exhaustive descriptions about the system requires an important effort from the devices, especially if reduced capabilities devices are deployed and we consider a large-scale system [27]. Additionally, this approach does not allow turning independent the different abstraction layers in the system, so managers must have technological skills and economy of scale is not promoted (knowledge and experience cannot be easily shared as every deployment is an ad hoc design).

In order to reduce this effort and, then, accelerate the configuration process we have designed a mechanism through which devices may share information about them using binary vectors (which might be transmitted very quickly and processed with simple logical operations). At low-level, archetype-based templates are defined. These templates are instantiated using binary information (yes/no answers) which may be understood as simple binary vectors. The vectors may be aggregated using logical operations. This information is collected by a medium-level data service manager which translates the aggregated binary descriptions into a traditional services description document (such as an XML).

Automatic self-configuration policies may be used as self-adaptation policies if, when a change occurs, the system is entirely re-configured. However, some specific adaptation technologies have been also described.

Systems configured by means of mechanisms based on the virtualization of the hardware platform (see above) tend to implement (over the same virtual infrastructure) complex intelligent algorithms to get adapted to the changes. For example, in [28] and [29] self-adaptation functionalities are supported by an intelligent scheduler installed in each device, communicating with a central core where decisions are taken. Similar to those proposals, in [30] self-adaptation process is based on software agents (called Multilevel self-configuration acting units). When a change occurs in the system, agents execute a recurrent algorithm of protocols negotiation and session establishment. In SOCRADES project [31] a self-adaptation technology based on petri nets is proposed. However, every reconfiguration results in a manual effort. Finally, AutoPnP [32] project deals with automatically adding new components (one of the most common adaptation cases, see above) using artificial intelligence algorithms.

In all those proposals, due to the need of executing complex algorithms both locally and remotely, modifying the virtualized hardware platform (which may include several entities is large-scale SE) and renegotiating the session parameters, the adaptation process tends to be slow, which makes difficult its execution under real-time constraints.

In our proposal, the most complex step is service composition. In order to avoid having to refresh this process when any change occurs in the SE, abstraction layers in the architecture are totally independent. In general, hardware devices present redundancies, backup solutions, etc. which cause that many times changes in the physical platform do not affect the provided services. Our proposal considers this fact and provides mechanisms to detect if a certain change in the hardware platform must be translated into a new service composition process. Thus, it is only necessary to refresh the service composition phase if new services must be added.

On the other hand, self-adaptation solutions based on the exchange of information among the different devices making up the SE have been described. The operating principle is similar to the automatic configuration solutions described above. However, while configuration policies only can restart the system operation (see previous description), self-adaptation mechanisms consider the past system configurations and update the system state dynamically by taking into account the new information without stopping the operation. In [33], a solution for self-adaptation based on control loops is proposed. A decision making unit receives a continuous data flow from every device describing its state, analyzes together all received data and acts over the system. Additionally, semantic solutions are really popular. Thus, in [34], [35] and [36] semantic technologies are used to process the information about the system and decide if a service has to be added, deleted or modified (later we are describing this topic in more detail). Finally, solutions based on the generation and transmission of event notifications have been also proposed [37].

As main advantage, with this second type of solutions, the adaptation process is much faster than which obtained when using genetic and other types of complex algorithms. However, these solutions present the same disadvantages as studied configuration solutions (a great effort from the system is required and abstraction layers are not independent).

A binary data format has been included in our solution in order to address these problems. Nevertheless, this information may not be enough in some occasions, especially when a certain service is present but its performance it is not adequate (if, for example, degradation due to interferences wants to be considered). Thus, QoS data are also included. These data do not affect the service composition process, so all the previously cited benefits are not penalized, but are employed to dynamically decide if a high-level service is available or not. In this way, the system may get adapted much more dynamically to changes and several different types of situations.

As final detail, the key element in our proposal is the semantic service composition based on templates (although many other proposals have been included in order to design a vertical solution).

Traditionally, service composition has been a manual task. However, in the last decade various works on semi-automatic service composition have appeared, most of them following the named *process-driven* paradigm [20]. Basically, in this type of solutions, low-level or atomic services are automatically collected and users must create composed services by connecting several of these atomic services [38][39]. Problems associated with these solutions have been described above.

In order to reduce the human intervention in the service composition process, semantic technologies have been applied. In general, these proposals are designed to build the future Web 3.0 [40], so web services are employed. The general idea is to include a semantic reasoner being able to decompose a high-level service into low-level services depending on their meaning [41][42], user preferences [20] or a set of constraints [43]. At the end, all these works try to execute the service with the most similar meaning to that desired by the users [20]. In this approach, however, the reasoning phase occurs each time a service is invoked during the operation time. Thus, in scenarios involving a great number of services and/or devices, the required time to resolve this process prevents the use of these solutions in real-time applications [20]. Moreover, the reasoning algorithms are typically platform dependent, so they cannot be freely applied to new scenarios or applications [20].

Our solution, on the contrary, performs the service composition during the pre-operational time (in the configuration procedure), so system operation is not affected. We are, moreover, applying the technology of Semantic Annotations [44] in order to include semantic in the lightest way. Additionally, in order to promote the collaboration among users and the economy of scale (typical of the SE revolution) we are basing the proposed semantic service composition on hardware independent templates which may be shared and employed in very different scenarios or applications.

Finally service-oriented designs present the advantage of being able to be applied to hybrid systems (including devices and humans). In fact, our proposal may be extended to this type of SE, and we present (see Section 3) a first possible way to design that extension. Nowadays, no configuration solutions for Smart Environments involving humans and physical devices at the same time have been proposed. Furthermore, proposals on self-configuration for the problem of humans-in-the-loop are based on previous works about self-organizing systems (different from Smart Environments, as self-organizing systems are more focused on the ability of reacting to stimuli -in any way-, than on improving the individuals experience). Among all the existing technologies, Collective Intelligence (COIN) has been the most studied [45] [46]. This technology is particularly useful as it allows incorporating user satisfaction functions in the configuration algorithm executed in the system manager. Other technologies such as ant-based models [47] or Multi-agent coordination [48] have been also proposed, although they are not focused on SE.

3. Proposal: a fast self-configuration solution for Smart Environments

This section analyzes the functional architecture which supports the proposed self-configuration and self-adaptation solutions, describes their theoretical formalization and presents their operation.

3.1. Solution overview. Functional architecture

Our solution is based on three different independent abstraction layers. Figure 2 shows the general proposed scheme (reference architecture).

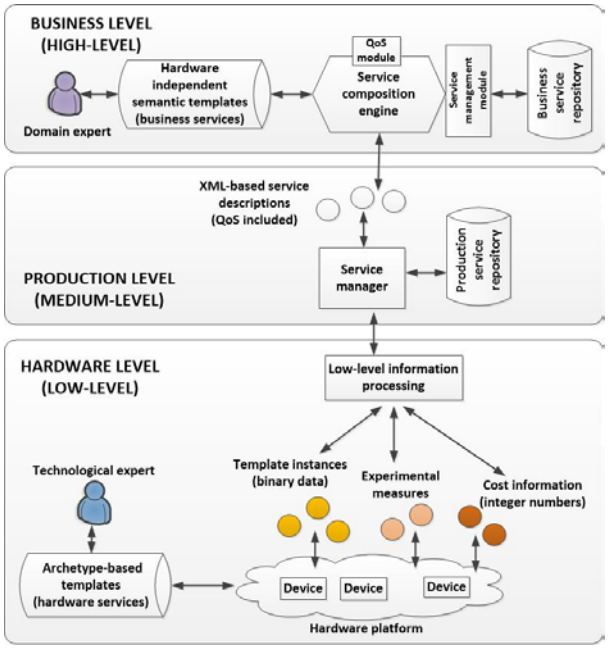

Fig. 2. Reference architecture

In the lowest level (called hardware level) a first expert appears. A technological expert (who is required to deeply know the base technology but which is not required to have knowledge about the application scenario) must indicate the hardware services provided by the infrastructure (currently or, if desired, including future extensions). These services must be embedded into hardware and depends on the devices to have certain capabilities. Then, every hardware service may be expressed as the collection of capabilities which together support the service. A certain capability (such as measuring temperature or adding integer numbers) may be present in several devices, so it can be understood as an archetype. An archetype is the original pattern or model of which all things of the same type are representations or copies [49]. As a certain capability either is present in a device or is not (there are no more options) the instances of these archetypes are binary (yes/no). In conclusion, every hardware service may be represented as a template listing all the archetypes (or capabilities) involved in its execution. The technological expert must create these templates (one for each service) which will be automatically instantiated by hardware devices (using binary data) during configuration process.

During the configuration procedure, binary information from the cited instances may be processed at hardware level in order to reduce the amount of data to be transmitted. The obtained instances, besides, must include a first simple semantic characterization also provided by the technological expert with the low-level templates. The result of this processing is sent to the next level.

At medium-level layer (called production level) services are abstracted from the underlying devices. In particular, obtained instances are translated into a generic service description language (such as XML) where semantic information is automatically completed and configured as Semantic Annotations. Moreover, proprietary APIs and interfaces (typical of hardware devices) are substituted by a uniform and generic API (using REST technologies, for example). The obtained description is totally agnostic in respect to the underlying infrastructure. A first service repository may be created at this level.

The generated generic service description at medium level, however, still contains services which have no value for final users in a SE. In order to create this value, it is added a final high-level layer (called business level). At business level a domain expert must indicate the relevant services for his application, including their most significant parts and their meanings. This information (provided using a domain description language) makes up a collection of semantic templates. These templates are introduced in a composition engine together with the description obtained from the production level. Then, the engine finally creates and defines the collection of services which are in theory available in the SE (by means of semantic matching).

The process previously described, however, does not take into account factors such as the minimum expected performance by SE users from a service. Therefore, in our solution, basic information about available services is complemented with additional data about Quality-of-Service. Moreover, these QoS information is employed in a self-adaptation mechanism which is triggered once finalized the configuration process (see Section 3.3).

In this work we call “QoS information” any representation of the system performance pertaining a certain service. This representation can be numerical (quantitative), but it might consist of only one value or of a collection of them. QoS may be represented by means of technical measurable parameters (jitter, delay, packet losses rate, etc.) or through dimensionless values obtained from the application of functions to the cited parameters in order to obtain representative ratios (availability, invocation cost, etc.). QoS is, at the end, employed to indicate the current value of certain important network parameters. Besides, QoS is used to evaluate if network parameters are compat-

ible with the expectances of users (which should also indicate the desired QoS, usually through representative ratios as employing network parameters requires technical knowledge). Other definitions of quality (not based on technical parameters and/or including the user perception) are not considered in this work.

Two different information sources are considered into the QoS calculation.

On the one hand, periodical experimental measures about some remarkable network parameters are done (delay, jitter, transmission rate, etc.). These measures are performed by the low-level processing module. The results depend on the device positions (if mobile), time, interferences, etc. As these measures may change very dynamically, the evaluation period will be a critical parameter in our solution (during that period service catalogue will not be updated).

On the other hand, information about the cost of executing a certain service in the hardware platform is collected. This information is provided by devices to the low-level processing module in an autonomous way (without a previous request). Costs are integer numbers representing the expense in battery charge, RAM memory, CPU and processing time (among other factors) which the execution of a certain service (or capability) causes in each device. The algorithm used to obtain these costs is not the focus of this article (various proposals may be found in the literature). Cost information, in general, changes in a very slow way (it depends on hardware and embedded software which is not variable), so devices simply notify variations to the processing module when occurring.

The low-level information processing module analyzes the received information (costs and experimental measures) and obtains hardware-independent remarkable ratios representing the offered QoS (both the maximum QoS and the guaranteed QoS). In that way, abstraction layers continue being independent as low-level information is not sent to the production level. Moreover, the entire system does not have to be dimensioned to accept a data flow which grows as the hardware-platform scales. It is enough to adequately design the infrastructure.

At production level, the QoS information is integrated into the same XML service descriptions previously described. Finally, this information, at business level, is employed to calculate a weighted quality value which is compared with some threshold defined in the system. The service management module, then, depending on this comparison decides if a service is available, unavailable or deleted.

It is important to note that recalculating the service composition is not necessary in order to update a service state, as it depends only on QoS information. In that way, service catalogue is updated in a very fast way, as the most time consuming step (the service composition process, as evaluated in some papers [50][51]) is avoided. As we are seeing later, with this self-adaptation mechanism (as can be seen is totally automatic) the service composition process must be executed again when a new service has to be added.

The technologies described above must be supported by various functional components. Several functional architectures for service-oriented Smart Environments have been proposed [52] [53]. Any of them may be complemented with the proposed technology, if adequate components are added. In that way, the proposed functional architecture for supporting our proposal can be seen in Figure 3. Various components can be distinguished.

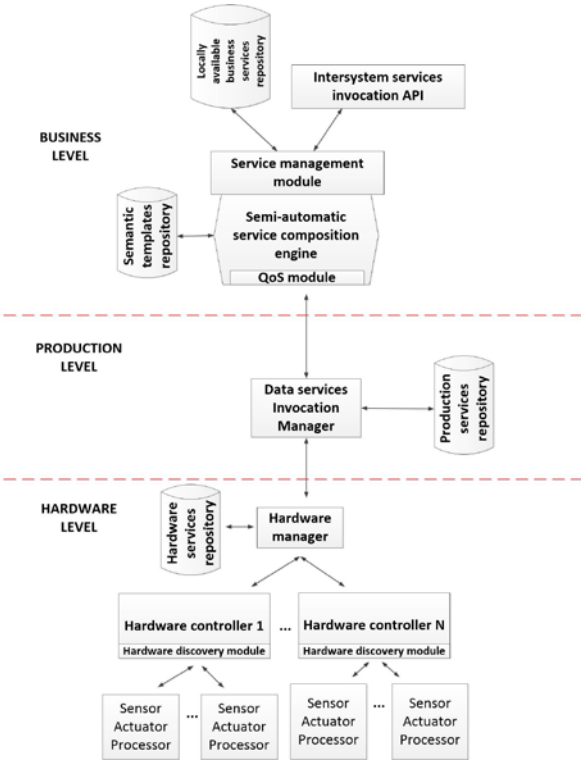

Fig. 3. Functional architecture

– *Sensors, actuators and processors*: They include all the traditional devices present in the hardware infrastructure of Smart Environments and ubiquitous computing systems. From the sim-

plest sensors to the called Smart Objects [54] are included.

- *Hardware controller*: This component is responsible for discovering, setting up and managing the hardware devices. Its main utility in our solution is to obtain and instantiate the low-level archetype-based templates (from *Hardware manager*). The architecture includes as many controllers as required by the hardware platform. Typically, each smart device [55] implements its own hardware controller and traditional devices are connected to a different hardware controller acting as broker. Each hardware controller includes a *Hardware discovery module* which obtains the necessary information to instantiate the cited archetype-based templates (using any of the existing technologies [56])
- *Hardware manager*: It coordinates the operation of the different *hardware controllers*. It manages the archetype-based templates and the instances generated by the underlying *hardware controllers*. Basically, it processes all the archetype-based templates describing every hardware service, in order to create a unique template (called master template) containing all the relevant capabilities (i.e. capabilities which are named at least one time in the service templates). This master template is sent to the hardware controllers which have to instantiate it. Moreover, it processes the low-level data to be submitted to medium-level components. It also maintains information about the different locations (devices) where a certain capability is available (although this problem is more related to the SE operation and it is not analyzed in this work).
- *Hardware services repository*: It contains all the necessary information about the hardware services (sometimes also named as atomic services). It stores the archetype-based templates, as well as the simple semantic information associated with each one. It also contains the master template and the generated instance. The *hardware manager* uses that information to decide which atomic services are available.
- *Data services invocation manager*: This production level component makes independent the hardware platform (at low level) and the service domain (at high level). It translates the archetype-based template instances into a generic service description document and offers to the business level components a service interface based on generic technologies (such as REST) which hides the proprietary and heterogeneous APIs employed by hardware devices. Thus, the resulting description is totally agnostic from the underlying devices.
- *Production services repository*: This is a secondary component which stores the list of production services described by the data services invocation manager in the corresponding document. It also stores the relation between the proprietary hardware interfaces and the generic one (based on REST technologies) offered to business level components.
- *Semi-automatic service composition engine*: This component composes high-level services using semi-automatic techniques and the information provided by the *Data Services Invocation Manager*. Semi-automatic techniques require some human intervention, but in a time previous to the composition process. In our case, domain experts must create a collection of semantic templates describing high-level services (business services). Following a matching mechanism, the semantic content of the medium-level service description is employed to instantiate the semantic templates. As a result, a traditional executable XML service description will be obtained. In this paper we are not discussing the particular matching algorithm employed in the process of services composition. Any of the proposed solutions in the research literature, such as [57] or Hydra Project [58] will be valid.

  This component, besides, include two important modules. The *QoS module* transforms the production level data about quality into a unique parameter (for each business service) through a weighting and aggregation procedure. The result is sent to the *service management module* which compares it to the minimum thresholds indicated by users and decides if a service is available, unavailable or deleted.

Finally, once determined the available composed services, they may be registered in a *Locally available business services repository* so they can be called from local applications, and/or in a public remote repository through the Intersystem services invocation API, so they can be called from applications in the cloud.

3.2. Self-configuration solution: mathematical formalization

Using the architecture proposed on Figure 3, the following self-configuration process may be executed in order to determine the available services in a Smart Environment.

First, each *Hardware controller* discovers the sensors, actuators and processors which it controls, as well as the capabilities of these devices. This process might be performed by means of any of the existing technologies such as IEEE 1451 [59] or the Composite Capability/Preference Profiles (CC/PP) W3C recommendation [60].

In parallel, the *Hardware manager* processes the defined archetype-based templates describing the hardware services. These templates may be described in various languages, although the most appropriate is ADL2 (Archetype Description Language, version 2.0) [61]. From the processing of all templates a master template will be obtained, containing all the capabilities employed in, at least, one service (see Figure 4). Notes describing the basic semantic of services and capabilities are also included.

```
archetype (adl_version=2.0)
hardware-service-example
language original_language = <iso_639-1:en>

definition
 SERVICE_1[id1] matches { --meaning service_1
  capability_1 {'True','False'} --meaning cap_1
   …
  sub_services cardinality matches {0..*} matches{
    SUBSERVICE_1[id2] matches { --meaning sub_serv
        …
    }
  }
}
```

```
…

definition
 MASTER_TEMPLATE[id1] matches {
  capability_1 {'True','False'} --meaning cap_1
  capability_2 {'True','False'} --meaning cap_2
   …
  capability_N {'True','False'} --meaning cap_N


}
```

Fig. 4. Example of an archetype-based template (top) and of a possible master template (bottom) using ADL2

An additional advantage of employing a service-oriented solution is that it is not necessary to model the universe of all possible devices [62]. It is enough to model the hardware capabilities and hardware services of interest for the particular application scenario. The list or the ontology of relevant capabilities may be based on an ad hoc definition or on any of the exiting generic proposals [63][64].

It is important to note that devices may present other capabilities not included in the archetype-based templates. Even, new devices with new capabilities could be added. However, in our proposal (in order to speed up the configuration process) only the required capabilities to execute hardware services are considered. The others, as no service employs them (probably they are not of interest in the application scenario), they will never be invoked, and it has no sense to include them in the configuration process. On the other hand, it must be said that SE are planned systems, not ad hoc deployments. Thus, new devices are always added by experts and (in that case) they also could include the corresponding archetype-based template (if necessary). In future work, besides, this procedure will be investigated to be automated.

Then, each *Hardware controller* obtains the master template and instantiates it. In general, instances of archetype-based templates are XML documents. However, in this case, all the fields in the template are Boolean variables, so the instance may be understood as a binary vector. Thus, after the instantiation process each *Hardware controller* creates a descriptor hd. The descriptor of the i-th *Hardware controller* can be denoted as indicated in Eq. (1).

$$hd_i(t) = \{v_i(t), q_i(t)\} \quad (1)$$

The descriptor includes two elements. v_i is a binary vector where each bit determines the presence or absence of a certain capability (the instance of the master template). q_i is a cost vector where each position indicates the cost of accessing a certain capability (if available) in the i-th hardware controller. As previously said, this cost includes the expense in battery charge, RAM memory, CPU and processing time (among other factors representing the hardware and software configuration). Both elements depend on time, as new sensors could be added, more efficient procedures could be programmed, etc.

In the third step, all *hardware controllers* send their descriptors to the *Hardware manager*. Then, in the *Hardware manager* three collections may be described: HD the collection of descriptors, V the collection of binary capabilities vectors and Q the collection of cost vectors.

$$HD(t) = \{\, hd_i(t)\,, i = 1, \dots, N(t)\} \quad (2)$$

$$V(t) = \{\, v_i(t)\,, i = 1, \dots, N(t)\} \quad (3)$$

$$Q(t) = \{\, q_i(t)\,, i = 1, \dots, N(t)\} \quad (4)$$

Where $N(t)$ is the number of *Hardware controllers* in the Smart Environment. All the variables are

time dependent, as changes in the hardware platform may cause changes in these values.

Strictly speaking, the collection $HD(t)$ already contains $V(t)$ and $Q(t)$. However, for more clarity, three different collections have been defined. This is because the purpose of the sets is very different. While $HD(t)$ is destined to be stored into the hardware services repository (in order to allow the future execution of atomic services), $V(t)$ and $Q(t)$ are going to be employed, independently, in the next steps of the configuration process (so two separated collection are needed).

Using Equations (2), (3) and (4) the *Hardware manager* generates a partitioned matrix $DM(t)$ describing the available atomic services in the hardware platform, as well as the QoS offered for each service.

$$DM(t) = \left(\sigma_d(t) \mid Q_g(t) \mid Q_c(t)\right) \quad (5)$$

As can be seen in Eq. (5), $DM(t)$ is composed of three submatrices.

$\sigma_d(t)$ is a binary column matrix where each row indicates if a certain atomic service is available (each hardware service, when defined through an archetype-based template is provided with a unique numerical integer identifier, which may be employed to determine the position in this column which is referred to it). Q_g is a matrix describing the guaranteed QoS for each atomic service in one particular moment. Each row in Q_g contains the quality parameters of the atomic service to which it is referred the same row in σ_d. Finally Q_c is a matrix describing the context-depending QoS (no guaranteed) for each atomic service in one particular moment. Sometimes, the system presents surplus resources that could be assigned to the existing services in order to improve their quality of execution. These resources, however, cannot be guaranteed, and their availability depends on the system's context.

The relation between the collections $HD(t), V(t)$ and $Q(t)$ and the matrix $DM(t)$ is denoted as a vector function F, represented in Eq. (6).

$$\left(\sigma_d, Q_g, Q_c\right) = \vec{F}(V, Q) = \\ = \left(f_1(V), f_2(Q, \Sigma), f_3(V, Q, \Sigma)\right) \quad (6)$$

As can be seen, this vector function has three components.

Function f_1 employs the collection of instances of the master template (bit vectors) produced by hardware controllers to determine if each one of the described hardware services in the corresponding repository is supported by the infrastructure or not. In a traditional approach, an instance transformation would be applied (using matching techniques) in order to obtain the instances of all archetype-based templates (one per hardware service, equal for all hardware controllers) from the instances of the master template (one per hardware controller, equal for all hardware services). However, this process is complex and heavy [65], and it may not fit teal-time constraints. This problem may be solved if master template instances are understood as bit vectors, as, then, f_1 might be described as a sequence of logical (binary) operations which are executed in a very dynamic and fast way.

For that, when creating the master template, *Hardware manager* should define a new collection of binary templates $C(t)$, identifying for each hardware service the capabilities involved in its execution (7).

$$C(t) = \{c_i, \quad i = 1, \dots, M(t)\} \quad (7)$$

Being $M(t)$ the number of available hardware services in the infrastructure in a certain moment.

The format of these binary templates must be the same as the master template instances: each bit determines the presence or absence of a certain capability in a hardware service. Then, function f_1 may be described as in Algorithm 1.

Algorithm 1. Implementation of function f_1 using pseudocode

N	**Algorithm 1 Function f_1**
1	**Input:** Collection V
2	**Output:** Column matrix σ_d
3	Generate a binary vector of zeros ds
4	Generate an integer i equal to one ($i = 1$)
5	**for each** element v_i in V **do**
6	ds is equal to v_i **bitor** ds
7	**end for**
8	ds is equal to **compl** ds
9	**for each** binary template c_i in C **do**
10	VM = (c_i **bitand** ds)
11	**if** VM != 0 **then**
12	BR is equal to zero
13	**else then**
14	BR is equal to one
15	**end if**
17	σ_d^i is equal to BR
18	Increment i in one unit
19	**end for**

In summary, Algorithm 1 condenses in a binary vector ds (which is initialized in line 3) all the existing capabilities in the *hardware controllers*. For that, an OR operation is applied to all master template instances expressed as bit vectors v_i (lines 5 to 7 in Algorithm 1). Later, for each atomic service it is ap-

plied (using an AND operation) a mask c_i (the corresponding binary template described above) to the binary vector ds in order to identify if it is required any capacity not available on the hardware platform (line 10). The result (VM) is a vector of zeros if all required capabilities are present in the hardware platform or a vector with non-null positions if any capability is not present. Then, the value of the VM parameter may be used to decide if a certain hardware service is available (VM = 0) or not, and, in that way, fixing the corresponding value (the i-th position of the σ_d column) in consequence. Then, in order to advance to the next position in the σ_d column, an integer named as i (initialized with value one in line 4) is updated (line 18).

Functions f_2 and f_3, which together the described function f_1 make up the vector function F, are presented in Equations (8) and (9).

$$Q_g^i(t) = f_2\big(Q(t), \Sigma(\mathrm{t})\big) = \begin{pmatrix} \max\left(\left\{\sum q_j c_i \;\; j = 1, \dots, L(t)\right\}\right) + q_{co}^i \\ R_1\big(Mo(\Sigma)\big) \\ \dots \\ R_S\big(Mo(\Sigma)\big) \end{pmatrix}^T \quad (8)$$

$$i = 1, \dots, M(t)$$

$$Q_c^i(t) = f_3\left(V(t), Q(t), \Sigma(\mathrm{t})\right) = \begin{pmatrix} \min\left(\left\{\sum q_j c_i + \sum \mu_i q_j v_j \;\; j = 1, \dots, L(t)\right\}\right) + q_{co}^i \\ R_1\left(P_{90}\left(\Sigma\right)\right) \\ \dots \\ R_S\left(P_{90}\left(\Sigma\right)\right) \end{pmatrix}^T \quad (9)$$

$$i = 1, \dots, M(t)$$

Briefly, function f_2 creates a vector describing the guaranteed global cost of executing a certain hardware service (the i-th service) and the guaranteed quality parameters associated. In particular, the first component, it adds the cost of accessing to all the capabilities which make up an atomic service, considering the maximum value among all the possible location where the service is available (being $L(t)$ the number of locations where it is available at a certain moment). In fact, in general, the guaranteed cost is the worst case (although certain services have reserved resources, but this is not the focus of this paper). The cost $q_{co}^i$is a correction factor applied if the i-the hardware service is provided by the coordination of various devices and there is a cost of coordination.

The rest of the components are relevant ratios $R_1, \dots, R_S$ (defined by technological experts according to users' needs and the application scenario) calculated from the experimental measures $\Sigma(\mathrm{t})$ (10).

$$\Sigma(\mathrm{t}) = \{\sigma_i(t - pT) \;\; i = 1, \dots, K \;\; p = 1, \dots, P\} \quad (10)$$

$\Sigma(\mathrm{t})$ is a collection of K experimental measures about QoS on the system σ_i (for example, its response time, packet loss rate, etc.). These measurements are periodically repeated and refreshed with a period of T. As we design a solution to maintain updated the system at real time, the value of this period will be critical in our proposal. The collection of experimental measures considers the P past evaluations of the experimental measures.

Examples of relevant ratios are availability, blocking probability, etc. In order to obtain the guaranteed value of these ratios, in their calculation it is employed the most probable value (mode) of the experimental measures, $Mo(\Sigma)$.

On the other hand, the function f_3 creates a vector describing the context-depending costs and QoS (i.e. values for the cost and QoS parameters which are not guaranteed and may be reached or not depending on the system context). The first component, the execution cost, is calculated in the same way than in function f_2. However, in this case, more than one device may be used at the same time to support a certain capability. Then, a correction factor is applied. In particular, μ_i is a numerical vector where each position indicates the modification in the QoS due to the existence of additional available capabilities.

```
<?xml version="1.0"?/>
  <application id=”prosumer_service_1”>
  [rdf:type annot: OntoMatAnnotation
   annot:label “operation description”]
    <grammars>
      <include href="document.xsd"/>
    </grammars>
    <resources>
      <resource>
        [rdf:type annot: OntoMatAnnotation
         annot:objetive “expected result”]
         <method name="GET" id="ID">
          <request>
            <param name="NAME" type="xsd:string"/>
          </request>
          <response status="200">
                    …
          </response>
         </method>
           …
       </resource>
    </resources>
  </application>
```

Fig. 5. Example of a WADL document with Semantic Annotations

In the same line, hardware independent relevant ratios employed to inform about QoS to production

level components are the same than in function f_2. In this case, nevertheless, in their calculation it is employed the 90-th percentile P_{90} (i.e. the value such 90% of measures are equal or worse than it) instead of the mode of measures.

Once $DM(t)$ matrix has been constructed, it is transmitted to the *Data services invocation manager* (production level). In the *Data services invocation manager*, matrices are mapped over a description file (typically an XML file) where atomic services are described as production-level services. For example, using REST services we would use a Web Application Description Language (WADL) file. Moreover, the short notes about semantics included in archetype-based templates are processed and condensed in various Semantic Annotations [36] (for that the master template is also sent to the production level). Figure 5 presents an example of this document.

As can be seen on Figure 5, Semantic Annotations are included in square brackets, describing the "meaning" of each application (prosumer service), request, etc. Once created, XML files are sent to the *Semi-automatic service composition engine*.

Then, the engine composes a collection of high-level (or business level) services $S(t)$ using the production services described in the WADL document, as expressed in Eq. (11). The collection of composed services is time dependent as it may be change depending on the hardware platform configuration.

$$S(t) = \{s_j, \quad j = 1, \dots, I\} \tag{11}$$

Where $I(t)$ is the number of composed services created in the semi-automatic engine.

In order to create this collection of business services, the composition engine must apply a semantic matching technology to the production service description file, together with the semantic templates for business services defined by domain experts. These templates describe the parts (invocations) a service should have, as well as their meaning. Comparing these templates with Semantic Annotations in XML description documents the collection of service is created (applications and requests with the same meaning than desired by users are connected in order to build the high-level services). Business services (as well as semantic templates) are described, then, using any of the existing executable workflow description language (YAWL, for example [66]), probably complemented with Semantic Annotations. In respect to the semantic matching technology, any of the techniques described in the literature may be employed [67][68].

In that way, the j-th service of the collection can be denoted as in Equation (12).

$$s_j = \{XML, QoS_g(t), QoS_c(t), BC(t)\} \tag{12}$$

Where XML describes the service's input, output and its decomposition in prosumer services. QoS_g represents the final guaranteed QoS for the composed service (expressed as a numerical vector where each position represents the value of one quality parameter). QoS_c expresses the context-depending QoS for the final composed service. And BC represents the behavior constraints for the composed service (such as time or range constraints for service availability).

The XML and BC elements are generated during the service composition process, by means of the cited semantic matching technique. However, QoS_g and QoS_c are calculated from quality parameters of all production services (see Eq. (8) and Eq. (9)) which make up a composed service, using Equations (13) and (14) in the *QoS module*

$$QoS_g = qual\_g\left(Q_g^1, \dots, Q_g^{R_1}\right) \tag{13}$$

$$QoS_c = qual\_c\left(Q_c^1, \dots, Q_c^{R_1}\right) \tag{14}$$

Where Q_g^r are the guaranteed quality parameters of the r-th production service provided by physical devices which is part of the composite service. The Q_c^r parameters have the same interpretation as the above, but applied to context-depending quality parameters. Finally, R_1 is the number of production services provided by physical devices which are part of the j-th composite service.

Several implementations for the functions $qual\_g$ and $qual\_c$ may be proposed. In this paper, we define them as indicated in Eq. (15) and Eq. (16). In that way, QoS_g and QoS_c are two real vectors, although in other implementation could be B-dimensional complex or integer vectors.

$$QoS_g = \sum_{i=1}^{R_1} \beta_i Q_g^i \tag{15}$$

$$QoS_c = \sum_{i=1}^{R_1} \beta_i Q_c^i \tag{16}$$

Where β_i is the relative weight of the QoS due to the i-th production service provided by physical devices in the total QoS.

Once obtained the entire collection of composed services by the *Semi-automatic service composition engine* (and the corresponding QoS), they are registered in the *Locally available services repository*. Through this repository, composed services can be

called from local applications, and/or from remote applications through the Intersystem services invocation API.

3.3. System self-adaptation: mathematical formalization

The described procedure in the previous section generates a fixed list of services, associated each one with two parameters representing the aggregated quality. However, the situation of the hardware platform may change, and the provided services should get affected in consequence.

Among all the described steps, the most time consuming is service composition in the *Semi-automatic service composition engine* (as indicated by various authors [51][65]). The rest of operations are simple matrix calculations which might be executed at real-time. The objective, then, is to define a self-adaptation policy which does not need to repeat the entire services composition process.

Table 1. Services' states in Smart Environments

State	Description
Available	A service is available when all the atomic services which compose it are available, and the guaranteed QoS of the composite service surpasses the established QoS thresholds.
Unavailable	A service is unavailable either when one of the atomic services which compose it becomes unavailable, or when the guaranteed QoS of the composite service falls below the established QoS thresholds. A service which is considered unavailable may return to the available state without refreshing the service composition process.
Deleted	A service is considered as deleted when gets unavailable for a time which passes a certain limit. A service which is considered deleted only may return to the available state after refreshing the service composition process in the semi-automatic engine.

In order to reach this objective, we associate to each composed service a *service state.* Thus, each composited service may present three different states: available, unavailable and deleted. Table 1 describes the conditions which must fit a service to present each state.

As can be seen, in Table 1 it is referred some QoS thresholds which all services must pass to be considered available. These thresholds are applied in a two-phase process in the *Service management module*.

In the first phase each quality parameter in QoS_g must pass a limit k_i as expressed in Eq. (17).

$$QoS_g^i > k_i \quad i = 1, \dots, B \tag{17}$$

Where B is the number of considered quality parameters in the system (i.e. the number of elements in QoS_g). Considering the functions proposed in Eq. (15) and Eq. (16), B is equal to the unit (in other implementation it could present greater values). In order to abbreviate the notation we also can denote Eq. (17) as indicated in Eq. (18).

$$QoS_g > K \tag{18}$$

Where $K = \{k_i \quad i = 1, \dots, B\}$

If the first phase is verified, then it is evaluated a second phase. In this new phase the total guaranteed QoS QoS_{to} must pass a limit QoS_{th} as indicated in Equation (19).

$$QoS_{to} > QoS_{th} \tag{19}$$

The expression which allows obtaining the total guaranteed QoS from the vector QoS_g is as expressed in Equation (20).

$$QoS_{to} = \sum_{i=1}^{P} \omega_i \frac{QoS_g^i}{\lambda_i} \tag{20}$$

Where λ_i is the standard value of the i-th quality parameter and ω_i is the relative weight of the i-th parameter in total estimation of the offered QoS.

Using our proposal typical fluctuations in the state of hardware devices do not causes the execution of a new service composition process. This process only has to be initiated in two situations:

- When a previously deleted composed service is available again
- When a new hardware service is discovered by the *Hardware manager*

In the first case, even when the service is considered *deleted*, both the *Semi-automatic service composition engine* and the *Locally available services repository* maintains in memory the service information. Thus, the complete composition process does not have to be executed, and only a new collection

modifying the affected service has to be generated in the engine.

In the second case, the complete composition process has to be repeated. New composed services could appear and they must be calculated and incorporated in the repository. Nevertheless, if the employed service composition technique allows conditional composition [50], this process could be simplified by restricting it to an expansion of the composed services due to the appearance of a new atomic service. If this solution is available, the *Data services invocation manager* would create a special XML description file indicating to the *Semi-automatic service composition engine* the new physical platform situation and the condition on which the new composition process should be based.

It must be noted that any change in the physical layer which does not cause a modification in the atomic services is not propagated towards the *Data services invocation manager*. As the proposed functional architecture is made of independent abstraction layers, changes in the hardware level which do not affect the hardware services are directly managed by the *hardware controllers* or the *Hardware manager*. This approach allows an important reduction in the average needed time to update the system state.

3.4. Extension to hybrid systems

Service-oriented architectures for SE present the possibility of including humans as service providers. However, most configuration and adaptation solutions for SE do not consider this situation (see Section 2). On the contrary, our proposal may be easily extended to hybrid systems (those which include both humans and devices in their low-level infrastructure).

Previous works [69] have proved that the number of humans in a Cyber-Physical System, together with their motivation state (which represents the execution quality), may characterize the available human provided services in the system. Therefore, in order to configure human provided services, our solution must collect and include this information in the previously described configuration and adaptation processes (Section 3.2 and 3.3).

Then, in the case of existing humans in a SE, sensors focused on acquiring information about people should be included into the hardware platform (called *user-focused devices*). These devices are controlled by *user-focused controllers*, where data analysis algorithms are included (in order to deduct the number of people and their motivation state from sensor information). With this information, a description matrix $UM(t)$ should be defined (21) (equivalent to the matrix $DM(t)$, employed to describe the low-level services provided by hardware devices).

$$UM(t) = \left(\sigma_h(t) \mid HQ_g(t) \mid HQ_c(t)\right) \qquad (21)$$

As matrix $DM(t)$, $UM(t)$ it is made of three submatrices. σ_h is a binary column matrix where each row indicates if a certain low-level service provided by humans is available. HQ_g is a matrix describing the guaranteed quality and HQ_c is a matrix describing the context-depending QoS (no guaranteed). Both matrices are calculated from the users' motivation state, which may be completed with other estimations such as the stress level.

Algorithms and functions employed to construct the matrix UM from sensor information have been described in previous works [69]. Thus, in this proposal we are focusing on the configuration process more than in the data analytics. Moreover, for human provided services, archetype-based templates are not required; only one "master pseudo-template" must be built. The objective of this pseudo-template is to register the services a certain quantity of people may provide in the application scenario (never to be instantiated as regular templates). Then, this pseudo-template lists these services (categorized depending on the number of people) with their meanings (see Figure 6).

Once matrix UM is constructed, the appropriate module (usually called "*user manager*" [69], see Figure 8) sends it to the production level (to the *Data services invocation manager*).

```
…

definition
 SERVICES_ONE_PERSON [id1] matches {
  Service_1 {'True','False'} --meaning serv_1
  Service_2 {'True','False'} --meaning serv_2
   …
  Service_N {'True','False'} --meaning serv_N

}
SERVICES_TWO_PEOPLE[id2] matches {
   …
}
…
```

Fig. 6. Master pseudo-template for human provided services

At production level, the *Data services invocation manager* includes human provided services in the created XML service description document, together with services provided by hardware devices. This module makes no difference between both types of services. If required by business level components, it

may include a new label indicating if a service is provided by devices or by humans but it is not mandatory.

From this point, the configuration process could be performed in the same way as if only devices were included into the low-level infrastructure. However, the calculation of the total QoS in Eq. (15) and Eq. (16) affects the self-adaptation solution, which should consider the dynamical behavior of humans, in order to avoid unnecessary calculations. Thus, in case of existing humans as service providers, we propose to modify these equations along expressions (22) and (23).

$$QoS_g = \alpha_1 \left(\sum_{i=1}^{R_1} \beta_i Q_g^i \right) + \alpha_2 \left(\sum_{i=1}^{R_2} \gamma_i \left(\delta_1^i \mathcal{LP}[HQ_g^i] \right) \right) \quad (22)$$

$$QoS_c = \alpha_1 \left(\sum_{i=1}^{R_1} \beta_i Q_c^i \right) + \alpha_2 \left(\sum_{i=1}^{R_2} \gamma_i \left(\delta_1^i \mathcal{LP}[HQ_c^i] \right) \right) \quad (23)$$

Where α_1 is the relative weight of the QoS due to the physical devices in the total QoS, and α_2 is the relative weight of the QoS due to production services provided by humans in the total QoS. β_i is the relative weight of the QoS due to the i-th production level service provided by physical devices in the total QoS due to hardware devices. γ_i has the same meaning of β_i but applied to services provided by humans. Finally, $\mathcal{LP}[\cdot]$ represents a low-pass filter, focused on removing the very high-frequency variations in the quality parameters due to humans, as their behavior tends to be very dynamic, but QoS is not really affected for these modifications [69]. The particular configuration of this filter will depend on the application scenario.

No more modifications are required.

3.5. Implementation

Once described the mathematical formalization, in this subsection we present a completely useful specific implementation of the proposed self-configuration technology (others could be proposed and developed).

For this first implementation we have selected as deployment scenario a manufacturing scenario where a Smart Environment for product control, traceability and workers performance monitoring has been deployed. In this scenario several types of sensors, actuators and physical devices could be included. However, in this first evaluation only three instruments (smart objects) were considered: a RFID-enabled glove [70], a worktable provided with NFC readers and LEDs for user notification [70], and an autonomous wireless device with e-paper display and low-energy communication for information presentation [71].

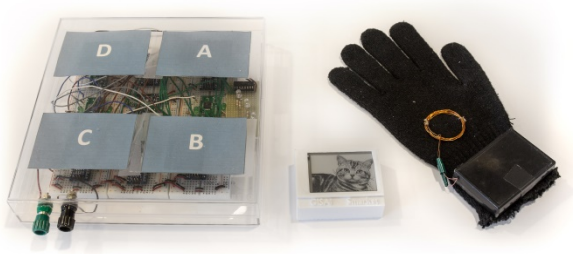

Fig. 7. Prototypes implementation

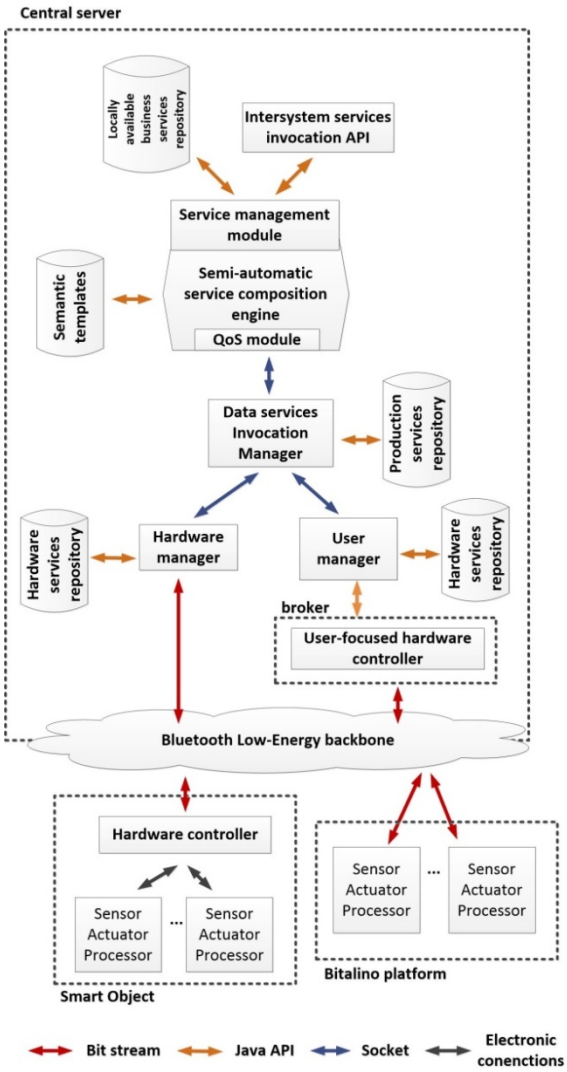

Fig. 8. Implementation architecture

All these devices were based on commercial technologies. Figure 7 shows a prototype of each one of the described devices. The final deployment scenario would include as much devices as necessary, so various copies of each one of the presented prototypes in Figure 7 could be considered.

On the other hand, human service providers were monitored by means of a Bitalino [72] platform (the user-focused device), whose generated data about the humans' state (stress level, heart rate, etc.) were used as input to determine the people's motivation state using any of the proposed algorithms in the existing literature [69].

The software components (such as the *Semi-automatic service composition engine*) were implemented in Java technology. These components were deployed in a central server. All the physical components were connected through a Bluetooth Low-Energy backbone. This technology (due to its low power consumption) fits perfectly the requirement of the proposed scenario, where devices are provided with a small battery as power supply.

Figure 8 shows a detailed implementation architecture where all the defined components in the functional architecture (Figure 3) are mapped. It is important to note that components dedicated to control humans acting as service providers (the called user-focused components) have also been included. In particular, as can be seen, the prototypes showed on Figure 7 implement an embedded *Hardware controller* but Bitalino platform requires a broker acting as *User-focused hardware controller* which is implemented using Java technologies in the central server.

Data transmission among the components implemented using Java technologies is based on the common Java serialization solutions (such as the *ObjectOutputStream* class). The transmission of the data generated by the Bitalino platform and the *user-focused hardware controllers* deployed as brokers in the central server employs the proprietary solutions provided by Bitalino software.

Finally, the transmission of the descriptors hd_i from the *hardware controllers* (deployed in the smart objects) to the *Hardware manager* (running in the central server) is based on a Bluetooth binary stream. Descriptors hd_i may be easily serialized sending at first the binary vector v_i and later the cost vector q_i. In the same way, binary stream is employed to transmit the master archetype-based template from hardware manager to hardware controllers.

Now, considering the proposed implementation, the structure of the different vectors, matrixes and collection described in Section 3.2 may be fixed. Moreover, the selected semantic service composition technology must be also presented.

Table 2 presents the vector and matrix format for all the data elements described in the previous subsection (3.2). As can be seen, for simplicity (templates are much more easily created), in this case we have selected the hardware services coincident with some hardware capabilities, although that is not mandatory (in general, an atomic service uses various hardware capabilities). This option, nevertheless, does not affect the number of mathematical operations performed during the self-configuration process, so results about the convergence time and scalability are valid. Moreover, reduced-dimensions vectors and matrix have been selected in order to clarify the proposed technology, though in practice they may be as big as needed.

On the other hand, matrices HQ_g and HQ_c are not included in Table 2, as they are usually made of dimensionless parameters which do not match with any common definition. In this work, we are assuming these matrices are numerical values (real numbers) obtained as indicated in any of the previous works on this topic [69].

In general, the particular choices described in Table 2 are stablished with the objective of obtaining results which could be easily interpreted in Section 5. Matrices v_i, q_i and σ_d are directly deducted from the capabilities of hardware prototypes showed on Figure 7. Human provided services in matrix σ_h are selected to be simple activities, so no specific formation or great (physical or mental) efforts are required from people (what makes simple the development of the experimental validation). In respect to QoS measures and matrices (i.e. matrices Σ, Q_g, Q_c QoS_g and QoS_c), parameters are selected as they are the most adequate to represent the proposed production and business services. Other typical parameters such as bandwidth, reliability of data precision have no sense in a scenarios based on RFID readers (as they do not affect in a determinant way that technology). Moreover, the selected parameters are easily understandable which is helping us to understand the results obtained in Section 5. In any case, the two parameters chosen represent two basic dimensions of a quality service: its delay and its reliability.

In respect to the semantic service composition, a technology focused on web service matching is implemented [43]. Finally, in the next sections, for simplicity in the experiment design, all relative weights have been fixed following a uniform distribution.

Table 2. Data format in fast self-configuration process for Smart Environments

Element	Pattern
v_i and q_i	$(NFC\ reading \quad Power\ LEDS \quad Signal\ processing \quad Measure\ temperature \quad Show\ a\ picture)$
σ_d	$(NFC\ reading \quad Power\ LEDS \quad Measure\ temperature \quad Show\ a\ picture)$
σ_h	$(Carrying\ object \quad Remove\ object\ off\ table \quad Place\ object\ on\ table)$
Σ	$(Response\ time \quad Availability)$
Q_g , Q_c	$\begin{pmatrix} Cost(element\_1) & Response\ time(element\_1) & Availability(element\_1) \\ Cost(element\_2) & Response\ time(element\_2) & Availability(element\_2) \\ \cdots & \cdots & \cdots \\ Cost(element\_N) & Response\ time(element\_N) & Availability(element\_N) \end{pmatrix}$
QoS_g and QoS_c	$(Cost \quad Response\ time \quad Availability)$

4. Experimental validation

The proposed solution presents the advantage over previous proposals to be able to be applied in real-time applications and scenarios. In order to reach this objective three different designs have been included:

- The semantic service composition is not performed at operation time but during the configuration phase. As this time consuming process does not have to be executed each time a service is called, the service invocations are highly accelerated.
- A self-adaptation policy based on the concept of QoS is deployed. As service composition is performed before the system operation it is required a solution to update the list of available services at real-time. Our proposal classifies services as available, deleted or unavailable using simple mathematical operations for QoS calculation.
- Information from the low-level platform in order to calculate the QoS is collected using binary data formats and simple logical operations. In that way, adaptations may be performed with a much reduced time consume, as almost no information must be transmitted (so system resources are not consumed).

In this section, the proposed technology is evaluated as a valid solution for real-time operation, considering the three mentioned designs. Additional advantages such as the possibility of including humans as service providers, and the independence among the different abstraction layer in the architecture are not evaluated as they are directly deducted from the previous descriptions.

The proposed experimental validation mixes two different techniques. On the one hand, a real deployment was performed in the first experiment; but, on the other hand, a simulation environment was employed in the second, third and fourth experiments.

In the first experiment the required convergence time for the initial system configuration and the medium convergence time needed by the system to react to changes in the physical level are evaluated. For that, a Smart Environment as described in Section 3.5 employing the proposed self-configuration technology was deployed at the Technical University of Madrid. Three different scenarios were evaluated:

- In the first scenario fifteen smart objects made up the hardware platform: five cyber gloves, five worktables and five electronic ink displays.
- In the second scenario only seven smart objects were included: three electronic ink displays, two cyber gloves and two worktables. Additionally, eight humans acting as service providers are also part of the environment.
- Finally, in the third scenario, exclusively fifteen humans acting as service providers made up the Smart Environment.
- For each scenario, the experiment consisted of two phases.
- In the first phase the system was powered on and the time employed in the initial self-configuration measured. Then, the system was powered off again. This routine was repeated twenty-four times four each scenario.

– In the second phase, for each scenario, the system was powered on, and it was kept operating for three hours. Data about the different system adaptations performed, and the convergence time employed in each one of these adaptations were collected. As said in Section 3.2, the refreshing period T of the experimental measures $\Sigma(t)$ employed to calculate the QoS and update the list of available services is critical. Different values for this parameter were also considered in each in of the proposed scenarios during the second phase. Table 3 shows the number of registers obtained for each scenario and value for the period T in this second phase.

Table 3. Number of registers in the second phase of the first experiment

Scenario	**Refreshing period (s)**			
	5	**10**	**30**	**60**
First scenario (fifteen devices)	142	106	97	89
Second scenario (seven devices and eight humans)	248	173	145	109
Third scenario (fifteen humans)	346	263	207	192

In the second experiment, the evolution of the convergence time of the proposed self-configuration technology when increasing the number of elements in the physical platform (both, humans and devices) is evaluated. In this experiment a simulated deployment was employed (mainly, due to the difficult of performing experiments involving greats amount of people).

The simulation was performed using the NS3 simulator [73]. In that simulator every element was represented by an agent, which implements the proposed self-configuration technology. Data gathered in the first experiment were used to model the behavior of the different devices and humans (then the obtained results present a good quality). Twelve different simulated scenarios were evaluated. In the first scenario only one element was included in the physical platform; in the last scenario 10000 (ten thousands) elements were considered. For each one of these twelve scenarios, three simulations were performed. Each simulation represented three hours of system operation. Data about the self-adaptations performed were gathered (specially the notifications about the start and the end of the self-configuration process).

The third experiment was designed to evaluate the evolution of the resource consumption (rate) in the system due to the proposed technology when increasing the number of elements in the physical platform (both, humans and devices). The experiment was really similar to the one performed in the second case. However, some modifications in the simulated scenarios were developed; basically various sinks in order to control the signalization load in the system must be added.

With this new simulation scenario, the same procedure as described for the second experiment was followed (twelve different scenarios including from one physical element to ten thousands, and three simulation performed for each scenario). Then, data about the signalization in the system was collected.

Finally, in the fourth experiment, previous results are compared to traditional configuration and adaptation solution for service-oriented SE. In particular, a traditional smart configuration solution based on semantic service compositions and the transmission of description files among components [36][20] was selected.

5. Results

In this Section, results of the proposed experiment in Section 4 are presented and analyzed.

Figure 9 presents the results of the first phase of the first experiment. Temporal values in the Y-axis were normalized by the maximum value, as the important information is the relation among the results obtained for each scenario.

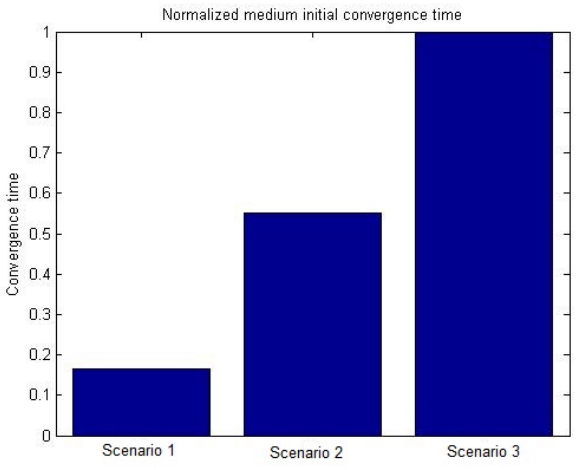

Fig. 9. Normalized medium initial convergence time

Figure 9 represents the medium time obtained from the twenty-four realizations developed. As can be seen, the maximum time is required in the third scenario, when fifteen humans are included in the Smart Environments. This time is 45% less (approximately) if only eight humans are considered (as in the second scenario). If only fifteen hardware devices are included, the required time is a 15% of the required time when fifteen humans were considered (third scenario), and a 65% less than the required time when eight humans and seven devices were included (second scenario).

As main conclusion, it is clear that Smart Environments including humans require more time to configure than system only made of hardware devices. This fact may be caused by two factors.

First, determining the humans' state requires more resources. Hardware components only must transmit two vectors to the *Hardware manager*; however, *user-focused hardware manager* needs to collect various frames of data and apply a data processing algorithm before determining the stress level in humans.

And, second, as the *user-focused hardware controller* is implemented in a software broker running in the central server (instead of in the same hardware device as in the case of smart objects), communications between sensors and controllers tend to be slower and need more time.

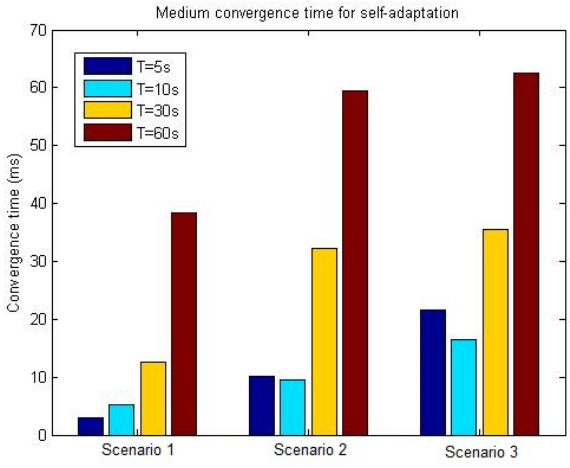

Fig. 10. Medium convergence time for self-adaptation

In respect to the second phase of the first experiment, first we are analyzing data on Table 3. As can be seen, as the refreshing period goes down, the number of adaptation processes performed grows. These additional records correspond to temporal fluctuations which are addressed if the refreshing time is enough short, but which are not detected if not (and then during a certain time the list of available services is not updated).

Figure 10 shows the results of the second phase of the first experiment. Figure 10 represents the medium time required for system adaptation in each one of the proposed scenarios, for the different proposed refreshing periods.

As can be seen in Figure 10, in general, the required time for self-adaptation continues being higher when humans are considered. In particular, the medium time when considering fifteen humans in the Smart Environment is up to three times higher than which obtained in the case of only existing hardware devices. Moreover, in the case of the second scenario (eight humans and seven hardware devices) the convergence time is around a 40% fewer than the value obtained for the third scenario.

Apart from these tendencies, two important facts about the refreshing period must be considered. First, very small values for the refreshing period do not guarantee a shorter convergence time. As can be seen on the results of the third scenario, the convergence time for $T = 5s$ is higher than which obtained for $T = 10s$. That is due to the fact that, for $T = 5s$, the system invests a lot of processing time on doing measures, and algorithm for motivation state calculation (then) cannot take profit from all resources in the user-focused controller.

And, second, if the refreshing period is too high, the convergence time is practically equal to it. In practice, as the required calculation time to perform an adaptation is relatively small, when the refreshing period goes up a certain threshold (around $T = 20s$), the convergence time does not depend on the processing time, but on the waiting time to the next experimental measure.

Nevertheless, despite these observations, the obtained convergence time is very low. In comparison, for example, similar experiments for LTE networks [44] showed that the required time for a handover is around 20ms (and network reconfiguration is perfectly executed at real-time in most cases).

From these results, it is possible to guarantee the proposed solution as useful in real-time applications.

Figure 11 shows the results of the second experiment, where the evolution of the convergence time of the proposed self-configuration technology when increasing the number of elements in the physical platform is evaluated. Results are normalized to the

maximum value as the interesting information is the tendency, not the particular temporal values.

As can be seen, the convergence time is almost invariant with respect to the number of elements in the physical platform. The medium normalized convergence time presents only around a 7% of variation when the number of elements in the physical platform increases in four magnitude orders (from 1 to 10000).

This result, actually, greatly improves the precedents. For example, some previous proposals [27] reported variations of a 600%, approximately, when the number of elements in the physical layer is multiplied by three.

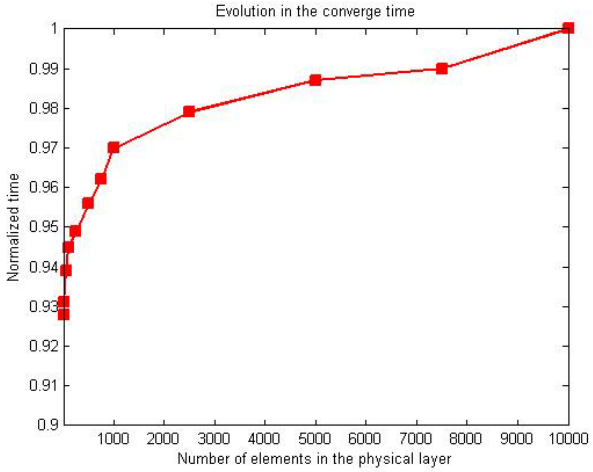

Fig. 11. Evolution in the convergence time vs. number of elements in the physical platform

The explanation of this improvement is a mix of various causes. In the proposed solution, the described functional architecture (Figure 3) presents a tree topology. In that way, information may be compiled in the low layers (i.e. *hardware controller*), and in the top layers (i.e. *hardware manager*, *service composition engine*, etc.) algorithms are independent of the number of elements included in the physical platform. Moreover, in the *hardware controllers* information about the system's state is codified using binary vectors and algorithms are based on logical operations. Thus, the convergence time increases lightly when increasing the number of elements in the physical platform, but the impact is very limited as showed in Figure 11.

Therefore, it is possible to say the proposed technology guarantees the scalability of the Smart Environments, at least considering the required convergence time.

Figure 12 shows the results of the third and last experiment. In this Figure the evolution of the resource consumption in the system is represented versus the number of elements included in the physical layer. The resource consumption seen in one simulation (as percentage) is calculated using the expression of Equation (24).

$$\frac{Bytes\ transmitted\ dedicated\ to\ conf\ or\ adapt}{Total\ of\ bytes\ transmited} \cdot 100 \qquad (24)$$

As can be seen, the resource consumption increases linearly when increasing the number of elements in the physical layer.

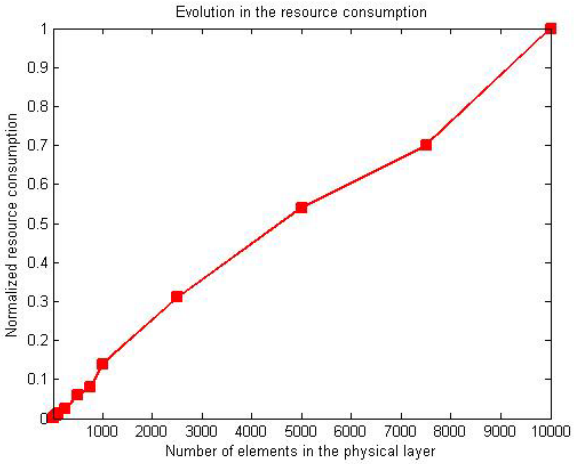

Fig. 12. Evolution in the resource consumption vs. number of elements in the physical platform

This result is coherent with the proposed scenario, where every additional smart object incorporates one *hardware controller* which transmits a fixed amount of bytes of signalization. The same reasoning is valid in the case of humans.

A linear increase improves the behavior of the previous proposals [19], which traditionally present an increase of n^2 order. However, it is a limiting characteristic of the proposed technology which must be taken into account (in particular when designing the Bluetooth backbone or when selecting the appropriate central server for the Smart Environment).

Finally, Figure 13 shows the results of the fourth experiment. In this figure, the required calculation time to perform the system configuration in our proposal is compared with the results obtained using previous technologies.

As can be seen, previous solutions require much more calculation time to perform the system configuration than the proposed technology (up to an order of magnitude higher). The evolution in both cases (the configuration process of the proposed technology and previous proposals) present an exponential evolution created by the service composition engine (whose reasoning time usually grows following this law). However, in the analyzed previous technology the increase is much faster as to the reasoning time of the composition engine it must be added the required time to collect the information from the hardware platform and processing it (which also evolves with and exponential law). On the other hand, the proposed processing algorithm based on simple logical operations, if programmed adequately, requires a constant calculation time, independent from the number of offered hardware services. Thus, the calculation time is only due to the service composition technology selected.

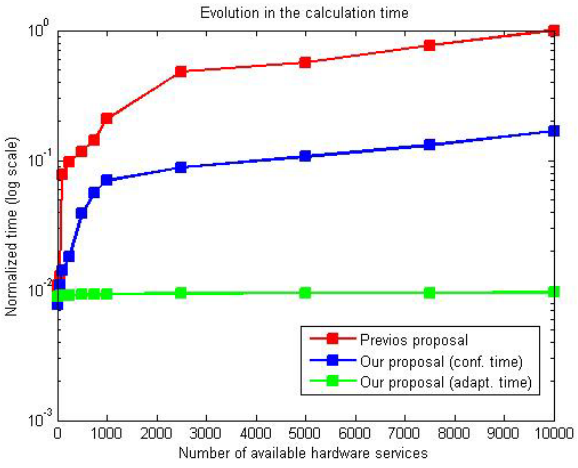

Fig. 13. Evolution in calculation time vs. number of available hardware services in the physical platform

Furthermore, there is a basic difference. The showed calculation time for previous proposals (red line) must be assumed each time a service is invoked. On the contrary, the configuration delay in our solution is only present at the pre-operational time and if a hard refresh is required. In any other case, the self-adaptation solution maintains updated the list of available services (green line). In that case, as seen previously, the calculation time remains almost constant and it is much lower than in the other cases (as the service composition process is not performed).

In conclusion, the proposed technology fulfills the objective of improving the existing technologies in order to be able to be applied to real-time scenarios.

6. Conclusions

Smart Environments are greatly extended, for example, as support for Ambient Intelligence applications. In order to become a real useful technology, Smart Environment must operate at real-time and support automatic self-configuration and self-adaptation.

The purpose of the current paper is to describe a new fast self-configuration technology to be applied in Smart Environment for real-time applications. Additionally, as a novelty, the proposed technology is able to include both service provided by hardware devices and services provided by humans.

The proposed solution is based on a functional architecture made of various independent components which, using low-level templates, simple binary and numeric vectors and matrices, transmit the state of the physical layer to a service composition engine. The engine, then, calculates the available services in the system using sematic templates. The solution is service-oriented and employs the concept of quality-of-service in order to reduce the time required in the self-adaptation process. For that, the quality-of-service is defined as a vector where various parameters are represented (humans' physiological state, available processing slots, availability, etc.). Then, a service calculated by the composition engine is available if each one of the parameters is higher than a certain limit and, besides, the global quality (calculated using a cost function and the quality-of-service vector) is also higher than a threshold. Otherwise, the service is unavailable or deleted.

In this work, an experimental validation consisting of three experiments about the scalability and the convergence time of the proposed solution was also performed. Then, the following conclusions can be drawn from the present study.

First, the proposed solution requires a convergence time which allows its application to real-time scenarios. In particular, the achieved value for the convergence time is lower than obtained for other technologies such as Long-Term Evolution (LTE) networks.

Second, the described solution includes both humans and hardware devices. However, performance is better in scenarios where only hardware compo-

nents are considered. Mainly, this fact is due to the higher resources needed to determine the humans' state; and because an external broker is required as user-focused hardware controller. In future works, mechanisms to obtain the same behavior independently of the physical layer composition should be developed.

Third, in terms of the convergence time, the scalability of the proposed solution is guaranteed. The impact of the number of elements of the physical layer in the convergence time is extremely limited, so large scale deployments may be considered. This is very important, considering concepts such as the Internet of Things and Pervasive Sensing, based on increasing the density of hardware devices in daily scenarios.

And, finally, the limiting characteristic of the proposed solution is the resource consumption. The proposed technology highly improves the performance of the previous proposals, but the resource consumption continues depending linearly on the number of elements in the physical platform. Future works should improve this aspect.

Acknowledgements

The research leading to these results has received funding from the Ministry of Economy and Competitiveness through SEMOLA project (TEC2015-68284-R) and from the Autonomous Region of Madrid through MOSI-AGIL-CM project (grant P2013/ICE-3019, co-funded by EU Structural Funds FSE and FEDER). Borja Bordel has received funding from the Ministry of Education through the FPU program (grant number FPU15/03977).

References

[1] D.J. Cook, Youngblood, L.B. Holder and E.O. Heierman, *Automation intelligence for the smart environment*, in: Proceedings of the International Joint Conference on Artificial Intelligence, 2005.

[2] D. J Cook, S. K. Das, How smart are our environments? An updated look at the state of the art. *Pervasive and mobile computing*, ***3*(2)** (2007), 53-73.

[3] R. Alcarria, T. Robles, A. Morales, S. González-Miranda, *New service development method for prosumer environments*. In: Proceedings of the 6th International Conference on Digital Society , pp. 86-91, 2012

[4] A. W. Colombo, T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, R., J. L. Lastra. *Industrial cloud-based cyber-physical systems*. The IMC-AESOP Approach. 2014

[5] F. Jammes, S. Karnouskos, B. Bony, P. Nappey, A.W. Colombo, J. Delsing, T. Bangemann, *Promising technologies for soa-based industrial automation systems*. In: Industrial Cloud-Based Cyber-Physical Systems, pp. 89-109. Springer International Publishing. 2014

[6] C. L. Wu, C. F. Liao, L. C. Fu, Service-oriented smart-home architecture based on OSGi and mobile-agent technology. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **37(2)** (2007), , 193-205.

[7] D. Cook, D. Das. *Smart environments: Technology, protocols and applications* (Vol. 43). (2004). John Wiley & Sons.

[8] A. Helal, D. J. Cook M. Schmalz. Smart home-based health platform for behavioral monitoring and alteration of diabetes patients. *Journal of diabetes science and technology*, **3(1)** (2009), 141-148.

[9] N. K. Suryadevara, S. C. Mukhopadhyay, R. Wang, R. K. Rayudu. Forecasting the behavior of an elderly using wireless sensors data in a smart home. *Engineering Applications of Artificial Intelligence*, **26(10),** (2013), 2641-2652.

[10] D. J. Cook, N. C. Krishnan, P. Rashidi. Activity discovery and activity recognition: A new partnership. *IEEE transactions on cybernetics*, **43(3)**, (2013), 820-828.

[11] N. Roy, A. Misra, D. Cook. Ambient and smartphone sensor assisted ADL recognition in multi-inhabitant smart environments. *Journal of ambient intelligence and humanized computing*, **7(1),** (2016), 1-19.

[12] G.M. Youngblood. *MavHome: Managing an Adaptive Versatile Home*, Oct. 2003.Website: mavhome.uta.edu [accessed: 02/2017].

[13] S. Munir, J. A. Stankovic, C. J. M. Liang, & S. Lin. *Cyber physical system challenges for human-in-the-loop control*. In: Presented as part of the 8th International Workshop on Feedback Computing. (2013).

[14] D. De, W. Z. Song, M. Xu, C. L. Wang, D. Cook, & X. Huo. *Findinghumo: Real-time tracking of motion trajectories from anonymous binary sensing in smart environments*. In: Distributed Computing Systems (ICDCS), (2012) IEEE 32nd International Conference on (pp. 163-172). IEEE.

[15] V. X. Tran, H. Tsuji. *Owl-t: A task ontology language for automatic service composition*. In: Proceedings of IEEE International Conference on Web Services (2007) , pp. 1164-1167. IEEE.

[16] D. D. Hoang, H. Paik, & C. K. Kim. Service-oriented middleware architectures for cyber-physical systems. *International Journal of Computer Science and Network Security*, **12(1)** (2012), 79-87.

[17] R. Alcarria, T. Robles, A. Morales, D. López-de-Ipiña, U. Aguilera. Enabling flexible and continuous capability invocation in mobile prosumer environments. *Sensors*, **12(7)** (2012), 8930-8954.

[18] F. Zeshan, R. Mohamad. Semantic web service composition approaches: overview and limitations. *International Journal of New Computer Architectures and their Applications* **1(3),** (2011), 640-651.

[19] J. Wang, Q. Zhu, Y. Ma. An agent-based hybrid service delivery for coordinating internet of things and 3rd party service providers. *Journal of Network and Computer Applications*, **36(6)** (2013), 1684-1695.

[20] S. Mayer, R. Verborgh, M. Kovatsch, F. Mattern, Smart Configuration of Smart Environments. *IEEE Transactions on Automation Science and Engineering*, **13(3),** (2014) 1247-1255

[21] H. Hu, J. Zhang, X. Zheng, Y. Yang, P. Wu, Self-configuration and self-optimization for LTE networks. *IEEE Communications Magazine*, **48(2),** (2010), 94-100.

[22] A. Erradi, P. Maheshwari, V. Tosic, *Policy-driven middleware for self-adaptation of web services compositions*. In: proceedings of International Conference on Distributed Systems Platforms and Open Distributed Processing, 2006, (pp. 62-80). Springer Berlin Heidelberg.

[23] Y. Charif, N. Sabouret, An overview of semantic web services composition approaches. *Electronic Notes in Theoretical Computer Science*, **146(1),** (2006), 33-41.

[24] M. S. Familiar, J. F. Martínez, I. Corredor, & C. García-Rubio. Building service-oriented smart infrastructures over wireless ad hoc sensor networks: A middleware perspective. *Computer Networks*, **56(4)**, 1303-1328. (2012).

[25] M. S. Familiar, J. F. Martínez & L. López. Pervasive smart spaces and environments: a service-oriented middleware architecture for wireless ad hoc and sensor networks. *International Journal of Distributed Sensor Networks*, (2012).

[26] C. Gouin-Vallerand, B. Abdulrazak, S. Giroux, & A. K. Dey,. A context-aware service provision system for smart environments based on the user interaction modalities. *Journal of Ambient Intelligence and Smart Environments*, **5(1),** (2013), 47-64.

[27] U. Mönks, H. Trsek, L. Dürkop, V. Geneiß, & V. Lohweg,.. Assisting the design of sensor and information fusion systems. *Procedia Technology*, **15,** (2014), 35-45.

[28] T. Dillon, V. Potdar, J. Singh, & A. Talevski. *Cyber-physical systems: Providing Quality of Service (QoS) in a heterogeneous systems-of-systems environment*. In: Proceedings of the 5th IEEE International Conference on Digital Ecosystems and Technologies Conference (DEST), (2011) (pp. 330-335). IEEE.

[29] T. S. Dillon, H. Zhuge, C. Wu, J. Singh, & E. Chang. Web - of - things framework for cyber–physical systems. *Concurrency and Computation: Practice and Experience*, **23(9),** (2011), 905-923.

[30] A. Smirnov, A. Kashevnik, N. Shilov, A. Makklya, & O. Gusikhin. *Context-aware service composition in cyber physical human system for transportation safety*. In: 13th International Conference on ITS Telecommunications (ITST), (2013) (pp. 139-144). IEEE.

[31] A. W. Colombo, S. Karnouskos, J. M. Mendes. *Factory of the Future: A Service-oriented System of Modular, Dynamic Reconfigurable and Collaborative Systems*. In: Artificial Intelligence Techniques for Networked Manuf Enterprises Management, pp. 459–481. Springer, 2010.

[32] AutoPNP consortium. AutoPNP – Plug and Play for Automation Systems 2014 Available from: http://www.autopnp.com/ (accessed on 6th July 2016).

[33] H. C. Cheng, R. de Lemos, H.Giese, P. Inverardi, J. Magee. *Software Engineering for Self-Adaptive Systems: A Research Roadmap Self-Adaptive Systems*, Springer. Berlin Heidelberg (2009), pp. 1–26, 2009.

[34] T. Wang, C. Niu, & L. Cheng, *A Two-Phase Context-Sensitive Service Composition Method with the Workflow Model in Cyber-Physical Systems*. In: Computational Science and Engineering (CSE), (2014) IEEE 17th International Conference on (pp. 1475-1482). IEEE.

[35] L. Baresi, S. Guinea, & A. Shahzada. *SeSaMe: towards a semantic self adaptive middleware for smart spaces*. In: International Workshop on Engineering Multi-Agent Systems (pp. 1-18). (2013, May). Springer Berlin Heidelberg.

[36] S. Mayer, N. Inhelder, R. Verborgh, R. Van de Walle & F. Mattern, *Configuration of smart environments made simple: Combining visual modeling with semantic metadata and reasoning*. In: International Conference on the Internet of Things (IOT), (2014) (pp. 61-66). IEEE.

[37] L. Baresi, & S. Guinea, *Event-Based Monitoring of Service-Oriented Smart Spaces*. In: IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA) (pp. 123-130). (2015) IEEE.

[38] K. Baina, B. Benatallah, F. Casati, F. Toumani, F. *Model-driven web service development*. In: Proceedings of International Conference on Advanced Information Systems Engineering. 2004. (pp. 290-306). Springer Berlin Heidelberg.

[39] M. Rietzler, J. Greim, M. Walch, F. Schaub, B. Wiedersheim, M. Weber, M. *Homeblox: Introducing process-driven home automation*. In: Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication, 2013, (pp. 801-808). ACM.

[40] J. Hendler, Web 3.0 Emerging. *Computer*, **42(1),** (2013), 111-113.

[41] H. N. Talantikite, D. Aissani, N. Boudjlida. Semantic annotations for web services discovery and composition. *Computer Standards & Interfaces*, **31(6)**, (2009) 1108-1117.

[42] Y. Charif, N. Sabouret. An overview of semantic web services composition approaches. *Electronic Notes in Theoretical Computer Science*, **146(1),** (2006), 33-41.

[43] E. Karakoc, P. Senkul, Composing semantic Web services under constraints. *Expert Systems with Applications*, **36(8),** (2009), 11021-11029.

[44] E. Oren, K. Möller, S. Scerri, S. Handschuh, M. Sintek, What are semantic annotations. *Relatório técnico*. DERI Galway, **9**(2006), pp.62.

[45] D. H. Wolpert, K. R. Wheeler, & K. Tumer. Collective intelligence for control of distributed dynamical systems. *EPL: Europhysics Letters*, **49(6),** (2000). 708.

[46] G. D. M. Serugendo, A. Karageorgos, O. F. Rana, & F. Zambonelli. *Engineering self-organising systems: nature-inspired approaches to software engineering* (Vol. 2977). (2004). Springer.

[47] G. Di Caro, M. Dorigo. *Ant colonies for adaptive routing in packet-switched communications networks*. In: Proceedings of PPSN V - Fifth International Conference on Parallel Problem Solving from Nature, volume 1498 of LNCS. SpringerVerlag, (1998).

[48] N. Jennings, K. Sycara, M. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, **1(1)** (1998) pp. 7–38

[49] V. Bicer, O. Kilic, A. Dogac, G. B. Laleci. Archetype-based semantic interoperability of web service messages in the health care domain. *International Journal on Semantic Web and Information Systems* **1(4),** (2005), 1-23.

[50] E. Sirin, B. Parsia, D. Wu, J. Hendler, D. Nau. HTN planning for web service composition using SHOP2. *Web Semantics: Science, Services and Agents on the World Wide Web*, **1(4),** (2004), 377-396.

[51] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q. Z. Sheng. *Quality driven web services composition*. In: Proceedings of the 12th international conference on World Wide Web, 2003. (pp. 411-421). ACM.

[52] V. Degeler, L. I. L. Gonzalez, M. Leva, P. Shrubsole, S. Bonomi, O. Amft, & A. Lazovik. *Service-oriented architecture for smart environments* (short paper). In: IEEE 6th International Conference on Service-Oriented Computing and Applications (pp. 99-104). (2013) IEEE.

[53] F. Furfari, M. Girolami, S. Lenzi, & S. Chessa. A service-oriented zigbee gateway for smart environments. *Journal of Ambient Intelligence and Smart Environments*, **6(6),** (2014), 691-705.

[54] G. Kortuem, F. Kawsar, D. Fitton, V. Sundramoorthy. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*. **14**, 2010, 44–51.
[55] J. W. Gardner, V. K. Varadan, & O. O. Awadelkarim. *Microsensors, MEMS, and smart devices* (Vol. 1). (2001). Chichester: Wiley.
[56] B. Bordel, D. Sánchez de Rivera, R. Alcarria. *Plug-and-Play Transducers in Cyber-Physical Systems for Device-Driven Applications*. In: proceedings of 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2016 (pp. 316-321). IEEE.
[57] J. Domingues, A. Damaso, R. Nascimento, N. Rosa, An Energy-Aware Middleware for Integrating Wireless Sensor Networks and the Internet, *International Journal of Distributed Sensor Networks*, (2011), pp. 1–19, 2011.
[58] Hydra Project Available from: http://www.hydramiddleware.eu/articles.php?article_id=68. (accessed on 6th July 2016)
[59] K. Lee. *IEEE 1451: A standard in support of smart transducer networking*. In: Proceedings of the 17th Conference on Instrumentation and Measurement Technology, (2000). p. 525-528.
[60] Composite Capability/Preference Profiles (CC/PP) current status page Available from: https://www.w3.org/standards/techs/ccpp#w3c_all (accessed on 6th July 2016)
[61] T. Beale, S. Heard. *Archetype Definition Language*. openEHR Foundation, 2007, 47.
[62] P. Kostelnik, M. Sarnovsky, J. Hreno, M. Ahlsen, P. Rosengren, P., Kool, M. Axling. *Semantic devices for ambient environment middleware*. In: Proceedings of EURO TrustAMI, Internet of Things and Services Workshop. 2008.
[63] M. Compton, P. Barnaghi, L. Bermudez, R. GarcíA-Castro, O. Corcho, S. Cox, V. Huang, The SSN ontology of the W3C semantic sensor network incubator group. *Web semantics: science, services and agents on the World Wide Web*, **17**, 2012, 25-32.
[64] A. Bandara, T. R. Payne, D. de Roure, G. Clemo An ontological framework for semantic description of devices. 2004. Available online: http://eprints.soton.ac.uk/262689/1/DOPoster2.pdf (accessed on 6th July 2016)
[65] N. Silva, J. Rocha. *Semantic Web Complex Ontology Mapping*. Web Intelligence, 2003, pp. 82-88
[66] V. M. Van Der Aalst, A. H. Ter Hofstede. YAWL: yet another workflow language. *Information systems*, **30(4),** 2005, 245-275.
[67] F. Giunchiglia, P. Shvaiko, M. Yatskevich. *Semantic matching*. Encyclopedia of Database Systems, 2009, pp. 2561-2566. Springer US.
[68] M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara. *Semantic matching of web services capabilities*. In: Proceedings of International Semantic Web Conference, 2002. (pp. 333-347). Springer Berlin Heidelberg.
[69] B. Bordel, R. Alcarria, D. Martín, T. Robles, D. Sánchez de Rivera. Self-configuration in humanized Cyber-Physical Systems. *Journal of Ambient Intelligence and Humanized Computing*, 1-12.
[70] B. Bordel Sánchez, R. Alcarria, D. Martín & T. Robles. TF4SM: A Framework for Developing Traceability Solutions in Small Manufacturing Companies. *Sensors*, **15(11),** (2015), 29478-29510.
[71] D. Sánchez de Rivera, R. Alcarria, D. Martín de Andres, B. Bordel, T. Robles, *An autonomous Information Device with E-Paper Display for Personal Environments* In: Proceedings of IEEE International Conference on Consumer Electronics (ICCE), (2016), Las Vegas, NV, 2016
[72] Bitalino homepage. Available from: http://www.bitalino.com/ (accessed on 6th July 2016).
[73] Ns-3 Simulator homepage. Available from: https://www.nsnam.org/ (accessed on 6th July 2016).